02288 Artificial Intelligence and Multiagent Systems
Group: *Deep Plan*
June 5^{th} 2020

Morten Holm Thomsen s164501
Gustav Hartz s174315
Jakob Vexø s152830
Simon Jacobsen s152655

Abstract

This paper presents a multi-agent architecture for solving transportation tasks in the simplified hospital domain. The client implements individual problem solving and communication between agents by the *cooperative distributed problem solving* framework, where problem decomposition and synthesis is key. When synthesising the sub-problem solutions, eventual conflicts are handled online, and the behaviour of agents involved in conflicts is defined by social laws, always favoring the collective solution compared to any individual agent. By performing an extensive level preprocessing, which included locating constructs such as tunnels, wells and connected components, this client was able to solve just under 50% of the submitted levels for the competition for DTU course 02285. The extensive and computationally heavy preprocessing, and an implementation that favored cooperation between agents instead of single-agent intelligence were the main reasons that the client could not solve the remaining 50%.

Introduction

This project is inspired by mobile transportation robots in a hospital. The domain have independent agents solving individual transportation-goals in a shared dynamic environment and requires an implementation of social, interactive behaviour between the agents. As a simplification of a true hospital, the environment is in this paper defined as a grid thus making the temporal movement of the agents discrete. For the agents to be able to maneuver in this domain, they need to know how to behave in certain, and sometimes unforeseen, situations if the system is to be robust. This boils down to creating agents that are able to communicate to solve collisions or resource conflicts.
Based on the CDPS fundamentals, BDI architecture and online replanning, this paper propose a client where a central entity perform the deliberation for the agents and solve their conflicts, however letting each agent control its own means-end reasoning in a relaxed perception of the world state. Agents are by nature benevolent, and prioritises helping each other over anything else. Conflict solving between agents is enforced by social laws, determined by the agents hierarchical social status given by the task they are currently performing as well as placement in critical environment.

Background

In a multi-agent environment, centralised offline planning is computationally hard and usually considered infeasible, because of the amount of different combinatorial outcomes a search with multiple agents would have. Thus we resolve to online replanning enabling the agents to divert from their initial plan as a reaction to unaccounted for conflicts.

To further decrease the computation a cooperative distributed problem solving (CDPS) approach can be taken, solving the problem through: Problem decomposition, subproblem solution and solution synthesis.
This approach requires a central planner for allocating the subgoals to the different agents who then solve the planning individually, thus reducing the time complexity of planning to almost linear with regard to the number of agents. Lastly, the solutions to the subproblems are synthesised into a shared data structure to keep track of action effects. This is very similar to what was done in the Martha Project (Alami et al. 1998).

When distributing tasks this central planner should construct a goal schedule which is one of the permutation of an ordered list of n goals. This is inspired by the work done by (Demaret, Lishout, and Gribomont 2008), where they constructed such a schedule, but in the classic Sokoban domain where it is of even higher importance. Ideally it finds the effective schedule that is the order in which the problem is solved most effectively, but often it is forced to settle with just a "good schedule" from an initial analysis of the state of the environment.

This CDPS method implies a benevolent design of an agent incorporating a very altruistic behaviour. In other words helping should be prioritised. This is somewhat inspired by (Clark, Morton, and Bekey 2009) where they show *that altruistic actions, where robots assist with each other's tasks, can lead to decreased path costs for individual robots*.

For online replanning to work, the agents need some sort of execution monitoring that identifies when to replan, and (Russell and Norvig 2009)[chapter 11.3.3] distinguishes between three levels of execution monitoring: *action monitoring, plan monitoring and goal monitoring*, where

our client implements both action- and goal monitoring

The system architecture is inspired by the belief-desire-intention (BDI) agent architecture, which is well suited for a more dynamic environment and constitutes of continuously alternating between deliberation and then means-end reasoning.
Belief is how the environment is regarded by the agents. Desires are the different subgoals the problem have been decomposed into. Intention is the one desire allocated to an agent which it has committed to. Further the belief can be manipulated by changing the agents percept done by reducing its perceptual input thus relaxing the problem.

