Cows and Fences: JIAC V - AC'09 Team Description

Axel Hessler, Tobias Küster, Oliver Niemann, Aldin Sljivar, and Amir Matallaoui

DAI-Labor, Technische Universität Berlin, Germany

Abstract. The agent contest of 2009 has significantly increased the complexity of last years scenario. In this paper we present our approach to tackle this challenge. Based on last years work and methodology we introduce some refined collaboration strategies. While last years scenario gave rise to discussion of some destructive strategies, we think that this year such strategies play an important role and thus we try to address them. Again this years contest is strongly appreciated not only as testing ground and for evaluation purposes but also as contest of applied game strategies.

1 Introduction

As in the last year's edition of the contest, the JIAC V agent framework [?] will be used for implementing the multi-agent system. The framework is the successor of the time-honored JIAC IV [?], which has been created along with an accompanying toolkit in the course of a series of projects at DAI-Labor. Compared to AC'08 few things have changed. Still we believe, that these small changes will demand a lot more from the teams w.r.t. inter-agent communication and coordination.

This year, the *JIAC V Agent Team* has been prepared by the students of a university course at Technische Universität Berlin which is supervised by members of the Competence Center Agent Core Technologies of DAI-Labor, TU Berlin. From this we got some fresh ideas, and also have gained some more insight in how well our agent framework can be used by developers being unfamiliar with it.

2 System Analysis and Design

Intuitively, all students took a role-based approach to analysis and design, a role meaning the aggregation of functionality and interactions regarding a certain aspect of the domain. In the following, we name and explain the roles regardless of whether they have been implemented or not in the end.

Obviously, the agents need to explore their environment in order to get to know it: find cows, find all obstacles in order to calculate the best way to the own corral, find the opponent's corral. This is what is subsumed in the *Explorer* role. We then need to drive one or more cows to the corral, the *Herder* role. We also assume that cows may escape from the corral when someone is using the fence switch, so we presumably need a *Keeper* role. The additional feature of this year's scenario, the switch, leads to another role: the *ButtonPusher*, although we expect that this is not a full time job.

We also identified implicit roles which all agents must be capable of: connect to the server, receive perceptions from and send actions to the server: the *Server-Connector* role. An agent must also be capable of parsing the server message and update its world model, the *Perception* role. The perception role also notices if actions failed or not. Furthermore, each agent should be capable of talking to all other agents of its own team to share its perception and its intention, the *TeamCommunicator* role.

A third group of roles that has been identified concerns the opponent: analyse opponent behaviour in the the *OpponentAnalyser* role. Based on the analysis the agents can then interfere with opponent agents' actions, the *TroubleMaker* role is born. If applicable to the situation, the own agents may try to steal cows from the other corral. These capabilities and interactions can be concentrated in the *Thief* role. We also must take into account that the opponent has the same skill, so we will have to expand the Keeper role with the ability to prevent that opponent agents steal our cows. When discussing the roles, there was no consensus on the Thief and TroubleMaker roles. Some stated that it is not worth to have these extra roles, because when making trouble and stealing these agents could have driven cows to one's own corral.

Last but not least, we identified the necessity to analyse the behaviour and performance of our own team, the *TeamAnalyser* role.

3 Software Architecture

Our contribution is realised using the JIAC V agent framework which we are currently developing and extending. It is aimed at the easy and efficient development of large-scale and high-performance multi-agent systems. It provides a scalable single-agent model and is built on state-of-the-art standard technologies. The main focus rests on usability meaning that a developer can use it easily and that she is supported by the right set of tools depending on what she is doing. JIAC V is implemented in the Java programming language.

The aforementioned roles are implemented with components (agentbeans) which are the behavioural structures of the agent. They access and modify the agent's state, generate knowledge and trigger the actions. We also use two sensor/actuator components. One component, the standard communication component of the framework, is used for the information exchange between our agents. The other component gathers the perception messages from and delegates the action message to the competition server.

According the the rules of the contest, we have ten agents, forming the team, acting autonomously on the environment, each of them having the same set of roles. In Figure 1 we show the principle structure of each contest agent drawn in the JIAC V Agent World Editor.

