General deSouches Commands Multi-agent Army for Performing in Agents Assemble III Scenario: FIT-BUT at MAPC 2022

Frantisek Zboril(✉), Frantisek Vidensky, Ladislav Dokoupil, and Jan Beran

Department of Intelligent Systems, Faculty of Information Technology, Brno University of Technology, Brno, Czech Republic
{zborilf,ividensky,iberan}@fit.vutbr.cz
https://www.fit.vut.cz/.en

Abstract. The Multi-agent Programming Contest provides a good opportunity to compare different approaches to creating a multi-agent system for given scenarios. In this edition, the FIT-BUT team tested solutions based on their own design of a multi-agent system, both on its organisational levels and on the actual architectures of the agents used. The three-level organisation of our multi-agent system was based on the principle of central strategic planning, which results in strategic goals for some subset of agents. These agents execute the goals as their mission and are coordinated by the agent that lies between them and the central element in the hierarchy. The resulting system has stood up to competition and some aspects of it may inspire the design of other multi-agent systems.

Keywords: Artificial Intelligence · Multi-Agent Programming · Decision-making Planning · Self-organisation · Rational Agents

1 Introduction

The third participation of our team in the Multi-agent Programming Contest (MAPC 2022) was also the third opportunity to design a multi-agent system for the assignment called Agents Assemble. The scenario is based on composing specific patterns from blocks in a discrete 2D space. Agents move in a discrete environment with obstacles. Other objects there are places where blocks are dispensed (dispensers), zones where the role of the agent can be changed (role zones), zones where tasks are submitted (goal zones) and possibly assemble the patterns to be submitted. An agent does not know its absolute position, but it has a line of sight to a certain distance. If it sees another agent, it does not know which one it is specifically, but only whether it is an agent from its own team or from another team.

During the competition round, tasks appear randomly. Agents must assemble certain pattern from basic blocks and submit it within a limited time to complete

T. Ahlbrecht et al. (Eds.): MAPC 2022, LNAI 13997, pp. 120–150, 2023.
https://doi.org/10.1007/978-3-031-38712-8_5

the task. Blocks can be acquired at several positions and are of different types. A correctly composed pattern from the corresponding blocks can be submitted. Roles give the agents various abilities including the range of their oversight, the ability to carry multiple blocks, the ability to take multiple actions in one turn or perform actions to remove obstacles or attack other agents. The dynamics of the environment are due to the fact that so-called ‘clear events’ can occur in the environment. When a ‘clear event’ occurs in some area, obstacles are rearranged and agents near the location of the event can be paralysed and deprived of the blocks they are currently carrying. For a more detailed description of the scenario, we refer to the competition website.

In this text, we present a description of our solution and specifically the architecture and the implementation of the multi-agent system we developed for the competition. We have developed our own system which is not based on any existing agent or multi-agent system as part of this competition. We understand that the right technical work should use the best and most appropriate solutions that have been developed in the field, but the desire to try and build a system of our own based on our knowledge prevailed. We have done this for the following reasons:

- Practical experience to find out the pitfalls and problems not necessarily visible to the user when building a multi-agent system in the still widely used programming language.
- Some frameworks and platforms that are now popular for implementing multi-agent systems tend to be already used by other teams in this competition [2,5]. For reasons of diversity, we wanted to come up with a new approach and see if it can compete with those systems.
- We intended to discover new possible approaches in the design of multi-agent systems.
- We intended to use a generic, non-agent programming language for wider possibilities to implement some parts of the agent (planning, group reasoning, coordination).

We summarise how this approach has worked for us in the conclusion of this text.

Our system is named after General deSouches. He is a historical figure who is connected to the city of Brno, where the technical university from which our team comes is located. The hierarchical decomposition of the problem at hand into coalitions and individuals controlled by a central agent, which is used in this architecture, may resemble military organisations, and it is General deSouches who was chosen as the person connecting the military and our city.

The following text aims both to describe this architecture and to implement a concrete multi-agent system for deployment in MAPC competitions, more specifically for the tasks called Agents Assemble. In the second section, we provide a description of our architecture, and in the third section, we provide a concrete implementation of our system for the competition. We then discuss the results achieved in the competition and the advantages or disadvantages of the chosen solution.

2 deSouches Multi-agent Architecture

As in previous cases of our participation, we announced that Java will be our programming language. In the previous two editions [12,13], a system based on hierarchical planning was used, combining proactive planning on individual layers with their activation according to priorities in the hierarchy. This architecture was inspired by Brooks' system [3].

In this year's competition, we designed a new system architecture. In order to specify it, we will present its structure as the interconnection of the expected roles in the whole system. By roles in the system, we will mean general roles in the system architecture. Later the term 'roles' in a different sense will also be applied to agents at the lowest situated in the environment, so when we talk about roles in the system we will mean roles in the deSouches architecture. Next, we describe the different roles in the system and the interactions between them.

Our architecture is based on a hierarchical grouping of roles in a system of agents, where there is a control agent (deSouches) at the top level. At the second level, then there are coordinating agents, and the third level consists of agents that are situated in the environment and act to achieve specified goals in that environment. Agents at the lowest level interact with the environment they are in, perform actions in it and receive inputs. Agents at higher levels do not interact directly with the environment, they only interact with other agents in the system, as shown in Fig. 1.

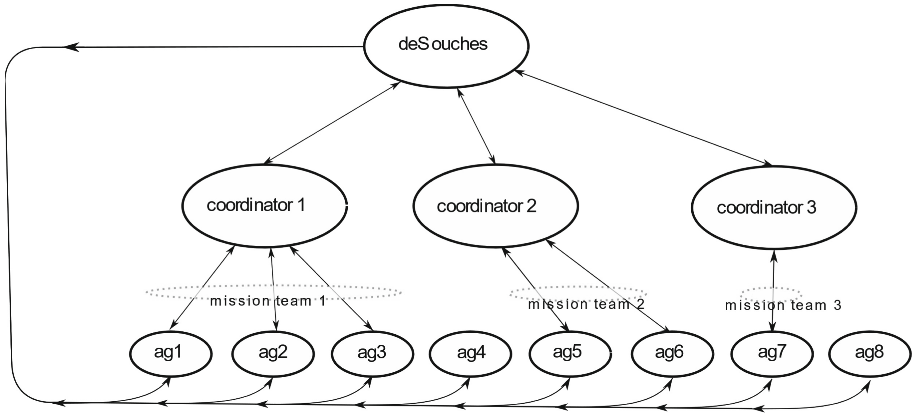

Fig. 1. Architecture of the deSouches Multi-agent System

Situated agents, as we will call third-level agents from now on, have their internal state defined by their mental states, including beliefs, goals or desires and intentions. This is common in agent systems; for example, in BDI [9] or 2APL [4] systems, beliefs, desires (as possible goals of the agent) and intentions are the basic elements. The system can also identify options and exceptions that can lead to changes in the mental states of one or more agents. Similar to the distinction we made between agent and organisational levels in the arrangement of individual agents, we will also distinguish beliefs, goals and options according to the purposes they serve within the multi-agent architecture. As for beliefs and goals, we will distinguish them according to whether they serve to make decisions at the situated agent level or to make organisational decisions. We will talk about strategic beliefs if they serve as practical reasoning about strategic goals, and strategic goals will be assignments for an organised group of agents. Next, at the organisational level, we will talk about options that can trigger reasoning and exceptions that can cause a revision of the currently specified goals. Situated agents at the lowest level will be driven by intentions, as persistent goals, that have been assigned to them from higher levels.

The meaning of options and exceptions as well as the meaning of the adjective 'strategic' for our system deserves further explanation. It will be presented in the following paragraphs, where we describe how goals are organised in the system and the plans that are used to achieve them. We then introduce the different organisational roles and their interactions.

2.1 Hierarchy of Goals and Missions

A common approach to representing the hierarchy of plans for artificial agents is the Goal-plan tree [11]. Such a model is based on BDI systems and is designed to explore the possible ordering of an agent's intentions. In our system, we have three levels of goals. Strategic goals are the top-level goals and are set by deSouches. A strategic goal can be atomic in the sense that it is given directly to an agent and if the agent achieves it, the goal is considered satisfied. Non-atomic goals can either be decomposed for processing by multiple agents, or into a sequence of goals for a single agent, or both. Similar to the JaCaMo system [6], we call a mission a group goal for agents acting coherently to achieve it. Then we first consider decomposition within an agent group and possibly subsequently into other subgoals. While the first decomposition can be described as an AND decomposition, in the case of the decomposition of the lowest-level goals, we have an OR decomposition of some sequence of agent mission goals that are to be (all) satisfied sequentially and is thus again an AND decomposition. In Fig. 2 we show these decompositions such that the sequences are words from a language.

Each mission has a goal, which is divided into subgoals for individual agents. If we assume that each agent must satisfy its subgoal to complete the mission, we can view such a decomposition as an AND decomposition. This goal need not be atomic but can be viewed as some sequence of lower-level subgoals whose successive execution achieves the agent mission goal. Specifically, it is represented by a finite state automaton, where states are subgoals and transitions are determined by the results of attempts to achieve these subgoals. These results are usually "succeeded"/"true" or "failed", but are not limited to them. Thus, an agent for executing a mission goal may execute different sequences of subgoals. Since the state automaton accepts regular language, we can think of such sequences as words of some language accepted by the automaton.

