The 16th Edition of the Multi-Agent Programming Contest - The GOAL-DTU Team

Jørgen Villadsen(✉) and Jonas Weile

Algorithms, Logic and Graphs Section, Department of Applied Mathematics and Computer Science, Technical University of Denmark, Richard Petersens Plads, Building 324, 2800 Kongens Lyngby, Denmark

jovi@dtu.dk

Abstract. We provide an overview of the GOAL-DTU system for the Multi-Agent Programming Contest, including the overall strategy and how the system is designed to apply this strategy. Our agents are implemented using the GOAL programming language. We evaluate the performance of our agents in the contest and, finally, we discuss how to improve the system based on an analysis of its strengths and weaknesses.

1 Introduction

In 2020/2021 we participated as the GOAL-DTU team in the annual Multi-Agent Programming Contest (MAPC). We are using the GOAL agent programming language [1–4] and we are affiliated with the Technical University of Denmark (DTU). We participated in the contest in 2009 and 2010 as the Jason-DTU team [5,6], in 2011 and 2012 as the Python-DTU team [7,8], in 2013 and 2014 as the GOAL-DTU team [9], in 2015/2016 as the Python-DTU team [10], in 2017 and 2018 as the Jason-DTU team [11,12] and in 2019/2020/2021 as the GOAL-DTU team [13,14].

In 2022 we had the *Agents Assemble III* scenario which further expands upon the *Agents Assemble* scenario used in the 2019/2020/2021 contests. Several new features are introduced to add to the scenario from the previous iterations. One such addition is the possibility for agents to switch between different *roles* that each have a set of strengths and weaknesses; one role might be able to move faster but carry less load. Another addition is that *goalzones* are no longer static and can relocate during a simulation.

In this paper, we describe the strategy behind our implementation and evaluate our performance at the competition. The paper is organized as follows:

- Section 2 covers the overall strategy of our agents and our implementation.
- Section 3 evaluates our agents' performance in the contest.
- Section 4 reflects on the differences between the teams
- Section 5 makes some concluding remarks.

T. Ahlbrecht et al. (Eds.): MAPC 2022, LNAI 13997, pp. 151–164, 2023.
https://doi.org/10.1007/978-3-031-38712-8_6

2 The Strategy and Implementation of Our Agents

Our system for the agent contest is implemented in the multi-agent programming language GOAL [1–4] because GOAL has been our recent framework of choice for student projects as well as for several research collaborations.

In GOAL, each agent is an independent entity that can communicate with other agents through an extensive messaging system and perform actions to interact with the environment. These actions are scenario specific and defined for the particular environment that the agents operate in. The agent keeps both predefined and learned knowledge of the environment in a Prolog knowledge base, which constitutes the mental state of the agent. The agent then continually perceives its environment and updates its mental state based on the changes made to the environment. These changes could be the result of an action that the agent itself performed, or they could be the result of some other entity, including what we might call *nature* (i.e. changes that are not the result of some entity acting in the environment).

The agents operate by simple rule-based decision-making. Predefined rules determine the next action of the agent based on its current mental state and the state of the environment, thereby enabling the agents to react to changes in their environment.

2.1 Discovery

We employ different strategies based on the state of the simulation. Initially, in what we call the *discovery phase*, the agents are only tasked with exploring the map and locating dispensers, goalzones, rolezones, and each other. This is necessary because an agent only perceives objects and entities within its (very limited) field of vision, and all agents start out knowing nothing about the environment they are placed in. Agents then store the location of the resources they discover relative to their starting position, which means that all agents will have different coordinate systems. When two (or more) agents meet they will therefore try to establish a connection by learning the origins of the other agents, thereby enabling the agents to efficiently translate between their respective coordinate systems, allowing the agents to communicate about specific resources.

As agents do not perceive the identities of other agents that they encounter, the agents must instead verify their identities by comparing the perceivable objects in their shared field of vision. Thus, all agents continuously broadcast their immediate surroundings to all other agents during the initial discovery phase. Agents then check for broadcasts that match their own surroundings (including the location of the agents themselves), and if there are no other identical broadcasts, they can safely establish a connection to the other broadcasting agent. To optimize the process, agents will share their knowledge of the coordinate systems of all the other agents they have already connected to when establishing a new connection.

