Improving Multi-Agent Systems Using *Jason*

Steen Vester, Niklas Skamriis Boss, Andreas Schmidt Jensen, and Jørgen Villadsen*

Department of Informatics and Mathematical Modelling
Technical University of Denmark
Richard Petersens Plads, Building 321, DK-2800 Kongens Lyngby, Denmark

Abstract. We describe the approach used to develop the multi-agent system of herders that competed as the Jason-DTU team at the Multi-Agent Programming Contest 2010. We also participated in 2009 with a system developed in the agent-oriented programming language Jason which is an extension of AgentSpeak. We used the implementation from 2009 as a foundation and therefore much of the work done this year was on improving that implementation.
We present a description which includes design and analysis of the system as well as the main features of our agent team strategy. In addition we discuss the technologies used to develop this system as well as our future goals in the area.

1 Introduction

In the Multi-Agent Programming Contest 2010 we have worked on improving the system of artificial herders used by the Jason-DTU team in 2009, which is described in [1] and [2]. The team is a computer science research group focusing on symbolic logic, knowledge-based systems, declarative programming and related approaches to automated reasoning. We still have some members from the team of last year, but there have been a number of changes to the team.

We have used the implementation from 2009 as a skeleton, but have made a number of changes and additions to this implementation. This means that our agents are still implemented using the AgentSpeak multi-agent language interpreted by Jason. Jason is based on Java and a lot of the algorithms have been more natural to implement in Java, but the overall control of the agents is still handled in Jason which can make calls to the algorithms implemented in Java. More specifically, we used Jason 1.3.2 and Java 1.6 to develop our system. During the competition we used a desktop computer running Fedora Linux.

It has been challenging to participate in the contest and this paper will present some of our findings and results.

* Contact: `jv@imm.dtu.dk`

2 System Analysis and Specification

The Prometheus methodology was used originally as a guideline when developing the system, but this year we have spent most of our time refining the algorithms from the 2009 agents and have not in the same sense followed a specific multi-agent system methodology.

The agents from 2009 needed a small update to deal with the rules of the 2010 implementation. However the protocols were almost identical, so this was not a major issue.

Much of our time has been used on refining the algorithms used in 2009 and therefore the foundation of the system has not been changed too much.

However, we have made some changes of the agents which include use of tactical path-finding when herding cows as opposed to the shortest-path technique used in 2009. This is especially important in the 2010 scenario given that the cows are much more sensitive than in 2009. Since the system uses 20 herders instead of 10 it is also of increasing importance to investigate different possibilities when forming groups of agents that work on the same subgoal. We have also worked on agents that can sabotage the herders of the opponents since it is just as important that the opponent performs badly as that our own team performs well given the scoring system of the contest.

As last year we have a scout agent. The scout agent will move around the environment to obtain information about it. Since our agents share a belief base, all of our agents will obtain the information gained by the scout.

We have introduced three agents that are committed to sabotaging the opponent. These three agents also work as scouts until the enemy corral has been located, in which case they will obstruct the opponent as much as possible.

The rest of our agents are herders that are responsible for moving cows to our home corral. One of the herding agents has a leader role, which means that he will assign targets to all the herding agents. Thus, we are still working with a centralized approach when herding but the sabotaging agents have their own autonomous group with a leader agent telling the others what to do.

Except for the scout agent, the agents are not very autonomous. A leader assigns targets to both herding and sabotaging agents, but the individual agent will autonomously choose the best way to get to these targets. Thus, communication between our agents is close to one-way with the leaders sending orders to the other agents. The only exception is that agents will inform their leader whenever they have reached their desired target.

We still use a shared belief base for all our agents with an internal representation of the world that is updated every time step.

3 System Design and Architecture

Agents are programmed by using goals and plans to accomplish the goals which are very natural when we use an extension of the AgentSpeak language that is closely connected to the Belief-Desire-Intention model.

As an example, our herding agents will have the goal `!move` to move at all times. Therefore the agents will have different plans for handling the goal to move depending on the circumstances. A plan can sometimes involve subgoals like obtaining a target from the leader agent or it could simply be a matter of making a call to our strategic path-finding algorithm to get find the best path to a given target.

Which plans the agents choose depend on their beliefs about the current state of the world. Since the environment is not fully observable at all times, the agents will collect the perceptions they get from the environment and collect it in a common belief base. The notions of belief bases are at the center of the AgentSpeak language and in the Jason implementation it is relatively easy to customize the way a belief base works, so our agents can obtain the same beliefs about the world at each time step.