This BDI model is suitable for online replanning, because of the alternation between deliberation and means-end reasoning. Moreover the model should take into account how often to reconsider its intentions (online deliberation) reassuring that the agent is behaving optimal, yet this is computationally costly. From the main results of the experiments by Kinny and Georgeff assessing degree of boldness of an agent, it states that given a low rate of world change a bold agent will do better than a cautious agent, as the latter waste time reconsidering while the first work towards the goal (Wooldridge 2002)[chapter 2].

Related Work

In *Multi-Robot Cooperation in the MARTHA Project* (Alami et al. 1998) a similar problem of mobile robots for transshipment tasks at ports was researched.
They developed a system that distributes missions to the different robots through a Central Station. Each mission is first simplified as if the robot is alone which motivates the need for a process for validating when a robot came up with a new plan in the multi-robot context referred to as a plan-merging operation. Just like the CDPS, they decomposed, solved sub-problems and synthesised.

In *Hierarchical Planning and Learning for Automatic Solving of Sokoban Problems* (Demaret, Lishout, and Gribomont 2008) they tried to mimic human in-game reasoning by first decomposing into sub-problems, just as the MARTHA project did, and try to learn from the deadlocks in Sokoban. Further they added a preliminary goal arrangement which helped improve their results.
The paper faced some limitations of handling the cases with several entrances to goals, why it was suggested for further work to not compute a full goal schedule and instead recompute a list of possible targets throughout the process.
This partly inspired our approach, which will be elaborated later in this paper in the section *Goal decomposition*.

The paper *Altruistic Relationships for Optimizing Task Fulfillment in Robot Communities* (Clark, Morton, and Bekey 2009) is set in a multi-robot domain where the agents have individual tasks as opposed to the typical setup with a global cost to minimised. Here a strategy of altruistic behaviour was examined. They had robots with a set of tasks auctioning of some of them. Bidding on such task is then altruistic. The initial result of it was that such action can cause decrease in overall cost.
Though they concluded that considerable work still is required a parallel can be drawn from this ”individualised” setup to the decomposed problem, where each agent solve their own sub-problem in a common workspace, why the agents in our implementation has very altruistic behaviour as well.

In *Implementing a Multi-Agent System in Python with an Auction-Based Agreement Approach* (Ettienne, Vester, and Villadsen 2012) they solved a multi agent system with a goal to conquer as much space as possible with the agents having different roles entailing properties and abilities.
Here they presented an auction-based agreement algorithm for cooperation between the agents and different strategies. The algorithm determines the price of each goal given a bid from an agent. Each agent has some measure of benefit, and will bid on the goal that gives the most netvalue (benefit - price). The auction-based agreement showed promising results for pathfinding in discrete time.

Methods

The client we have developed is a mix of multi-body and multi-agent setups. The most complex objectives, distributing goals and solving conflicts, are performed centrally, but once an agent have been assigned a task all elements of planning lies within the agent itself. This carries the benefit of being able to relax the world state by limiting the percept of the individual agent, to simplify the search-space.

Level preprocessing

The client proposed in this paper first of all attempts to identify some of the properties of the given level that enables it to optimise its goal assignments, conflict management, and especially agent planning, by preprocessing the level. One of these properties is identifying critical paths in the level such as tunnels(agent can only move in 2 directions) or wells(tunnels with only one exit). Collisions in tunnels and wells are complex to manage and the knowledge of where these paths are allows the client to handle these conflicts more effectively with some hardwired social laws dependent on what environment the conflict occurs in. Another property that is identified is what components are connected in the level. Some levels can consist of multiple sets of connected cells that are completely separated, which basically means they should be treated as separate levels. Boxes that has no agent of the same colour are treated as walls, so that the connected components identification can assist with decreasing the search space. Also, identifying the connected components optimises the assigning of tasks by ensuring a box or agent from a different connected component does not get assigned to the task, which is important since there is no way for an agent to determine an assigned task unsolvable. The client also generates grids of true distances from all cells in the connected component to the goal locations as seen in the example in figure 1. These grids are generated by running BFS from each goal location reaching all

