International Journal of Energy, Information and Communications
Vol.5, Issue 5 (2014), pp.13-22
http://dx.doi.org/10.14257/ijeic.2014.5.5.02

Multi-agent System Learning Support Software with Fighting Games

Keitarou Busaki, Yasue Iijima and Susumu Konno

Department of Electrical, Electronics and Computer Engineering,
Faculty of Engineering, Chiba Institute of Technology

{keitarou.busaki, yasue.iijima, konno}@ga.aais-lab.org

Abstract

Beginners may find it hard to learn multi-agent systems because they need to study the *structures and concepts of agents as well as an agent* programming *language. Th*is *paper describes a prototype system in which users can learn structures and concepts of multi-agent systems without studying any agent* programming *language.*

Keywords: *Learning Support Software, Multi-agent system.*

1. Introduction

Multi-agent system technologies are used in various fields[1][2][3]. To build a multi-agent system, programmers must understand the structures and concepts of agents as well as an agent programming language (an agent description language). However, it is hard for beginners to learn these simultaneously because they have to use an agent programming language and framework corresponding to the intended application and purpose to create a multi-agent system. Even with two multi-agent systems that behave in the same manner, the methods of creating those systems depend on individual description languages. Figure 1 shows a description of a simple search agent using JADE[4] and LISP[5].

Simple JADE based search agent

```
public class MyOneShotBehaviour
       extends OneShotBehaviour {
    public void action() {
        // perform operation X
    }
}
```

Simple LISP based search agent

```
(ask-all
   :reply-with FA-20000518-9861
   :protocol sage_ask-all_1.0
   :language-encoding euc-jp
   :language KIF
   :ontology ebisu
   :aspect (?category ?name ?price)
)
```

Figure 1. Description of a simple JADE- and LISP-based search agent

To execute the procedure, “perform operation X” in the JADE-based code can be replaced by a process where attribute information must be added to the LISP-based code. The statements in these programs vary widely because of differences in the specifications of the languages although the same behavior is programmed. Consequently, learners often confront the following problems:

ISSN: 2093-9655 IJEIC
Copyright ⓒ 2014 SERSC

- They must study a programming language to create an agent corresponding to the intended use.
- They must recognize the notation of a language as the agent's characteristic.
- They fail to create an agent if the language used is different from the one that is familiar to them.

To solve these issues, we developed a new multi-agent system learning support software called "Agent Theory Learning System (ATLS)," which enables users to learn the structures and concepts of multi-agent systems regardless of the language-specific notation used. The paper's objective was to verify the efficacy of ATLS at educational sites.

2. Existing Agent System Learning Support Software

2.1. StarLOGO

StarLOGO, based on the LOGO programming language, can produce dynamic, autonomous, and intelligent programs [6]. Users can visually learn programming and agents' behavior. StarLOGO also provides tutorials explaining how to learn. Figure 2 shows a screenshot of StarLOGO. Knowledge for LOGO-based agents is described in the panels on the left side, in which characters are written, and users can check the described behavior of agents in the field on the right side. For example, the LOGO-based agents of Figure 2 are small car icons on the right side of the screen, and the actions of those cars are programmed in the panels on the left side. Programmers can define actions such as "make a round trip" or "change direction at a constant distance" to check the behavior on the right side of the screen. They can also intuitively understand agents' behavior through the tutorials.

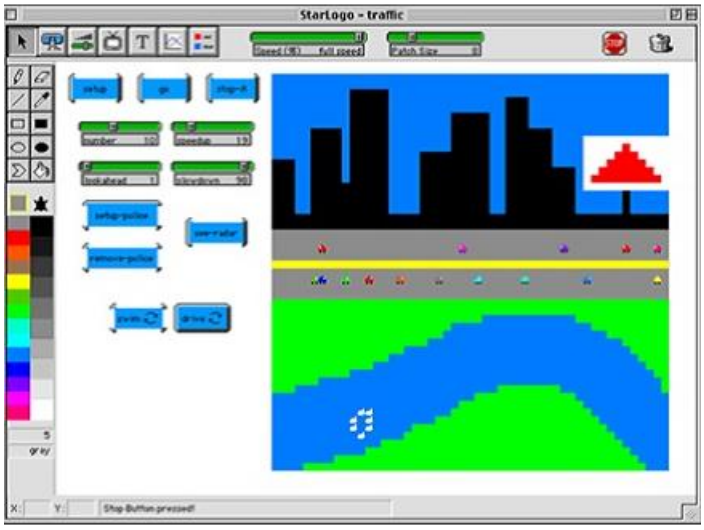

Figure 2. StarLOGO screenshot

2.2 AgentSheets

The most remarkable feature of AgentSheets is that users can create agents using a production system instead of a programming language [7]. To program agents, users just

 Copyright ⓒ 2014 SERSC

select if-then production rules in the window shown in Figure 3. No programming language is needed. Conditions and actions can be selected in the panels for if- and then-statements, respectively.