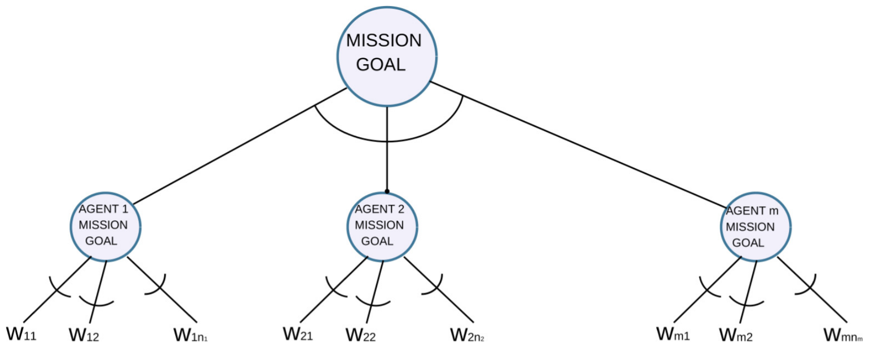

Fig. 2. Goal Hierarchy in deSouches Multi-agent System

Consider that $g1$ is the subgoal of the agent to reach the dispenser and pick up a block, $g2$ is the subgoal of getting to the goal area, $g3$ is the subgoal of waiting for other agents to arrive at their positions, and $g4$ represents the subgoal of connecting the block to other agents. If an agent sequentially executes all these subgoals, then their ascent is a success, which is represented by the terminal symbol t and the automaton should accept the word $t\ t\ t$. However, if the agent fails to execute goal $g2$, for example, due to a clear event, the agent must retry to get the block and execute the whole process from the beginning. Thus, the result is a sequence $t\ fce\ t\ t\ t$, where the terminal symbol fce denotes the failure to reach the goal due to a clear event. However, despite the failure in the subgoal, the mission goal was achieved and hence this sequence will be accepted by the automaton. In the goal decomposition shown in Fig. 2, we then represent the above as an OR decomposition into the individual words accepted by the automaton for a given subgoal, and the individual sequences must then be executed as a sequence of subgoals according to the states over which the automaton accepted the word.

We will mention how these goals are set and evaluated later in the section on coordinator agents. We now return to the hierarchical structure of system roles and describe these roles in turn, and in the case of situated agents, their architecture and operating principle. To better organise the description of the roles, we define the roles according to the GAIA methodology [14] using their responsibilities, activities, interaction protocols, and the permissions (access to data) they work with.

2.2 Top Level: deSouches/central Agent

As we mentioned earlier, we chose the name deSouches for the control-level agent which is at the top of the system management hierarchy. It is responsible for setting the strategies of the entire multi-agent team during its operation. Conceptually, deSouches creates missions and assigns agents to them from the population of situated agents at the third level. As a rational agent, it is tasked with assigning goals in a way that best fits the stated goals of the entire system. deSouches works with beliefs passed to him by the situated agents and also keeps the information about the current state of assigning agents to missions. It performs group reasoning in each cycle after it receives information from all the situated agents. This group reasoning results in options that can lead to the creation of missions. The creation of missions is also related to the creation of a mission coordinator, i.e., a second-level agent, and the assignment of initial goals to the agents assigned to the mission.

More specifically, the responsibilities of this role can be described as follows

- It creates a population of situated agents at the third level and assigns them primary goals. The population size is constant.
- It assigns goals to agents on request.
- It processes strategic information from lower level agents (as it comes in)
    - Based on this information, it identifies strategic options.
    - It processes mission termination information.
- It processes strategic options (after the barrier, which will be mentioned in more detail later)
    - It defines strategic goals.
    - It creates a mission coordinator and assigns situated agents to the strategic objectives. At the second level, agents are created and terminated as missions are created and terminated.
- It assigns a goal to a situated agent based on options, or when asked to do so by such an agent.

The activities are based on the responsibilities stated above:

- An agent must be able to process information provided by other agents, identify options in the system within which it operates, be able to rationally identify missions, and select appropriate goals for agents when requested. Related to this, it maintains a record of missions assigned to agents.
- In terms of data, it maintains information about the teams formed. It also manages aggregated data from each situated agent in the environment. It also creates strategic goals and objectives for individual agents based on the strategic decisions made.
- It receives synchronisation information (salut) from the agents and starts making strategic inference after receiving salutes from all agents in a given interpretation cycle.
- Is informed by the coordinators of the completion of missions, whether successful or unsuccessful.

Protocols determine how a role interacts with other roles in the system. deSouches is the bottleneck of the entire system. If deSouches fails, mission teams will stop forming. We have not addressed this issue in this version of the system, however, to guarantee the robustness of the multi-agent system, we will consider some possibility of decentralization or substitutability at the control level for the following competitions.

2.3 Second Level: Organisational Agents

We have already introduced agents at the organisational level in the section on organising goals in systems. They are created by the central agent after it has created a mission. Their task is to assign goals to the situated agents and to process information from them about the outcome of their actions. We have stated that we understand sequences of results of specified goals as words of some language. These words lead to the achievement of a goal to be accomplished by an agent within a mission. In our system, the language is accepted by finite state automata and thus is regular. Agents with behavioral models tend to be implemented as automata [7], and even in the case of this system, the automaton will receive as input the results of the agent's efforts to execute the mission subgoal, and the output will be the agent's new/next mission subgoal. If the automata reaches the goal state, the agent has also fulfilled its mission goal. Since a mission can be performed by multiple agents, the coordinator operates the automata for each of them. In addition, it can synchronise these automata. Thus, if an agent has finished processing a goal and is to be synchronised with one or more agents, the coordinator will only issue another goal to the agent when it can issue all other goals to other agents that are in a barrier.

The agent may report exceptions to the coordinator. These have already been mentioned above and are worth commenting on here. This is a situation that the agents discover that could lead to a change in mission control. They are not informed about the outcome of the agent's efforts, but they may lead to a change in the assigned goals of one or more agents in the mission team, or to the termination of the mission. Again, the exceptions are not specified and depend on the system implementation for the task.

The responsibilities of the agent coordinator are:

– Creating missions and informing assigned agents that they are assigned to a mission and assigning them primary objectives.
– In the event of mission termination, informing the assigned agents that they are released from the mission.
– Processing the results of the agents' efforts to accomplish the missions.
– Assigning goals to individual mission agents on an ongoing basis.
– In the event of mission termination (successful or unsuccessful), informing deSouches of this.
– Reacting to exception communicated by agents.

The data they are working with is just the reported output from the situated agents' efforts to meet the assigned goal and the reported exceptions from those agents as well. They produce data in the form of goals to these agents and a report on the outcome of the deSouches mission when it is completed. The activities and protocols associated with this are first initialising the agent team for the mission, processing the data from the agents, processing the mission completion, i.e., releasing the agents, and then informing deSouches.

2.4 Third Level: Situated Agents

The agents operating at the lowest level are the agents that actually work on specific tasks and operate in the environment in which the multi-agent system is situated. The architecture we use for these agents is inspired by current agent architectures, especially those of BDI, but we have retained the freedom to implement it and have omitted some specific constraints that are present elsewhere. For example, some parts that are specified in BDI systems, such as beliefs, events, or even the processes of selecting plans to selected goals, are not so constrained in our case and the implementation can use the resources provided by Java.

Below we present the control loop of our agent, which we describe in more detail later:

Algorithm 1: Agent's control loop

```
Loop
    Procedure sensing
        get information from environment;
        if ordered by coordinator or deSouches then
            adopt new goal;
        propagate last action result to intention pool;
        update belief base;
        send strategic information to deSouches;
        salut deSouches ;    /* sending strategic information finished,
          synchronise */
    Procedure practical reasoning
        select intention from intention pool;
        if last action demanded a subgoal then
            push the subgoal to the intention;
        get goal at the top of the intention;
        revise plans for that goal;
        execute one step of highest priority plan of that goal;
```

In general, however, an agent makes inferences based on data that represent the agent's perception of the state of the environment it is situated in. The types of data depend on the particular implementation of the system. They can be atomic elements of the types available in Java, as well as complex structures with their own reasoning mechanisms, for example for building a map of the environment they are in. Agents modify the beliefs in their belief base appropriately in each cycle. Agents also pass strategic information to deSouches. What is considered as strategic information depends once again on the specific implementation of the overall system. After sending all such information, they send the information that everything needed for this cycle has already been sent. We call this last message a salute.

The next stage in the agent's control cycle is practical reasoning about goals and the means to achieve them. The agent is supposed to work proactively towards its goals. These are persistent goals that the agent follows until it achieves them or finds that it is not possible. This is how we understand the agent's intention in our system as well. What differs from current BDI systems is that the goal for which the intention is formed is specified by deSouches or the mission coordinator, which we have already introduced in the previous sections. Thus, it is not created in response to an event but is set from outside.

