Assignment 3 report - Group: YSoSirius
02285 AI & MAS

Bogdan Sorlea
s121075

Rasmus Krøyer Jørgensen
s090487

Abstract

The abstract goes here. Please read this document carefully before preparing your manuscript.

To ensure that all reports have a uniform appearance corresponding to published papers at the major AI conferences (like IJCAI and AAAI), authors must adhere to the following instructions.

1 Introduction

Introduction goes here.

2 Background

Stuff about the theories that our program is based on.

3 Related work

Research needs to be done. This section could be moved.

4 Method

This section contains a detailed explanation of the algorithm our client uses. Each step will be explained in its own section. There will less focus on the theory behind this solution as this should be clear from the previous section.

4.1 Preprocessing

The first step opon startup is the preprocessing step. This step is not part of the core loop of the client and will only be executed once. In this part we first read the input to the server to create our perception of the map. After this we perform a simple form of goal decomposition. Each goal cell is treated as a independent goal that must be achieved. Then for each goal we calculate the goal priority. The priority is based on whether or not other goal cells block this goal cell - that is if by solving the other goals you will then block access to this goal. Once each goal cell has a priority we then calculate the exact distance from each cell to each goal cell. This will help us have a more efficient heuristic when we want to know the distance from the boxes to the goal cells.

Distance map While incrementally developing the solution, we took the decision to simplify as much as possible, wherever it was possible. One such simplifications relied on pre-computing a distance map for each of the goal cells. The rationale behind it was that if we could obtain a solution that is not computationally intensive for computing the minimum distances from each cell to each goal cell, as the goal cells stay constant (a goal cannot be moved), then we would have a very precise metric to include in our heuristic function, which would ultimately make the solution more robust.

For computing this distance map we took an approach based on starting from the goal cell and, using the 4-vicinity, continuously add neighbors into a frontier list, that would be continuously processed until empty. The processing consists of always storing the minimum distance to a specific cell, which is based on the distance to its parent, incremented by one.

The usage of the frontier data structure and of the way the distance is computed can be considered to be structurally similar to how the best-first strategy algorithm work - which itself is an important part of the solution, in exploring the state space.

Specific measures were taken in order to increase the performance of the algorithm, most important one being a check based on which the iteration would just continue if a specific cell has been already processed (i.e. due to a different - and therefore possibly shorter - path from the goal to that specific cell).

The result of applying the aforementioned algorithm is a map of distances similar to the one in Figure 3, which corresponds for the level in Figure 2 and the goal denoted by *b*.

It is noteworthy that the algorithm itself does not take into account situations where a level is split into two or more disjoint areas, where one area is not reachable from another and this might lead to exceptions in some levels. Therefore, this scenario can be improved by map initialization to an arbitrary flag that indicates that the cell has not been reached in any way from the goal.

4.2 Find solution

In relation to the BDI architecture this step involes updating the Beliefs, Desires and Intention of each agent. In this step each agent will first update their desire, which in our context is to find a new subgoal (found from the goal decomposition mentioned above). If the agent does already have a goal and this is not yet achieved then no new goal is given. If the agent is not done and is not quarantined (explained later) then it is given an updated version of the map and the positions of agents and boxes. From this state the agent uses a graph search tree with a heuristic function to find a state that satisfies its goal. During this search process the agent assumes that the map stays unchanges, except for the changes made by the agent. The agent also cannot see obstacles in the form of boxes of another color or other agents, so it will plan as if they are not there (but these can still influence the heuristic function).

Heuristic function The heuristic function we use is a relaxed, admissable heuristic function. It uses the A* as its basis which means that it will add a $g()$ function to all heuristics. The $g()$ functions return the number of steps to the initial state of the search tree. The $h()$ part of the heuristic function is found in the following way. It will find the box closest to the goal cell that the agent is trying to solve and then find the distance between the two, using the goal distance map we calculated during the preprocessing. It will then find the manhatten distance between the agent and the box chosen and add that to the earlier distance found and this will be the foundation of the returned value. There are few other things that influence the heuristic:

- Moving a box that is not the box found earlier will add to the returned value
- Moving a box or the agent into a cell the is occupied by another box or agent will add to the returned value.
- Moving a box or the agent into a cell that is a solved goal cell will add greatly to the returned value.

Since the search function will chose the state whose heuristic function returns the lowest value, adding to the value returned effectively discourages the agent from chosing that state. *Possibly explain why heuristic is admisable and relaxed.*

4.3 Execute plan

Once each agent has a plan the will solve its current subgoal, the plan will be executed. The loop that executes the plan will terminate if the server responds *false* to any of the actions given - that is if the action an agent is trying to do is not possible. The client has a blackboard architecture and has a shared variable that holds the current state of the map. This variable is updated each time and agent has performed an action successfully. It is the same variable that an agent copies when it updates its beliefs in the previous stage. Once all agents have executed their plan the loop terminated, assuming it was not terminated prematurely because of a conflict.