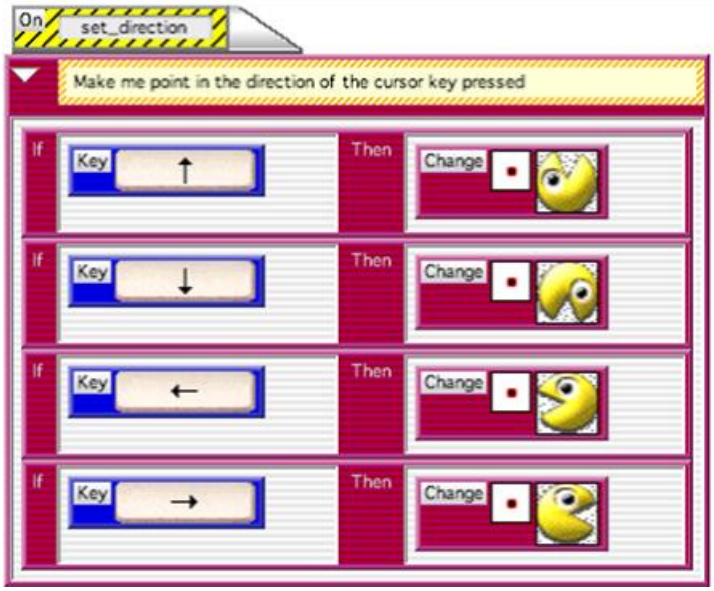

Figure 3. AgentSheets screenshot

2.3 Problems of the existing agent system learning support software

Both StarLOGO and AgentSheets have their unique characteristics. StarLOGO enables users to intuitively understand agents' behavior whereas AgentSheets does not require a programming language for creating agents.

However, to use these existing software systems, programmers must understand the unique languages of the systems as well as structures and concepts of agents because neither of the systems is designed for learning agents only.

The existing software systems require programmers to study a programming language; thus, there may be no software through which beginners can mainly absorb the basic knowledge of the structures and concepts of agents.

ATLS multi-agent system learning support software was developed to solve these problems.

3. ATLS design and prototype

3.1 Configuration of learning content

ATLS users are able to understand structures and concepts of agents through learning four key notions: autonomy, reactivity, social ability (or coordination), and persistence. To learn these notions in a proper order, ATLS learning content consists of six levels. Programmers, including beginners, can advance step-by-step from one level to the next to deepen their understanding of multi-agent systems using a single agent (levels 1–4) and multiple agents (level 5 and above).

Copyright ⓒ 2014 SERSC

ATLS users learn:

Level 1: How to create agents after they become familiar with ATLS.

Level 2: Autonomy of agents (behavior selection responding to contexts).

Level 3: Reactivity of agents (appropriate reaction to contexts).

Level 4: How agents can react to complex contexts appropriately.

Level 5: Social ability (or coordination) of multi-agent systems through allocation of roles to agents.

Level 6: Persistence of multi-agent systems (support of other agents).

3.2 Use of fighting games

To learn the six levels defined in 3.1 efficiently, we embedded fighting games, in which users can create game characters as agents. They can understand agents' behavior corresponding to contexts by playing fighting games and acquire knowledge of agents through the completion of a fighting game at each level. Random behavior was excluded to adjust opponent characters in accordance with learning levels; thus, users can clearly understand the learning purpose and criterion of each level.

Levels 1–4 are one-on-one games, and levels 5 and 6 are three-on-three fighting games. Users' and opponent characters deliver attacks in rotation. Characters behave differently depending on the preset contexts and knowledge (the details of contexts are described later). After an attack is executed, the state of the character is changed from behavior mode to monitoring mode until the next turn. When either of the two fighting characters exhaust hit points (HPs) (the initial value is 1,000), the game is over. To win a game, learners must reduce HPs of opponent characters to zero before their own HPs become zero.

3.3 Method of creating agents (characters)

ATLS agents are created in the same manner as those created in AgentSheets, using if-then production rules in which users select knowledge for agents. Users just select if- and then-statements from lists. ATLS does not allow users to describe agents' behavior by themselves. Therefore, they are able to create agents without learning a method of describing production rules.