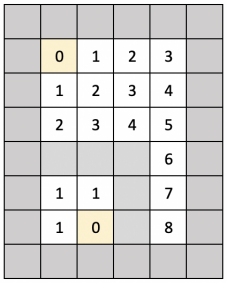

Figure 1: An example of a BFS-mapping of a level

cells in their connected component and storing the shortest-path distance for each cell. This grid increases the precision of the heuristic function as opposed to using the Manhattan distance. This is used e.g. when assigning a goal location to the closest object or during planning towards a goal location. Although this BFS-grid can help the client decrease its branching factor, its efficiency decreases with the increase in the number of goal-tasks and the size of the level, as the time complexity is bounded by $\mathcal{O}(n \cdot m)$ per goal location, where n, m are the dimensions of the level

Goal decomposition

When assigning tasks to agents, the hierarchical planning method is used to first of all decompose the overall goal state into independent subgoals for each box or agent, who needs to be located in a specific cell in the goal state. Furthermore, subgoals that involve moving a box to a location are decomposed into serialised subtasks of 1) move to the box and 2) move the box to the location. The location can either be a specific goal cell or out of a list of coordinates. This decomposition keeps the branching factor down, since the agent will only search with *Move* actions when searching to the box and only *Push* or *Pull* actions when searching with the box to a desired location. Before assigning the tasks, some subgoals will be prioritised over others, and goal schedules are in specific cases used to ensure the goals with higher priority are performed first. This is useful in wells with multiple goal locations.

Goal assigning

The problem of assigning objects to the right tasks is solved using a multibody approach. For a box goal location, the nearest box is selected using the BFS-grid and the closest agent to that box is then selected using Manhattan distance. The goal assigner will only distribute one subgoal to an agent at a time, and only if there are no box goals possible to solve for a given agent, i.e. free goal locations requiring a box of the same colour and in the same connected component, can that agent be assigned to an agent goal. The goal assigner attempts to reassign agents, whenever an agent reaches its goal or when its current plan length is zero. Furthermore, it also informs agents about when they are allowed to execute the next subtask of their current intent, had the intent been serialised. If an object is removed from its intended goal location, e.g. if another object needs to pass through and can find no way around, then the goal assigner detect that this task is no longer fulfilled and reassigns it at the next opportunity.

Relaxation of problem

The client is build with the agents' percepts of the world being a relaxation of the actual world state. The agents are not able to see other agents nor boxes that are not currently in their intent. This means, if an agent is planning to solve a task of moving a box to a goal location, the agent can only see its own box, the goal location and the walls of the level. Other boxes and agents may move simultaneously and it would not make sense to have their initial locations influence the agents plan in a future state. This relaxation allows the agent to plan a direct route towards its goal location.

Planning

When agents are planning for a task of moving a box to its goal location, the agent uses the A^* search strategy with the data from the BFS-grid as heuristic function. This is not the case when planning towards a box or for a helping task, and the search will often take longer, because the agent A^* search has to rely on the Manhattan distance as its heuristic function, which in some cases is less precise. Agents behave differently with regards to replanning depending on the environment and the type of conflict that has occurred. In some cases, e.g. if the agent collides with a stationary object that it can move around, it is sufficient to just temporarily change the agent's percept of the world state by adding the blocked locations as walls, and then letting the agent plan repair towards the next unoccupied coordinate in its plan. In other cases, it is less simple to replan, e.g. if the colliding objects are agents going opposite directions in a tunnel and therefore can not move around each other without leaving the tunnel. Thus, agents have also been provided with the ability to move themselves out of a given set of coordinates, which could be other agents' plans. The replanning is generated using BFS. Though the search space explores all nodes within a specific distance before progressing, the branching factor will for the most part be quite low as this scenario typically occurs within tunnels or wells.