The way to accomplish the goal, if accepted as an intention, is the execution of some relevant plan. This means that the goals in our system are procedural. This plan is a linear sequence of actions. The planning and re-evaluation of plans are performed during each cycle of the agent for the chosen goal in this cycle, thus guaranteeing its reactivity. This is an example of the difference from classical BDI systems, where reasoning is first performed to process events and find relevant and applicable plans to them. In our system, practical reasoning is performed by first selecting a goal and selecting plans for it. Of course, a change of plan does not have to take place every time, as long as the previously constructed plan for the goal is suitable. After the plan re-evaluation phase, the agent uses the plan and executes one step from it.

An agent can create multiple plans for a single goal with different priorities. For example, if an agent is following a path according to a plan and sees an enemy agent, it can create a short plan with a higher priority to attack the enemy. Once this is completed, the agent will move on to the lower priority plan. Based on this example, we introduce one more attribute of a plan, namely the one that determines whether its execution will result in the goal being met or not. The goal is then satisfied only when the plan that is set to be final is fully executed.

The agent may also set subgoals during the re-evaluation of plans. We introduced this capability into the system despite the fact that goals are primarily set by agents at higher levels, and the ability to create a priority hierarchy of plans may mimic the way goals are hierarchically created in BDI systems, where plans may include declared subgoals. In the case of our system, subgoals are rather considered as procedures that are executed by agents for different goals. For example, a 'block pick up' goal may have as a subgoal to get to a given position (near the dispenser) and a goal for 'approaching goal area' may use the same subgoal to approach a given position.

Since an agent can have multiple intentions at the same time, and since we have not limited the possibility of an agent being in multiple teams at the same time, the agent's intention pool will be part of the situated agents. Figure 3 shows an example of an intention pool with two intentions. Each intention has a goal and one or two subgoals, where these goals or subgoals have one to three plans with given priorities (to the right of the plan name) at any given time, some are final and some are not (green or red box). In addition, the intention pool stores a reference to the most recently processed intention and a reference to the most recently executed plan for each goal or subgoal. This is in order to

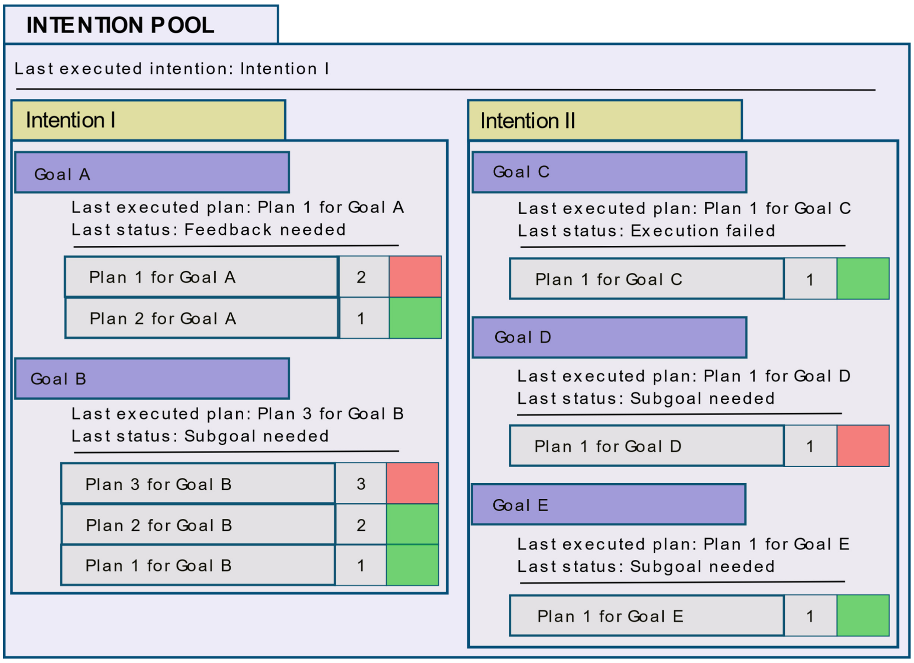
INTENTION POOL
Last executed intention: Intention I
Intention I
Goal A
Last executed plan: Plan 1 for Goal A
Last status: Feedback needed
Plan 1 for Goal A
2
Plan 2 for Goal A
1
Goal B
Last executed plan: Plan 3 for Goal B
Last status: Subgoal needed
Plan 3 for Goal B
3
Plan 2 for Goal B
2
Plan 1 for Goal B
1
Intention II
Goal C
Last executed plan: Plan 1 for Goal C
Last status: Execution failed
Plan 1 for Goal C
1
Goal D
Last executed plan: Plan 1 for Goal D
Last status: Subgoal needed
Plan 1 for Goal D
1
Goal E
Last executed plan: Plan 1 for Goal E
Last status: Subgoal needed
Plan 1 for Goal E
1

Fig. 3. Structure of Intention Pool (Color figure online)

properly propagate the result of the last executed plan action. In the example above, *Goal A* is subgoal of *Goal B*, *Goal C* is subgoal of *Goal C* and *Goal D* is subgoal of *Goal E*. *Goal A* can be achieved by executing *Plan* 2 for *Goal A*. However, *Plan* 1 for *Goal A* should be executed first at that point in time because it has a higher priority. This can be changed during the re-evaluation of the plan. If *Goal A* was achieved, *Plan* 3 for *Goal B* which triggered it may continue. If this plan succeeds, *Goal B* would still need to be achieved for Intention I to be fulfilled. This is because *Plan* 3 for *Goal B* is not final. It can be achieved by executing plans 1 or 2 for this goal.

The structure of the intention pool is re-evaluated twice during the control loop. Firstly, after the agent receives the results of an action execution and secondly, when a goal is to be followed. The reason why both do not occur at the same time is that there may be some additional cycles between the processing of the action result and the execution of the intention again. Because of this, the result must be processed immediately in the next cycle by the currently selected intention, while another intention can be executed later in the same cycle.

The result of an action can be interpreted in different ways. The actions and plans themselves are some objects, and an object may have as property information about how to handle the failure of execution. An action may cause the plan to fail and be removed, or conversely, the action may be retained and be repeated, or retained for a given number of repetitions or with a given probability.

Following a goal can end in failure if no plan is available after the plan revision. If agent's mission goal fails, the coordinator is informed. If a subgoal fails, this is then treated as a failure of the action of setting the subgoal in some higher-level plan, and treated as we described for failures of actions as such. A summary of the role of the situated agent is that the responsibilities of these agents are to try to achieve the specified goals and to report the result to the coordinator. Furthermore, agents must share strategic beliefs with deSouches and inform it also about exceptions according to the specific implementation. On the other hand, an agent may ask deSouches for strategic information if it has to make some decision that should be in line with the strategy of the whole team. Thus, while communication with the coordinator is given in the form of success/failure of the goals, communication with deSouches depends on the specific implementation.

This concludes the discussion of the deSouches agent architecture. In the following paragraphs, we will discuss a concrete implementation of a multi-agent system using this architecture, specifically for the MAPC 2022 competition.

3 Multi-agent System Design for the Agents Assemble III

The first and so far only implementation of deSouches is a system for the Multi-agent Programming Contest and Agent Assemble tasks. In the following paragraphs, we describe the concrete implementation of this system at the control, organisational, and situated agent levels. First, however, we will present our approach to the implementation of the situated agents in Agent Assemble III.

3.1 Strategy and Mission of the FIT-BUT Team for Agent Assemble III

The idea behind Agent Assemble scenarios is to get as many points as possible within a given number of steps. Points are earned by the agent (from now on, unless we use the term agent with further specification, we will mean situated agents) submitting an object of a specified pattern at one of the designated locations. The pattern can be a single block but it can also be more complex patterns that can be assembled from multiple blocks. We have identified all possible constructions for two to four blocks. From experience in previous years, we have estimated the maximum number of blocks to be just four since more complex patterns have not appeared before and the probability of successful delivery of a pattern decreases with the number of blocks needed. In total, we considered 42 possible patterns for these numbers of blocks.

Our solution was based on the execution of a task by a group of agents to deliver every single block of the desired pattern to a goal area. If all the agents in the group successfully transport their block to the goal area, they interconnect them together and one of them then submits the pattern to agents. It is therefore necessary that such agents have knowledge of where the pattern will be constructed and where each agent should bring its block. To do this, however, the agents must know each other's position. Therefore, in the first phase of the run, the agents try to coordinate with each other and achieve a shared coordination system. After their mutual positions are known for a group of agents, as well as their positions with respect to the dispensers and the goal area, such agents can be used for a mission during which they accomplish the task. The next question is what capabilities such agents must have. Although the roles of agents can be defined in any way, the teams were informed that the set of roles would be given in advance and there would be no need for a special reasoning mechanism to analyse offered roles. In our case, in addition to the default role that was assigned to the agents at the beginning, we limited ourselves to the worker and digger roles. The worker role gave the agents the ability to request blocks from the dispenser, assemble them, and submit the required constructs. The digger role had better skills for removing obstacles in space and possibly attacking other teams' agents.

