Fuzzy Logic in Agent-Based Game Design

Yifan Li, Petr Musilek and Loren Wyard-Scott
Department of Electrical and Computer Engineering
University of Alberta
Edmonton, AB T6G 2V4, Canada
yifan, musilek, wyard@ece.ualberta.ca

***Abstract* – The underlying artificial intelligence of computer games is constantly in need of improvement to meet the ever-increasing demands of game players. This paper discusses how intelligent agents and fuzzy logic can help increase the quality and amount of a computer game's most important element: interaction. The applications of fuzzy logic in behavior design are illustrated in detail through implementation of an arcade-style game.**

***Keywords* – Agent, BDI, Fuzzy Control, Combs, Game**

I. INTRODUCTION

What is a computer game? In his influential book, *The Art of Computer Game Design* [1], Crawford identifies four fundamental elements of computer games: representation, interaction, conflict, and safety. A Game presents a virtual reality in which players enjoy interacting with the virtual beings and taking challenges presented by the environment without facing any physical risk. Of the four elements, interaction is considered the most important. The advent of computers has brought forward an unprecedented interactive capability, making computer games truly different from conventional games.

Intelligent agents [2] and fuzzy logic [3] are two techniques that can help increase the quality of interaction in games. Agents provide a new architecture of game design that leads to more flexible interaction, and fuzzy control affords a practical method for generating subtle behavior.

As defined in [2], an agent is an autonomous software entity that has its own goals, sensory inputs, reasoning, reaction, and social ties. In the agent-oriented approach to game design, the game creatures are modeled as purposeful 'living' entities. This approach bears multiple benefits: 1) the distributed nature of the architecture facilitates the creation of complex system behaviors; 2) agents interact with each other through high-level communication languages instead of rigid low-level interfaces, making it simpler to manage large number of interactive behaviors; and, 3) when immersed in the virtual environment, players tend to perceive the virtual creatures as intelligent individuals; modeling them as such from the beginning helps to meet that expectation during later development stages.

In the domain of games, fuzzy control is not just another control method: it is a quality behavior generator. Creatures with subtle behaviors enhance the perceived complexity, enjoyableness and credibility of a virtual environment. While classical rule-based systems often fail to attain subtlety, fuzzy rule-based systems allow the nuances among inputs to be captured and further reflected in the decisions at relatively low computational cost. Compared to some other intelligent techniques, fuzzy rule-based systems are easy to design and implement as well – they are very similar to the crisp rule-based systems, differing only in the approach used to partition the input space. As a natural extension of classical set logic, fuzzy logic is playing an increasingly important role in games.

This paper provides an anatomy of how agent technology, incorporated with fuzzy logic, can be applied to game design. A brief introduction to the Belief-Desire-Intention (BDI) agent theory and an overview of the BDI.net agent framework is provided in Section II. Section III closely examines BattleCity.net, a re-implementation of an old arcade game. Section IV outlines the extended framework and describes how fuzzy rule-based computing can be used to improve the perceived quality of computer games. Finally, Section V summarizes the conclusions and indicates the direction of future work.

II. BACKGROUND

A. Software Agents

An agent theory defines the internals of agents. Among the many theories surveyed in [2], the BDI model [4][5][6] is the most popular and influential.

During the past decade, the BDI model has been well-studied and formalizations have been developed [7][8]. The strong theoretical basis is supplemented by a number of successful industrial applications ranging from early NASA projects [9] to more recent air-traffic management [10] and air combat simulation [11].

The BDI model is rooted in the philosophical work of Bratman [12], which studies intention and its relation to other mental attitudes. As its name implies, BDI features three major mental attitudes as its building blocks - belief, desire, and intention:

- *Belief* is the agent's knowledge about its environment and itself.
- *Desire* describes the agent's goal: a system state that the agent wants to achieve.
- *Intention* is the course of action that the agent has chosen to achieve that goal.