Conflict Management

With the use of online replanning follows a method to identify and solve resource conflicts. We have implemented this in a central module called *the Conflict Manager*. This module is responsible for both the action- and goal monitoring. The conflict manager collects all agents current locations and the next action in their respective plans. With this information, it is possible to create a blackboard-like structure containing information on all agent and box locations at the current time, T, and after the actions are performed, $T + 1$. With this information the client is able to check preconditions and effects of all actions, to see if any conflicts are present. This setup enables for action monitoring, but also allows for goal monitoring. If any agents encounter conflicts with boxes or agents that are

in the way of their plans, and these objects are *stationary* (object did not move in previous time-step), the client will instantly identify an agent that is able to help. This is the goal-monitoring approach, as agents are able to change their intent before the current action is executed. Having established a hierarchy that helping-tasks is prioritised over everything else, the designated helper-agent now immediately drops its current plan and performs a search on how to help.

The Conflict Manager thus handles all potential conflicts. To efficiently handle situations in different parts of a level, e.g. in a tunnel, we have hardwired a range of social laws, determining how conflicting agents should act. If an agent collides with a stationary box or agent in a tunnel or a well, it will automatically start to move out into open space, so the helper-agent is able to reach the box without first having to ask the agent to move out of the way. If an agent collide with a stationary object in open space (not in tunnel or well), it will automatically try to navigate around it. If this is not possible, the client will assign a helper agent. If non-stationary agents or boxes are colliding and they are moving in opposite directions, the one with the most important job will get right of way (helping $>$ box-to-goal $>$ agent-to-goal), and the losing agent will have to plan to get out of the winners plan. Ties are decided by the index value of the agents. If an agent collides with a non-stationary object that is not moving in the opposite direction, it will receive a NoOp action from the client, as it is assumed the object will be gone in the very near future. Agents that are receiving help will gain a social status *pending help*. This status makes the agent passive; after eventually moving out of a tunnel or a well, it will receive NoOp actions until the helper-agent has finished the helping task. This is to make sure that the agent will not be assigned to other tasks while waiting for the help to arrive. Upon termination of the help task the involved agents have empty plans and are automatically reassigned by the goalassigner if there are eligible subgoals remaining. Note, the helper agents are not necessarily assigned to the same subgoal they gave up as other agents could have solved them in the meantime. Once all actions are approved by the conflict manager, they are appended into the joint action and are executed.

Experiments and Results

In the competition the client solved 28 out of 60 total levels, resulting in a completion rate of just under 47%. The result was split even, with the client solving 14 single agent and 14 multi agent levels. During the run on the competition levels, we discovered prior unseen bugs that needed some debugging. After fixing these the client performance increased to $\frac{33}{60} = 55\%$ solved levels with an overall decrease in joint actions. The development of the multi-agent system, was primarily focused on solving the multi-agent part of the competition, which was also evident from the results. The system was out of 30 participants, ranked in the bottom twenties in single-agent levels and top 4 in multi-agent levels.

In general we saw that especially large levels with a lot of goal locations, such as `MaReftAI.lvl`, were not solved by our client as the preprocessing of the levels simply took too long under the given time constraints. Furthermore, the client performed best in levels with low amounts of nested conflicts; conflicts engaged by agents that are already helping solving other conflicts. This is exemplified in a level such as `MAFPHPOP.lvl`.

As part of the development of the client, the following experiments were done to asses the impact of performance.

- Agents taking a random legal action.
- Replanning around objects.
- Goal-location penalties in heuristic function.

The results of these implementation can be seen from figure 2.

Agents taking a random legal action was inspired from Stochastic Local Search concepts and was implemented to help the agents out of otherwise inevitable dead-lock states. The idea is, if an agent, who currently has some sort of task assigned, does not seem to go anywhere (after a specified number of NoOp-actions), then it should take a random legal action to get out of the state.

Without this agent randomness the client would be prone to reach infinite loops of help requesting and conflict resolution, and would not be able to solve the `MAeasypeasy.lvl` level. As can be seen in table 2 with this feature the level is solved in 348 moves, highlighting that this solution is not efficient. Compared to the results after applying the two other features, the number of actions used is very high but nevertheless leads to a solved level.

