





MMD: The Block Building Agent Team with Explainable Intentions

Miklós Miskolczi  and László Z. Varga ^(✉) 

Faculty of Informatics, ELTE Eötvös Loránd University, Budapest 1117, Hungary
{psbdho,lzvarga}@inf.elte.hu

Abstract. The Multi-Agent Programming Contest (MAPC) is an excellent test ground to stimulate research on the development and programming of multi-agent systems. The current Agents Assemble III scenario is a nice example for cooperative distributed problem solving in a highly dynamic environment, and it requires that the agents are normative agents. For MAPC 2022, we have implemented the MMD multi-agent system from scratch in the Python programming language to find out if a multi-agent system can be developed efficiently in a general programming language using multi-agent concepts. We describe the implementation details, including the coordination and the optimisation algorithms of the MMD multi-agent system to solve the complex and dynamic tasks, and also including the testing aspects that use explainable intentions as well. The performance indicators of the implementation are the development time, the development efforts, and the quality of the job done by the implemented multi-agent system. The development time of the MMD system is not more than any other system at MAPC 2022, including those that were implemented with agent-oriented programming. The comparison of the development efforts of the contest participants is difficult because the performance of the systems are also different, but the development effort is more likely to be independent from the implementation language used. The first position of the MMD system at MAPC 2022 seems to indicate that the implemented MMD multi-agent system is competitive with the systems developed with agent-oriented software engineering methods.

Keywords: Practical reasoning architecture · Blackboard architecture · Explainable intention

The work of L.Z. Varga was supported by the “Application Domain Specific Highly Reliable IT Solutions” project which has been implemented with the support provided from the National Research, Development and Innovation Fund of Hungary, financed under the Thematic Excellence Programme TKP2020-NKA-06 (National Challenges Subprogramme) funding scheme.

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2023
T. Ahlbrecht et al. (Eds.): MAPC 2022, LNAI 13997, pp. 54–97, 2023.
https://doi.org/10.1007/978-3-031-38712-8_3

1 Introduction

The Multi-Agent Programming Contest¹ (MAPC) is an excellent test ground to stimulate research on the development and programming of multi-agent systems. In the current Agents Assemble III scenario the agents explore a grid world and execute dynamically announced tasks. The goal of a task is to create a structure of blocks in goal areas, which requires that the blocks are collected, delivered and assembled by a group of agents. This is a nice example for cooperative distributed problem solving [5]. The Agents Assemble III scenario requires that the agents are normative agents [16]. The capabilities of the agents depend on their current role. Roles can be adopted at specific role areas. The agents have to adopt roles that are better suited for their specific goals, but they also have to take into account the norms that regulate the adoptable roles. The norms are dynamically created.

According to the previous experiences of MAPC, the systems that were developed with multi-agent programming languages usually performed better than those that were developed in a general programming language [2]. On the other hand, the history of multi-agent system research seems to show that the theoretically crafted platforms seldom lead to practical applications. The prime examples of agent systems are like Siri, Alexa, Cortana, high frequency algorithmic traders [17], massive fleet of warehouse robots [18], and the IT giants have their own implementations of multi-agent technologies like negotiation mechanisms [6].

When preparing for MAPC 2022, we thought that the agent-oriented software engineering methods and the related planning systems would involve restrictions for us. In addition, an experimental Python communication client for the 2020/21 edition of the Multi-Agent Programming Contest, used in the WESAAC 2021 short course [1], became available². Therefore we decided to implement the MMD multi-agent system from scratch in the Python programming language to find out if a multi-agent system can be developed efficiently in a general programming language using multi-agent concepts.

We present our work in the following order. Section 2 describes the logical agent team architecture and how it is mapped to the software architecture of the implementation. Section 3 describes how the agents represent their beliefs of their environment, and how they find their way. Section 4 describes how the team is coordinated. Section 5 describes the building blocks of the individual agent behaviours. Section 6 describes the debugging of the system and how the agents helped this by explaining their intentions. In Sect. 7 we analyse the matches at the contest. Finally, in Sect. 8 we conclude our work.

¹ <https://multiagentcontest.org/2022/>.

² <https://github.com/agentcontest/python-mapc2020>.

2 Architecture

2.1 Agent Team Architecture

The architecture of the MMD multi-agent system is based on two architectural concepts: the blackboard architecture [7] and the practical reasoning agent architecture [3] as shown in Fig. 1. This is a reasonable architecture, because the MAPC requires cooperative distributed problem solving [5]. The team level problem solving is done on the blackboard, while the individual problem solving is done in the agents.

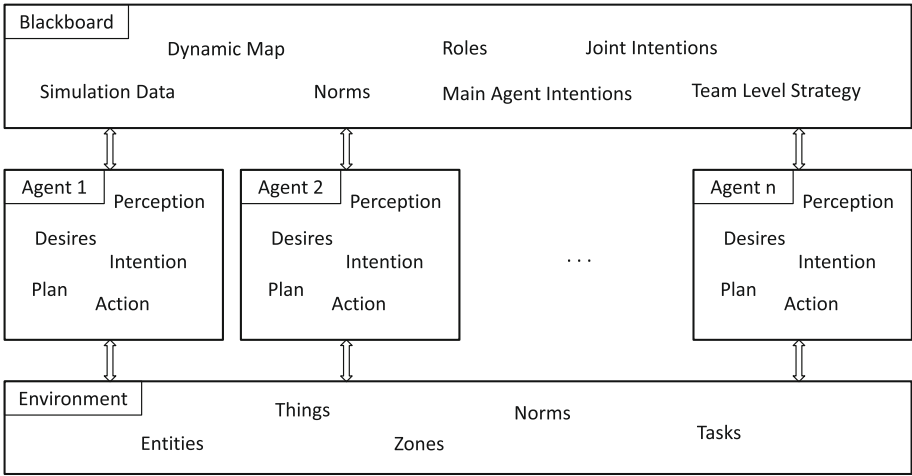


Fig. 1. The architecture of the MMD multi-agent system.

The agents have their own perceptions, desires and intentions. The perception is immediately submitted to the blackboard, where the perceptions from all the agents are processed, and the agents use this processed perceptions in their reasoning. The blackboard is not just a passive data store, because it has active reasoning capabilities as well, like reasoning on the team level strategy, and making decisions on the individual and the joint intentions [10] for the agents. This way, the blackboard is a coordinator similar to the general deSouches of the FIT BUT solution in 2019 [15]. The individual intentions on the team level are called main agent intentions in the MMD system. The main agent intentions are assigned to the agents, and the agents execute their own plan to achieve the goal of their main agent intention. The plans may involve other intentions which are managed on the agent level. When the agent executes its plan for the intention, it submits the agent actions of the plan to the MAPC simulation server.

Due to the highly dynamic nature of the MAPC environment, the main operation cycle of each MMD agent is the following:

- Check if there is a team mate in the perception area. If yes, then try to identify the agent. If the identification is successful, then handle map merging and map size determination.
- Generate desires.
- Filter the desires and commit to the best desire which becomes the current main intention.
- If execution of the main agent intention requires, then commit to another agent level intention as part of the main agent intention. Compute the next action of the current intention.
- Execute the action and process the dynamic perception from the action.

The agents basically communicate via the blackboard. If there are agents involved in the same joint intention, then they communicate via their intentions, which are directly connected to each other. The direct connection between the intentions is because of implementation considerations.

The blackboard contains all the information shared among the agents. In the beginning, the agents do not know each other, and each agent stores its perceptions in its own dynamic map on the blackboard. When the agents perceive other team members, then they try to identify each other, using an identification algorithm similar to the one in the LFC solution in 2019 [4]. If the identification is successful, then the blackboard merges the dynamic maps of the involved agents into a single dynamic map, which becomes the own dynamic map for each involved agent. In the beginning, the dynamic maps are infinite. If the map size determination algorithm described in Sect. 3.5 is able to determine the height or width of the map, then the dynamic map becomes finite in the respective direction. If both the height and the width of the dynamic map are determined, then the dynamic map becomes finite. The looping of the coordinates on the map is handled by the dynamic map.

The blackboard reasons on the team level strategy by considering the dynamic maps on the blackboard. If there is not enough information on a dynamic map, then the blackboard assigns exploration intentions to the agents of the dynamic map.

If the blackboard finds that enough information is gathered on a dynamic map for a task of the MAPC competition, then it may ask further information from the agents, for example the current desires of the agents or the bidding of the agents for given tasks. Based on the information on the blackboard and the information gathered from the agents, the blackboard decides the team level strategy. It selects the tasks of the MAPC competition to be achieved for each shared dynamic map, and selects one or more agents for each task. If the task requires only one block, then the best single agent is selected, and the agent receives the corresponding main agent intention. If the task requires several blocks, then the best group of agents is selected, and the agents receive the main agent intentions of the joint intention of the group. The joint intention is initiated by the blackboard. The execution and the termination of the joint intention is managed by the coordinator of the joint intention.

In order to keep to the rules of the MAPC competition rules, the agents have to manage their roles both individually and on the team level. The current roles of the agents are also shared on the blackboard. When a main agent intention is assigned to an agent, then the blackboard checks if the agent has the suitable role for the given intention. If not, then the blackboard posts a “role reservation” for the given agent on the blackboard. The “role reservation” is needed to facilitate the team level management of the role of the agents. When the agent creates its plan and finds that it needs another role, then the agent takes the role reservation from the blackboard and modifies its plan to adopt the role. The blackboard keeps track of the current norms of the MAPC competition, and reasons on the norms. If the norm is not considered to be “harmful”, then it is just ignored. If the norm is considered to be “serious”, then the blackboard changes the role reservations on the blackboard, or if it is necessary, then directly modifies the intentions of the agents to switch role or drop blocks.

This agent team architecture allowed a flexible intention management of the agents both on individual and on team level to achieve good results at the competition. In the beginning of a match, the agents know only their local perception area, so they start to explore the environment and their team mates. They do this, because their exploring desire does not have any pre-requisite, so they can commit to it by default. Once enough agents meet each other, and they share their map to be able to work on a task of the competition, they generate task execution desires, and then they commit to a joint intention to work on a task which is the most promising for the team. At any time, if an explosion event threatens an agent, then the agent generates a desire to escape from the threat. The escape is an important thing, therefore the agent drops its current intention and commits to the escape. If the agent with the escape desire is involved in a joint intention, then it first releases the other agents from the joint intention. The agents have open-minded intention management strategy, so they keep their desires during the escape, and when they get to a safe area, they continue with their normal operation. The normal operation means that they commit to task execution as soon as possible.

2.2 Software Architecture

Architecture. The system uses a unique combination of the repository and the layered architecture, which is shown in Fig. 2.

In the architecture, one client and two server layers can be found. One of the latter is the MAPC server, which is an independent service. The other is the MMD server, which can be divided into two parts: the blackboard and the scheduler. The client layer communicates with both of the servers in a bidirectional way.

The system’s logical architecture is decentralized, each agent can be interpreted as an independent process, although it is implemented in a centralized way. The reason behind it is the simplicity: this way it was easy to design the communication and the synchronization, although parallelization could not be exploited.