This year, norms were added to the scenario. In multi-agent systems, a norm is a collective commitment by members of the organisation to follow certain rules, and penalties are a way of enforcing compliance with such norms. Norms here took the form of specifications, for example, which roles agents could not take on or how many blocks they could carry at most. Since we assumed, according to the provided settings, what the penalties for violating them were, and took into account what agents would have to perform to not violate these norms, for example, to reach a role zone and change their temporarily forbidden roles, we decided to ignore the norms and accept the penalties. Without analysing this in detail, this seems to us to be a rational solution and our system was still able to compete in the competition with its rivals.

From what we have now outlined, some of the activities and responsibilities that the various system roles in our solution are expected to perform. We will again make a description of each of these system roles in our solution.

3.2 deSouches in MAPC

If we are to specify the control agent's/deSouches' responsibilities, activities, protocols and permissions, let us start with the primary responsibility of creating a multi-agent population. Our system uses JADE but only to a limited extent. We do not make use of the more complex behaviors of these agents, the means of communication between agents, or the other functionality that JADE [1] provides. Basically, we have used this system to create agents as threads that can execute a method in cycles (i.e., we use the cyclic behavior of the JADE agent). deSouches itself is a JADE agent, and so are all situated agents created as the first activity of deSouches. Next, it connects these agents to the competition server, and from then the system runs with these agents until termination (i.e. when the agents receive information from the server that the game is over). The primary goals of situated agents follow the described behavior of our system. Agents must be able to cooperate with other agents of their team, and to do so they must synchronise with others. However, the very first goal that the situated agents receive is different, for purely practical reasons. During the previous years, the whole multi-agent system had to be restarted if there were some undetected errors or when the communication with the organisers' server collapsed. We solved the problem of possible connected blocks to agents by first instructing all agents to try to get rid of any connected blocks. Then, even after a restart, they can proceed as if the game has just started and the agents have no connected blocks. After the agents have performed actions in accordance with the initial 'DetachAll' goal, they will use their permissions against deSouches and ask it to assign a goal or mission. At this point, deSouches assigns the agents a 'DivideAndExplore' mission, which we will present in more detail later.

In the following paragraphs, we describe deSouches responsibilities. We will start with those related to setting up coordinated agent groups and managing their maps.

Assigns Roles to Situated Agents. The first of the responsibilities deSouches has to its situated agents is making decisions about what roles these agents should take on. Now, if we talk about roles, we will refer to the roles in the competition scenario. In our system, when situated agents find themselves in a role zone, they try to take the opportunity to change their role and ask deSouches what role would be appropriate for them. This decision is straightforward in our current implementation; deSouches responds with either the worker or digger roles in some ratio that we set to 5:1.

Establish and Coordinate Groups of Agents. The idea for coordinating individual agents is to group them together in a way that they can unify their coordinate systems. Initially, there are groups of agents corresponding to the number of situated agents, and each group contains just one agent. Thus, for some population of agents A, we have a set of groups $G = \{g_1, g_2...g_n\} = \{\{a_1\}, \{a_2\}...\{a_n\}\}, |A| = n$ and each agent is in one group. deSouches works with strategic information, including what distance vectors individual agents see other agents in their team. We can synchronise the groups g_i and g_j of agents if there are agents from these groups that can unify the coordinates and thus merge the two groups into one. We can write that $\exists a_i, a_j, a_i \in g_i \wedge a_j \in g_j \wedge g_i \neq g_j : d(a_i, a_j) + d(a_j, a_i) = (0, 0)$ where $d(a_i, a_j) = (x, y)$ is the distance on the axes x and y and a_i, a_j are the only pair of agents in A for which this holds. The $d(a_j, a_i)$ represents a vector that gives the distance between these agents in the horizontal and vertical directions. New group of agents is created, the two groups' maps are merged into one, and the coordinates of the elements on the two maps are aligned. For example, if two agents at distance $d(a_i, a_j) = (3, 2)$ from the perspective of agent a_i who sees another agent of his team three positions away from it in the horizontal direction and two positions away in the vertical direction, for example east and south of each other, then $d(a_j, a_i) = (-3, -2)$ because the second agent sees the first agent west and north at the same absolute distance values.

We could synchronise multiple groups in a single cycle if the pairwise distances were different, i.e., one pair of agents would see each other at some distance and these distances would be different for one and the other pair, but in our solution we synchronise groups only when there is only one such pair. Since we have done this in the same way in the previous two editions of the contest, we refer to our first publications on MAPC for a closer look at this algorithm [12].

Maintain Group Maps. Agents act based on the map, just as deSouches makes decisions based on the map. In the case of agents, they work with the map of their group, which we understand as strategic information provided by deSouches. It is in fact responsible for keeping information about which group the agent belongs to and what information this group shares. In each cycle, agents transmit information about what they see, and based on this information, their group maps are updated. In addition to the actual objects recorded on the map, the map provides methods for finding nearby objects of a given type (objects, friendly or hostile agents, dispensers, obstacles...). The map also indicates each position when the agent last saw it. This serves as a pheromone and can be used by agents to better search the area. Furthermore, the map can provide an agent with information about where an object is located that can or cannot be traversed or destroyed, what objects appear to be attached to an agent, and so on. Thus, a group map is some object shared by a group with appropriate methods that allow agents to make better reasoning. When merging groups, deSouches also merges the maps of both groups into one, which is then the group map for the newly created group.

Detect Master Group. Another of deSouches' responsibilities in completing Agent Assembly scenarios was determining the Master map. This is a necessary prerequisite for agents to start completing tasks that will earn them points. During the process of exploring the environment and bringing groups of agents together, situations may arise where deSouches learns that a group is capable of performing tasks. There are a sufficient number of agents who have worker roles, i.e. they can manipulate the blocks that make up the desired patterns, there are goal zones on the map and dispensers are known. In our case, we set the required number of agents in the worker role to three, we required at least one goal zone, as well as one role zone and dispensers of at least two types. If any such group map satisfies this, it becomes a Master map, the corresponding group becomes a Master group. From now on, every other group that can join this group will join it, and every group newly formed in this way becomes the Master group.

Detect Dimension of Master Map. Another deSouches' responsibility is estimating the size of the environment. Since it is cyclic, agents cannot detect its edge and it may appear infinite to them. Our approach how to decide the width and height of the grid is based on the fact that the immobile part on the map is the dispensers. For the Master map, deSouches uses dispensers to estimate the width and height of the environment when in a certain number of cases (in our case set to three) the dispensers are in the same horizontal position but at a certain vertical distance that is the same for all the three cases. Thus, deSouches then assumes that these are the same dispensers that the agent encountered when it crossed the edge. The height of the area is estimated in a corresponding manner.

3.3 Task Fulfilment by Agent Groups

The behavior of the system, or rather the deSouches policy, changes when a Master group is established. From this point on, it is already clear which group will be active and perform tasks. deSouches is now trying to find a coalition of agents to work on the tasks. The information about possible tasks deSouches receives from the system via situated agents, who receive this information from the competition system and then report it towards deSouches as a strategic option. It then searches each cycle, if any tasks are active, for a one - to four-agent coalition of free agents in that group that would be able to perform any of the tasks now (Fig. 4). By free agents we mean those agents in the group in the role of worker who are not currently ordered to participate in the assembly of a task. Related to this is the change in deSouches' assignment of goals to situated agents. If an agent in the Master group has nothing to do and asks deSouches for a goal assignment, then as long as it is still in the default role, it will have to go to the zone role, which must be at least one known role on the Master map, and accept some new role. Again, deSouches will decide what that role will be when it is needed. If the agent applying for the job is in the worker role, it will be ordered to roam and explore the environment. If the agent is in the digger role, it will be tasked with removing obstacles and attacking agents of other teams.

It is worth mentioning one more activity that deSouches performs during strategic decision-making. In each cycle, for all active tasks, it checks whether there is a strategic exception that will prevent the achievement of the task. And this could occur for two reasons. The first reason occurs when it is clear that the agents cannot accomplish the task in the time allotted because each task has a specified step up to which it is valid. The second reason is that the goal zone is no longer valid. This year, after a task is submitted, the goal zone can be

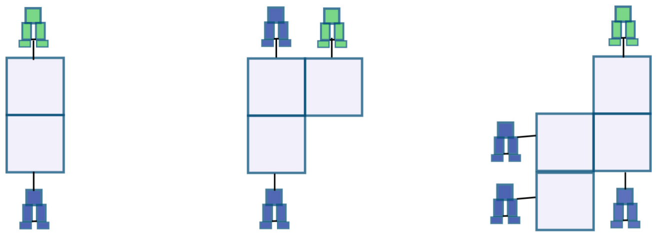
Fig. 4. An example of possible patterns made from two, three and four blocks. The green agent is the one tasked with committing the result. (Color figure online)

moved with some probability. Therefore, it is also checked each time whether the intended goal zone for the task submission still exists. If one of these exceptions occurs, deSouches will cancel the mission and release the allocated agents.