The *replan around objects* feature would allow for agents to search around stationary objects it has collided with, by letting the agent temporarily percept the object, plan around and resume its plan if possible. This option is opposed to an initial implementation where agents could merely ask for help if facing an obstacle. The method was implemented using a plan repair approach, where the agent looks ahead in its plan for the next ”free” location. The method solved the major problem faced in the level `MAaicecube.lvl`, depicted in figure 5, since it allowed for agent 1 to replan around box A without requesting help from agent 0, which otherwise would have led to an infinite help-request loop, where agent 0 would not be able to help, since box B blocks the inner area of the level. The effects of this implementation is also noticeable in MAeasypeasy.lvl, where the feature results in a significant reduction $\frac{83}{348} = 23.9\%$ of the actions used without replanning.

Goal-location penalties are implemented in the heuristic function and seeks to make agents choose paths not crossing other goal locations when searching to- and with boxes. This experiment evolved as a response to cases like the one represented in fig 4, where agent 6 has a plan along the map edge until it reaches its own goal-location, as this was the shortest path according to its percept. This implementation dramatically reduced the occurrence of this type of conflict by encouraging the agent to move through the search-space

	Agent randomness	**Replanning around objects**	**Goallocation heuristic penalties**
Method:	Allow for the multi-agent system escape state-loops	Replan around objects in open space scenarios	h+ ϵ if box or agent in a goal location
		Small search space	
Level	`MAaicecubes.lvl`	`MAaicecubes.lvl`	`MAaicecubes.lvl`
Time used (s)		0.491	0.494
Actions used		64	64
Solved	False	True	True
		Large search space	
Level	`MAeasypeasy.lvl`	`MAeasypeasy.lvl`	`MAeasypeasy.lvl`
Time used (s)	3.022	4.765	1.236
Actions used	348	83	56
Solved	Yes	Yes	Yes

Figure 2: Benchmarks. The features are enabled from left to right meaning "Replanning around objects" also has "Agent randomness" enabled and so on. Device specs: 2.2 GHz 6-Core Intel Core i7 processor, 16 GB 2400 MHz DDR4 ram, with a 10.000 action limit and limit of replanning depth to g=5

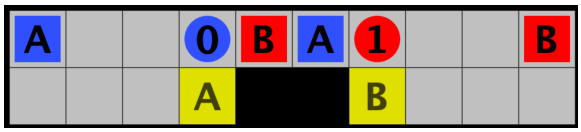

Figure 3: Simplification of `MAReftAI.lvl` with box-wall consisting of "active" boxes. Case unsolvable due to goal assigner

in a path that is less prone to conflicts. As can be seen from table 2, this implementation was highly efficient in `MAeasypeasy.lvl` as it saved 27 moves and almost cut the execution time down to 25% compared to the results without this feature. The drawback of this solution is that if an agent can not avoid crossing a goal location, e.g. in a well, it will branch out further in the search before crossing the goal location.

After the competition we examined further on some of the levels the client was not able to solve, and found that especially one situation was causing major problems for the agents. This is illustrated by the cut-out of the bottom left corner of the level `MAGlados.lvl`, shown in figure 6. Hardwiring an agent to always try to replan around an object as the first solution to a collision, agent 0 in the figure would only be able to percept one of the boxes that is is trapped behind, thus would end up in an infinite replan-collision-replan loop as it would never consider the option to just push the box out of the way. Disabling the feature *Replanning around objects* and letting the agent request help (from itself) to move the box resulted in the client solving the given level in 456 moves, using 8.3 seconds. Though disabling the feature certainly helped in this level, it is a major benefit for most levels that the agents always try to replan around objects as the first solution.

Discussion

Multiple elements in the client can be considered as a tradeoff between solving some types of levels particularly well and others particularly worse. Because of the bfs-grid the preprocessing encountered problems in levels like `MAReftAI.lvl`, where the search-space and count of goal-locations is large. This level required `search_space*goal_locations` $= 48^2 * (48 * 5.5) \approx 6.000.000$ nodes explored just for the prepossessing. For the identification of connected components, the motivation was to better assign the agents and also limiting the search space for agents, which also showed its worth in several levels including our own. However, larger levels were as mentioned the weakness of this preprocessing element, and a more dynamic client that only used these preprocessing methods on certain levels depending on its size and number of subgoals may have been preferable. Another improvement would be to only use this mapping if the subgoal is not fulfilled already in the initial state, since we observed some competition levels where this was the case.