The architecture used also makes inter-agent communication quite easy. In Jason, the agents communicate using a language close to *KQML* (Knowledge Query and Manipulation Language). A formal semantics for the communication features in Jason are described in [5]. We use these features of Jason to implement the communication between our agents.

Coordination of agent movement is done by the leaders of the herding agents and the sabotaging agents respectively which leave little room for autonomy to the herders and sabotagers themselves. Thus our agents have very fixed roles with specific responsibilities. The only proactiveness of the individual agents is that they will ask their leader to delegate them a target if they do not have one or have just reached the target they were delegated.

4 Programming Language and Execution Platform

The system was developed using Jason v1.3.2 which is an interpreter for an extended version of the agent-oriented language AgentSpeak. The extensions include inter-agent communication facilities and an interface making it easy to execute Java code from the agent programs. The overall decision making and communication is handled in Jason whereas more low-level technical algorithms like path-finding algorithms and clustering algorithms are implemented in Java. We used Java v1.6.

For developing the system we used jEdit 4.2 which can be downloaded together with Jason. Eclipse was also used when implementing Java code. During the competition the agent programs ran through jEdit.

Each type of agent (herder, leader, sabotager, sabotager leader, scout) was implemented in an .asl (AgentSpeak Language) file where goals and plans specify the behavior of each type of agent. Goals are implemented with exclamation marks and an example of a goal is simply `!move` which denotes that the agent has a goal to move. Every type of agent is implemented with plans to handle possible goals and belief changes. Each plan has a triggering event which specifies when the plan is relevant. In addition a plan has a context that dismisses all relevant plans that are not applicable in the given situation. Finally, a plan consists of a body which specifies how to execute the plan. The syntax of a plan is:

```
triggering event : context -> body.
```

An example of a plan used by our herding agents is:

```
+!move : pos(X,Y,_) & target_pos(X,Y)
   <- -target_pos(_,_);
      !get_new_target;
      !!move.
```

This plan is relevant if an agent gets the goal `!move`. The plan will be applicable if `pos(X,Y,_) & target_pos(X,Y)` can be unified with the belief base (using unification much like in Prolog). In other words the plan is applicable whenever the current position of the agent is the same as its target position. If this is the case the agent will delete its target position from the belief base, accomplish the subgoal of getting a new target (handled in another plan) and then pursuit the goal of moving again.

By using this agent-oriented language for implementing agents we automatically get an implementation that is very close to concepts like goals, plans and beliefs, which makes much of the programming quite natural and understandable.

The more technical algorithms like strategic path-finding and clustering are handled in Java, but can be called from plan bodies by the agents. This is done because the AgentSpeak language is not very well suited for implementing domain specific algorithms like this. And also, since Jason is built on Java there is a nice interface between the two languages.

5 Agent Team Strategy

5.1 Tactical Path-finding

The use of tactical path-finding is inspired by [3]. It is a search technique that our herding agents use to find the best path to a given target instead of the shortest path.

The reason for using a different path than the shortest when herding cows is illustrated in figure 1.

In this figure our scout agents will start investigating the environment. When they notice the two clusters of cows, our leader agent will split our herders into two groups that each will have a cow cluster as responsibility.

As in the 2009 version, the herders will be assigned targets behind the clusters, so that they can "push" the cows towards the corral in the top left corner. Some possible targets are illustrated by the red X's in Figure 1.

The problem with using a shortest-path algorithm to reach the targets is that the shortest path from the agents to the targets move through the cow clusters.

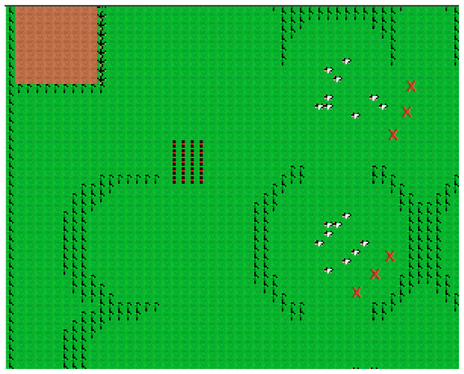

Fig. 1. Initial map where agents are red, cows are white, the corral is the red area in the top left corner, the dark green fields are obstacles and the herding targets illustrated as red X's

Since the cows are very sensitive to agents and will flee when agents come close the cow clusters will be chased away, which is not what we want. We would like the agents to move around the clusters before they get close to the cows so they will be pushed in the right direction.