5 Results

Discuss our results.

6 Discussion

Why did we do what we did? What was good and bad etc.

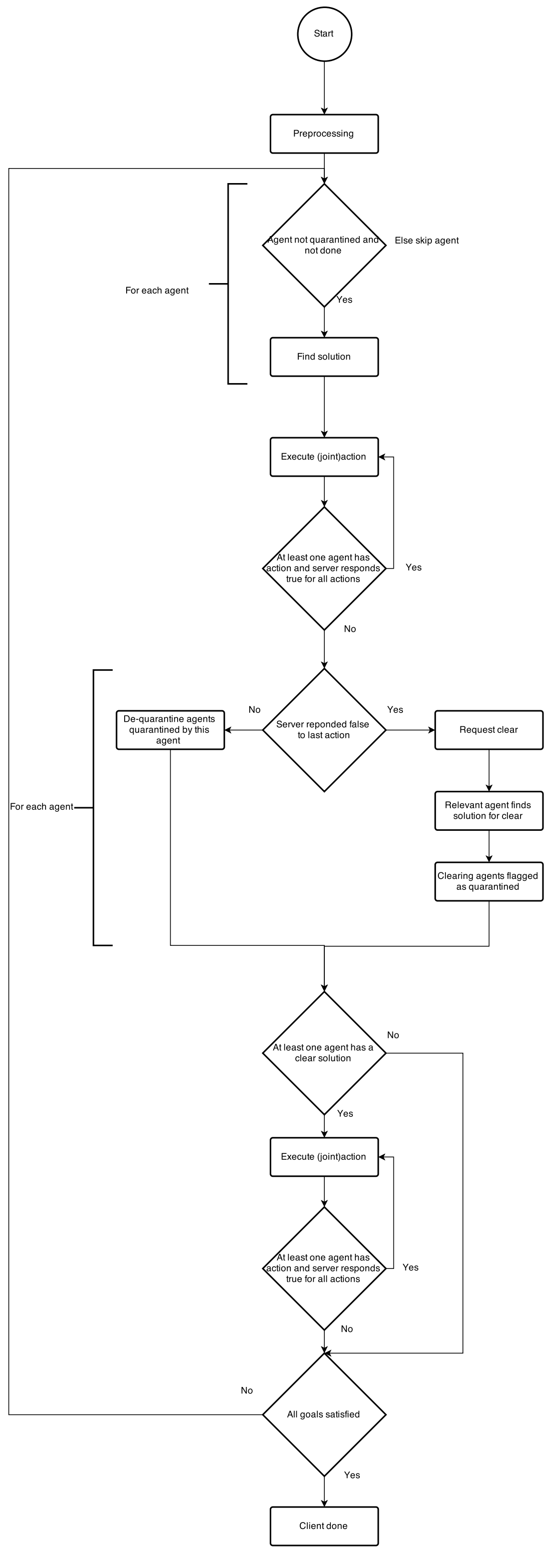
7 Future work

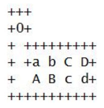
How could we most effectively improve our client.

8 Bibliography

Your bibliography should be formatted using `aaai.bst` as this document. Citations are included like so (**?**). Multiple citations appear like this (**?**; **?**). All references to be cited should be included in BibTeX format in the file `bibliography.bib`.[1]

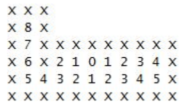
[1]Almost anything ever published can be found in BibTeX format via Google Scholar, but if using this method, you need to check the BibTeX entry for sanity before including it in the `bibliography.bib` file.


Start
Preprocessing
For each agent
Agent not quarantined and not done
Else skip agent
Yes
Find solution
Execute (joint)action
At least one agent has action and server responds true for all actions
Yes
No
For each agent
Server reponded false to last action
No
Yes
De-quarantine agents quarantined by this agent
Request clear
Relevant agent finds solution for clear
Clearing agents flagged as quarantined
At least one agent has a clear solution
No
Yes
Execute (joint)action
At least one agent has action and server responds true for all actions
Yes
No
All goals satisfied
No
Yes
Client done



```
+++
+0+
+ +++++++++
+ +a b C D+
+  A B c d+
+++++++++++
```

Figure 2: Original level



```
X X X
X 8 X
X 7 X X X X X X X X X
X 6 X 2 1 0 1 2 3 4 X
X 5 4 3 2 1 2 3 4 5 X
X X X X X X X X X X X
```

Figure 3: Distances to *b*

Figure 1: Flowchart of the base client loop