We now present the algorithm by which deSouches finds a suitable coalition of agents for the tasks. The inputs to this algorithm, in addition to a description of the desired pattern, are the free agents from the Master group, a list of dispensers that are listed on the Master map, and a list of goal zones. To avoid collisions of custom agents in goal zones, one goal zone is always considered for the currently solved tasks. In other words, the goal zones input to the algorithm are those to which no coalition is currently directed. Since goal zones tend to be a contiguous region that covers multiple positions in the grid, only one position is chosen among them. This is chosen according to the current state of the Master map so that the area around it is as free of obstacles as possible. With the inputs, we can search for a suitable coalition using the algorithm we present in the next section.

3.4 Building Coalitions to Accomplish Tasks

The essence of our solution is the creation of coalitions for given tasks. Since finding the optimal coalition structure, in general, is an NP-complete problem [10], and the number of agents in the system could be several tens, it was necessary to consider the time complexity of the assumed algorithm for forming such coalitions. The specificity of the problem that the coalitions are intended to solve gives us the opportunity to design a solution that has reasonable time requirements for an agent population on the order of tens of agents.

A coalition game is defined by a coalition structure that maximizes its value such that CS is a partitioning over A then $V(CS) = \sum v(C_i)$ [8], where $v(c_i)$ are values of individual coalitions in CS. The fundamental question is how to evaluate the coalitions. The number of points that agents in such a coalition can achieve is offered. The value of such coalition structure would be the sum of the points that each coalition can achieve. Other aspects that we could include in the value function would be the probability of achieving these points and the time in which these points are achieved. However, in working on the system for MAPC 2022, we have established only a simple principle such that the value depends on the number of steps its slowest member must take to complete its subtask. Thus, we will try to complete the task as quickly as possible, and the output of the coalition formation algorithm will be the coalition that promises to complete one of the given tasks the fastest.

We formalise the task of forming one coalition by having a set of agents in the form of their position on the map $A = \{a_1, a_2, ..., a_n\}$, a set of dispensers $D = \{(bt_1, \{dp_{1_1}, dp_{1_2})...\}, ...(bt_n, \{dp_{n_1}, ...\})\}$, where bt_i is the block type, which can be obtained from the dispensers at positions $dp_{i_1}, , dp_{i_2}...$, a set of subtasks $T = \{t_1, t_2, ..., t_o\}$, where t_i is the block type for the subtask, and a set of goal zones' positions $G = \{g_1, g_2, ..., g_p\}$. Estimation of the shortest number of steps that agent a_1 needs to achieve a subtask from T is $c(a_1) = \arg\min_{d_{pi}, g_j}(ds(a_1, dp_i) + ds(dp_i, g_j))$ where $ds(a, b)$ is Manhattan distance between positions a and b. If we specify block type and goal zone position, then arguments are also a set of appripriate dispensers for the type D and a goal zone position g we write $c(a_1, D, g) = \arg\min_{d_{pi} \in D}(ds(a_1, dp_i) + ds(dp_i, g))$. However, each agent must cater for different block from the task T. The choice of dispensers must be such that all the block types T are processed by the coalition. Therefore, we examine coalitions $A_i = \{a_{i1}...a_{io}\} \in A$ of size $|T|$ such that each agent has to bring one of the desired blocks. The cost of such a coalition which will submit the resulting pattern in goal zone g is then $v(A_i) = 1/max(c(a_{i1}, D_1, g)...c(a_{io}, D_o, g))$ where D_i to D_o are dispensers for requested block types t_1 to t_o and a_{i1} to a_{io} are agents that are obligated to bring these types of blocks.

Our coalition selection algorithm searches for the optimal coalition for each of the currently given tasks and selects the best coalition among them that is optimal for all given tasks. The algorithm for finding the optimal coalition for a single task T is shown below as Algorithm 2.

Algorithm 2: Coalition Algorithm for Agents Assemble

```
Phase 1;
foreach agent $a \in A$ do
    foreach goal $g \in G$ do
        foreach subtask $t \in T$ do
            d = bestDispenser(a,g,t);
            price = distance(a,d,g);
            taskCandidate.insert(t,a,d,g, price) ;          /* insertSort */
Phase 2;
CC = {} ;
repeat
    tc = taskCandidate.bestTaskCandidate();
    taskCandidate.remove(tc);
    foreach $C \in CC$ do
        if tc is for agent not in C and the goal positions for tc and C are the
          same then
            CC.insert(C ∪{tc});
    CC.insert({tc});
    /* if any C from CC can complete T, return C                  */
    foreach $C \in CC$ do
        if C can complete T then
            return C;
until taskCandidate is empty;
```

In the first part of the algorithm, for each given subtask and each goal area, each agent is given a dispenser over which it could execute the subtask from T. This dispenser does not have to be the closest to the agent, but it must be on the shortest agent-dispenser-goal zone path among all dispensers issuing a block of the desired type. Then we have the best possible candidate solutions for each agent, subtask, and goal zone. We then compose coalitions from these candidate solutions. Each time we select a candidate solution that has not yet been used and whose path length is the shortest among all other unused candidate solutions. With it, we first form a single-element coalition with this candidate solution, and second, we extend each already formed coalition with this candidate solution if the goal zones match. Each coalition will thus consist of candidate solutions that target the same goal zone. The algorithm terminates if any coalition is able to process all subtasks of the given task.

The number of iterations in phase 1 is obviously $|A| \times |G| \times |T| \times |D|$. It can be assumed that there are four types in T at most, i.e., that the desired pattern consists of at most four parts. This number is derived from the fact that more complex patterns have not been offered in the competition so far, and our system is not even able to compose patterns with more than four blocks. We set the maximum number of dispensers for each of the subtasks from experience to five, the same way we estimate the maximum number of goal positions. Thus, we estimated that finding the optimal coalition would require at most $100 \times |A|$ iterations, where $|A|$ is the number of agents (in the order of tens). Then several thousand iterations our system is able to perform in each cycle, since in the competition the time between each step was typically on the order of seconds. This was also confirmed after the implementation of this algorithm, which was not a computational burden on the operation of our system.

In total, there will be $|A| \times |G| \times |T|$ candidate assignments for individual agents to bring a block of the given type to the goal area (hereafter we will call this candidate solution) and based on them, we are trying to find a possible coalitions. We always add the next best candidate solution not yet added to the possible coalitions. If we had a set of k of candidate solutions, then the number of possible coalitions formed from these solutions would be 2^k. However, we only form coalitions where there are unique agents and the same goal position. Thus, if we expand the set of coalitions by adding a solution with agent a_i that points to the goal zone g_i, then firstly a single candidate coalition with this solution will be formed, and secondly, coalitions will be formed by adding this solution to such existing coalitions in which the agent a_i is not included, and the individual candidate solutions in the coalition point to the same goal zone g_i. In addition, we only extend coalitions that have less than $|T|$ members, because, by the nature of the problem solution, the coalition being sought will have exactly $|T|$ members. The number of coalitions for a four-block task that we can traverse is then given by the number of possible four-block coalitions from the agent population multiplied by a permutation of four, which covers all possible subtask assignments to these agents, because the cost of such a coalition also depends on which specific subtask from T is assigned to which agent.

This number is then equal to $4! * \binom{|A|}{4} = \frac{|A|!}{(|A|-4)!}$. While the number of possible coalitions to explore is still high and can be estimated as the fourth power of the number of agents, i.e., over half a million iterations for 25 agents, we have not approached such a number in practical use for competition. By incrementally building coalitions by adding the next cheapest candidate solution the first suitable coalition, where each subtask from T is processed, is also optimal. Therefore, the maximum number of coalitions we could find before we discovered the optimal coalitions was $\sum_{k=1}^{3} k! * \binom{|A|}{k}$ which for $|A| = 30$ is 24360 three-agent coalitions respecting the assigned tasks, 870 such two-agent coalitions, and 30 one-agent coalitions. Thus, our solution was found at the latest after the establishment of 25270 coalitions, which for today's serial computers is manageable in the time our system had in one step of the competition.

We give an example for two agents, a task demanding two blocks of different types and three goal positions. For both agents at a_1 and a_2, for both block types t_1 and t_2, and for all three possible goal positions, dispensers were found for which the path from the agent's current position through the dispenser location and the goal position is the shortest. Assume that for both types of tasks, there are, respectively, a group of agents knowing the location of dispensers D_1^1, D_2^1 and D_3^1 for the first type and D_1^2, D_2^2 and D_3^2 for the second type. After computing the optimal dispensers, we get a total of twelve candidate solutions as shown in Fig. 5. The number of steps required for every agent to move from its current position through the assumed dispenser to the goal zone is given below each illustration of these solutions. In the figure, solutions are numbered in a circle above the triangle.