Production rules can be executed in several ways: first-match execution (if-statements are checked according to the description order to execute matching rules in order from the start), last-match execution (if-statements are checked according to the description order to execute matching rules in order from the end), and reflection (if-statements are checked to skip rules that have already been executed). ATLS uses first-match execution.

3.4 Design of learning courses

3.4.1 Outline of learning courses: As explained in 3.1, ATLS learning content consists of six levels, which can be divided into two parts: levels 1–4 using a single agent and levels 5 and 6 using multiple agents. The purpose of learning with multiple agents is to understand agents' behavior. Therefore, communication and synchronization are not covered.

Different levels have different learning courses. Each learning course starts with a tutorial, and users can create a character (or characters) to understand the learning content of the intended level. After the tutorial, learners must win a fighting game with an opponent

 Copyright ⓒ 2014 SERSC

character to prove that they fully understand the learning content of the level. Learners must create a new character (or characters) to play a game, and can move on to the next level when they beat an opponent character. If learners lose a game, they must repeat the tutorial of the present level.

3.4.2 Level 1: Level 1 is designed to teach the right way to create an agent with ATLS as well as the flow of a fighting game. Learners can create three rules and four patterns of behavior per agent. Basically, they can win the game if the preset rules to attack an opponent character are correct. In case learners select a wrong condition such as "attack the opponent only when its HPs are full," they may lose the game because they can attack the opponent character only once.

3.4.3 Level 2: Level 2 is designed to teach the autonomy of agents. Agents are able to change their behavior in response to contexts. Learners at this level can create five rules and six patterns of behavior, including agent self-recovery of HPs using magic points (MPs) (the initial value is 100). To execute the newly-added behavior, they must allow the agent to recover depending on its HP status. To win the game, learners need to set a rule that the agent can take an action for self-recovery when its HPs drop below a certain value. In case learners set a wrong condition such as "recover self-HPs depending on the opponent's HPs," they may lose the game because of the limited behavior. In addition, the agent is switched to defense mode after receiving an attack if the first rule is "act in self-defense when HPs are reduced."

3.4.4 Level 3: Level 3 is designed to teach the reactivity of agents. Agents have the capability to take appropriate actions according to context. Learners at this level can create eight rules. Moreover, two behavioral patterns related to recovery of MPs become available. Programmers must create an agent that can select the optimal behavior from all patterns, including the actions added at level 2 (recovery of HPs using MPs).

Programmers generally need to set a rule that the agent recovers MPs before HPs. The balance of recovery (how many MPs and HPs have to be recovered) is the key to winning the game. For example, programmers should set the first rule that the agent takes a recovery action when MPs drop below 30 to avoid depletion of MPs. Then the second rule must be recovery of HPs so that the agent can continue to perform. In case of a rule to recover HPs before MPs, the agent executes a recovery action even when MPs are zero; accordingly, the agent cannot beat an opponent character because the agent has no HPs.

3.4.5 Level 4: Level 4 is designed to teach complex behavior of agents. Agents are able to change behavior depending on timing. Programmers can create ten rules with eight additional behavioral patterns such as attacks using MPs; the agent can implement more complex actions by attacks using MPs. In addition, a new agent parameter "attribute" becomes available. Because this parameter controls attacks using MPs, programmers must decide the agent's behavior strategically, considering advantages and disadvantages of inter-attributes.

Because of the addition of the attribute parameter, it is difficult to win the game with the simple strategy of level 3 (the agent recovers HPs and MPs to attack an opponent character). Learners of this level must identify the attribute of an opponent character to attack weak points or restrict behavior. However, if learners attempt to execute all of the added behavior, they may lose the game because the conditional statements become too complex for the agent to control behavior.

Copyright ⓒ 2014 SERSC

3.4.6 Level 5: Level 5 is designed to teach social ability (or coordination) of multiple agents through the allocation of roles. Programmers can create as many rules as possible at level 4. However, more options for conditional statements are available because this is a team competition; thus, the behavior of agents also becomes more complex. Programmers must decide the rules and behavior of agents according to roles.

To win the game, they need to create two types of agents: one attacks opponent characters and the other takes recovery actions. It is particularly important for the agent that takes a recovery role to understand the status of opponent characters.

3.4.7 Level 6: Level 6 is designed to teach persistence of agents through a three-to-three game. Learners at this level aim at winning the game with teamwork of agents. Dividing the roles of agents into two (attack and recovery) is sufficient to win the game at level 5; however, the strategy does not work at level 6 because opponent characters may focus on attacking one agent. Agents must take actions to support other agents in their roles and cover their weaknesses. Therefore, programmers have to create agents that can implement monitoring tasks to assist other agents.