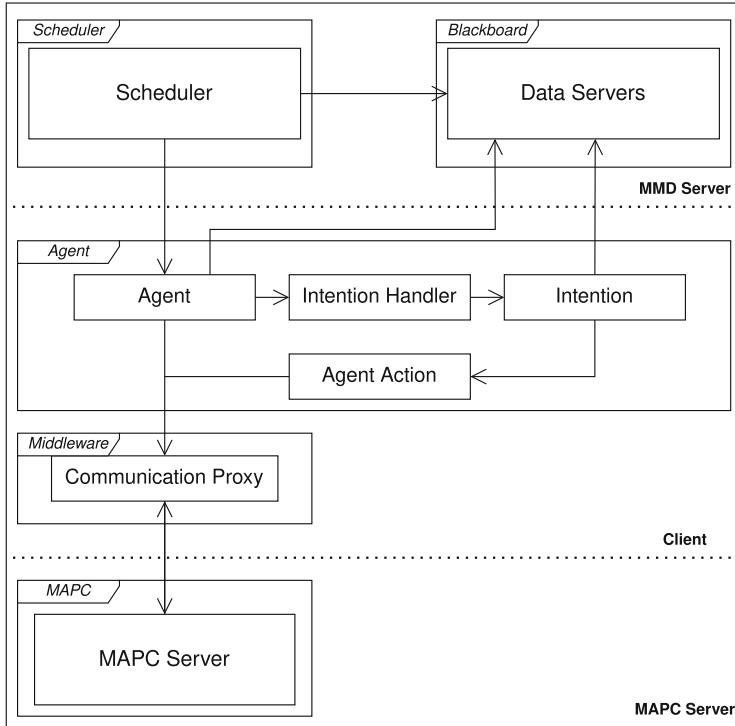


Fig. 2. System architecture.

Due to the centralized implementation, there is an individual component, the scheduler, which controls the system's operations.

Another effect of this decision is that there is no need for a separate communication layer between the MMD servers and the client. We needed a communication layer only between the MAPC server and the client due to abstraction.

Scheduler. The scheduler is a component, which controls the operation of the *data servers* and the *agents*. It does not persist any data and provides no functionality. Its only responsibility is to initialize the system and to initiate communication between the other components in every step.

Data Servers. The blackboard consists of data servers. The data servers are repositories and services, which gather information about a specific data category and provide business logic services on it.

Each data server collects its relevant type of input from the agents. The accumulated data are transformed, stored and can be accessed by any system component without any restrictions.

Some of them not only share the gathered data, but also build complex functions on them. Usually these functions are the ones that require data from multiple sources, such as global reasoning and coordination of several agents.

The following data servers exist in the blackboard.

- Simulation data server: basic repository, which stores raw data and parameters. It’s functionality is similar to a data lake and it is accessible for everyone.
- Map server: map building manager, described in Sect. 3.2.
- Intention data server: stores agents’ *intention* related information, such as their job type and data related to their current job.
- Role and Norm server: manager of the agent roles and simulation norms, described in Sect. 4.2.
- Task coordination server: service which organizes simulation task related jobs. Its functionalities are explained in Sect. 4.1.

Agents. In architectural perspective, agents are individual clients, that work together using the servers as communication channels.

They can directly communicate with the blackboard repositories, but a proxy is required to accomplish the communication with the MAPC server. Their business logic is mainly located at the *agent intention* components. Only the state of their intentions is stored by them, the rest of the incoming data is persisted at the data servers.

From a functional point of view, agents are standalone entities, that work together to accomplish shared goals, but individual ones, too.

Agents have intentions, which represent a complete job. The desires are optional intentions which may be selected for execution. At any given moment, an individual agent can have multiple desires. The selection of the best desire is based on the priority value of the desire. The priority value expresses the urgency of the desire. At every step, the desire with the highest priority is selected for the agent’s current intention, and the agent commits to the intention.

The desires are either generated by the agent itself or by a data server. When an intention is finished, then it’s removed from the agent’s current desires.

Some jobs, like task achieving, require the involvement of multiple other agents. Agents can not directly communicate with each other, but it can be done through the specific data server.

Agent Intentions. This is the business logic layer of the agent component. This layer contains only those functionalities, whose results directly affect only the agent itself.

Each intention is a representation of a specific job related behaviour. Every one of them has its own purpose and an algorithm that leads closer to its goal.

Agent intentions define an interface, shown in Fig. 3, which must be implemented.

- Determine the next *agent action*, which leads closer to its purpose. The calculation is performed by the job related algorithm.

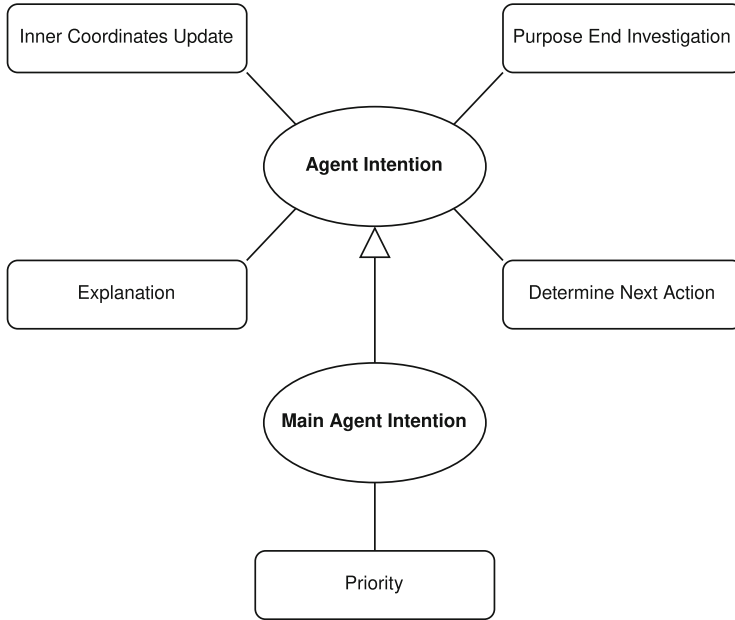


Fig. 3. Agent intention interface.

- Investigate if its purpose is reached, in that case the intention is finished.
- Update inner coordinate in case of coordinate system change (Sect. 3.4 and Sect. 3.5).
- Write explanation, which purpose is described in Sect. 6.

By implementing this interface, complex behaviours can be built easily by integrating separate intentions. The benefit of this property is that intentions can be structured hierarchically and they can be reused in other intentions.

Main agent intentions are a special type of intention. From an abstract point of view, they can be interpreted as the standalone, complete agent jobs. They can be prioritized, which defines an order if multiple top main agent intentions are active at the same time.³ The execution of main agent intentions may invoke one or more agent intentions.

Agent Actions. The agent actions are simple data transfer objects used between the agent component and the middleware layer. The transfer is unidirectional from the agent component. The middleware layer transforms the object, so it can be sent to the MAPC server.

³ The implementation used constant priorities for the main agent intentions, however, they could have depended on the current environment for better performance.

Actions are the representations of agent operations. Usually, they are initialized as the result of the planning of the next action of an intention, and they are sent to the simulation server indirectly at the end of each step.

Every agent action represents an allowed operation and its parameters. Their only prerequisite is to be deserializable, so the communication module can handle them.

Server Communication. The communication with the server is arranged by a bidirectional transport protocol. The request/response communication mode is used, using the *JSON* data format. Each agent uses its own channel to communicate with the server.

There are two types of requests that the agents use: the connection related ones and the agent actions. The former are the usual ones: connect and disconnect. The latter are the agent action requests. The client sends the given action and its response is a *perception*. The perception contains the result and the impacts of the action and much more. The content of the perception and its processing is explained in Sect. 3.1.

Due to the multiple different team sized simulations, the transition between the simulations must be handled in some way. The team sizes are only known at the start of each simulation, so always only one agent is connected to the server in the first simulation step. The single connected agent sends the information contained in the initial static percept to the scheduler. The rest of the team is created and connected to the server in the second simulation step.

3 Orientation

3.1 Perceptions and Observations

Perceptions. Information about the simulation and the environment is gathered through *static* and *dynamic perceptions*. Both of them are raw, unprocessed data sent by the MAPC server.

Static perception is invariable information, that is valid for the entire simulations. It is agent and state independent data and accessible by everything. It is uploaded to the simulation data server, by the first connected agent, without any transformation.

On the other hand, dynamic perceptions are based on the simulation's actual state. Some of them are agent independent, their content is equal to all agents. The rest of them are heavily agent dependent. These are narrow environment data from the receiver agent's point of view and only the receiver has access to it. The dynamic perception includes information about the agent itself and the things which are visible by the agent. The positions of the latter are relative to the agent, which means agents are not aware of their position of the actual map.

Dynamic perceptions are uploaded in raw format to a *dynamic map*, which is defined in Sect. 3.2, using the map server. In the beginning of every step, these are gathered from all agents.

Observation. The data which is required for the intentions to determine the next agent action, is all wrapped up in *observations*. These contain processed data, generated by data servers and the agents from the perceptions. Also access to several data server is granted by them, so not every data server related information must be stored in these objects.

Observations are generated from the data gathered from different sources at every simulation step by the agents. The agents use the observations to determine their next action. The observations provide an accurate view of the simulation, so that the intention calculations are as effective as possible.

For example, the perceptions only contain that a thing is attached to anything or not, but it does not tell where the thing is attached. Agents must be aware of their attached things, especially the blocks they are carrying. Therefore attached things must be tracked, which is done with the observations. Observations track every gain and loss of attached things. In agent oriented terms, we can say that the observations are the beliefs of the agents.

Observations are introduced into the software design, because of the architectural design. Intentions are the agent's business logic layer, therefore the intentions are not aware of the agent itself. Basically, observations are a combination of data transfer objects and proxies.

These objects are suitable for every type of intention, although not every intention requires all its data.

3.2 Map Building and the Dynamic Map

Dynamic maps are collections of data structures, which store things and their coordinates. At the same time, they provide different services, which are built on the stored data. Dynamic maps serve as repositories and services, too.

In the beginning, each agent orientates with its own map. The agents are not aware of their global location, so each of them builds its own map, using its own coordinate system: the agent starting position is the origo. The map is built using the data received from the perceptions, which is adjusted to the origo and to the agent position, too.

The following data is stored in the dynamic maps:

- Dynamic things, like agents, obstacles and blocks
- Dispensers
- Role and goal zones
- Marker zones

Dispensers and role zones are static, none of them are changed during the simulation. They are simply stored and managed separately due to their latter property.

On the other hand, dynamic things, goal and marker zones are stored and managed in a different way. A time stamp is associated to these data, so later it can be decided which information is more up to date compared to another one coming from a different source. The time stamp is needed when two agents

merge their maps and they have to decide which one of them has the most recent information.

When the map is built, whether a coordinate from a perception is undiscovered or not, the coordinate and the thing associated to it are added to the map, because the current perception is the most up to date data source.

The dynamic maps are managed by a data server, called the map server. The map server's only purpose is to associate maps to agents, ensuring that the agents coordinate their activities via the map which is associated to them.