Plan is another important element of the BDI model. A *plan* is a recipe of action for achieving a certain goal and is used to guide the deliberation process of the agents so they need not search through the entire space of possible solutions [13].

B. BDI.net

BDI.net [14] is a Microsoft Visual C# [15] implementation of the AgentSpeak [16] BDI model. It is designed to be a lightweight framework for easy BDI implementation by casual programmers. This design philosophy leads to some favorable characteristics in comparison to the other existing BDI implementations, such as UM-PRS [17], JACK [18], JAM [19], dMARS [7], and ZEUS [20]. In particular, BDI.net offers the following advantages:

- Mainstream support and portability: BDI.net is written using a popular programming language and work seamlessly with all programming languages that support the .NET framework;
- Better alignment with standard programming knowledge: models can be described using common programming syntax rather than specialized symbolic logic;
- Testability: BDI.net is designed with testability in mind, providing special facilities and considerations to aid the debugging process;
- Explicit communication support: facilities for agent communication are built-in; and
- Tool library: apart from basic infrastructure, the framework implements many intelligent algorithms (including fuzzy inference) ready for use by the application programmers.

A BDI.net agent operates through iterations of its execution cycle as depicted in Fig. 1.

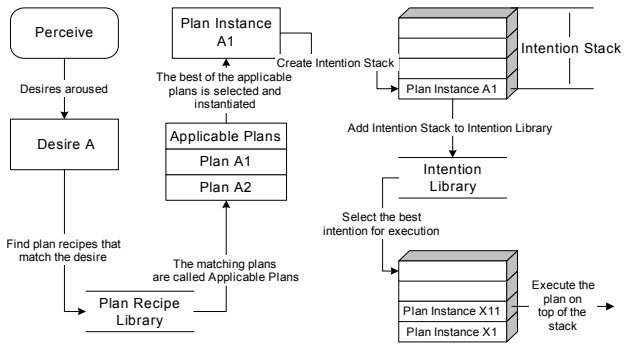

Fig. 1 BDI.net main execution cycle

At the beginning of each cycle, the agent senses its environment and updates its belief. Consequently, desires may arouse in response to the newly received stimulations. The agent then matches its stock plans against the desires to find out the best applicable plan for each desire (based on the plans' fitness values) and initiates new intentions of execution. Upon the end of the cycle, the intention that serves the most urgent desire is chosen from the intention library to be actually carried out. During execution, one plan may occasionally need to achieve a certain sub-goal that is beyond its capabilities to obtain. In such scenarios, the plan can stimulate a new desire which the agent will try to fulfill in a fashion similar to that described earlier. For the originating plan, the call is synchronous – it waits until the sub-goal is achieved or believed unobtainable before devising further actions.

Communication is of paramount importance in any serious agent application: without communication, it is almost impossible to exercise control over a multi-agent system, let alone collaborative problem-solving. Agent Communication Language (ACL) [21], along with content language and ontology specifications, is an enabling technology of agent communication. BDI.net implements the Foundation for Intelligent Physical Agents (FIPA) [22] agent communication language standards for message encoding, semantics and pragmatics. FIPA ACL is chosen over the equally popular Knowledge Query and Manipulation Language (KQML) [23] due to its formal semantics, support for XML (which is relatively easy to parse), and other benefits such as specifications of interaction protocols.

C. Fuzzy Rule-based Systems

Fuzzy rule-based computing is the most widely used form of fuzzy logic. A fuzzy rule-based systems is generally composed of four components: 1) the *fuzzifier* converts the crisp inputs into fuzzy sets; 2) the *rule base* holds the relations between the inputs and the desired output, expressed as a collection of IF-THEN rules; 3) the *reasoning unit* applies the inputs to the rules; 4) the *defuzzifier* converts the result of reasoning back to crisp output values.