`MAReftAI.lvl` also revealed another hurdle for the client, which lies within the generation of the desires presented to the agents in the deliberation phase. The multi-agent system discussed in this article has a central framework generating all "jobs" for the agents to asses. These jobs consist of an assignment of a specific box to a specific location, which cannot change, hence the agent can get stuck in states, that it cannot recover from. This is exemplified in figure 3. In this case, reading from left to right, the second A box will be assigned to the A goal-location and the first B box to the B goal location, because they are the closest. As this is a static assignment, the agents $0, 1$ will be stuck in an infinite help-request loop where they are asking each other to move unreachable boxes out of the a path. One way of handling this problem would be to have the client updating its beliefs as conflicts occur and updating goal tasks thereafter, leading to another

box being assigned to the goal location.

Remembering conflicts and updating beliefs could also be a solution to the deadlock in `MAGlados.lvl` mentioned earlier. The agent's world is relaxed from the true world state to such a degree that it repeatedly believes it can go one or the other way without pushing any of the boxes aside. The thought process behind this relatively big relaxation of the agents' world was that boxes and agents can be anywhere in the future, and there is no way of telling if a conflict is going to occur in the future without implementing plan monitoring, which can be very complex and computationally expensive.

Besides what is discussed above, our client performed satisfying on multi-agent levels in general, however its performance in the single-agent domain did not live up to the same standards. This is first and foremost because the client was build specifically to the more complex case of having multiple agents in the same environment that are able to interact, where the single-agent domain carries some other challenges that were not prioritised. In the single-agent domain, there is no immediate reason to relax the agent's world so much that it can not see any boxes, because all boxes are movable by the agent and will not move anywhere by any other agents. Also, the general approach in the conflict manager is to first try and move around the colliding object and only if this is impossible will it try and request for help and then respond to its own request. Here in the single-agent domain, it would likely be preferable for the agent to just go ahead and help itself while remembering its current subgoal, which it would otherwise forget in the current client design.

Future work

In relation to some of the weaknesses of the client elaborated in the discussion, this section assesses some of the improvements or further experiments that are worth doing to the client.

As mentioned, the preprocessing on levels with a larger size or a larger amount of goal locations is very time-consuming, and this problem caused some levels to remain unsolved. A suggestion for future work would be to solve this problem by implementing a threshold on how large a fraction of cells can be goal-locations in the level for the client to make use of these prepossessing methods.

Another approach that would help our current implementation would be to only generate the BFS-grids, if the subgoals are not satisfied already in the initial state. This implementation will result in more agent planning without the help of the BFS-grid, and it would also be interesting to see the effects of this on the collective results.

Another property that would be interesting to implement in the client is the ability to gradually adjusting the agents' percepts and thereby their beliefs. As mentioned, the general course of action that the conflict manager follows leads to infinite deadlocks in some levels, because the agent will repeatedly try to replan with no memory of its previous conflicts. In relation to this it would be interesting to experiment with implementing some sort of gradual change in what is perceived by the agent depending on the previous conflicts. Another solution could be for the agent to always perceive everything that is a small deterministic radius (e.g. 1 action) from the agent.

The current goal assigner with a first comes first serve approach will in some cases prove suboptimal. It would therefore have been interesting to implement an auction based method in the goal assigner similar to in (Ettienne, Vester, and Villadsen 2012), where the agents all have some sort of benefit measure related to every subgoal thus making the goal assignment near optimal with regard to this benefit measure.

References

Alami, R.; Fleury, S.; Herrb, M.; Ingrand, F.; and Robert, F. 1998. Multi-robot cooperation in the martha project. *IEEE Robotics Automation Magazine* 5(1):36–47.

Clark, C. M.; Morton, R.; and Bekey, G. A. 2009. *Altruistic Relationships for Optimizing Task Fulfillment in Robot Communities*. Berlin, Heidelberg: Springer Berlin Heidelberg. 261–270.