In Figure 2 we illustrate the shortest path and the path we would like the agents to follow.

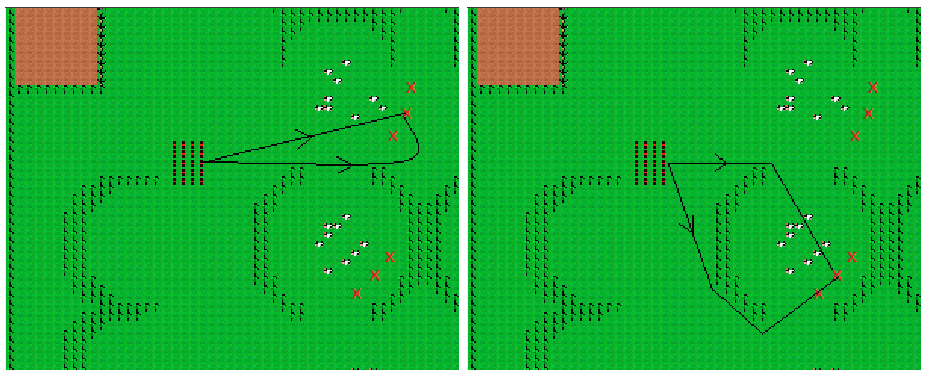

Fig. 2. The preferred path versus the shortest path for both clusters

The tactical path-finding algorithm we use is based on A*, but instead of simply using the step cost function g(s)=1 for each state in the map, we use quadratic filters which are laid on top of each state that holds a cow. These filters depend on the direction of the shortest path from the cow to the corral in such a way that states in front of the cow (i.e. between the cow and the corral) will receive a punishment that is increasing in size the closer we get to the cow, whereas states behind the cow will not receive punishments.

An advantage of the tactical path-finding algorithm is that it works exactly as a shortest-path algorithm when there are no cows nearby.

The algorithm in Figure 3 is used to calculate the step cost function of all states in the map.

Algorithm STEP-COST-CALCULATION
inputs: C, the set of cows; F, a set containing a $(2n+1) \times (2n+1)$ filter for every cow; w, width of the map; h, height of the map
output: A matrix containing step-costs for each state of the map

```
$G \leftarrow I[w, h]$ //A $w \times h$ matrix containing only 1's.
for all cows $c$ in $C$
    for $i \leftarrow -n, ..., n$
        for $j \leftarrow -n, ..., n$
            $G[c.x+i][c.y+j]$ += $F(c)[n+i][n+j]$
return $G$
```

Fig. 3. Algorithm for calculating step costs

The algorithm runs in $\Theta(n^2 + c)$ time where n is the width of the filter matrices and c is the number of cows. Since filters are typically kept quite small (because they are only interesting when the cows can be affected by agents close by) this algorithm is not too time consuming. In practice it is only necessary to run it once every time step for our whole team of agents since our agents have a shared knowledge base.

In our implementation we use 8 different filters which are used when the direction of a cow towards a corral is respectively North, East, West, South, Southwest, Southeast, Northwest and Northeast. By simple trial and error we have created filters that work reasonably well with respect to our goal. However, it is probably possible to improve these filters by using learning techniques. Also, it would probably work even better with more different filters than only eight.

Our implementation of the search algorithm has also been improved from last year by using a hashmap for the closed set of states. This gives us fast lookups of states that are already processed. We also use a priority queue for the open set of states which keeps the unprocessed states sorted in order of the most promising ones while keeping insert and remove calls quite cheap. These data structures replaced dynamic arrays in both cases.

A test sequence of the scenario introduced above can be seen in Figure 4 where our two groups of herders move around the clusters roughly the way we want them to. Note that the three agents to the far right in the first two images are scouts and not herders.