After sorting the candidate solutions from cheapest to most expensive, we can start forming coalitions. In the first step, we add only one solution to form one coalition $\{\{8\}\}$. After adding the coalition marked as 11, we have the possible coalitions $\{\{11\}, \{8\}\}$. We could not extend the $\{8\}$ coalition with candidate solution 8 with candidate solution 11, since it is directed to the same goal zone, bud it is made by the same agent as is in 8. We cannot extend any of the existing coalitions with candidate solutions 3 and 1, so after adding them to the game we have possible coalitions $\{\{1\}, \{3\}, \{11\}, \{8\}\}$. We can combine the candidate solution 10 with the first one and we get $\{\{10\}, \{1\}, \{3\}, \{11\}, \{8\}, \{1, 10\}\}$ and coalitions with members $\{1, 10\}$ covers both types of blocks with different agents heading to the same goal position. And this is the take solution that is optimal according to our criterion and estimates the transport of both desired blocks in 35 steps.

At this point, we would like to have a short discussion about our solution. The evaluation of the coalitions is crucial for their correct determination and in our case, only an initial and rather naive approach was used. We do not compute the entire coalition structure but search for the optimal free agent coalition in a given cycle. However, this does not guarantee an optimal coalition structure of agents, because there may not be an optimal coalition as recommended by our algorithm, in the optimal coalition structure of agents at the specific time.

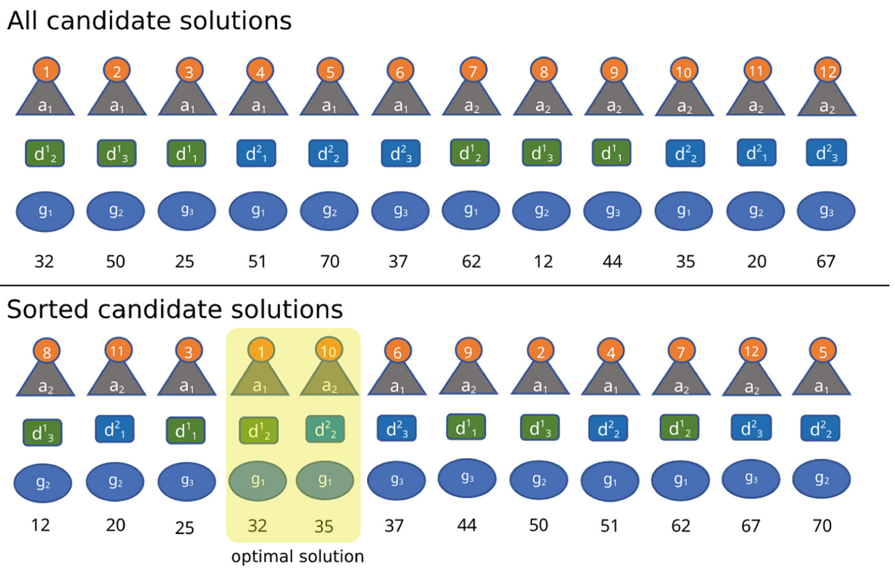

Fig. 5. Coalition Search Example

Although this statement is obvious, as an example we can consider some problems where the grand coalition value is say $v(A) = 8$ and the coalitions B, C, which are disjunctive decompositions of A, have values $v(B) = 5$ and $v(C) = 4$. Then the optimal coalition structure is formed by coalitions B, C and not by coalition A, which however our system would recommend. We also do not take into account the dynamics of the environment. By examining it, agents may find that the coalition is not optimal because the system has changed. For example, they may discover a new dispenser or even a separate deferred block, or agents may add other agents to the agent group if this (master) group is merged with another group. The above shortcomings are thus a topic for further improvement of the system for MAPC tasks. We anticipate that the coalition solution may be suitable for other multi-agent tasks if the Agent Assemble task is not repeated in the next edition. If it does, we will try to ensure that at each point in time, our system knows the optimal coalition structure of our agents to improve its performance. We will now return to the deSouches system and present our solutions for MAPC at other levels of the system.

3.5 Coordinator Agents and Missions in the System

Most of the goals specified by deSouches need some missions. We have mentioned three missions in the text above. First, a mission for exploring the environment and coordinating groups, then a mission for clearing the environment of obstacles, and we have devoted the main part of this section to creating missions for task completion. Introducing the different missions and their execution at the second and third levels of the deSouches system, we start with those that are performed by a single agent in order to explore and clear the environment in which the agents are situated.

Divide and Explore Mission. The coordinator is working with one agent and all it does is repeat the 'Explore' goal assignment, whether the previous effort to achieve this goal succeeded or failed. An agent fulfils this goal when it performs a walk within some distance. This is specified when the mission is created, and in our system, we set it around a value of ten. The direction is calculated using a pheromone algorithm, which improved the system performance and made finding the Master group faster.

Compared to random walks where we found the Master group for a 70×70 environment with twenty agents between steps 100 and 150, using pheromone we established the Master group around step 50. This mission can be terminated only by deSouches by using the agent for another mission.

Fighter Mission. This mission is similar to the previous one in that one and the same goal is given whether its previous follow-up was successful or unsuccessful. This mission is executed by all agents who adopted the 'digger' role at the time they adopted the role and, unlike the previous mission, it is executed throughout the entire runtime of the system. The goal given to the agents is 'Go And Shoot'. The agent walks randomly and if an obstacle or foreign agent appears within its firing range, it will fire at it. To avoid being blocked from firing at a foreign agent, who may already be knocked out for a few steps due to the fire and thus there is no point in attacking it further, the agent keeps a record of who it has ever fired at. Based on these records, he deduces whether to attack the enemy or pursue other targets.

Block Missions. Unlike previous missions, block missions for two or more blocks are performed in coalitions. We have described how deSouches forms a coalition in the Subsect. 3.4 above, and we now discuss the individual goals that agents must achieve during the mission and their possible coordination.

Whether the product is made up of one, two, three or four blocks, it is always the job of each agent solving this block mission to find a block of the desired type, transport it to a goal position, and connect its block to the other blocks. If the mission is for a task with more than one block, agents end their activity after linking their block to another block or blocks by disconnecting from their block, except for one agent who remains attached to its block until the entire pattern is composed and can be submitted. In Fig. 6 we have demonstrated the behavior of the coordinator for a two-block mission. One agent has to execute the sequence of goals 'Detach All', 'Get Block From Dispenser', 'Go To Position', 'Connect' and 'Submit'. The names of these objectives are, we hope, self-explanatory. The first agent first gets rid of everything attached to it, if there is such a thing, just in case. It then walks to the dispenser that was found while searching for the coalition, picks up the block, walks with it to the given location at the goal zone, connects its block to the other agent's block, and submits the block. The other agent does the same, but only until it connects his block to the other block and disconnects from it (part of the goal 'Connect' is to disconnect from his block).

You can see two things on the automata. First, the transition between the 'Go To Position' and 'Connect' goals is barrier synchronised. The 'Connect' goal can only be issued to both agents at once. If one agent has completed its 'Go to Position' objective and the other has not yet, the first agent must wait. You can also see the coordinator's behavior if either goal fails, or an event occurs where the agent is subjected to an attack (clear event) that causes it to lose any attached block. If it fails to travel to a goal zone, it performs a random walk and tries again to reach that goal. In the event of a clear event, the agent must backtrack to reach the dispenser for a new block. In the diagram, it is the connection above the depicted states.

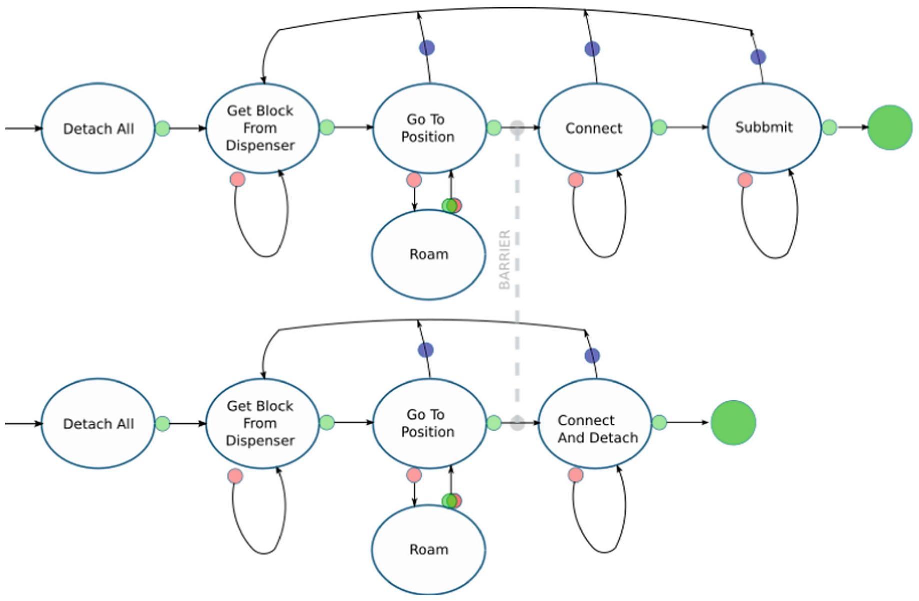

Fig. 6. Two Blocks Mission Control Automata

3.6 Situated Agents Level

The goals that are given to the agents in this realisation follow from the missions mentioned above, and there are also two atomic goals that are given directly by deSouchces. We begin our description of the situated agents behavior with one of them. This is the 'Force Detach' goal, which we mentioned in the introduction of Sect. 3.2, and which is given to all situated agents immediately after their creation. Its purpose is that, if our system needs to restart during a match, the agents start with no blocks attached and can proceed as if the match had just started. To accomplish this, plans for disconnecting blocks are created sequentially in all directions, and after the last plan is executed, the goal is considered accomplished, and the agent then asks deSouches for more work.