3.3 Agent Identifications

Agents of the same team are not aware of the positions of other team members by default. To work together efficiently, they must know the location of each other.

When more than one agent observe in their perception an other agent from the same team, then the identification process is initiated. An algorithm similar to the LFC solution in [4] is performed for the identification. The idea is that, if an agent notices another one, then asks the others if there is an agent in their perception in a reversed point of view. If this condition is met, then the common viewable things are checked if they match, filtering out non possible candidates. At the end, if there are more than one candidates, then the identification is unambiguous and fails. The identification is only accepted if the number of candidates equals to one.

After a successful identification, the result is used for map merging (Sect. 3.4) and map size detection (Sect. 3.5). Both have a positive impact on efficiency, so agent identifications are always performed at the start of every step.

3.4 Map Merging

Only those agents can work together, that share a common map. The goal is to use as few maps as possible, so more agents can cooperate and the map's shared resources, like dispensers and goal zones, can be managed more efficiently.

In the beginning, none of the maps are shared. When an agent identification is performed successfully and the participating agents belong to different maps, then the map merge process begins.

Map merge is done by integrating a map into another. Only two maps are merged at the same time, but more can be done at the same time step. The maps use different coordinate systems, so one of them is kept, while the other is shifted to the other one. The shift value is calculated by the participating agents.

The shift value can be calculated by the difference of the coordinates of the agents, which merge their maps. The calculation is performed by Eq. 1.

$$\begin{aligned}
 \text{shiftValue} &= ac_1 - ac_2 \\
 ac_1 &= \text{agent coordinate in own map} \\
 ac_2 &= \text{other agent coordinate in own map}
 \end{aligned} \tag{1}$$

Those map elements that only exists in one map, are instantly added to the merged map. The elements that exists in both maps, are chosen by their time stamp: the more up to date value is stored, the other is discarded. The result is an union of the maps, which is shown in Fig. 4.

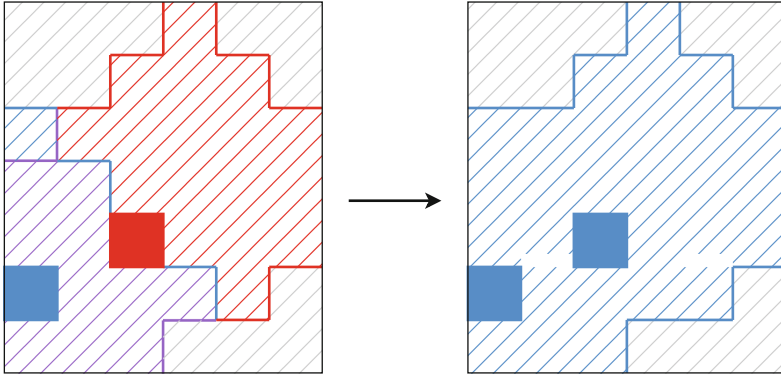


Fig. 4. Map merge result.

The map server guarantees, that all affected coordinates are shifted. It is done by the map merge and also by alerting the agents whose map has been integrated. The shift value is sent to all affected agents and it is their responsibility to update their coordinates inside the intentions.

After a map merge is completed, the involved agents belong to the same map, which ensured by the map server. From now on, these agents build the same map. They are able to manage the shared resources of the map to reduce possible conflicts and to work together to accomplish a given job.

In the beginning, the agents try to explore the world to build and merge maps. Task achievement is always preferred to map building. Often, there is no need to explore the whole map to complete tasks effectively.

3.5 Looping Grid and Map Size Detection

The end of the map cannot be detected directly, because the map repeats itself. For efficiency benefits, it is important that the dimensions of the map are known, if it is possible to calculate them. If it is not determined, then it can lead to performance decreases and to inefficient path findings. Because of the previous consequences, this must be dealt with.

The result of the agent identification process (Sect. 3.3) is used in map size detection as well. When an agent identification is performed successfully, and it turns out that the newly identified agent already belongs to the same map, but its map coordinate differs from the newly identified coordinate, then it means that the agent looped at least once on the map. Note that it is required, that the

agent coordinates are always maintained correctly. In this case, the dimensions are determined by Eq. 2 using the agents map and relative locations.

$$\begin{aligned}
 (\text{width}, \text{height}) &= |ac_1 - ac_2 + rc| \\
 ac_1 &= \text{first agent coordinate} \\
 ac_2 &= \text{second agent coordinate} \\
 rc &= \text{relative coordinate between } ac_1 \text{ and } ac_2
 \end{aligned}
 \tag{2}$$

Those dimensions can be determined for which the map has been travelled through. The map size calculation is always performed after agent identification. The size, calculated by Eq. 2, may be one or more times the real size of the map, because the other agent may have looped on the map more than once. If a size has already been determined at least once, and in a new agent identification a new size is calculated, then the size is updated only if the new size is less than the previous one.

After a valid new map size determination, all the coordinates in the system are normalized to the new dimensions. The coordinate update is organized just like at map merging. It results in a much smaller map, which is visualized in Fig. 5.

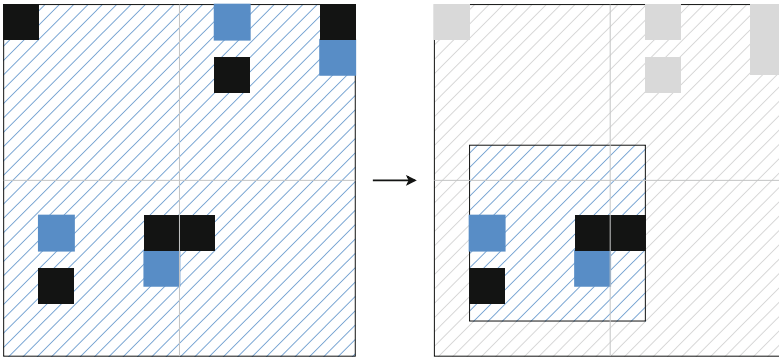


Fig. 5. Map size detection result.

Map size detection has a lower priority than merging all the maps and exploring every part of it. This does not necessarily mean that the latter precedes the former. According to our experiences, the dimension detection usually occurs earlier than the full map discovery, but not earlier than the merge of all maps.⁴

Map size detection has many advantages, but our agents may start task achievements even before the map sizes are determined, because it has proven to be more effective. Maybe it would be different at longer simulations.

⁴ Despite of not prioritizing complete map exploration, our experience shows that usually all maps are merged into a single one by the half of the simulation, while the dimension detection occurs some time later. The full map discovery happens late in the simulation, although, it is exceptional.

3.6 Pathfinder

Modified A* Algorithm. The main idea of our pathfinding is based on the A* algorithm [9] and the anytime search algorithm [8]. Agents have limited time to find an optimal path, therefore a constant node visit iteration threshold has been introduced to meet the anytime requirement. If the threshold is reached, then the closest estimated coordinate to the end is treated as the end coordinate. The heuristic estimation for the distance to the end node in the A* algorithm is the euclidean distance to the end node. The found path may be not be optimal, because the pathfinding may be terminated before the full path is discovered. However, the non optimality of the found path is not critical, because the MAPC environment is highly dynamic, and the agents create a new plan for their intentions in every simulation step. Therefore only the first elements of the found path matter, and the starting direction will be approximately right in every simulation step.⁵

The first elements of the found path is an agent action, which can be either a move or a clear action. Agents can travel multiple coordinates, depending on their role and carried entities. Always the maximum possible movement is applied, until a clear or rotate action is needed, or maximum movement limit is reached.

However, we limited the maximum number of move actions per simulation step to two. When more than two move actions are performed in a single step, and they are only partially successful, then the agents may lose tracking of their absolute position, because the count of the successful movements are not known. The loss of the absolute position completely confuses our maps and consequently all our algorithms. This is why we had to introduce this limit.

Pathfinding with Attached Blocks. In order to support pathfinding while carrying attached entities, the algorithm had to be further improved. The basic principle is that agents pull the blocks behind them, so upcoming obstacles can be cleared under any circumstances. Complexity and various edge-case scenarios are introduced by this idea, so a simplification had to be applied. The simplification prescribes that only one block may be carried, per agent, at all times. This principle ensures that agents find a route under any circumstances, without clearing large sections. This is achieved because, regardless of the role, the agent can always clear the obstacles in front of it.

In order to ensure that blocks are pulled, agents have to rotate in certain situations. The basic idea is that if an agent moves to a direction, which causes the attached block not to be behind the agent, then a rotation is needed before that. However, this causes lot of unnecessary rotations, which slows down the agent's travel time. To avoid unnecessary rotations, an optimization has been

⁵ Restarting the A* algorithm in every simulation step may not be efficient in terms of computation time. The D* Lite algorithm [11] would have been more efficient. On the other hand, the D* Lite algorithm uses more memory, especially if there are many agents.

introduced, that weakens the previous principle. Rotation is only required, if it is not possible to proceed without it: the attached block can not be moved due to blocking obstacles.

4 Team Coordination

4.1 Task Achievement

Task Completion Prerequisites. Task achievement has several essential prerequisites, which is visualized in Fig. 6. In order to begin a task, a group of agents must share a map, and a given set of conditions must be met on their shared dynamic map. This is because there is no sharing of resources between maps.

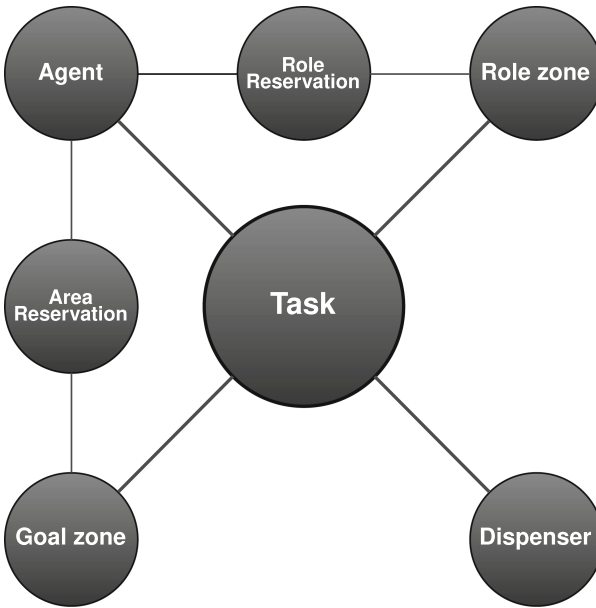


Fig. 6. Task prerequisites.

Regardless of the task itself, the location of at least one role and one goal zone must be known. The agents need to adopt the right role for the task, and they need a location where the submission can be done. If either of the previous conditions are not fulfilled, then the task cannot be started.

The same rule applies to dispensers, but it is task dependent. There must be at least one dispenser for each block type required for the task.

In order to begin a task, a goal zone location must be provided to the agents, where the blocks can be assembled and submitted. Therefore, the coordinates, where the assembly and submission are performed, should be reserved to avoid

possible conflicts. The neighbouring coordinates are also included in this area, so agents have enough space to hand over blocks. The reservation makes sure, that other agents from the same team on the same map do not assemble blocks at the same location.