In the case of multiple inputs, the antecedents of a fuzzy rule are usually connected with the logical AND connective, and the rule base should cover every possible combination of all linguistic terms defined on each input variable. This requirement of completeness leads to a combinatorial explosion of the rules: for a system of n inputs with m terms each, a total of m^n rules are needed. Combinatorial rule explosion poses a severe problem for applying fuzzy control in game design as it complicates the design process and taxes precious CPU resources.

Combs has proposed the Union Rule Configuration (URC) [24][25] to alleviate the rule explosion problem. A typical URC is expressed as follows:

$$\begin{aligned} &\text{IF } X_1 \text{ IS } A_{11} \text{ THEN } Y \text{ IS } B_1 \\ \text{OR } &\text{IF } X_1 \text{ IS } A_{12} \text{ THEN } Y \text{ IS } B_2 \\ \text{OR } &\text{IF } X_2 \text{ IS } A_{21} \text{ THEN } Y \text{ IS } B_2 \\ \text{OR } &\ldots \\ \text{OR } &\text{IF } X_i \text{ IS } A_{ij} \text{ THEN } Y \text{ IS } B_k \end{aligned} \quad (1)$$

Where X_i denotes the input variables, Y denotes the output variable, and A_{ij} and B_k denote the linguistic terms defined on the fuzzy variables. Instead of presenting the

relationship between the connected antecedents and the consequent, the URC relates each input variable directly to the output variable, thus reducing the total number of rules required to $m \cdot n$.

Although the theoretical foundations of URC are still under debate [26][27], there is evidence of its applicability in a relatively large number of situations. It can serve as a practical and economic alternative to other approaches (such as clustering and hierarchical systems) to solve the rule explosion problem.

III. The Game

BattleCity is an old Nintendo Entertainment System (NES) game released by Namco group in 1985. Like most early video games it employs a simple plot (cf. Fig. 2): 1 or 2 players control their tanks and strive to protect their own base (presented as an eagle at the bottom of the map) and destroy all enemy tanks, then move on to the next stage. The game provides different terrain elements – rivers, bricks, stones, and trees, each featuring unique properties. The enemies have various shapes, colors, speeds, and armor strengths. However, all enemies share the same stochastic behavior, making them relatively easy targets. During game play, the human contenders are to devise a terrain-relative strategy to effectively clear the hostile tanks while remaining out of harm's way.

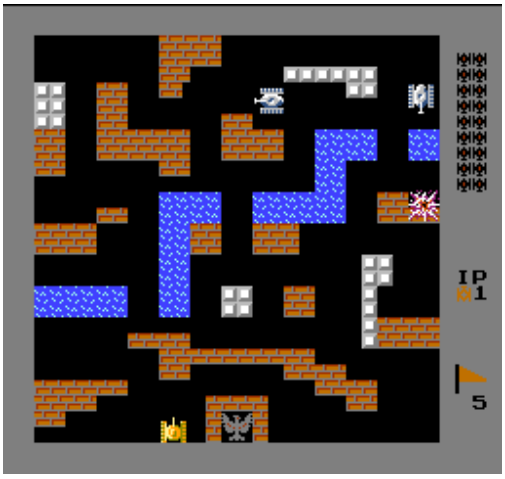

Fig. 2 Original BattleCity Game (single player)

BattleCity.net [28] is a remake of the original BattleCity game using the BDI.net framework and fuzzy logic. The rules and graphic elements are transplanted as is, but for the convenience of research the new game changes the plot to a computer vs. computer battle. While the enemy tanks remain stochastic, the player tanks are managed by intelligent agents in order to eliminate the need for human intervention.

A. Design

In an agent-oriented approach the target system is decomposed into autonomous entities and inanimate objects. The first step is to decompose the system properly and identify the agents. The nine types of entities are four terrain elements, three different enemy tanks, the base, and the player's tanks. In theory, all entities can be agents, but that would result in a waste of computing resources. As with many other design issues there is no straightforward answer as to which entities should be designed as agents. However a simple rule of thumb is applied: if an entity is to perform actions that change the environment, and the motivation for performing these actions originates from the environment, it is a candidate to be made an agent.