Since it is almost certain that a Master group is not established after four actions, then the agent is assigned to the 'Divide and Explore Mission' if it is not in the 'digger' role, or the 'Fighter Mission' if it is in that role. The second case can occur when the system is forced to restart during a match and the agent realises that it is a digger. In the case of the 'Divide and Explore Mission', the agent is repeatedly given the 'Explore' goal. This goal is not just about exploring the environment by walking around, but the agent can perform other activities during these walks. During the re-evaluation of plans, it can build a plan to change roles if it is standing in the goal zone and its role is still the default. It then asks deSouches to tell it which role to adopt and builds a higher priority plan with one action, switching to that role. This plan is not final and executing it does not achieve the 'Explore' goal, so the agent continues to follow that goal. With a lower priority, the agent builds a plan for its movement to some chosen location. The agent uses pheromones on our group's map to select this location, and the agent prefers to explore those parts where no pheromone is listed or the pheromone is old. It takes a relatively aggressive approach to movement and tries to get directly to the chosen location, although it often has to remove obstacles along the way.

As mentioned, if an agent is in one of other than default roles, they may be assigned some different missions. In the case of agents in the 'digger' role, this is the 'Fighter Mission'. Within this, the goal is to 'Go And Shoot' the essence of which is to go to a designated position and in the process remove obstacles and attack other teams' agents. As the agent in the 'digger' role has a larger range of the clear action, which is the removal of obstacles and attacks on agents, part of the planning is to find a suitable target within a range of this action.

The most complex missions involving multiple objectives are the 'Block Missions' described above. During these, agents, after disconnecting all previously connected blocks for safety, try to pick up the block whose type they are in charge of. Multiple plans of different priorities can also be created to fulfil the 'Get Block From Dispenser' objective. If an agent, while re-evaluating a plan, finds that it has a block of the desired type on either side, then it attaches it and the goal is achieved. However, this is only if there are no other agents of its team standing around the block. Then they respect the right hand rule with priorities north, east, south and west. An agent only attaches a block when there are no other agents of its team standing next to the block in the higher priority direction. The agent waits otherwise. If it can join the block, it creates a plan to join the block and by successfully executing the plan, the 'Get Block From Dispenser' goal is met. The agent proceeds similarly if it is standing next to a dispenser and intends to pick up a block from it, but then the plan includes both the request from a dispenser action and the subsequent attach action. A third possible situation is that the agent stands directly on the dispenser and then tries to take one step outside of it in some direction. If neither of the above applies, the agent searches for the nearest dispenser that dispenses a block of the desired type and sets off towards it. Here we use the 'Approach Type' subplan declaration. This plan is constructed by method A* as the optimal sequence of steps to reach the position where the object of the desired type, in this case, the dispenser, is located, and this plan is executed unless a higher priority plan is constructed. The most flexible dispenser should be the one that was considered when the coalition was built. In our implementation, however, we always had the agent search for the nearest suitable dispenser in case a new and more suitable dispenser was discovered.

The mission continues by entering the goal 'Go To Position'. This goal is similar to the 'Approach Type' goal, but the position is given here precisely because it was set when the team for this mission was assembled. Again, the agent plans the path using the A* method. Now we want to point out that the path found takes into account the block attached to the agent and also the desired rotation of the block at the goal position. However, this can be tricky in situations where there are obstacles and other agents at the goal position and the execution of the path may not be and often has not been, successful. Therefore, we have incorporated a 'Roam' goal into the mission, which is given to the agent when such a failure occurs. As part of its execution, the agent attempts to take a few steps aside and then reach the position again for the re-assigned goal 'Go To Position'. The remaining targets are for connecting the individual blocks and then submitting them by the selected agent.

Finally, we mention one more goal that deSouches specifies as atomic. If a master group is established and some agents have not managed to change their role from 'default' to 'worker' or 'digger', deSouches instructs such agents to reach the role zone and change the role. For this 'Change Role' goal, they again use the 'Approach Type' subgoal, this time with the desired type at the goal location corresponding to the role zone, and the agent, once it reaches this location, asks deSouches to tell it what role to adopt, and then adopts it.

4 Evaluation of Participation in the Tournament

In total, our system played twelve games against four teams, so each team played each other in three games. As could be observed, the settings of each match varied in terms of duration, i.e., number of steps, map size, and number of agents per team. Overall, we placed third out of five teams with a score of 19 points. We won six games out of twelve, drew one, and lost five. In one case we recorded zero points, the most points we recorded in our last appearance was 1640 points, which was the most points anyone recorded in this tournament. Although some task parameters were already estimable after the warm-up matches with Paula Böhm from the school organising the competition, namely the number of agents and the size of the area in each match, we did not set these parameters and let the agents act without such estimates, even though the team would have gained some advantage. We observed the following facts from the matches. As expected, we didn't have many chances in the first matches, which lasted only 400 steps. Since our solution required some initial familiarisation with the system, the agents had only a very short time to work on the tasks. This then meant that we only won once and lost three times against four opponents. As the number of steps increased, our handicap with the initial warm-up decreased. In the other eight matches, which were played at six hundred and eight hundred steps, our team did much better, losing only once and by only thirty points, to GOALdigger, who finished second in the end. We played more close results, even drawing with the same team, beating the GOAL-DTU team in one of our matches by only forty points and beating the later overall winners MMD by eighty points.

It could be observed that not only our team but also the MMD team recorded significantly higher scores in cases when the opponent was not doing well. Although we tried to coordinate our agents so that each group submitted scores at different goal zones, having multiple teams working at the same locations introduced collisions and neither team was as efficient as in situations where they had this space to themselves. It is the improvement of precommit manoeuvring that is the challenge for us to improve the performance of our solution.

5 Conclusion

Even though our system has finished worst in the overall competition so far compared to the systems we have used in previous competitions, we do not see our new system as a step in the wrong direction. While the original system was able to perform the tasks well, our system did not perform downright worse. In addition, we had to deal with new aspects that were added to the assignment this year. The norms that our system did not consider would have been ignored anyway by the system we used in previous years, but without the coordination of the individual agents in the mission, the rather reactive approach would have had problems and would not have been able to perform the tasks without major modifications. We are also aware of a number of shortcomings and opportunities for improvement in this year's deSouches system, ranging from improvements to the coalition planning algorithm to better end-of-task execution. We found it beneficial to use our system in situations where it was advantageous to be reactive to emerging stimuli as well as in line with the goals currently being pursued. For a more general use of our system, it would be necessary to specify the individual functionalities of the system more precisely, to create an environment and support for programming, including a suitable graphical environment, and a set of libraries on which the solution would be based. As we anticipate continuing our participation in this competition, we will continue to work on the development of the deSouches system.

Acknowledgement. This work was supported by the project IT4IXS: IT4Innovations Excellence in Science project (LQ1602).

A Team Overview: Short Answers

A.1 Participants and Their Background

Who is part of your team? This year, members of the teams were František Zbořil, František Vídeňsky and Ladislav Dokoupil

What was your motivation to participate in the contest? We have participated in the previous two years and we intended to continue this year, even though our main man from previous years is no longer on the team. We wanted to try some new strategies for the current scenario and have some fun competing.

What is the history of your group? (course project, thesis, ...) The course on agent and multi-agent systems has been taught at our school for more than ten years, during which a number of diploma and bachelor theses have been written. This year our team consisted of a professor, one PhD student whose thesis topic are BDI systems, and one Master's student who focused on environment search in his Bachelor's thesis.

What is your field of research? Which work therein is related? In particular, BDI agents, interpretation of BDI languages and modeling of distributed systems.

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 system used in the competition has been developed for the 14th edition in 2019. This system has been modified in accordance with the current competition rules.

How much time did you invest in the contest (for programming, organising your group, other)? Approximately six man-weeks

How was the time (roughly) distributed over the months before the contest? Half of that time was in the month before qualifying and the other half was between qualifying and competition.

How many lines of code did you produce for your final agent team? Our implemented system has approximately 12,000 lines, most of which have been created previously for earlier years and about one-third for this year.

A.3 Technology and Techniques

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

Agent programming languages and/or frameworks? We specified Java as the programming language in the competition entry form, and did not specify any other system. However, we use the JADE library in our solution but only for managing agent threads and our system is a custom work inspired by BDI systems but also multi-agent approaches like JaCaMo.

Methodologies (e.g. Prometheus)? No

Notation (e.g. Agent UML)? No

Coordination mechanisms (e.g. protocols, games, ...)? Yes, agent coordination consisted of forming coalitions for assigned tasks. In each cycle, a coalition was sought for free agents that were not currently working on one of the tasks. This was a crucial conceptual idea for our solution this year.

Other (methods/concepts/tools)? Nothing significant to note here.