A reservation is made, when a task is started, and its lifecycle needs to be managed. The reservation needs to be ended when the task is finished. The task may be finished either because it is submitted or because it is dropped due to some critical reason. Another possible event is when multiple maps are merged, and the reservations are conflicting with each other. In this case, only the reservation that belongs to the team closest to the reserved target zones is kept. The rest of them are cancelled, therefore the tasks associated with them are dropped.

Dropping a task is not a serious problem, because the agents involved in the task will be freed, and they will bid for task execution in the next simulation step. If the same task is the best candidate next time, and these agents are in the best position for the same task, then these agents will continue with the task execution, but with a different goal zone which is not occupied by other team members. If the same task is not the best candidate next time, or these agents are not in the best position for the task, then the team will select a better task and/or group of agents for task achievement.

The size of the group of agents needed for each task is equal to the number of block requirement of the task plus one. The extra agent is the coordinator which organizes the rest of team which are the block providers. The coordination process is similar to the *Contract Net Protocol* [12, 13]. The coordinator divides the task into sub-tasks, and the generated sub-tasks are allocated among the group. It is the block provider agents' responsibility to supply the correct amount and type of blocks. Meanwhile, the coordinator clears the assembly area, then acquires the blocks from the block providers, and submits the task at the end. The contract ends, when a task is finished either because it is submitted or dropped.

If the task requires a single block, then only one agent is needed, which performs the block provision and the coordination too.

Task Options. Task options are generated in each simulation step for each dynamic map. This is an agent and resource reservation independent pre-filtration. For each map, the available tasks are stored as possible options. The availability of a task is defined by a subset of the task prerequisites listed above (Fig. 6) with some additions.

- If there are no known role or goal zones for the given map, then no task can be started.
- A task cannot be started on the map, if no dispensers are known on the map for the type of blocks required by the task.
- If the block count of a task violates a norm, then it is ignored and skipped.
- A task should not be started, if surely there is not enough time to complete it. A simple lower estimation was used to calculate the minimum completion

time, which assumed that every agent required at least four simulation step to finish their part.⁶

Task Selection and Bidding. The generated task options are filtered by the available resources for each map. From here on, the beginning of a task depends on resource management: agent, role and goal zone availabilities.

For resource management a simple, local optimization algorithm was used. The algorithm always selects and starts only one task, and it is repeated until none of the tasks can be started. One task can be selected multiple times, there is no upper limit for that. Although, there should be some kind of limitation, because once a task is submitted a certain number of times, then it can not be submitted again. Therefore, the teams, which are still working on that task, have to change to another.

In each task selection iteration, the following conditions must be valid.

- There is at least one goal zone area free for reservation. The size of the area depends on the required block formation.
- There must be at least the given number of free⁷ or explorer agents described in the prerequisites sub-section above.
- The required roles for the task can be reserved, i.e. they comply with the norm management of the team, which is described in Sect. 4.2.

After the filtering, the remaining tasks are sorted based on a value function. The value function defines a ranking, that determines which tasks worth the most in terms of rewards and efforts. The value of a task is calculated by Eq. 3.

$$value = \frac{\frac{reward}{agents} \cdot remainingTime}{1 + crowdednes} \quad (3)$$

where the variables are the following.

- *reward*: The reward points for the task.
- *agents*: The number of agents needed for the task completion.
- *remaining Time*: A quotient of the remaining time until the task deadline and a constant.
- *crowdedness*: Agent density estimation at goal zones. It is a rough constant multiplier based on how much space is needed for the block assembly, how many agents are performing task related jobs and how many goal zone coordinates are known on the actual dynamic map. The multiplier is calculated by Eq. 4. The *crowdedness* value is positive, if the number of agents already working on task achievement (*busyAgents*) plus a space proportional to the block requirements of the given task (*blocks*) increases three quarter of the goal zones. In this case, if there are more busy agents and there are more requirements of the task, then the crowdedness is higher.

$$crowdedness = \max(busyAgents + 2 * blocks - goalZones * 0.75, 0) \quad (4)$$

⁶ It was a very simple estimation, which have not been improved due to lack of time.

⁷ An agent is free if it is not involved in task achievement.

The value of a task (Eq. 3) is higher if the team can produce more rewards per agent. The value is also higher, if it is more likely that the task can actually be submitted, because there is more time until the deadline. If the available free goal zones drop below one quarter of the known goal zones, then the value of the tasks with less block requirements will be higher, because these tasks need less space for submission. The one quarter limit for the goal zones is just a ballpark value that takes into account that the other team also occupies a part of the goal zones.⁸

The task with the highest value is started. The required amount of agents are assigned to the task to form a task achievement group. The members of the group are selected on the basis of the bids of the agents, which is described in the next subsection. For the selected agents, the right roles are reserved, if needed, and their intentions are inserted into their desires. The required goal zone coordinates are reserved for the team, which is selected by the coordinator. This is the end of the algorithm, and the agents start their task in the simulation next step.

Agent Task Bidding. When agents are considered to be part of a group for a given task, they are ranked by their bids. The bid of an agent estimates how much step is required for the agent to accomplish a given job. Agents that require the least amount of time are selected for the given sub-task.

A job can be a coordination, block provision or single block provision. Therefore, for each job, the agents bid differently.

Bid calculations consist of several different parts, where one depends on the other. For example, when the agent bids for a coordination, and does not have the right role for that, then the bid contains the cost of travelling to the nearest role zone, plus going from the role zone to the goal zone that is closest to the role zone.

Each bid calculation has a common part, the role adopt time. In the beginning, if the agent’s actual role is not suitable for the given job, then the time to adopt the right role is included. It may occur, that a job requires more than one role. In this case, the calculation takes into account that more than one role is needed.

The jobs require that the following calculations must be made.

- *Coordination*: the time to get to the goal zone which is closest to the agent. The chosen agent’s closest goal zone is reserved for the team.
- *Block provision*: the sum of time required to get to a right type of dispenser and then to the reserved goal zone.
- *Single block provision*: the sum of coordination and block providing costs.

⁸ Although we had this complex task evaluation function, we are not sure that it really played an important role in the MAPC 2022 contest, because there were always only 2 active tasks at the contest, and there were not many options to choose from.

4.2 Role and Norm Management

The MAPC competition rules define the roles and the norms. The norms regulate the type of the roles and the number of the agents with given roles during the competition. Role and norm management is needed to keep to the rules of the competition, which may require team level coordination.

Role Categorization. The possible roles of the simulation are not known in advance, their available actions and other parameters are only known at runtime. Proper management of agent roles is key to effective operation. Choosing the wrong role for a given job can disable the agent from doing it or significantly reduce its performance.

At the beginning of the simulation, the agent roles provided by the MAPC simulation are categorized according to the job they are suitable for. This is needed, because for example, if an MMD agent wants to do the coordination job, then it needs to adopt a MAPC role which has the capabilities to attach a block, to connect two blocks and to submit a task. The role categories are determined by the possible agent jobs that are used in the MMD system:

- Coordination
- Block provision
- Single block provision
- Explorer or inter task role

In the future, the role categories might be extended with saboteur and surveyor categories.

The current design assumes that, that these roles exist as single roles, not as combination of more. Only the single block provision is excluded from that assumption.

Each role category has mandatory conditions, without them, the role is not suitable for the given job. Examples of such conditions are the permitted actions and the ability to move with attached things. There are optional conditions. The optional conditions can be used to rank the roles which one can perform better in the given job, usually determined by skill parameters.

Coordination roles require submit, attach and connect action options. The coordinator must be able to get blocks from others, using the attach and connect action. For task submission the submit action is mandatory. The coordinator should be able to protect itself from saboteur agents, so a role, which has more promising clearing parameters, should be preferred.

The essential capabilities of the block provision are the request, attach and connect actions. The reasons for the attach and connect actions are the same as in the previous case. The request is used for block acquisition.

Single block providers require request, attach and submit actions for reasons similar to the previous ones. Should there be no single role for single block provision in the simulation, then the coordinator and block providing roles substitute the single block provider role. Only one role is allowed at the same time, therefore in this case at some point, the role must be changed.

There is no separate explorer category, but there is a so called *inter task role* category instead. Agents are optimized to complete tasks as soon as they appear. During exploration, a role from one of the previous three categories is chosen, so agents are always ready for a task job.

Usually the roles with higher speed are preferred, but because of role distribution balance, it was not included in the selection. The vision of a role could have been used at exploring, however, it has been treated just like the agent speed. The reasons behind the role distribution balance are described in the next subsection.

The importance of the role's attributes is visualized in Fig. 7.

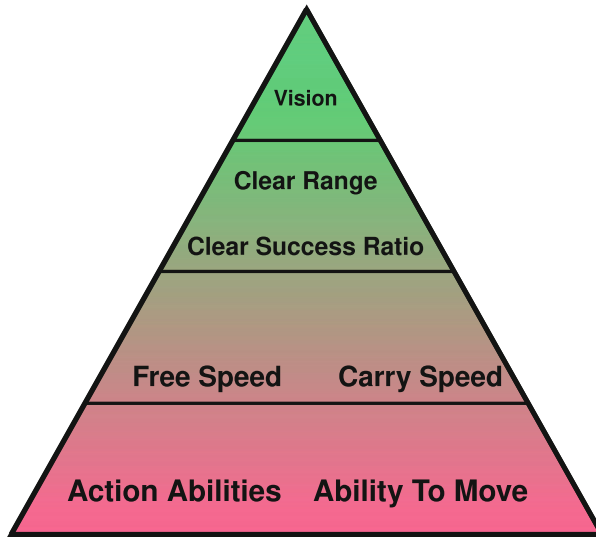


Fig. 7. Role property prioritization.

Role Assignment and Reservation. The management of role assignments and reservations are controlled by the *role server*. Agents use the role server to obtain a role of a specified category, usually for a specific job. However, agents do not choose the specific role within the category themselves, they choose from the options offered, because norms must be handled on team level. Team level coordination of roles is facilitated by first reserving the role for future use by the team, and then assigning the role to a given agent when needed.

When the roles of a given category are requested, the role server filters them by their availability. The availability of a role depends on the norms, which limits the role usage at team level. If the sum of the count of assigned and reserved roles reaches a limit defined by a norm, then the role is not offered in any category, making the role unobtainable for the agents. The count of both assigned and

reserved roles matters, because even if not at the given moment, but in the worst case, in the near future, the limit of the norm might be exceeded.

The role server service requires the tracking of the assigned and the reserved roles. When an agent requests a role of a given category, then the role server offers the agent the available options. The agent selects one of them, then reserves it. This indicates that the agent will take on the chosen role in the near future and it must be guaranteed to it. The role is removed from the reservations after either adoption or resignation. In the former case, the role is assigned to the agent.

When multiple roles are offered in a given category, then the agent chooses randomly. In this way, it is less likely that every agent adopts the same role, thus the team of agents is in a better position for upcoming role limits.

Norm Filter and Consideration. Norms are limitations, which can regulate agents at individual or team level. Violating them causes the agents to lose energy, which is a critical threat to them. Some norms are even extremely harmful, so they must be dealt with.