In BattleCity.net, the player-tank entity and the base entity are designed as agents, leaving the rest as plain objects. To generate a more realistic behavior, the player-tank agents have limited sight, which in turn demands that the base (also with limited sight) monitor its own vicinity and alert the player-tank agents when enemies draw close. Additionally, the base constantly evaluates its own situation and limits the player tank's distance to a level commensurate with the currently perceived threat. This strategy is to avoid a situation where the player tanks are too far away to rescue a threatened base.

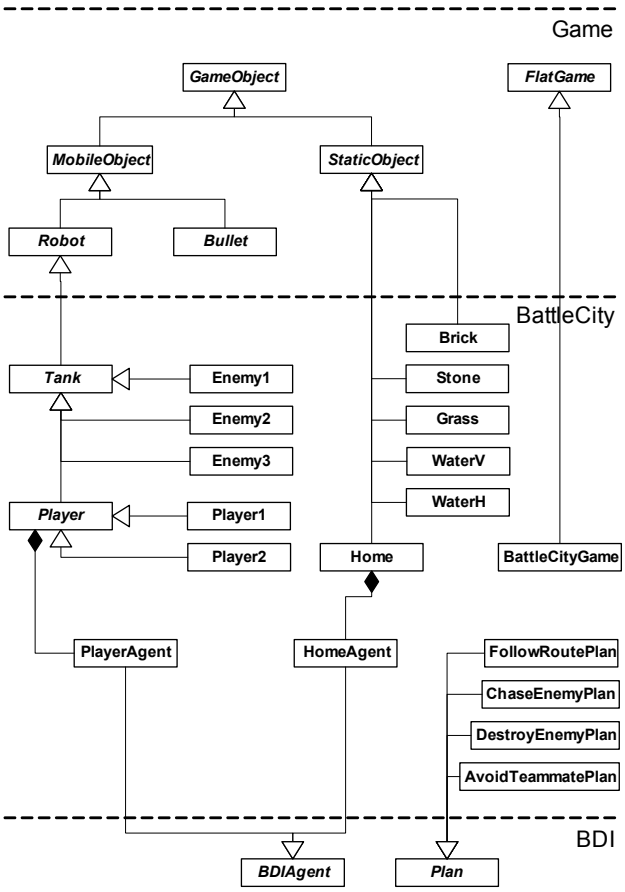

Fig. 3 BattleCity.net Design (Selected Components)

From a design perspective, an agent-oriented solution is a natural extension of object-oriented programming. Agent-based games can still take advantage of existing game

development frameworks. BattleCity.net makes use of a 2D grid game development framework that provides rendering, sound, collision detection, and basic object prototypes. BDI agents are embedded into the game objects to control their behaviors, but the actual actions are still performed by the objects, cf. Fig. 3.

B. *Implementation*

The game framework provides a control loop driven by an external timer to handle animations and collisions, and to give the other game entities an opportunity to perform their own processing. The enemy tanks use this time slice to make random moves and occasionally open fire, while the player tanks and the base pass the control to their 'brains', i.e. the agents. The terrain elements are not active entities and therefore do not require processing time.

While the agent framework covers the mental operations of an agent, plans have to be set up to inform the agent what can be done and how to do it. There are six major tasks for the player agent: to avoid bullets, to protect the base, to destroy enemies that can be fired at, to track the enemies down, to avoid collision with team-mates, and when no other task is pressing, explore the battlefield. Each of these situations has to be completely handled by at least one plan. Taken that the game is not complicated, one plan is conceived for each respective task. Similarly, for the base agent, only two plans are needed - calling for help and restricting the roaming range of the player tanks.