2.2 Task Plans

Once enough agents have connected to one another (we, somewhat arbitrarily, chose this to be half of the agents), a single agent is chosen to become the *task master*.

The task master is responsible for creating plans to solve available tasks in order to score points. The task master ranks the available tasks and tries to solve a specific task at a time. The first step in creating a plan is to request information from all other agents relevant for the task at hand. This includes their current attachments and knowledge of available resources necessary to complete the task. The other agents will then respond with their distance to each type of resource and an estimated delivery time.

The task master then collects all responses and checks whether it is feasible to solve the task by attempting to create a *task plan*. A task plan is a detailed description delegating each block of the pattern to a specific agent, specifying a point of assembly at which the pattern is to be assembled. If the task master is unsuccessful in creating a task plan for the task at hand, the task master will restart the process with an alternative task. On the other hand, if the task master successfully creates a task plan, the plan is sent to each agent that is to take part in executing the plan. Each of the agents receives only part of the plan, namely the steps that the agent is responsible for. An agent will only be assigned to deliver blocks of a single type, and from the task plan, the agent is able to infer exactly where it must position itself to deliver the blocks it is responsible for.

2.3 Solving Tasks

To solve a given task, an agent must first collect the necessary block types, and then move to the point of assembly. In order to efficiently solve tasks, the movement of the agents must be direct and fast. However, because the environment is extremely dynamic, the agents do not maintain an up-to-date representation of the map, and thus we cannot use simple path-finding algorithms without modifications.

Our approach is to mix a heuristics-based approach to global path-finding, with optimal local (i.e. within the field of vision) path-finding using A*. The heuristics based approach naively selects moves that bring the agent closer to its destination as long as possible. Once this simple approach is no longer possible, i.e. there might be something blocking the path, the agent chooses a *way point* among the locations within its field of vision. This way point is chosen to be the location within the agents' field of vision that is closest to the final destination. The agent then uses A* to find the optimal path to the way point within its field of vision, and once the agent reaches its way point, the process then starts over. This very simple approach to path-finding is only made feasible by the availability of the clear action. The possibility to clear any blocks that are in the way of the agent means that it will not get stuck in dead-ends; the agent can always clear its way out. However, there is of course an overhead cost attached to heavy utilization of the clear action, but we think that it is a relatively small price to pay in order to achieve such simple path-finding logic.

When the agent reaches the point of assembly for the task plan, it will position itself relative to the agent submitting the pattern, which will locate itself exactly at the predetermined point of assembly. This means, that assembling the pattern is fast and easy, but also extremely inflexible. The agents know exactly where to position themselves in advance, but if the predetermined point of assembly is blocked and cannot be cleared, the agents must find another place to assemble the pattern.

If, at any point during the execution of the task plan, an agent realizes that its individual plan is either unfeasible or cannot be performed within the agreed-upon time-frame, the agent will try to replan. As long as the agent is able to find a plan that can be performed within the time-frame, there is no need to inform the group of the change of plans. Otherwise, the agent will inform the group that the task is no longer solvable and invoke the taskmaster anew. Likewise, once the agreed-upon deadline for solving the task is reached, the default behavior for the agents is to simply drop the task. This ensures that agents will not be stuck waiting for a failed agent.

3 Evaluation of Matches

In this section, we evaluate all of our matches and highlight what goes wrong and what goes right. A match consisted of three simulations, and a simulation started automatically when the previous simulation finished.

3.1 GOAL-DTU VS GOALdigger

SIM1 (0 : 180)

Clearly, something goes wrong for our team of agents. Around step 200 there are three agents that seem to attempt to assemble the pattern for task 4, but it is clear that they do not agree on the exact coordinates for the point of assembly. Maybe something went wrong with inferring map dimensions, leading to a mismatch in coordinates. But this inference should have been disabled at the contest, due to exactly the concern that it might lead to such a mismatch. Because the simulations were run by another student on his local PC, there is however a chance, that it was accidentally left on.

Later in the simulation, there is a long period of time where the agents do not even try to assemble a pattern. This could be further proof that the inferred coordinates are way off, leaving the master agent unable to create any feasible plans.