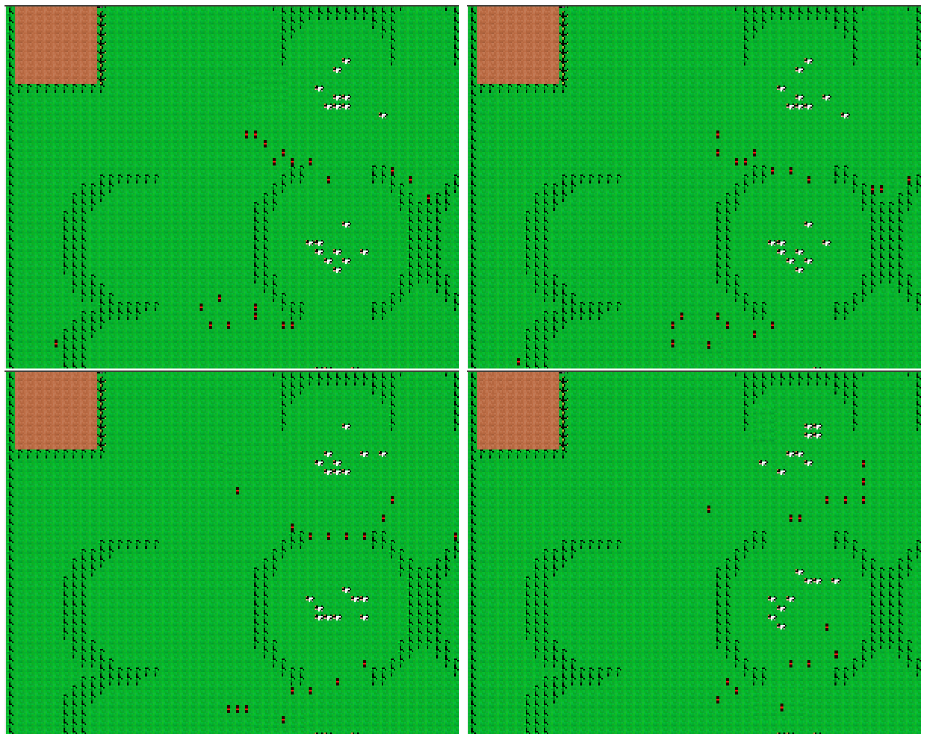

Fig. 4. Test sequence of the tactical pathfinding algorithm

5.2 Forming groups and cluster selection

From experience it seems that it is important to have groups of herders working together when collecting cows. The main reason is that single (or few) herders have trouble controlling large clusters by themselves and will often end up splitting the clusters into several smaller ones. We believe that it is in general more profitable to move large clusters towards the corral instead of splitting them up as was the approach last year.

When we started working on this year's agents our approach was to let all our herders work together in one big group. This worked quite well as long as clusters were large and there were a lot of room. However it was difficult to control such a big group in a dynamic environment and make it work together properly. In most environments we had more success by using 2-3 groups of 5-8 herders instead of all herders in one big group. The problem might be that our tactical path-finding algorithm and goal delegation algorithm is not well suited for controlling large groups because a large group in total control would probably be quite powerful.

We definitely want to investigate this area further and perhaps let the forming of groups be more environment-dependent and dynamic than now, where we use a fixed number of groups all the time.

The next question is how to choose which clusters to move. In our opinion important factors to consider seem to be:

- Cluster Size
- Distance from cluster to corral
- Distance from herders to cluster

Since we work in a limited timeframe and it is the average number of cows in the corral that counts we need to get as many cows into the corral as fast as possible. And clearly, it will be better to start moving the large clusters into the corral first, provided that it will take the same amount of time as moving smaller ones into the corral. This brings us to the other two points, which gives us an estimate of how quickly we can move a given cluster to the corral.

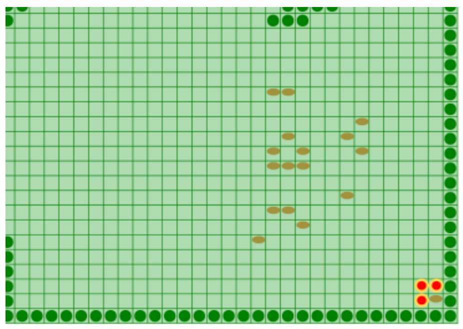

Fig. 5. Example of group of herders following the closest cluster (with only one cow)

When we started implementing our system, the only thing we considered was the distance from our group of herders to a given cluster. This gave some very suboptimal behavior where we could end up with large groups of herders chasing a few cows (or even a single one as in Figure 5) in the opposite end of the map than the corral. We than started weighting the distance from the cluster to the corral largest. This gave better behavior, but our herders would still leave anything they were doing if a single cow already captured would leave the corral again, which lead to all our herders spending most of the time in front of the corral. We also had some problems with weighting these distances, which gave rise to groups of herders redeciding between target clusters every few steps which led to a lot of inefficiency.