What hardware did you use during the contest? We ran the system on regular PCs and laptops. During the competition we used Acer Predator Helios 300 laptop

A.4 Agent System Details

Would you say your system is decentralised? Why? It is not decentralised completely. deSouches assigns missions to other agents, which the agents in the group then execute. Based on the missions, goals are assigned to individual agents, which they execute autonomously until they meet them or if their efforts fail. Thus, the missions again have a central element that works with the result of the agents' efforts. It also synchronises the agents, e.g., assigns additional goals and informs the controlling agent about the completion of the mission.

Do your agents use the following features: Planning, Learning, Organisations, Norms? If so, please elaborate briefly. Planning takes place in each cycle for the assigned goals. The agent may re-evaluate its plan, or if it is unable to make any plan for the goals, it informs the mission coordinator. Among the classical algorithms, we use A* for path finding in the environment. To organise agents, as we have already mentioned, a central agent is used to explore opportunities and create coalitions of agents for them, if possible.

How do your agents cooperate Agents work together at the mission level by informing the mission coordinator of their success or failure in achieving their assigned goals and are assigned additional goals by the coordinator on that basis.

Can your agents change their general behaviour during run time? If so, what triggers the changes? Yes, as we have already stated, the central agent analyses the current situation and assigns missions to agents accordingly. Initially, the agents synchronise and explore the environment, later they focus on completing the tasks if the right teams can be formed for them.

Did you have to make changes to the team (e.g. fix critical bugs) during the contest? We made some changes after the warm-up matches. We did not make any more changes during the competition.

How did you go about debugging your system? What kinds of measures could improve your debugging experience? We debugged mainly by doing a lot of system runs and observing the behavior of the agents, including what tasks they followed and how successful they were. At this point, we can't think of any major suggestions for improving the debugging options.

During the contest, you were not allowed to watch the matches. How did you track what was going on? Was it helpful? Based on the agents, maps are created for the groups they are in. We can display these maps on our side. We also have an overview of which tasks which agents are performing and how they are performing them. It was only useful during the competition in that this satisfied our curiosity.

Did you invest time in making your agents more robust/fault-tolerant? How? We tuned the robustness based on our experience with our system and experimenting with different settings. This was done incrementally and there is still room for improvement.

A.5 Scenario and Strategy

How would you describe your intended agent behaviour? Did the actual behaviour deviate from that? The basic idea was to first create as large a group of agents as possible that share a map and are synchronised in terms of knowledge of their positions relative to the other agents in the group. Then, if a group was large enough and knew about dispensers and goal zones on its shared map, it was labelled as a mastergroup. Coalitions were then sought for agents in this group that were capable of completing the current task assignments. These groups were sought as optimal given the expected distance they must travel to pick up a block and transport it to a predetermined goal zone. Our agents have mostly behaved according to these principles.

Why did your team perform as it did? Why did the other teams perform better/worse than you did? It seemed that teams that didn't rely on just one specific task might have had an advantage. This is the approach we have taken in previous years and this year we tried to be more specific in the assignments of tasks to agents. Also, the need for initial synchronisation cost quite a bit of time/steps and that was our disadvantage. For the next competition we intend to combine both approaches.

Did you implement any strategy that tries to interfere with your opponents? When the agent had a role in which he could attack opponents, he did so. However, since it was clear that several such attacks were needed to gain an advantage by such an attack, this was not relevant to our system and there were only a few agents with this role in our team. Moreover, they were primarily intended to clear the environment of obstacles.

How do your agents coordinate assembling and delivering a structure for a task? For each possible structure up to four blocks, we have predefined positions of the agents that deliver the blocks. If all the agents reach them, the blocks are connected and the product is submitted by one of them.

Which aspect(s) of the scenario did you find particularly challenging? The scenario as a whole was a challenge. We can't single out one aspect. For us the biggest problem was probably getting to the right places in the goal zones. There was often a lot of traffic and agents were in each other's way. Therefore, the ability of the agents to solve this problem fundamentally affected the performance of our system.

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 a number of ideas on how to improve the functioning of our system. For example, re-evaluating coalitions already formed or better maneuvering before submitting a task.

What can be improved regarding the scenario for next year? What would you remove? What would you add? The scenarios get more interesting every year. This year we have not been too concerned with norms because we felt that the penalties for violating them were low and behavioural changes in line with norms would be irrational. This may change by next year, as may the wider range of roles. Agents needed to be in one of the roles available to complete tasks, which could change for next year, making the competition more interesting but more difficult to solve.

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

What did you learn from participating in the contest? As in previous years, the creation of a multi-agent system for non-trivial tasks is a complex and quite challenging activity.

What advice would you give to yourself before the contest/another team wanting to participate in the next? As this was our third participation, we had already learned our lesson. Not to underestimate the functioning of the whole system when the opponent is present and to make sure that the system is able to cope with possible connection failures, which we fortunately did not encounter this year.

Where did you benefit from your chosen programming language, methodology, tools, and algorithms?
Since we were building our system from scratch, the disadvantage was that we had to invest a lot of time in creating a basic system. On the other hand, we could adapt the multi-agent architecture to the competition.

Which problems did you encounter because of your chosen technologies? Again, we mention the need to create it completely. That is why there were and probably still are untuned parts in it, which in already finished and time-tested systems work flawlessly.

Which aspect of your team cost you the most time? Certainly debugging the system and verifying its functionality.

A.7 Looking into the Future

Did the warm-up match help improve your team of agents? How useful do you think it is? The warm up match was crucial to our team's performance in the competition. It helped us uncover some fundamental flaws in our system setup. Without it, our results would have been much worse.

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? In previous years, it happened that teams lost connection with the organisers' server, which led to inconvenience and the need to cope with it. This year we have not encountered this and therefore have no objection to this method of running matches.

Do you think a match containing more than two teams should be mandatory? We wouldn't say it's mandatory, but it would certainly make the competition more interesting. We can even imagine some form of knockout phase if the number of teams is even larger next time.

What else can be improved regarding the MAPC for next year? The organisers have our full confidence that next year's event will again be at least as interesting, motivating, beneficial and entertaining as it has been so far.

References

1. Bellifemine, F.L., Caire, G., Greenwood, D.: Developing Multi-agent Systems with JADE. Wiley, Hoboken (2007)
2. Bordini, R.H., Hübner, J.F.: BDI agent programming in AgentSpeak using *Jason*. In: Toni, F., Torroni, P. (eds.) CLIMA 2005. LNCS (LNAI), vol. 3900, pp. 143–164. Springer, Heidelberg (2006). https://doi.org/10.1007/11750734_9
3. Brooks, R.: A robust layered control system for a mobile robot. IEEE J. Robot. Autom. **2**(1), 14–23 (1986)
4. Dastani, M.: 2APL: a practical agent programming language. Auton. Agent. Multi-Agent Syst. **16**(3), 214–248 (2008). https://doi.org/10.1007/s10458-008-9036-y
5. Hindriks, K.V., de Boer, F.S., van der Hoek, W., Meyer, J.-J.C.: Agent programming with declarative goals. In: Castelfranchi, C., Lespérance, Y. (eds.) ATAL 2000. LNCS (LNAI), vol. 1986, pp. 228–243. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44631-1_16
6. Hübner, J.F., Boissier, O., Kitio, R., Ricci, A.: Instrumenting multi-agent organisations with organisational artifacts and agents. Auton. Agent. Multi-Agent Syst. **20**(3), 369–400 (2010)
7. Kaelbling, L.P.: A situated-automata approach to the design of embedded agents. ACM SIGART Bull. **2**(4), 85–88 (1991)
8. Rahwan, T., Michalak, T.P., Wooldridge, M., Jennings, N.R.: Coalition structure generation: a survey. Artif. Intell. **229**, 139–174 (2015)
9. Rao, A.S.: AgentSpeak(L): BDI agents speak out in a logical computable language. In: Van de Velde, W., Perram, J.W. (eds.) MAAMAW 1996. LNCS, vol. 1038, pp. 42–55. Springer, Heidelberg (1996). https://doi.org/10.1007/BFb0031845
10. Sandholm, T., Larson, K., Andersson, M., Shehory, O., Tohmé, F.: Coalition structure generation with worst case guarantees. Artif. Intell. **111**(1–2), 209–238 (1999)
11. Thangarajah, J., Padgham, L., Winikoff, M.: Detecting and avoiding interference between goals in intelligent agents. In: Proceedings of the 18th International Joint Conference on Artificial Intelligence, pp. 721–726. Morgan Kaufmann Publishers (2003)

12. 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
13. Uhlir, V., Zboril, F., Vidensky, F.: FIT BUT: rational agents in the multi-agent programming contest. In: Ahlbrecht, T., Dix, J., Fiekas, N., Krausburg, T. (eds.) MAPC 2021. LNCS (LNAI), vol. 12947, pp. 23–45. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-88549-6_2
14. Wooldridge, M., Jennings, N.R., Kinny, D.: The Gaia methodology for agent-oriented analysis and design. Auton. Agent. Multi-Agent Syst. **3**(3), 285–312 (2000)