Demaret, J.-N.; Lishout, F. V.; and Gribomont, P. 2008. Hierarchical planning and learning for automatic solving of sokoban problems. *20th Belgium-Netherlands Conference on Artificial Intelligence*.

Ettienne, M.; Vester, S.; and Villadsen, J. 2012. Implementing a multi-agent system in python with an auction-based agreement approach. In *Programming Multi-Agent Systems*, Lecture Notes in Artificial Intelligence, 185–196. Springer. 9th International Workshop on Programming Multi-Agent Systems, ProMAS2011 ; Conference date: 03-05-2011.

GSHDK. 2020. 02285-ai-programming-project. `https://github.com/GSHDK/02285-ai-programming-project`.

Russell, S., and Norvig, P. 2009. *Artificial Intelligence: A Modern Approach*. USA: Prentice Hall Press, 3rd edition.

Wooldridge, M. 2002. Reasoning about rational agents. *J. Artificial Societies and Social Simulation* 5.

Appendix

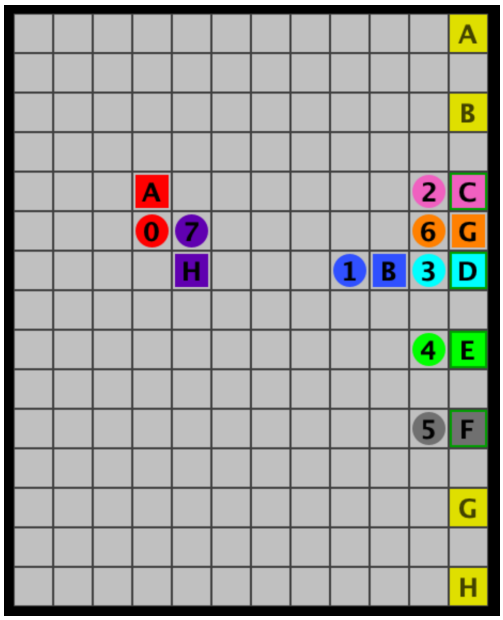

Figure 4: MAeasypeasy.lvl conflict

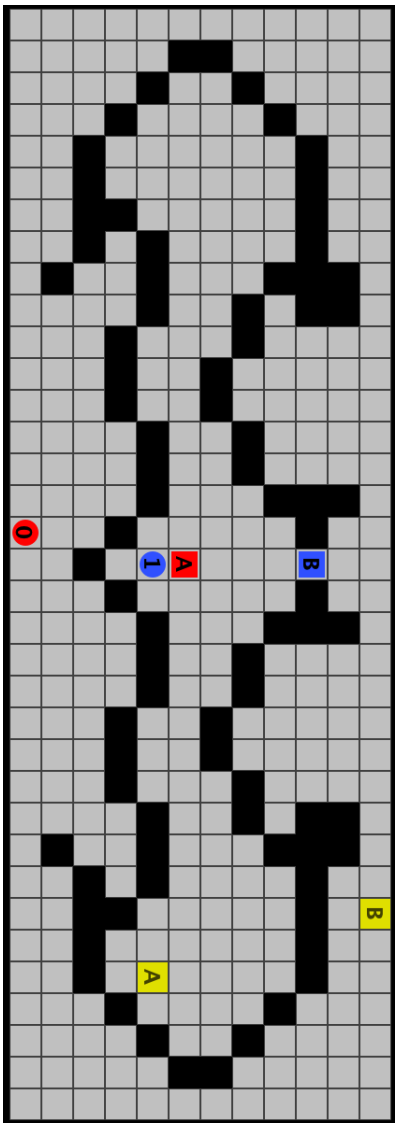

Figure 5: MAaicecubes.lvl conflict

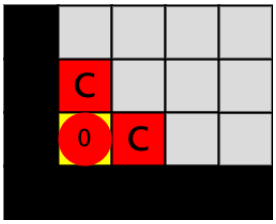

Figure 6: MAglados.lvl cutout