Compared to our team, we see that GOALdigger performs significantly better. It appears that the strategy of the GOALdigger team is to focus on the simple tasks—all of the 180 points that GOALdigger scores in this simulation are scored by submitting single-block patterns. However, we also see what seems to be an (unsuccessful) attempt to submit a more complex task, see agent 17 from step 160 and onward.

SIM2 (0 : 370)

In general, agents seem to move rather well even in maps like this one that are mostly blocked; the agents creates a path forward by clearing blocks and finding small passages. However, to highlight an instance where this is not the case, consider agent 19 at step 320. Evidently, the "explore score" that prioritizes different moves according to a number of heuristics is not properly tuned. The agent simply rotates in place for many turns before finally finding another path.

Otherwise, this simulation is very similar to simulation 1. We notice the same exact problem; namely that agents do not even bother trying to assemble any patterns, probably due to a mismatch in coordinate systems.

The strategy of GOALdigger also seems to be exactly the same as in the previous simulation. A clear majority of the points are attained by submitting single-block patterns; the GOALdigger team submits an impressive 31 tasks, 29 of which are single-block patterns, and the remaining two patterns are made up of two blocks.

SIM3 (520 : 410)

Unlike the previous two simulations, we actually succeed in submitting tasks and scoring points. In fact, we end up winning the simulation. We note, that most of our points are due to two submissions of the same complex pattern. This showcases some of the beneficial aspects of our simple strategy—the agents are very efficient at building complex patterns. However, this is only the case under optimal conditions.

We notice several occasions during the simulation where it gets extremely crowded at specific goalzones—especially the GOALdigger agents have a tendency to clutter. Crowded goalzones are a known weakness of our simple strategy, and it makes it difficult for our agents to successfully submit patterns. As an example (where the clutter is actually quite limited, but still our agents fail), consider agent 24 around step 360. The agent does not succeed in performing any action for a large number of turns, and instead simply times out. This could be due to problems with the path-finding algorithm. While observing agent 24, we also notice that it seems like two different groups of our agents are trying to assemble the same pattern at the same place, and they actually end up being in the way of one-another.

As another low-hanging fruit for improving our system, we see that our agents try to submit patterns in OLD goalzones. These old goalzones should simply have been removed from the knowledge base once an agent notices that they are no longer present. As an example, observe agent 18 during step 650 and on-wards. Clearly, the agents are trying to submit task 12, but the goalzone disappeared after step 361.

Finally, we notice that the GOALdigger team succeeds in submitting several two-block patterns; a lot more than in the previous simulations.

3.2 GOAL-DTU VS LI(A)RA

SIM1 (120 : 310)
We manage to submit a number of tasks, but we see a clear indication that our strategy is failing. Consider for example agents 4, 5, and 20 around step 225. It seems that their goal is to cooperate to build and submit task 5, but some slight error in the agreed-upon coordinates is stopping them from accomplishing this goal. The same scenario plays out just 45 steps later (at step 270), where two of the same agents, namely agent 4 and 20, fail at assembling task 6 due to the very same error. Fixing this miscalculation of relative coordinates should be top priority, but it also showcases another important problem—our agents *ought* to be more flexible. It should be easy for the agents to recognize this error, and then dynamically agree on a new point of assembly.

We see that the LI(A)RA team submits a large number of tasks compared to us, and seems to do so quite efficiently. Interestingly, however, the LI(A)RA agents fail to submit any patterns for more than 150 steps (between step 133 and 289), but it is not obvious as to why.

SIM2 (160 : 80)

Again, our agents fare reasonably well in a map with a lot of obstacles. Consider for example agents 8 and 18 from step 230 and on-wards. Both agents must clear their way through obstacles to complete the task, and they succeed in doing so in a quite efficient manner. However, agent 18 does perform some unnecessary clearing operations (where the agent does not even use the cleared space), and thus there is still room for improvement.

We also observe the same errors when trying to assemble patterns as in the previous simulation. From step 260, it seems evident that agents 5 and 9 wish to complete task 5. But despite being present in the goalzone with the necessary blocks, and getting within a single rotation from being able to complete the pattern, they never succeed in actually doing so. Here, our agents should be able to recognize the mistake in the agreed-upon plan, and then realize that agent 5 simply needs to perform a rotation in order to successfully complete the pattern.

SIM3 (0 : 270)

Our agents fail completely this simulation. They do not score a single point, nor do they even try to assemble/submit any task. Again, we think this is because of an error in inferring map dimensions, which can completely paralyze the agents.