Norms are handled serially: only the upcoming ones consecutively. This is by no means the most optimal approach, but based on our experience, it solved the problem relatively efficiently and correctly.

Norms can be classified as ignorable and considerable ones.

The ignorable norms are those that do not require direct major intervention. An example is the limitation of the default role, because it is usually not needed. This experience is based on the test simulations prior to the contest. Another example is that a two-block limit norm does not really matter for two-block tasks, because in the case of two-block tasks, the time when the agent has two blocks is negligible.

The norms must be considered by their energy loss effect. A norm must be complied with, if the average agent energy drops below a certain level due to the energy loss caused by it and other norms that are not complied with during its duration. The constant energy recovery is included in this calculation, which is not known in advance, it is discovered by the agents, using simple energy comparison between steps. Once the energy recovery is discovered, then it is stored in the simulation data server. The energy threshold is a constant percentage of maximum energy. The calculation is an estimation and the threshold tries to mitigate its inaccuracy. While block norm violation only punishes an individual agent, it is treated as a team-level energy loss, as not even a single agent is allowed to drop out.

Norms are complied with by their category, which can be either block or role regulation.

If a block norm has to be complied with, than any task is stopped where the block count of the task exceeds the limit. In this case, only the coordinator is affected, however it stops the whole group of agents involved in the task.

Role norms are managed using the role server. First, the role server provides the count of the task performing agents of the affected role, which indicates

how many roles have to be dropped. Then, the task performing agent group, which has the most given role reservations discards the task and the role reservations. This continues until the given role reservations drop below the limit of the norm or until the norm goes out of effect. If the norm is still in effect and all role reservations are discarded, then the algorithm is continued, but with the adopted roles. The algorithm starts with the reservations, because the reservations indicate that those agents are far from finishing the task, while the agents that have their proper roles are already working on tasks. The agents that do not participate in task jobs automatically change to available roles.

If more than one norm of a given type is active at the same time, the strictest one will always be taken into account.

5 Intentions

Intentions represent agent jobs, which can be either standalone or part of another intention. Every intention has a unique purpose, and when it is reached, then it is considered finished. Because the MAPC 2022 scenario is highly dynamic, the intentions determine the next action leading closer to its purpose in each simulation step using their observations.

Intentions are structured hierarchically. There are simpler and more complex ones, usually the latter include the former. Main agent intentions are at the top of this hierarchy, they can be interpreted as standalone jobs. Each main agent intention is associated with a priority, so it can be decided which of them is more important at the actual step.

In the following sections, intentions are categorized by their usage and explained in detail.

5.1 Common Intentions

The intentions listed here can be categorized as supporter intentions, they are usually included as building blocks in other intentions.

Idle. This is a placeholder intention, which never finishes and does nothing. Its only purpose is to make the agent's intention queue (described in Sect. 5.4) non-empty.

Skip. This simple intention is used for skipping the current simulation step. Has no purpose, never finishes.

It has an optional flag, which triggers that, if one of the agent's attached things blocks a dispenser, then it rotates to avoid to free the dispenser. This behaviour ensures that it does not hinder other agent's work. The flag is enabled only, if the attached blocks do not have to maintain a specified format for task achievement.

When an agent skips, it uses this intention to perform the skip action.

Travel. This intention wraps the pathfinder component. Its goal is to travel to a given destination with the invocation of the pathfinder.

If the given destination is reachable, i.e. the destination is not blocked by another agent, then it returns an action to get closer to it. The action can be either move, rotate or clear action. If the destination is not reachable, then it skips, because there is another intention to handle this case.

Wait. This is an extension of the travel intention. When the given destination is reached, then the skip intention is initiated. The wait condition is determined by another intention.

It has a simple defensive behaviour: after its destination is reached, it attempts to shoot at agents from the other team in the perception range to protect itself and to keep them away. This defensive behaviour is always active at the destination, but it can have a real effect only if the role of the agent allows that the clear action damages the other agent.

Agitated Travel. This is an improved version of the travel intention to handle the case when the target destination is unreachable. It applies the basic travel and wait intentions to achieve that.

It has two improvements. Firstly, it allows multiple destinations next to the target destination. When one of them is reached, then its purpose is considered reached. Second, if all of the destinations are unreachable, then it searches and travels to the closest reachable location between its current location and the destinations. By this behaviour, it is ensured that the agent is getting closer to the goal area at every step.

Agitated Wait. This is a combination of the agitated travel and the wait intentions.

After one of the goal locations are reached, it behaves just like the end of the wait intention. It waits unconditionally and attempts to shoot at hostile agents.

Distant Agitated Travel. This is a distance keeper version of the agitated travel intention.

It has only one destination, however, its purpose is not to reach it, rather to be in a given distance to it. It ensures, that the agent is near to a given destination, but not close enough, so the area around the destination is not crowded.

Detach Blocks. This intention detaches all the things which are attached to the agent. Terminates when nothing is attached to the agent.

Reset. This is a main intention, which is only initiated after the agent team got reconnected to the MAPC server.

Its only purpose is to drop every attached thing, by applying the detach block intention. This is because the individual agents can not remember their previous intentions and their task group. Therefore, the groups have to be recreated, but until then, the attached blocks unnecessarily hinder the agents.

Escape. When the agents notice that they are in a clear event area, then they start to flee. The flee process is handled by the escape main intention.

It uses the path finding component's calculations to find the shortest route to a destination, that is not affected by the clear event. Then the travel intention takes care of moving there. Before it tries to escape, it applies the detach block intention, which helps the agent to get to the destination as soon as possible. This may require some extra time, but the agent can reach its goal more flexibly and faster, and also, the pathfinder does not work correctly if more than one block is attached to the agent.

Clear Target. This is a simple intention to clear the given coordinate, if a block or obstacle is located at it. First of all, it moves to a coordinate adjacent to the target coordinate by the travel intention, and then clears the given target. If the target is unreachable then it skips, and this case will be handled in the intention from where the clear target intention is invoked.

Clear Zone. This is an extended version of the clear target intention, which allows the clearing of multiple targets, while preserving a minimal amount of energy. It chooses its current target as the closest one, making the clear process optimal.

If the agent's current energy is low, then it just skips to gain energy. The energy limit is defined by a constant parameter. By this, it is ensured, that not all of its energy is consumed and even preserves some, if suddenly an escape is needed.

At random occasions it rather shoots at hostile agents to keep them away from the given area the same way as in the wait intention.

Adopt Role. The purpose of this intention is to adopt the given role. It uses the agitated travel intention to get to the closest role zone area. After that, the intention adopts the given role.

5.2 Explorer Intentions

Explorer intentions ensure that the map is discovered and kept up to date as much as possible.

Explore. In the beginning, most of the map is completely unknown. The task of the explore intention is to discover as much parts of the map as possible, in the least time possible.

It begins to explore the agent's environment in a spiral like shape. It moves with the basic travel intention towards an unknown location, which is closest to the starting point of the agent and also to its current position. If there are more than one locations like this, then a random one is chosen, which may cause the agent to start exploring in another direction. The selected destination is a little bit further than the selected unknown location, in order to move the whole perception range of the agent out from the already explored area. Because of this exploration algorithm, the shape of the exploration is like a spiral. The shape is visualized in Fig. 8.

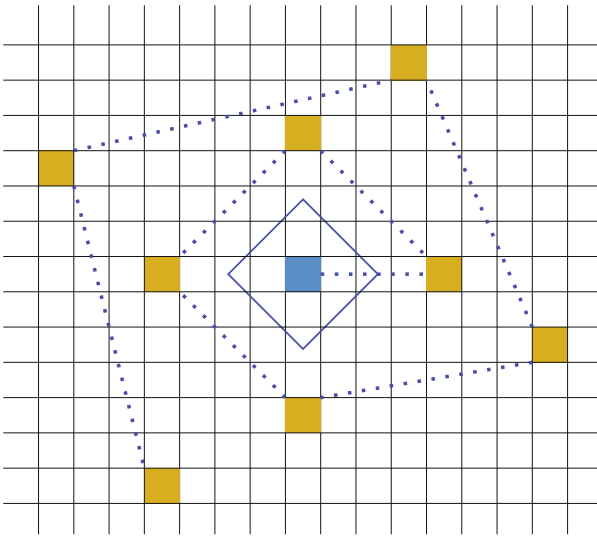


Fig. 8. Direction of the exploration.

This movement assists the agents to find each other as soon as possible, and also explore unknown areas with minimal effort. The explore algorithm is compatible with the map merges. The behaviour does not change, although the exploration does not keep the spiral like shape, but it still wanders around the edge of the already explored area.

The intention is finished, when the map dimension is calculated and all the points are known. These conditions are rarely fulfilled.

If there is a reserved role for the agent, then the adopt role intention is performed to adopt it during the exploration. Usually this scenario happens only, when the agent has to change a role due to a norm violation.

This main intention has lower priority than the task related ones, therefore the map is rarely explored completely.

Map Update. The map update main intention takes the explore intention's place, after the map is fully discovered.

It explores coordinates, which were discovered the earliest. The closer ones to the agents are always preferred. The exploration is performed by the basic travel intention.

This intention has the same role handling behaviour like the explore intention. Their priorities are also equal.

This main intention is never finished, because there is always a point to explore again, but a task related intention may make this intention inactive for a while.

5.3 Task Achieving Intentions

In this category, all the intentions are part of the task achieving process. Only the coordinators and block providers use the intentions listed here.

Block Collection. The responsibility of the block collection intention is to acquire a given type of block, that can be consumed in a task submission. It is used by single or regular block provider agents.

The intention searches for dispensers, from where the given type of block can be gathered. It uses the agitated travel intention to travel next to the dispenser. While going to the dispenser, if the agent perceives an abandoned block of the correct type, and the abandoned block is closer than the dispenser, then it targets the abandoned block.

However, not all the dispensers and abandoned blocks are measured the same way. The dispensers, that are not completely occupied by other agents, have always greater priority. There is a high chance at dispensers surrounded by agents, that there will be a conflict for the requested block. The conflict can be either waiting for other agents or suffering damage from hostile agents.

The abandoned blocks are not prioritized, rather they are filtered by their surroundings. If there is any agent or block besides them, then they are ignored, because the abandoned block may be attached to the agent or the block. We allow to hold more than one thing only for coordinator agents, because it would hurt the "maximum one attached thing per agent while moving" principle. The coordinator agents do not move if they have at least one attached block.

The "maximum one attached thing per agent while moving" principle can also be hurt when attaching a block from a dispenser. This may happen if more than one agent attach the block at the same time from the same team. When the agents belong to the same map, then an attach order is defined by themselves. However, if they do not belong to the same map, the previous scenario might happen. When this scenario is recognised by the agents, then the agent either detaches the attached block with 50% chance or skips with the same chance. Basically, the conflict resolution is random based, because after a while at least one agent detaches the block. If both agents detach the block, then they attach

it again in the next step. Sooner or later, only one of the agents holds the block and it is free to go with it.

The intention is finished, when the agent has acquired the block and there is no other thing attached to it.