We did not refine our solution much, but feel that a good function $f(s, d_c, d_h)$ of the size of a cluster s, the distance from a cluster to the corral d_c and the distance of a cluster to a group of herders d_h can be created. When a lot of cows are in the corral we also feel that it will be a good idea to commit some of the herders to keep these cows in the corral (especially if the enemy team starts attacking).

Another thing that can be refined is our definition of a cluster. How long a distance should we allow between two cows in a cluster? In our implementation we used a fixed value of 5 fields in max distance between cows in a cluster, but it might be worth investigating other approaches.

After the competition in 2010 we have found an important bug in our program that allowed two cows on each side of a large obstacle to be in the same cluster as long as the euclidean distance between them was small enough. This clearly has caused some trouble on some of the maps with lots of obstacles whereas on large open maps it has not caused too much damage.

5.3 Sabotaging Agents

As a new feature we have added three agents whose main responsibility is to sabotage the herding of the opponent team. In the beginning of a match the position of the opponent corral is in general unknown. Thus, the three agents will start exploring the map until they find it.

Since a lot of the maps are symmetric (either using a horizontal, vertical or diagonal axis), the width and height of the map is known and our own corral is known they start by exploiting this fact to investigate the most likely enemy corral positions first. However, if they do not find the enemy corral in any of these locations they will just act as our scout agent that is reused from last year's implementation until the enemy corral is found.

When the enemy corral is found and it contains cows, one of the agents will open the fence to the corral and the other agents will move towards cows in the corral to make them move.

The approach has not been refined too much and there is still room for improvement. In particular our agents have problems with chasing cows out of the enemy corral if they stand close against obstacles or in corners. However, it has proven very important resultwise to try to chase cows away from the enemy corral since this job seems quite a lot easier than catching the cows in the first place.

6 Technical Details

Our agents did not perform any background processing when they were idle, but we are considering this for the next contest.

We did not experience technical problems during the competition which was quite pleasant. We feel that the protocol for communicating with the servers were quite clear and we ran a number of tests both on the online test server and the local test server provided. In addition to these tests we also used a monitor showing the model of the world from the agents' point of view which was also helpful during debugging.

We are currently working on a local test server where the length of the time steps can be decreased since debugging took a very long time with the current local server.

7 Discussion and Conclusion

We are relatively inexperienced with respect to developing multi-agent systems, so we have gained a much better understanding of the challenges of developing multi-agent systems. Especially the dynamic and complex nature of the environment has forced us to work with a number of different areas and quickly give up on trying to find perfect solutions even to subproblems in the system.

In general our solution is very fixed with agents having particular roles. We feel that a good solution should have dynamic agents capable of reasoning autonomously and capable of doing different tasks depending on the situation. In that context we are interested in developing approaches building on logically reasoning agents, since we believe that this will give more interesting results than the approach we have chosen in this year's contest. Our system essentially consists of a number of independent centralized systems that are quite domain specific. This resembles a classical AI approach more than a multi-agent system approach.

Using Jason to implement the system seemed like the natural choice since the task was to improve the system of last year that was implemented in Jason. In addition the team had some experience with the language though much of it was theoretical. As explained earlier Jason has some nice features making things like belief base sharing, belief updating and communication an easier task than it would have been in a more traditional general-purpose programming language. Also, the structure of the Jason programs where one uses high level notions like goals and plans makes the programs easily understandable to humans which is beneficial both when implementing the system initially but also when changing agent policies during system updates.

We were generally satisfied with the challenges of the scenario in this year's contest. Though we feel that it should not be so rewarding and relatively easy to sabotage the opponent team as it was in the scenario this year. We think that the scenario should be focusing more on the constructive tasks. Besides this we liked the different challenges posed by this year's competition and still feel that the scenario is far from solved yet.

References

1. *Niklas Skamriis Boss, Andreas Schmidt Jensen and Jørgen Villadsen*: Building Multi-Agent Systems Using Jason. Annals of Mathematics and Artificial Intelligence, Springer Online First 6 May 2010.
2. *Niklas Skamriis Boss, Andreas Schmidt Jensen and Jørgen Villadsen*: Developing Artificial Herders Using Jason. Proceedings of the 10th International Workshop on Computational Logic in Multi-Agent Systems 2009.