3.3 GOAL-DTU VS FIT But

SIM1 (230 : 0)
We fare decently well in this simulation and end up winning it. However, we still see some of the problems touched upon in the previous matches. Further, it is very noticeable in this simulation how agents cannot decide on any tasks themselves—instead, only the taskmaster can delegate task plans. For simple 1 block patterns this approach is extremely inefficient, and we see several cases where agents could have completed tasks by themselves. As an example, consider agent 11 around step 90. The agent seems to be waiting for two other agents in order to complete task 3. However, the other agents fail to show up, and therefore agent 11 must drop the task. The agent actually fulfills all the requirements for task 5 but simply wanders out the goalzone with its block instead of submitting the task. This shows that our agents need more autonomy.

It is also observed, that once an agent is in place and waiting for remaining agents to complete a pattern, they completely ignore the threat of clearing events. Consider agent 3 around step 160. The agent is oblivious to the clear event and does not get out of the way, despite having to move only a single (or maybe 2) cells to get of the radius of the event. Agents should always try to avoid clear events, especially when they are about to complete a task.

SIM2 (630 : 670)

We perform fairly well this simulation but with the same problems of failing to complete task patterns. Consider for example agents 6, 8, 14, and 15 around step 470. The agents have built most of the complex pattern, but agent 8 fails to deliver the last remaining block just two rotations (or moves) away from completing the pattern. However, it seems that the reason is due to the pathfinding algorithm failing/timing out and not a disagreement in coordinates. This follows because agent 8 does not skip its turns (as an agent would do once it is in place) but instead fails to perform any action. Completing this pattern would have meant winning the simulation.

We also notice that, again, the goalzones tend to get very crowded and that our agents do not deal well with this.

SIM3: (0 : 1640)

Once again, a complete meltdown of our system. It is identical to the previous matches where we do not get a single point.

However, we would like to use the opportunity to praise FIT BUT and their active use of roles—in this simulation, it really shows how they benefit from using the diggers to remove obstacles from the playing field in order to gain easy access to goalzones.

3.4 GOAL-DTU VS MMD

SIM1 (480 : 910)

We do fairly well, but are not nearly efficient enough. We observe several cases where we begin assembling a pattern but are not able to finish it in time. Further, we notice the large contrast to MMD that always have many agents building patterns in the same goalzone—we do not utilize the large goalzones well enough. We could easily have more agents work in the same goalzones at the same time.

SIM2 (150 : 780)

A lot of cases of agents trying to submit tasks in nonexistent (old) goalzones. This is a severe waste of resources, and leads to a game where we score very few points.

SIM3 (0 : 1520)

Yet another complete meltdown; see the descriptions of the previous simulations where we score 0 points.

4 Discussion

In this section, we retrospectively discuss and review some assumptions of our strategy, and then we take a brief look at the different teams that took part in this year's contest.

4.1 Strategy

Our strategy is very much developed with the specific scenario in mind (the *Agents Assemble* scenario), both in regards to the actual problem to solve, but also in regards to the specific parameters of the scenario. In particular, our implementation is based upon an implicit assumption that the number of agents in a map is fairly limited—in the history of this scenario, the maximum number of agents per team has been no more than 50. This allows us to take a centralized approached to task planning which, if the number of agents were to increase by a few orders of magnitude, would quickly prove to be inefficient.

Our centralized approach is also built upon an implicit assumption of the actual layout of maps; in order to select the taskmaster, at least half of the agents must be connected. By experience, this rarely takes more than 60 rounds, but in a very large map this might not happen within the duration of a single simulation.

It would be interesting to introduce simulations that challenge these kinds of implicit assumptions in the contest which would force competitors to create more general systems. One possible approach could be to have each competing team design a simulation for the contest (in addition to those provided by the organizers), which we expect would lead to less similarity between simulations.