Connect. This intention hands over an attached block to another agent, usually to a coordinator. It is usually performed by a block provider agent to connect a block to a coordinator agent during the block delivery. The intention is initiated when the involved agents are close to each other.

The intention receives the exact location, where the block must be delivered. By the basic travel intention, the agent moves to one of the given destination's adjacent coordinates. After the travelling is finished, it rotates the attached block to the given location.

When the block is in the right location, then the agent either detaches the block or connects the block to one of the coordinator's attached blocks. The required action is determined by the block position. If the position is directly besides the coordinator, then a detach action is sufficient. If not, then a connection must be made, which must be closed immediately.

The intention is finished, when the block provision intention signals that the transmission was successful and the agent holds no longer the block.

Block Delivery. The block delivery intention ensures that an attached block is transported to a coordinator agent. It is only used by block provider agents.

The intention uses the agitated travel intention to go close to the coordinator agent and then to follow it.

When the block provider agent is close enough to the coordinator, then sooner or later, it receives a signal to exchange the requested block. The exchange process is managed by the connect intention.

The process ends, if the connection intention is finished and the coordinator agent approves the exchange, indicating it by a signal.

Block Providing. This main intention handles the block providing job by scheduling and managing the block collection and the block delivery intentions.

First of all, if the agent has the wrong role for the block providing, then it adopts the right one, using the adopt role intention.

If the agent has either no blocks or the wrong ones, then all the attached blocks are detached using the detach block intention. Then the block collection intention is initialized to acquire the right block.⁹ If the agent has the right block, either because already having it or from the block collection intention, then the block delivery intention is initialized to deliver the block to the coordinator agent.

⁹ There could have been an optimization, if the right block is included in the attachments, then only keep that one. It was not implemented due to lack of time.

The intention recognises if it loses its attached block, for example by a clear event. In this case, the process starts over by acquiring the right block.

The intention can be finished in several ways. The positive outcome is when the block is successfully delivered. The negative one is determined by the coordinator agent, usually when the task can not be completed in any way.

Assemble. This intention is used by a coordinator agent to acquire an attached block from another agent, usually from a block provider agent. The intention is started for each required block, when the corresponding block provider is close to the coordinator. The coordinator agent assigns a connect intention to its block provider agent to connect the delivered block at a given location. The intention is finished when the block is acquired, and if connection was required, then the connection is closed.

Coordination. This is the main intention of the coordination process. Its purpose is to accomplish a task submission flow, by its own logic, and to control the block provider agents of the task group.

It consists of the following steps:

1. If just initialized, the attached blocks are checked, and the wrong ones are detached.
2. Before determining the next step, the intention checks if the task can still be completed. If not, then the intention ends.
3. The current agent role is checked, if it has the wrong one, then the intention adopts the right one.
4. It travels to the chosen goal zone destination.
5. While none of the right block providers are ready for the block exchange, it clears the area from obstacles and abandoned blocks using the clear zone intention.
6. One by one, the blocks are exchanged from the providers when they are ready.
7. Once, it is in the possession of all required blocks, then it submits the task.

If the intention just got initialized, then it needs to make sure, that the agent has only the right type of blocks in the right positions. All the attached blocks that do not satisfy the condition must be detached, using the detach block intention.¹⁰

Before continuing, in each step, it must be determined, that the task can be still completed. The influencing factors, which can interrupt the flow, can be divided into two parts. The first part is the critical, which make the task completion impossible, like task expiration or a norm violation. The other part is the optional, which just set back the task completion. These hindering factors could be handled by the intention, however for simplicity purposes, they are handled like the critical ones: the intention is terminated, and if it is still relevant,

¹⁰ Due to lack of time, wrong block selection algorithm was not implemented, therefore if there is at least one wrong block, then all the attached blocks are detached.

then it will be restarted in the next simulation steps anyway. The optional factors are the following: either when the goal zone disappears or when a clear event appears on the agent itself or one of its attached blocks.

If the intention has to be interrupted, then it releases the block provider agents from the task group. Then the detach block intention is initiated to drop all the attached blocks.

If the agent has the wrong role for the coordination, then the adopt role intention adopts the right one.

The goal zone area, where the blocks can be assembled are determined by the dynamic map associated to the agent. Until one of the coordinates of the goal zone area is not reached, the intention always chooses the closest one and travels there, using the basic travel intention.

After the arrival to a goal zone, if none of the right block provider agents are ready for block exchanging, then it starts to clear the area. Its purpose is to clear the area from obstacles and abandoned blocks, to make exchange easier for the block provider agents. It is performed by the clear zone intention.

The order of the block exchange is determined on the basis of the task block requirements and the block provider agents' readiness. Only one exchange at a time can be completed for simplicity purposes. From experience, there were usually no complex tasks where parallelization could be used. The task block requirements itself defines a non linear order of the possible options, which is filtered further, by the available blocks from the block provider agents, at the given moment. If there is any hand over opportunity, then one of them is chosen randomly. The exchange is performed by the assemble intention. The block exchange loops until all the required blocks are attached.

When all of the required blocks of the task attached to the right positions, and none of the block provider agents are connected to the coordinator, then the task is submitted, and the intention is finished.

Although we have a goal zone reservation scheme to prevent that two groups of our team block each other by trying to submit a task at the same place, the block reservation scheme cannot take into account the other team of the match. If the coordinator finds that an agent from the other team stays for a longer time at the reserved goal zone, then the coordinator drops the task and releases its block providers. If the same task and the same group of agents are still preferable, then the group will be recreated with a different goal zone in the next simulation step. If not, then a new group will be formed.

Single Block Submission. The purpose of this intention is to deliver a single block to a goal zone, and then to submit the task. A prerequisite of this intention is that the task requires only one block. Only the single block provider agents have this intention.

The intention determines the closest goal zone in each simulation step the same way as in the coordination intention. The goal zone determination in each simulation step is useful, because when a goal zone disappears, then the intention can go to another goal zone with the same block. If the goal zone is reached,

then the intention rotates the block into the direction required by the task, and submits the task. If the location at the rotation target is blocked, then the intention either clears the location, if possible, or selects another location within the goal zone.

The intention ends, if the task is submitted successfully.

Single Block Providing. This main intention is a unique mix of the block collection and coordination intention. Just like the single block submission intention, only the single block providers can have it.

Its purpose is to accomplish a task submission flow, which requires only one block. It's the intention's responsibility, to collect the given type of block and to submit it in a goal zone, without the assistance of other agents.

First of all, if the agent has the wrong role for the block collection, then the adopt role intention adopts the right one.

Just like at the coordinator intention, if the agent has wrong blocks, then the detach block intention is performed. After that, the block collection intention is initiated to acquire the right block.

At this point, if the agent's role is not suitable for the task submission, then the adopt role intention is performed again, to change role.

Then the single block submission intention is executed to deliver the block and submit the task. If the agent loses the block, the algorithm starts over.

Just like at the coordination intention, the task completion can be interrupted by several factors. It is handled the same way, excluding the release of the block providers.

The intention is finished, when either the single block submission intention ends or if it had to be ended for some reason.

5.4 Agent Intention Management

Agents usually have more than one main agent intention at the same time, but only one is active at any given moment. Often these intentions are not performed sequentially. At every step, the most important one gets activated, which might be different than the one in the previous step.

The active intention is changed often, so all the agent intentions must be persisted with their actual state. Main agent intentions are stored in a priority queue. This priority queue is called *agent intention handler*, which stores and manages the main agent intentions. Each agent has its own intention handler.

In every step, the main agent intention with the highest priority gets activated. The priority is defined by the main agent intention itself.

The intention handler manages the insertion of the new intentions, and also the removals of the finished ones. Intention insertion is usually initiated from the outside, but the intention handler can also generate intentions from the inside. The inside generated intentions are usually the agent's individual intentions, like the escape and explore intention.

The transition from one intention to another one usually does not have to be handled specially, because they do not depend on other agents. The coordination intention is the exception, because if it must be cancelled immediately, then all the block providers must be released. This special transition is handled by the agent intention handler.

6 Debugging and Explanations

6.1 Challenges of the Debugging

Despite following the principle of simplicity, the multi-agent system had a lot of error sources. Fail-safe was a prerequisite for many algorithms, therefore the cause of all errors had to be spotted.

One of the most challenging factors in debugging was the randomness of the MAPC 2022 scenario. The agents have to operate in a random based dynamic environment. In this dynamic environment, the elements of the map, the norms, the opposing team and the success of the agent actions themselves vary. Even some intention algorithms contain randomness. The factors listed above all make the reproduction of the bug difficult.

Most of the times the reproduction of rare faults were the most challenging. It is almost impossible to start the MAPC server and our agent team from a given situation, because for example it is difficult to recreate dynamic maps and agent states. The server and the agents can only be started from the beginning of a match, and then the same situation may not occur due to randomness. Sometimes repeated execution was the only solution to find out more about the cause, due to the complexity of the error.

The occurring faults were also challenging to solve. Due to the system architecture, intention related fatal errors were difficult to associate with actual agents. Most of the times, the fault itself could not be perceived, only its consequences could be seen. For example, if two agents stick together, then their map becomes confused. Later when they are detached and everything seems fine, then they may identify the wrong size for the map, which may invalidate all the dynamic maps for the whole team. Therefore the pathfinding of the agents does not work correctly, and we see that the agents issue inexplicable actions.

6.2 Explanations

In order to help the debugging process, we created an option to make the agents explain their intentions. Explanations are brief strings that summarize the agent's current intentions. Due to their low level of detail, they are only usable with some kind of other debugging tool, such as visualization. Their level of simplicity is shown in Fig. 9.

In each step, all agents have an explanation, which is only shown for the given step. Explanations are not persisted, because their purpose is to provide brief information, so they can be read between the simulation steps, so the MAPC

agentA1	exploring		
agentA2	providing block	delivering	to agentA4
agentA3	providing block	connecting	to agentA4
agentA4	coordinating	assembling	with agentA3
⋮	⋮	⋮	⋮
⋮	⋮	⋮	⋮
agentAN	single block	collecting	block B1

Fig. 9. Explanation strings.

simulation server has to be paused. Unfortunately, the MAPC simulation server cannot be operated step-by-step, so the “continue” and the “pause” commands have to be typed in quickly on the console to see the progress of the explanations.

An agent’s explanation is equal to the agent’s active intention’s explanation string at the given time step. The explanation of an intention describes the actual state of the intention, such as its actual sub goal. These descriptions can be easily constructed, due to the hierarchic structure of the intentions. The explanation string is built by concatenating the embedded intention’s description with explanation string of the given intention.

Although the explanation strings of Fig. 9 look like simple print-outs on a console, they are a little bit more than that. The print-outs on a console runs away, and they are hard to follow, while the explanation window stays and displays the current behaviour of the agents. They are really explanations, because they describe why the agent is doing what it is actually doing. If we just see that the agent is going to somewhere, then we do not know why. If the explanation says that the agent is going to fetch a block from a dispenser to deliver it to a given agent, then it explains us why it going on its way. The explanation string may contain more complex explanations as well. For example we can put into the explanation string, that the agent became block provider of block x , because its bid for block provision had a given value. Such explanations were not needed for debugging. However, this type of explanation needs the modification of the source code. A more flexible solution would be to make the explanation feature interactive.