Aside from the main control loop described earlier, each BDI agent runs its own execution cycle in a separate thread. These loops are synchronized with the main control loop in order to avoid undesirable results. In other words, considering each timer event as one step, the agents should only be able to perform one set of actions at any step. BDI.net provides support for synchronization at two levels, both through semaphores: the agent execution cycle and the plans. At the agent execution cycle level, agents can be configured to wait for a ready-to-go signal at the beginning of each cycle. The plans can be interrupted and resumed at an execution cycle in the future. This is useful when a plan, such as chasing the enemy, takes multiple steps to finish. Special care has to be taken when implementing such plans by closely monitoring the agent's status to ensure that the plans are still valid under the current circumstances. This need arises since new situations may develop while a plan is temporarily suspended and other plans of higher importance are executed, potentially invalidating the original conditions.

Without centralized control, the game agents coordinate strategies among themselves. Calling for help is one situation that requires such coordination. As illustrated below in Fig. 4, when the base senses enemy tanks around, it sets up a *contract net* conversation with both player-tank agents by broadcasting a *Call for Proposal* (CFP) message that indicates the enemy's position and the deadline of proposal submission. If a player agent is not too busy it will reply with the cost of performing the rescue (the time it takes to reach the enemy's location). The base agent then chooses a winner from the submitted proposals. Once a contract is awarded to the player agent, it initiates a *subscribe* conversation message with the base agent so that it will be kept informed with up-to-date information about the threat.

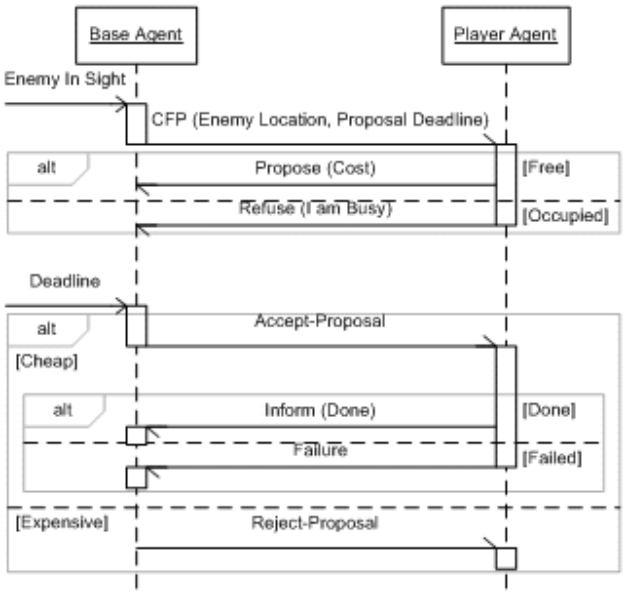

Fig. 4 Call for Help Contract Net Conversation

At the current stage the rule-based approach (in which a set of if-then-else statements define the stimuli and the corresponding reactions) is used to generate the behaviors of the agents, and the standard A* search algorithm [29] is responsible for path finding. The same configuration would be used without problem in a game that does not employ the agent-based design. However, since all the agents are planning their own paths locally, there is a possibility that the player-tank agents may run into each other and, in the worst case, result in a deadlock. A solution to this problem is to have the agents negotiate with each-other in case of collision, but a simpler approach is taken here: both parties in collision will stop and wait for a random period of time before re-evaluating their situation. In this way, the first agent to recover from the deadlock will have to find a new path.

IV. Fuzzy Logic in Game Agents

As stated earlier, the base imposes dynamic constraints on the player tanks' radius of roaming in accordance with its perceived level of threats. A fuzzy rule-based system is used here to increase the level of details of the decision process. The base considers three factors when evaluating the threats: 1) the elapsed time since the last attack; 2) its shield level (i.e. the status of the walls surrounding the base); and 3) the combined strength of the enemies in sight. The fuzzy rule-based system takes these three inputs and derives the desired radius of the player tanks, which is then directed to the player-tank agents via a *request* conversation. The settings of the

input and output variables are shown in Fig. 5 (elapsed time is expressed in unit of 30 milliseconds, or one game execution cycle, and the desired radius is denoted in number of grids).