4.2 Teams

Team	FIT BUT	GOAL-DTU	GOALdigger	LI(A)RA	MMD
Members	3	3	4	5	2
Time	6 man-weeks	30 h	1200 h prog.	80 h prog. + 40 h	896 h
Lines	12000	2000	10000	1100	5407
Platform	Java(+JADE)	GOAL	GOAL	Jason	Python
Returning	Yes	Yes	No	No	No

The table shows that the time each team has spent in order to prepare for the competition varies a lot. The hours spent differs as much as by a factor 40. However, it is also important to note that our team spent the least time preparing for this year's competition, but is a returning team. Thus, almost our entire codebase is reused from last year, but with a small number of updates. Likewise, the number of lines of code varies quite a lot between teams. We notice that the teams that have used dedicated multi-agent programming languages do not necessarily need fewer lines of code; both the two teams with fewest lines of code, as well as the two teams with the most lines of code, use such dedicated languages. Even within the same dedicated language, GOAL, the number of lines of code differs by a factor 5 between the two teams that use it.

5 Conclusion

During the contest, we observed some problems concerning robustness due to false information. When the problem did not occur, we generally achieved competitive scores. If an agent somehow incorrectly updates its location, this can lead to agents agreeing on incorrect dimensions of the map, and as a result the agents will be rendered useless. While this suggests that the system is not robust enough, one could also argue that, once false information is introduced into the system, we cannot expect coherent behavior.

Another observation is that, especially in larger maps with lots of agents, we have too many idle agents. Once the map is explored and an agent has connected two blocks, the agent will simply roam the map waiting to be assigned a task. Instead, the idle agents could be put to better use by e.g. protecting goalzones.

In conclusion, we are satisfied with our placement for the contest and with the improvements we have made to the system compared to last year, but at the same time, we have discussed several ways that the system could be improved. For the future we suggest to execute the agents on the same infrastructure by the organisers.

Acknowledgements. We thank the anonymous reviewers for helpful comments.

A Team Overview: Short Answers

A.1 Participants and Their Background

Who is part of your team?

This year, 3 people were involved: Jørgen Villadsen (PhD), Jonas Weile (MSc student) and during the contest days also Markus Fridlev Schlenzig (BSc student). For earlier iterations of the code that we have built upon, also Alexander Birch Jensen and Erik Kristian Gylling have been involved.

What was your motivation to participate in the contest?

To study multi-agent systems in a realistic, but simulated, environment and to enhance our knowledge of the GOAL agent programming language.

What is the history of your group? (course project, thesis, ...)

Our team name is GOAL-DTU. We participated in the contest in 2009 and 2010 as the Jason-DTU team, in 2011 and 2012 as the Python-DTU team, in 2013 and 2014 as the GOAL-DTU team, in 2015/2016 as the Python-DTU team, in 2017 and 2018 as the Jason-DTU team, and in 2019 and 2020/2021 as the GOAL-DTU team. We are affiliated with the Algorithms, Logic and Graphs section at DTU Compute, Department of Applied Mathematics and Computer Science, Technical University of Denmark (DTU). DTU Compute is located in the greater Copenhagen area. The main contact is associate professor Jørgen Villadsen, email: `jovi@dtu.dk`

What is your field of research? Which work therein is related?

We are responsible for the Artificial Intelligence and Algorithms study line of the MSc in Computer Science and Engineering programme.

A.2 Statistics

Did you start your agent team from scratch, or did you build on existing agents (from yourself or another previous participant)?

As our starting point, we used the code from the competition last year—which in turn built upon the competition in 2020.

How much time did you invest in the contest (for programming, organising your group, other)?

We have spent approximately 60 h to further develop the code from the previous iteration.

How was the time (roughly) distributed over the months before the contest?

Most of the time was spent in August leading up to the qualification. After qualifying, no further improvements were made.

How many lines of code did you produce for your final agent team?

We have about 2000 lines of code.

A.3 Technology and Techniques

Did you use any of these agent technology/AOSE methods or tools? What were your experiences?

Agent programming languages and/or frameworks?
We used GOAL which is a quite easy and intuitive agent programming language.

Methodologies (e.g. Prometheus)?
No.

Notation (e.g. Agent UML)?
No.

Coordination mechanisms (e.g. protocols, games, ...)?
No.

Other (methods/concepts/tools)?
We used the Eclipse IDE for programming (it has a GOAL add-on).

What hardware did you use during the contest? We used a laptop.

A.4 Agent System Details

Would you say your system is decentralised? Why?
The team communicates via messages and channels to share information and agree on plans. The approach is mostly decentralized, but planning tasks are currently delegated to a single master agent.