The main benefit of this low level of information providing is that the presence of bugs can be easily found out. Also, simple bugs can be eliminated without effort. However, in case of complex errors, these are only beneficial to approximate the error. For the identification of complex errors, the explanation strings would have to be made more complex, but the available space is not enough for that.

6.3 Logging

Log entries are detailed information about anything which might be involved in possible bugs. The most common entries are about intentions and blackboard data, such as intention actual state and decision or the modification of common data.

Generated log entries are detailed and therefore persisted. Due to their high level of detail, they can not be read properly between the simulation steps. This information are independently useable, although, more details are provided when used with server logs.

There is no exact schema for the content of log entries. The only exceptions are time step and the agents involved, which are almost always relevant. The rest of the content usually depends on the actual fault, which can be very diverse.

Contrary to the explanations, the implementation of the logging infrastructure is quite a challenge. A proper logging infrastructure has never been implemented. There were several reasons behind this decision. The main challenge was the implementation of a logging, which level can be regulated by various parameters. Due to the diversity of the faults, a lot of parameters would have been required, which could not be known in advance. By logging everything in high detail would have made the debugging nearly impossible without the right infrastructure.

Instead, a simple ad-hoc logging was implemented. This logging infrastructure was implemented every time after a fault occurred, and it was highly specialized to the fault. In the long term, this solution was not effective, because it had to be developed almost every time after a new fault occurred.

The main benefit of the logging is that complex faults can be tracked down more easily. Blackboard data change and the behaviour of the agent intentions are more traceable with this method.

7 Match Analysis

In order to evaluate the quality of the jobs done by the implemented MMD system, we summarize the performance of all the teams at MAPC 2022 in Table 1. The table includes the warmup matches against master student Paula Böhm from TU Clausthal, outside of the competition. The columns of the table show the points collected by the teams against each team and each simulation run (or match with another word). The total amount of collected points by the team is at the bottom of the table, together with the scores and placements of the teams. The points are the rewards for the tasks completed in the matches. The winner of a match is the team that collects more points. The scores are given for the matches: 3 for a win, 1 for a draw and 0 for a loss. The placement at the contest is determined by the total scores.

There is randomness in the matches, therefore the points collected by a team may vary with each match even if all the conditions are the same. When the MMD system was tested against the MMD system before the contest, sometimes

Table 1. Summary of the points at the contest. The columns contain the points of the teams against the teams in the rows.

		LI(A)RA	GOALdigger	MMD	FIT BUT	GOAL-DTU	Paula
LI(A)RA	Sim1		350	760	60	120	600
	Sim2		410	750	540	160	120
	Sim3		310	1600	780	0	1000
GOALdigger	Sim1	130		500	220	0	160
	Sim2	0		200	320	0	220
	Sim3	80		840	490	520	1270
MMD	Sim1	120	370		80	480	770
	Sim2	10	300		760	150	500
	Sim3	150	720		170	0	450
FIT BUT	Sim1	220	120	770		230	480
	Sim2	60	320	680		630	360
	Sim3	120	520	1140		0	250
GOAL-DTU	Sim1	310	180	910	0		710
	Sim2	80	370	780	670		610
	Sim3	270	410	1520	1640		570
Paula	Sim1	110	330	580	0	320	
	Sim2	60	700	790	330	280	
	Sim3	90	320	1690	60	0	
Total points:		1810	5730	13510	6120	2610	8070
Total score		9	22	30	19	9	N/A
Placement		4	2	1	3	4	N/A

there were big differences in the points collected by the two MMD systems, although the two systems had the same capabilities. Nevertheless, we can see in the table, that the total points mainly correlate with the scores and the placements.

Each team played 15 matches in Table 1, which is not a big number statistically, but the points collected by the MMD team in the matches at the contest and against Paula are similar to the points collected during our test matches when the MMD system played against the MMD system. The points collected by the MMD system in every match are mainly in the range 600–1600. There is one exception, the MMD system collected observably less points in the matches against the GOALdigger system. This is because the GOALdigger team had “saboteur” agents to block the agents of the competitor.

The design of the MMD system focused on the task completions in order to complete as much tasks as possible, and to collect as much points as possible. Table 1 confirms that the design goal was mainly achieved in comparison with other teams.

GOALdigger-AIG-Hagen vs. MMD. At the match against the GOALdigger system we noticed that many of the agents of the MMD system lost their energy and got deactivated for a while. After the contest, we have learned that the deactivation was caused by the attack from the “saboteur” agents of the GOALdigger system. Figure 10 shows such a situation. The MMD agent is surrounded by three GOALdigger agents, all of them issuing a clear action against the MMD agent. Because the agents are next to each other, the MMD agent loses its energy quickly, and gets deactivated.



Fig. 10. Three blue GOALdigger agents attacking one green MMD agent. (Color figure online)

The MMD system delivers multiple-block tasks with agents that stay around a goal zone for a longer time, either because they coordinate task delivery and are waiting for block delivery, or because they are delivering blocks and wait for their turn to hand over the block. This kind of task delivery strategy makes the MMD agents vulnerable to “saboteur” agent attacks which obviously reduced the task completion capability of the MMD system when played against the GOALdigger system.

Other teams may have had different kind of task delivery strategy, or they did not spend too much time to deliver multiple-block tasks at the goal zone, because they did not have salient performance degradation against the GOALdigger system, as it can be seen in Table 1.

The “saboteur” agent concept seems to be efficient in reducing the task delivery capability of a competitor like the MMD. However, if a team has several dedicated “saboteur” agents, then the team has less number of agents to deliver tasks, therefore the “saboteur” agents reduce the task delivery capability of their own team as well. The “saboteur” agents not only block the agents of the other team, but they block the goal zones as well if the attacked agent is on the goal zone, thus they reduce the task delivery capability of both teams. Our experience was that one of the bottlenecks of task delivery at the contest was the lack of enough free goal zone. Because of the lack of goal zones, we introduced the goal

zone reservation scheme, and we also took into account the availability of goal zones at the task selection.

The MMD agents also have the capability to attack other agents, but only while they are involved in task delivery, and they happen to have nothing important thing to do. Actually, in the configuration files of the contest, the roles that were able to deliver task had only maximum clear distance 1, which did not allow attacking other agents. On the other hand, the roles that were able to attack other agents did not have task delivery capabilities, so only dedicated “saboteur” agents could operate. Thus the MMD agents never attacked any other agent at the contest.

8 Conclusion

We have presented how we have implemented the MMD block building agents for the 2022 Multi-Agent Programming Contest (MAPC 2022). The system is implemented in a general programming language in order to compare the resulting system with other systems implemented in multi-agent programming languages. The key performance indicators of the implementation are the development time, the development efforts, and the quality of the job done by the implemented multi-agent system. The development time and the development efforts of the contest participants can be found in the introductory chapter of this book and in the answers to the questionnaire of each team. The quality of the job done by the MMD multi-agent system can be found in Sect. 7.

The development time of the MMD system is not more than any other system at MAPC 2022, because all the contest participants had the same deadline. In addition, the MMD team was formed only for the 2022 competition, while there were two other contestants who already had participated in the previous MAPC contest(s) with similar rules, so they could build on their previous implementation and experiences. Our experience showed that the implementation of our own version of the practical reasoning agent architecture for the individual agents and our own version of the blackboard architecture for the coordination of the agents could be done in time, and the use of a general programming language did not hinder us to meet the deadline.

The comparison of the development efforts of the teams is a bit difficult, because two teams already participated in the previous contests, and they only had to modify their systems. In addition, the declared development efforts are mainly rough estimates. Nevertheless, we can see that the modification of the already existing system required the least development efforts. The teams putting more effort into the development have achieved better placement. The teams using agent oriented programming languages had spent less effort for the development, but the less effort was reflected in their placement as well. We cannot conclude that the usage of a general programming language would require more development effort, because the more development effort resulted in better performance of the system. According to our experience, the biggest development challenge was the debugging of the system. The chosen debugging infrastructure

made the debugging easier, but even so, it was still the most time-consuming part of the development.

The contest results seem to indicate that the implemented MMD multi-agent system is competitive with the systems developed with agent-oriented software engineering methods. There is randomness in the contest and we cannot say that a better team always wins against the other team. In spite of this, the first position of the MMD team is reaffirmed by the total points of the contestants in Table 1. This indicates that the quality of the job done by an implemented multi-agent system mostly depends on the knowledge implemented in the system rather than the programming language used.

We could have implemented further improvements into the MMD system if we had more time. The different estimation and cost calculations in the system are only approximate, and more precise calculations would probably give better results. More efficient resource management of roles, agents and tasks would be another improvement. We did not exploit the “saboteur” capability of the agents, and although we thought of “saboteur” agents in other teams, we have not prepared any defensive behaviour. A surveyor agent could make the exploration of the world faster in the beginning of the match. The pathfinder algorithm could be improved by multi-agent pathfinding algorithms [14]. These are future works for the contests of the next years.

We think that we have implemented a basic agent architecture and a basic blackboard architecture for the collaboration of the agents, and these architectures can be used in other MAPC scenarios as well. Of course, modifications are necessary if the scenario changes, but this holds for multi-agent programming language implementations as well. If the environment changes, then the interaction with the environment, as well as the internal model of the environment have to be changed in both approaches. If the logic of the scenario changes, then the logic implemented in the agent team has to be changed in both approaches. Because the implementation effort of the MMD system is comparable to the implementation efforts of the other teams, we think that the modification effort would be comparable as well.

16th Multi-agent Programming Contest: All Questions Answered

A Team Overview: Short Answers

A.1 Participants and Their Background

Who is part of your team?

Miklós Miskolczi, László Z. Varga

What was your motivation to participate in the contest?

We wanted to do an experience with multi-agent systems, and of course we wanted to be the winner.

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

The MSc diploma work of Miklós Miskolczi.

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

Multi-agent systems, online routing game model, multi-agent path finding.

A.2 Statistics**Did you start your agent team from scratch, or did you build on existing agents (from yourself or another previous participant)?**

The agent team was started from scratch, but we used a modified version of the experimental [Python client](#) for 2020/21 edition of the Multi-Agent Programming Contest to communicate with the contest server.

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

We started in February 2022 and worked on the program 28 h per week, a total of 896 h.

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

Continuous development. The last two weeks mainly testing.

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

github.com/AlDanial/cloc v 1.94 T=0.13 s (649.6 files/s, 72314.5 lines/s)

Language	files	blank	comment	code
Python	83	1882	2050	4842
Text	1	121	0	553
Markdown	1	3	0	12
SUM:	85	2006	2050	5407

The above data include the modified experimental [Python client](#), which is: github.com/AlDanial/cloc v 1.94 T=0.05 s (20.9 files/s, 15498.5 lines/s)

Language	files	blank	comment	code
Python	1	115	102	526

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?**

No.

Methodologies (e.g. Prometheus)?