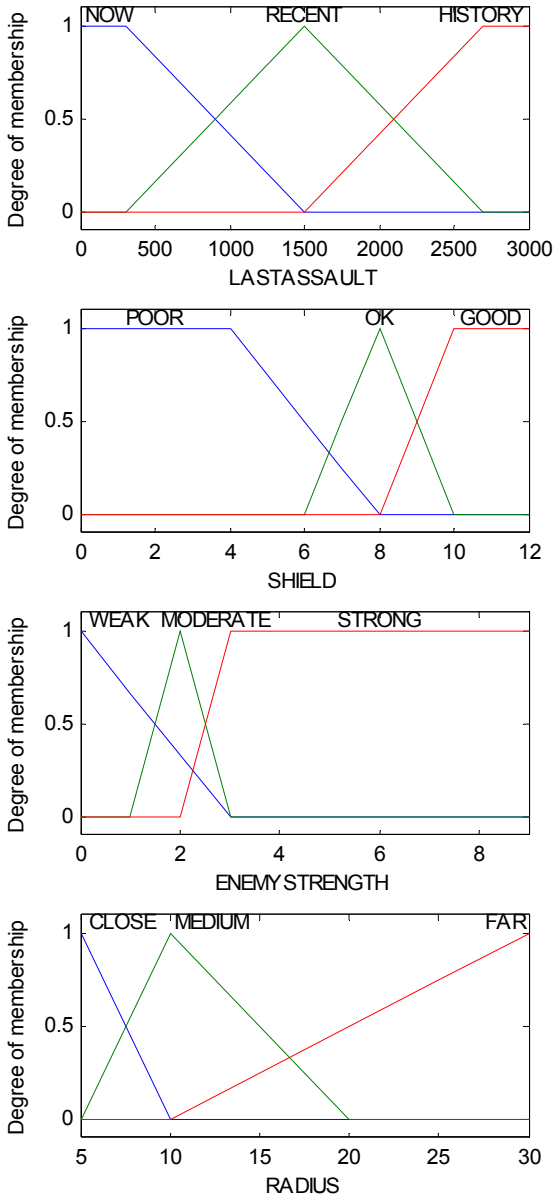

Fig. 5 Fuzzy Variables for Player Radius Control

The traditional and the URC configuration were evaluated and both met the expectations. However the traditional configuration employs 27 rules, which is three times the number of rules used in the URC configuration, the CPU usage is considerably higher as well. Obviously in this particular application the URC configuration is much more beneficial.

Below is a series of portraits of the game showing the effects of the fuzzy rule-based system in various situations. The legends ❶, ❷, and ❼ indicate the two player tanks and the base, with their respective sight limits denoted as ①, ②, and ⓪. The roaming radius limit of the player tanks is also shown on the pictures (denoted by ◎) for visualization purposes.