Do your agents use the following features: Planning, Learning, Organisations, Norms? If so, please elaborate briefly.
The agents use planning to choose the tasks to pursue. A single agent is chosen to do the planning, but this agent relies on input from all other agents, and the planning agent is chosen dynamically at run time. The planning agent will search through assignment combinations and choose the most promising.

How do your agents cooperate?
The agents reactively decide on their actions based on the current percepts, their beliefs and their goals. They use predetermined rules and actions.

Can your agents change their general behaviour during run time? If so, what triggers the changes?
An agent will change its behaviour when it is chosen to take part in solving a task.

Did you have to make changes to the team (e.g. fix critical bugs) during the contest?
We chose not to make changes during the contest.

How did you go about debugging your system? What kinds of measures could improve your debugging experience?
We used log files to record the agents belief base and percepts.

During the contest, you were not allowed to watch the matches. How did you track what was going on? Was it helpful?
We only did basic logging to the console.

Did you invest time in making your agents more robust/fault-tolerant? How?
As evident from the competition results, we did not spend enough time on this aspect.

A.5 Scenario and Strategy

How would you describe your intended agent behaviour? Did the actual behaviour deviate from that?
First, to explore and have our agents find other agents, goal-zones and dispensers. Once most agents have connected, they collectively decide on a master agent to do planning. Once this agent has been found, it will continuously inquire the other agents about their available resources and try to create task plans. The task plan is sent to all agents involved in the plan, and these will try to solve it as efficiently as possible.

Why did your team perform as it did? Why did the other teams perform better/worse than you did?
We defined the overall strategy. The task-plans are created autonomously. We would have liked more flexibility for the agents to evaluate their strategy and correct this strategy as needed.

Did you implement any strategy that tries to interfere with your opponents?
We worked on some clearing strategies to defend goal cells, and to scare off opponents. However, they seemingly did more harm than good at the competition.

How do your agents coordinate assembling and delivering a structure for a task?
The planning agent creates a structured plan describing which agent should deliver what blocks, based on the input it receives from other agents. All agents involved in delivering a task then continuously check the plan to see if it remains feasible, updating the plan if necessary.

Which aspect(s) of the scenario did you find particularly challenging?
The map was made even more dynamic than preceding years, which was definitely a challenge.

What would you improve (wrt. your agents) if you wanted to participate in the same contest a week from now (or next year)?
If the contest was a week from now, we would mainly focus on bug fixing and thorough testing. If we had a whole year, we would work on changing the way we solve tasks and do planning—we would further decentralize it, removing most of the responsibility of the planning agent, and make the assembling of blocks more dynamic. Also, we should make better use of agents not partaking in solving tasks, as well as improving defensive strategies

What can be improved regarding the scenario for next year? What would you remove? What would you add?
We suggest to execute the agents on the same infrastructure by the organisers and then it would be interesting to decrease the time available for the agents to decide on their actions.

A.6 And the Moral of it is ...

What did you learn from participating in the contest?
We learned a lot about using GOAL to write multi-agent programs. We were reminded of the care it takes to develop and test in multi-agent environments.

What advice would you give to yourself before the contest/another team wanting to participate in the next?
Start early, because unexpected problems will occur. Have a clear testing strategy. The coordination between agents is working quite well and the A* path finding helps agents to move directly. Agents could be more flexible in helping each other and prioritizing other agents' tasks over their own when it is better for the team.

Where did you benefit from your chosen programming language, methodology, tools, and algorithms?
GOAL has built-in functionality that allows agents to communicate with one another and it has a predefined agent-cycle that is suitable for the belief-desire-intention model. A* was used by the agents to determine movement actions for short distances.

Which problems did you encounter because of your chosen technologies?
Writing thorough tests for GOAL code can be challenging,

Which aspect of your team cost you the most time?
Some unexpected problems (unrelated to the contest) ended up costing us a team member, and another team member had less time to work on the project than anticipated. This led to a large loss of potential time.

A.7 Looking into the Future

Did the warm-up match help improve your team of agents? How useful do you think it is?
It was not really useful due to the lack of time for improvements.

What are your thoughts on changing how the contest is run, so that the participants' agents are executed on the same infrastructure by the organisers? What do you see as positive or negative about this approach?
Yes, it would be great if the agents are executed on the same infrastructure.