No.

Notation (e.g. Agent UML)?

No.

Coordination mechanisms (e.g. protocols, games, ...)?

We used a simple (one level) Contract Net protocol, which includes a simplified auction mechanism.

Other (methods/concepts/tools)?

We used our own version of the practical reasoning agent architecture. We used our own version of the blackboard architecture for the coordination of the agents.

What hardware did you use during the contest?

Hardware	Specification
Processor	AMD Ryzen 5 3600 6-Core Processor
RAM	16 GB
OS	Windows 11 Pro

Only about 20% of the processing power of the computer was used by the agent team.

A.4 Agent System Details**Would you say your system is decentralised? Why?**

Although the coordination of the agents is done by a central blackboard, and the implementation of the whole agent team is a single Python program, the system can be seen as a decentralised one in the sense that the planning and the activities of the agents are done individually. The agents were meant to be separate threads, but threading in Python is not fast enough, and we had to refactor the code to speed up the system. Python multiprocessing might be the solution for the next competition.

Do your agents use the following features: Planning, Learning, Organisations, Norms? If so, please elaborate briefly.

The actions needed to achieve a goal is basically hardcoded in the implementation. Simple learning is used to discover e.g. the cost of a clear action. In order to solve a multiple-block task, the agents are organised into a sub-team. The sub-team is connected through the intentions of the members. The intentions are assigned by the blackboard.

How do your agents cooperate?

Cooperation is done through the intentions of the agents (which is a simplified and direct communication between the agents) and the shared blackboard which includes the shared maps as well. The planning for the cooperation is hardcoded in the intentions.

Can your agents change their general behaviour during run time? If so, what triggers the changes?

Intentions are reconsidered in each simulation step. If there are changes in

the environment, then the agents may change their intention. The behaviour of the intentions may be different depending on the match configuration. The capabilities of the agents depend on the match configuration as well. The behaviour of the blackboard depends on the current state of the environment and the agents.

Did you have to make changes to the team (e.g. fix critical bugs) during the contest?

The code and the settings were not changed, but there was a problem with the connection to the server at the second and third simulation of each match, and the agent team had to be restarted manually after the first simulation steps. Interestingly, this problem did not occur during the warm-up match before the competition with the real competition server. Also, this problem did not occur with the localhost.

How did you go about debugging your system? What kinds of measures could improve your debugging experience?

The basic “debugging tool” was the printout on the console, but we also implemented an “explanation function”. If the system is run with the explanation function, then the agents give information on what they are doing. The given information might be their beliefs, or their current intention and its details. The server logs and replays were also used to trace back various complex cases.

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

The agents printed on the console the same information as those during the testing period before the contest. It was helpful in the sense that we could see that everything goes well.

Did you invest time in making your agents more robust/fault-tolerant? How?

Robustness and fault-tolerance was part of the development process.

A.5 Scenario and Strategy

How would you describe your intended agent behaviour? Did the actual behaviour deviate from that?

The agents mainly do what they are intended to do. Sometimes they produce strange behaviour, but we know that this may be due to the incompleteness of the solution. For example, individual route planning may produce deadlock like situation.

Why did your team perform as it did? Why did the other teams perform better/worse than you did?

The results at the competition were similar to those at the testing period, excluding the cases when the other teams were heavily aggressive against opponent agents.

We do not know much about the other teams.

Did you implement any strategy that tries to interfere with your opponents?

Yes.

The tolerant way: If our agents notice that the other team stay in a goal zone for a long time at the place needed for our team, then our team go to another place.

The aggressive way: When our agent is at the goal zone, then it tries to keep the agents of the other team away from the goal zone by shooting at other agents approaching the goal zone, assuming that the role capabilities of our agent allows this. This goal zone defence behaviour was not possible with the match configuration of the competition, so our agents did not shoot at the other team during the competition.

How do your agents coordinate assembling and delivering a structure for a task?

Multi-block tasks are delivered by a single coordinator agent and block provider agents for each block. The coordinator goes to the selected goal zone. The coordinator clears the surrounding of the goal zone until the first block provider arrives. Block providers fetch the block from a dispenser and take it to the surrounding of the coordinator. The block provider waits until the call from the coordinator. When the call arrives, then the block provider takes the block to the place requested by the coordinator, and then the two agents connect the blocks.

Which aspect(s) of the scenario did you find particularly challenging?

Map building, map merging, map update, map size determination, path finding on the looping map. Shortly: dynamic map management.

Limited (and in our opinion, not realistic) perception of the agents, which means, among others, the following: When the agent moves and there is a failure, then the agent does not know which step failed. The agent does not know which blocks are attached to which agent.

What would you improve (wrt. your agents) if you wanted to participate in the same contest a week from now (or next year)?

We have ideas, but we keep them for the next competition. Surely we have to prepare to defend our agents from the potential saboteur agents of the other team.

What can be improved regarding the scenario for next year? What would you remove? What would you add?

Perception capabilities of the agents (see above).

There were only two active tasks in the current scenario, and often there was no big difference between the two tasks. Therefore a good task selection strategy was not so critical in the current scenario. Bigger choice of tasks would be more challenging.

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

What did you learn from participating in the contest?

Good programming and debugging exercise in a non-deterministic and hardly reproducible environment.

Building an agent architecture from scratch in a general programming language.

What advice would you give to yourself before the contest/another team wanting to participate in the next?

Now we have more knowledge to build a cleaner agent architecture.

Where did you benefit from your chosen programming language, methodology, tools, and algorithms?

The main benefits were the development speed and the simplicity. We followed the “keep it simple principle” to ensure fault-tolerance and make components open for extensions and optimizations.

Which problems did you encounter because of your chosen technologies?

Performance issues. Full parallel operation would need another implementation approach.

Programming errors are signalled in Python only when the actual line of code is executed. This way, it is easy to make errors.

Which aspect of your team cost you the most time?

Architecture building, safe map management, path finding and ensuring fault-tolerance.

A.7 Looking into the Future

Did the warm-up match help improve your team of agents? How useful do you think it is?

We did not change anything after the warm-up match, but it was good to know that the connection to the server works.

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?

The positive aspect would be that all teams have the same conditions (for example network speed).

The negative aspect would be that we cannot correct any problem during the competition. For example we had to restart the team manually, because the connection to the server did not work the same way as at the warm-up match.

Do you think a match containing more than two teams should be mandatory?

This might be a possibility, but probably with not too big team sizes.

What else can be improved regarding the MAPC for next year?

Nothing more than those already mentioned above.

References

1. Ahlbrecht, T., Dix, J.: Multi-agent programming contest - Lecture 2 at 15th Workshop-School on Agents, Environments, and Applications. <https://www.youtube.com/watch?v=HgNlfKm7YdQ&t=1417s>. Accessed Nov 2022
2. Ahlbrecht, T., Dix, J., Fiekas, N., Krausburg, T.: The multi-agent programming contest: a Résumé. In: Ahlbrecht, T., Dix, J., Fiekas, N., Krausburg, T. (eds.) MAPC 2019. LNCS (LNAI), vol. 12381, pp. 3–27. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-59299-8_1
3. Bratman, M.: Intention, Plans, and Practical Reason. Harvard University Press, Cambridge (1987)
4. Cardoso, R.C., Ferrando, A., Papacchini, F.: LFC: combining autonomous agents and automated planning in the multi-agent programming contest. In: Ahlbrecht, T., Dix, J., Fiekas, N., Krausburg, T. (eds.) MAPC 2019. LNCS (LNAI), vol. 12381, pp. 31–58. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-59299-8_2
5. Durfee, E.H.: Cooperative distributed problem solving between (and within) intelligent agents. In: Rudomin, P., Arbib, M.A., Cervantes-Pérez, F., Romo, R. (eds.) Neuroscience: From Neural Networks to Artificial Intelligence. NEURALCOMPUTING, vol. 4, pp. 84–98. Springer, Heidelberg (1993). https://doi.org/10.1007/978-3-642-78102-5_5
6. Edelman, B., Ostrovsky, M., Schwarz, M.: Internet advertising and the generalized second-price auction: selling billions of dollars worth of keywords. *Am. Econ. Rev.* **97**(1), 242–259 (2007). <https://doi.org/10.1257/aer.97.1.242>
7. Englemore, R., Morgan, A.: Blackboard Systems; Edited by Robert Englemore, Tony Morgan (the Insight Series in Artificial Intell, 1st edn. Addison-Wesley Longman Publishing Co., Inc., Boston (1988)
8. Hansen, E.A., Zhou, R.: Anytime heuristic search. *J. Artif. Intell. Res.* **28**, 267–297 (2007). <https://doi.org/10.1613/jair.2096>
9. Hart, P., Nilsson, N., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Syst. Sci. Cybern.* **4**(2), 100–107 (1968). <https://doi.org/10.1109/tssc.1968.300136>
10. Jennings, N.R.: Coordination through joint intentions in industrial multiagent systems. *AI Mag.* **14**(4), 79 (1993). <https://doi.org/10.1609/aimag.v14i4.1071>. <https://ojs.aaai.org/index.php/aimagazine/article/view/1071>
11. Koenig, S., Likhachev, M.: D*lite. In: Proceedings of the Eighteenth National Conference on Artificial Intelligence and Fourteenth Conference on Innovative Applications of Artificial Intelligence, Edmonton, Alberta, Canada, 28 July–1 August 2002, pp. 476–483 (2002). <http://www.aaai.org/Library/AAAI/2002/aaai02-072.php>
12. Sandholm, T., Lesser, V.R.: Issues in automated negotiation and electronic commerce: extending the contract net framework. In: Proceedings of the First International Conference on Multiagent Systems, San Francisco, California, USA, 12–14 June 1995, pp. 328–335 (1995)
13. Smith: The contract net protocol: high-level communication and control in a distributed problem solver. *IEEE Trans. Comput.* **C-29**(12), 1104–1113 (1980). <https://doi.org/10.1109/tc.1980.1675516>
14. Stern, R., et al.: Multi-agent pathfinding: definitions, variants, and benchmarks. In: Proceedings of the Twelfth International Symposium on Combinatorial Search, SOCS 2019, Napa, California, 16–17 July 2019, pp. 151–159. AAAI Press (2019)

15. Uhlir, V., Zboril, F., Vidensky, F.: Multi-agent programming contest 2019 FIT BUT team solution. In: Ahlbrecht, T., Dix, J., Fiekas, N., Krausburg, T. (eds.) MAPC 2019. LNCS (LNAI), vol. 12381, pp. 59–78. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-59299-8_3
16. Vázquez-Salceda, J.: The Role of Norms and Electronic Institutions in Multi-agent Systems. Birkhäuser Basel (2004). <https://doi.org/10.1007/978-3-0348-7955-2>
17. Wooldridge, M.: Understanding equilibria in multi-agent systems. In: Keynote presentation at FTC 2021 - Future Technologies Conference 2021 (2021). <https://youtu.be/Iqm8UTXUG24?t=411>. Accessed Nov 2022
18. Wurman, P.R., D’Andrea, R., Mountz, M.: Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI Mag.* **29**(1), 9 (2008)