3. *Ian Millington*: Artificial Intelligence for Games. Second Edition, Morgan Kaufmann 2009.
4. *Stuart Russell and Peter Norvig*: Artificial Intelligence — A Modern Approach. Second Edition, Pearson 2003.
5. *Rafael H. Bordini, Jomi Fred Hübner and Michael Wooldridge*: Programming multi-agent systems in AgentSpeak using Jason. Wiley 2007.
6. *Michael Wooldridge*: An Introduction to MultiAgent Systems. Second Edition, Wiley 2009.

A Summary

1.1 *We are part of a computer science group that works with symbolic logic, knowledge-based systems, declarative programming and related approaches to automated reasoning.*

1.2 *We joined to get experience with the practical side of developing of multi-agent systems and to become part of an international network of researchers in the area.*

1.3 *We used Jason 1.3.2 and Java 1.6 to develop our system. During the competition we used a desktop computer running Fedora Linux.*

2.1 *We did not use a specific requirement analysis approach.*

2.2 *Our system uses five types of agents: herder leader, herder, scout, sabotager and sabotager leader. Agents may change roles over time.*

2.3 *The system was originally developed using the Prometheus methodology as a guideline.*

2.4 *Herding is controlled centralized by a leader giving targets to the herders. The sabotager agents are also controlled by a leader in the same fashion. Our scout works autonomously.*

2.5 *We have a number of centralized systems as described above. However, each agent has the responsibility of finding a good path to a given target. All the agents have a common belief base.*

3.1 *The Jason architecture makes it easy to implement agents individually with notions like beliefs, goals and plans. Also both agent-agent interaction and agent-environment interaction are relatively easy to implement.*

3.2 *Jason is built on the Belief-Desire-Intention model which also reflects the overall design of the agent programs. However, many algorithms are implemented in Java and can easily be called from the Jason programs.*

3.3 *Our agents have fixed roles. They communicate with the leaders only where they get quite precise orders and are therefore not very autonomous.*

4.1 *We used the jEdit 4.2 for implementing the agents in Jason and a mix of Eclipse and jEdit for the Java code. During competition we ran the agent programs through jEdit.*

4.2 *Each agent is implemented in Jason by the means of plans and goals. Internal actions (e.g. shortest-path algorithm) implemented in Java is called from the Jason agent programs as part of plans to accomplish goals.*

4.3 *Each agent type is written to a .asl file. It is written in AgentSpeak that is interpreted by Jason. In addition, there is a multi-agent system file used to specify each agent of a system by its type.*

4.4 *Overall structure of agents are in the .asl files, whereas specific tasks like shortest-path algorithms, clustering algorithms are implemented in Java in a package called* `jia`*.*

4.5 *Navigation of agents is done using a version of A*. In addition we have a clustering algorithm to represent cow clusters and various algorithms for selecting groups of agents and controlling groups of agents.*

5.1 *A* with different types of weights for different situations are used to handle navigation, cow herding, sabotaging, etc.*

5.2 *We have a couple of groups of herder agents responsible for herding cows and coordinated by a leader. We have a scout for discovering the environment and also a team of sabotaging agents controlled by a leader.*

5.3 *The leader of the herders will at fixed time intervals choose which agents should be in which herding group based on the position of the agents with respect to the chosen cow clusters.*

5.4 *Our agents communicate all percepts from the environment to each other. With a shared belief base they all have the same information about the environment when deciding on an action.*

5.5 *Herding agents ask their leader for targets whenever they reach their delegated target, which he provides. Likewise for the sabotaging team. The leaders can also delegate new targets without being asked.*

5.6 *More autonomy of each agent would be interesting. We also wish to implement a system with agents reasoning based on logical rules. But our current approach can still be refined in a lot of different areas.*

6.1 *Our agents did not perform any background processing when they were idle.*

6.2 *We did not have a crash recovery measure. We had an agent monitor showing the environment from the agents' point of view, which was used for debugging.*

6.3 *Our system proved to be quite stable and we did not experience any major crashes. Running the agents through jEdit also helped by giving us outputs about the connection status of each agent.*

7.1 *In general our solution is very fixed with agents having particular roles. A good solution should have dynamic agents capable of reasoning autonomously and capable of doing different tasks depending on the situation.*

7.2 *We are still relatively inexperienced with respect to developing multi-agent systems, so we have gained a much better understanding of the challenges of developing multi-agent systems.*

7.3 *Jason seemed like the natural choice since the task was to improve the system of last year that was implemented in Jason. In addition the team had some experience with the language though much of it was theoretical.*

7.4 *We do not have specific suggestions, but feel that it should not be so rewarding to sabotage the opponent team as it was in the scenario this year.*