Do you think a match containing more than two teams should be mandatory?
Maybe—perhaps if the agents are executed on the same infrastructure.

What else can be improved regarding the MAPC for next year?
We would prefer more or less the same scenario.

References

1. Hindriks, K.V., Koeman, V.: The GOAL Agent Programming Language Home (2021). https://goalapl.atlassian.net/wiki
2. Hindriks, K.V., de Boer, F.S., van der Hoek, W., Meyer, J.-J.C.: Agent programming with declarative goals. In: Castelfranchi, C., Lespérance, Y. (eds.) ATAL 2000. LNCS (LNAI), vol. 1986, pp. 228–243. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44631-1_16
3. Hindriks, K.V.: Programming rational agents in GOAL. In: El Fallah Seghrouchni, A., Dix, J., Dastani, M., Bordini, R.H. (eds.) Multi-Agent Programming, pp. 119–157. Springer, Boston (2009). https://doi.org/10.1007/978-0-387-89299-3_4
4. Hindriks, K.V., Dix, J.: GOAL: a multi-agent programming language applied to an exploration game. In: Shehory, O., Sturm, A. (eds.) Agent-Oriented Software Engineering, pp. 235–258. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-54432-3_12
5. Boss, N.S., Jensen, A.S., Villadsen, J.: Building multi-agent systems using Jason. Ann. Math. Artif. Intell. **59**, 373–388 (2010). https://doi.org/10.1007/s10472-010-9181-2
6. Vester, S., Boss, N.S., Jensen, A.S., Villadsen, J.: Improving multi-agent systems using Jason. Ann. Math. Artif. Intell. **61**, 297–307 (2011). https://doi.org/10.1007/s10472-011-9225-2
7. Ettienne, M.B., Vester, S., Villadsen, J.: Implementing a multi-agent system in Python with an auction-based agreement approach. In: Dennis, L., Boissier, O., Bordini, R.H. (eds.) ProMAS 2011. LNCS (LNAI), vol. 7217, pp. 185–196. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31915-0_11
8. Villadsen, J., Jensen, A.S., Ettienne, M.B., Vester, S., Andersen, K.B., Frøsig, A.: Reimplementing a multi-agent system in Python. In: Dastani, M., Hübner, J.F., Logan, B. (eds.) ProMAS 2012. LNCS (LNAI), vol. 7837, pp. 205–216. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38700-5_13
9. Villadsen, J., et al.: Engineering a multi-agent system in GOAL. In: Cossentino, M., El Fallah Seghrouchni, A., Winikoff, M. (eds.) EMAS 2013. LNCS (LNAI), vol. 8245, pp. 329–338. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-45343-4_18
10. Villadsen, J., From, A.H., Jacobi, S., Larsen, N.N.: Multi-agent programming contest 2016 - the Python-DTU team. Int. J. Agent-Oriented Softw. Eng. **6**(1), 86–100 (2018)
11. Villadsen, J., Fleckenstein, O., Hatteland, H., Larsen, J.B.: Engineering a multi-agent system in Jason and CArtAgO. Ann. Math. Artif. Intell. **84**, 57–74 (2018). https://doi.org/10.1007/s10472-018-9588-8
12. Villadsen, J., Bjørn, M.O., From, A.H., Henney, T.S., Larsen, J.B.: Multi-agent programming contest 2018—the Jason-DTU team. In: Ahlbrecht, T., Dix, J., Fiekas, N. (eds.) MAPC 2018. LNCS (LNAI), vol. 11957, pp. 41–71. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-37959-9_3
13. Jensen, A.B., Villadsen, J.: GOAL-DTU: development of distributed intelligence for the multi-agent programming contest. In: Ahlbrecht, T., Dix, J., Fiekas, N., Krausburg, T. (eds.) MAPC 2019. LNCS (LNAI), vol. 12381, pp. 79–105. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-59299-8_4
14. Jensen, A.B., Villadsen, J., Weile, J., Gylling, E.K.: The 15th edition of the multi-agent programming contest - the GOAL-DTU team. In: Ahlbrecht, T., Dix, J., Fiekas, N., Krausburg, T. (eds.) MAPC 2021. LNCS (LNAI), vol. 12947, pp. 46–81. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-88549-6_3