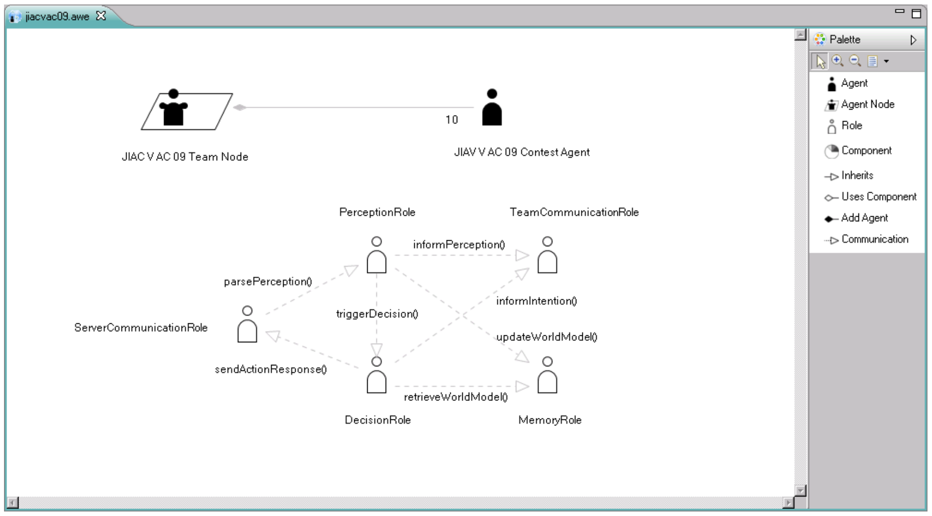

Fig. 1. Design of the competition agents using the JIAC V Agent World Editor (AWE)

4 Agent team strategy

Due to the similarity of the scenarios, large parts of last year's strategy can be adopted.

Firstly, every agent builds its own world model from what it is told by the server and its team mates. Every agent also plans for itself, by taking the intentions of its team mates into account. Further, they share both their perceptions and their intentions, preventing redundant actions and allowing to quickly re-enter the game in case an agent should need to be restarted.

Just like last year, the agents will navigate using the A^* algorithm both for calculating its own path and for calculating the path a cow should take and where to position itself to drive the cow in this direction. Last year, our agents were very proficient in driving single cows and smaller flocks of cows, and we found it impossible to separate cows from a flock of cows having a certain critical mass and shape. This year, due to the changed cow algorithm, single cows are harder to drive alone, and it is easier to drive smaller flocks and possible to drive even huge herds.

Last year we could see emergent behaviour while driving cows in a team of agents. This was a special feature of our cow driving algorithm [?], which has not been implemented explicitly. Although we wanted to implement explicit team strategies, we could see to our astonishment that our cow driving algorithm also adopts to the new terms, that cows do not dissappear in the corral, so explicit team strategy was not necessary. Even better, when cows try to escape from the corral or other agents try to drive them out, two or three agents alone take the *Keeper* role and drive them back. As we have not implemented such behaviour this can also be considered as emergent behaviour.

Finally, it was not necessary to implement the *Thief* role. Our agents do not distinguish between free cows and cows in the opponent's corral, they always drive cows not in the own corral.

One feature of the scenario made changes in the cowboy agents' behaviour necessary: fences. We try to solve the problem by introducing two new intentions: *OpenFence* and *GoThroughFence*. With these two intentions our agents tell their team mates if they just open a fence for others to drive cows through or if they just want to go through the fence in order to explore the world and find new cow herds.

5 Discussion

Again, this year's contest terms challenge the participating teams more then they did last year. But this was not the main reason why we wanted to take part int the contest. We use the contest as a test bed for the JIAC V agent framework to show how fast and reliable distributed computing can be done with agents from the shelf, and to improve the reliability and scalability of the JIAC V agent framework.

As JIAC V is so new, we have to improve the way how people can learn developing agents using the framework by documentation, tutorials and small examples, which show aspects of JIAC V agents and multi-agent systems in the nutshell. This is the result of watching the students while they look into the cow herding problem and try to implement the solution on top of the JIAC V architecture.

And not to forget, the supporting tools. We more and more learn which set of tools is essential to support agent-oriented software development. In this paper we have used a screenshot of the JIAC V Agent World Editor (AWE), a successor of the Toolipse AgentRole Editor [?]. The new AWE allows to design a multi-agent system without switching the view between platform level, agent level and agentrole level. Together with a code generation component, an application starter and a source code editor for the JADL++ agent programming language [?] on top of the Eclipse IDE [?] it makes the necessary toolkit for each agent developer.

6 Conclusion

The JIAC V team solved the problem of capturing as much cows as possible and to keep them in the corral. We used the contest to improve our new agent framework concerning reliability and scalability and tested the new AWE tool. The greatest pleasure was, again, to see emergent team behaviour while agents are driving cows, keep cows in the corrals and “steeling” agents from the other team. It is still not clear to us that sharing perceptions and intentions between agents is such a powerful concept. We also appreciate the higher scenario complexity.