3.5 ATLS prototype

We produced a prototype of ATLS using Java according to the contents described earlier. Figures 4 and 5 show the main screen and agent creation screen of ATLS, respectively.

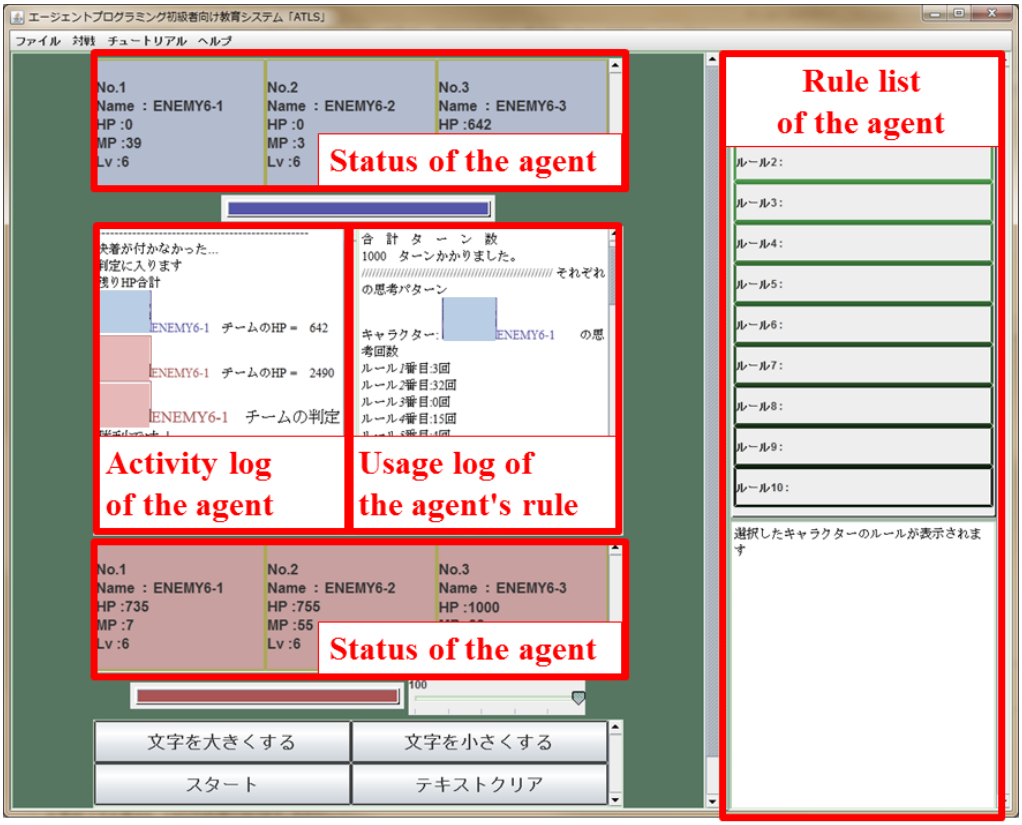

Figure 4. ATLS main screen

Copyright ⓒ 2014 SERSC

Figure 5. Agent creation screen

In the main screen, learners are able to see the agent status, the usage log of the agent's rule, activity log of the agent and the agent's rule list. Learner checks the learning status by using this main screen.

To program agents, learners just select if-then production rules in the window shown in Figure 5. No programming language is needed. Conditions and actions can be selected in the panels for if- and then-statements, respectively.

4. Operation and evaluation of ATLS

We conducted a questionnaire survey of five laboratories at three universities on multi-agent systems. The questionnaire included a comparison with the multi-agent system called IDEA [8, 9], developed by the students of these laboratories, and its tutorials.

The results of the questionnaire survey showed that many ATLS users failed to follow the tutorials of level 5 and above. This may have been caused by the gap in the number of controlled agents (levels 1–4: a single agent; levels 5 and 6: multiple agents). More detailed tutorials are needed for levels 5 and 6 to solve this issue. Moreover, half the students indicated that it was hard to see the window shown in Figure 4. We used a high-resolution monitor to develop ATLS. However, some students operated the system using a low-resolution monitor, and consequently part of the window was displayed off the screen. This problem can be solved by selecting a screen size before operating the system to optimize the window size. Students evaluated that ATLS is a more user-friendly learning tool than IDEA. This result indicates that ATLS is effective in learning multi-agent systems.

On the other hand, some students requested a function that judges the level of proficiency. According to them, those who are familiar with fighting games may create agents that can beat opponents without a full understanding of agents. We can solve this problem by adjusting the parameters for fighting and preparing simple tests.

5. Conclusion

We developed a multi-agent system learning support software called “Agent Theory Learning System” (ATLS), which enables programmers to learn the structures and concepts

Copyright ⓒ 2014 SERSC

of multi-agent systems without concern for language-specific notation. The objective of this paper was to verify the efficacy of ATLS at educational sites. The results demonstrate that programmers can learn simple structures and concepts of agents using ATLS.

Future challenges are to improve the tutorials for multi-agent systems and the interface.

References

[1] H. Uchida and H. Fujii, S. Yoshimura and S. Arai, "Learning Routing Policy for Changes in Road Network", IPSJ Journal, vol. 53, no. 11, in Japanese, **(2012)** November, pp. 2409 – 2418.
[2] M. Okaya and T. Takahashi, "A Framework of Evacuation Simulation Based on Agents Relationships and Behaviors", The IEICE Transactions on Information and Systems (Japanese Edition), vol. J94-D, no. 11, **(2011)** November, pp.1855-1865.
[3] M. Aoyagi and A. Namatame, "Effects of the Network Topology on Synchronization and Emergence of Flocking Behavior", IPSJ Transaction on Mathematical Modeling and Problem Solving, vol. 2, no. 1, in Japanese, **(2009)** February, pp. 37-46.
[4] F. Bergenti and G. Caire, and D. Gotta, "Interactive workflows with wade –jade", Proc. of the 21st IEEE Int. Work. on Enabling Technologies: Infrastructure for Collaborative Enterprises, Toulouse, France, **(2012)** June, pp. 10-15.
[5] A. Kawamura, N. Kitajima, Y. Teramoto, A. Satoh and R. Masuoka, "Building an Intelligent Agent System Using Lisp Language", Lisp User Group Meeting Japan, Talk 10, **(2000)** May.
[6] Massachusetts Institute of Technology Scheller Teacher Education Program: StarLOGO on the web (online), http://education.mit.edu/starlogo/, Accessed: **(2014)** August.
[7] AgentSheets, Inc., AgentSheets (online), http://www.agentsheets.com/, Accessed: **(2014)** August.
[8] T. Uchiya, T. Maemura, H. Hara and T. Kinoshita, "Interactive Design Method of Agent System for Symbiotic Computing", Proc. of the 6th IEEE International Conference on Cognitive Informatics (ICCI 2007), USA, **(2007)** August, pp. 312-320.
[9] T. Uchiya, S. Itazuro, I. Takumi and T. Kinoshita, "IDEAL: Interactive design environment for agent system with learning mechanism", Proc. of the 12th IEEE International Conference on Cognitive Informatics and Cognitive Computing (ICCI*CC2013), USA, **(2013)** July, pp. 153-160.

Authors

Keitarou Busaki

He was born 1986. He received B.E and M.E degrees in 2010 and 2013, respectively form Chiba Institute of Technology, Japan. He is with the city office of Katori, Japan since April 2013. His research fields are agent-based computing and learning support software.

Yasue Iijima

She was born 1991. She received B.E degrees in 2013, respectively form Chiba Institute of Technology, Japan. She is currently a graduate student at the Graduate School of Engineering, Chiba Institute of Technology, Japan. She is a member of The Information Processing Society of Japan. Her research fields are agent-based computing and knowledge engineering.

 Copyright ⓒ 2014 SERSC

Susumu Konno

He was born 1973. He received B.E, M.E and Ph.D in Engineering degrees in 1996, 1998 and 2002, respectively form Chiba Institute of Technology, Japan. He joined Research Institute of Electrical Communication, Tohoku University, Japan as a research associate in 2001. He was a senior assistant professor of the Graduate School of Information Sciences, Tohoku University since 2007. He has moved to Chiba Institute of Technology, Japan as an associate professor in 2008. He is an associate professor of the Department of Electrica1, Electronics and Computer Engineering at Chiba Institute of Technology. He is a member of The Institute of E1ectrical Engineers Japan, The Institute of Electronics, Information and Communication Engineers Japan, The Information Processing Society of Japan, The Institute of Electrical and Electronics Engineers, Association for the Advancement of Artificial Intelligence. His research fields are agent-based computing, knowledge engineering and symbiotic computing.

Copyright ⓒ 2014 SERSC

Copyright ⓒ 2014 SERSC