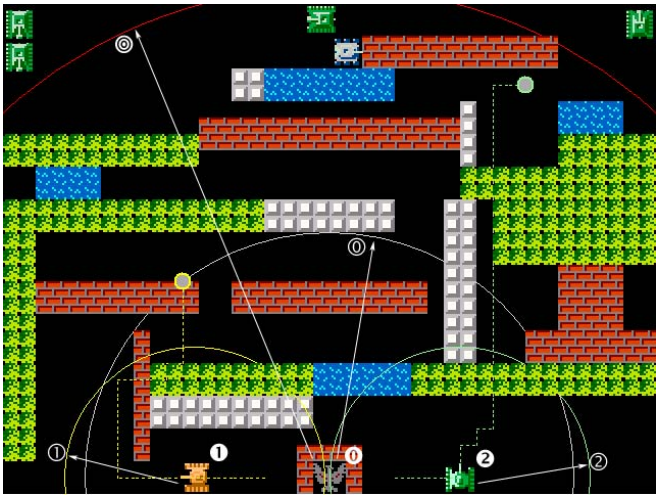

Fig. 6 Portrait of the game in action – beginning

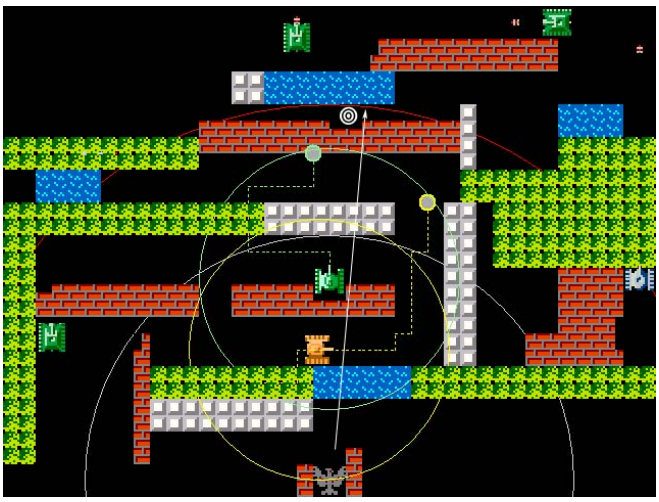

Fig. 7 Portrait of the game in action – slightly vulnerable

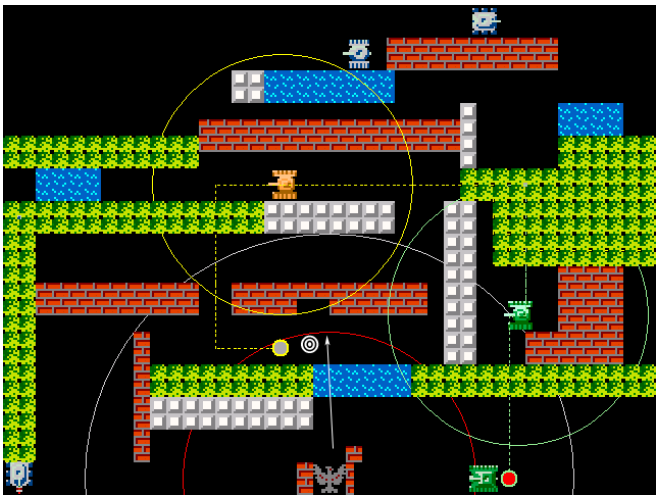

Fig. 8 Portrait of the game in action – emergency

In Fig. 6, the game has just begun and all conditions are favorable – the shields of the base are intact, no enemy in sight of the base, and there has never been any registered attack. As a result, the player tanks are allowed to roam almost freely. When the situation deteriorates in Fig. 7, the player tanks are informed to remain closer to the base. Finally, in Fig. 8 the base faces imminent threats – the shield on the right side is almost penetrated and an enemy is approaching, forcing one of the player tanks to engage the enemy and the other one to rush into the direct neighborhood of the base.

V. CONCLUSIONS

The Agent-oriented approach to game development offers many benefits throughout the development cycle. It provides a natural way of modeling the game creatures and a software architecture of high flexibility and low coupling which allows the flourish of behaviors. In addition, developers can easily integrate their old game development frameworks with the new design approach.

Agents will certainly play a key role in game development in the near future. However the agent-oriented approach is not without drawbacks. The most serious problem is the conflict between the need to maintain a storyline and the autonomous nature of the agents. The storyline often demands precise control over certain creature's properties, but the autonomous agents may exhibit undesirable emergent behaviors due to the absence of centralized planning and control. Such unwanted emergent behaviors can be eliminated on a per-problem basis, like the teammate avoidance problem described earlier. While patch works can also be effective, the general solution to this kind of problems will be a hybrid architecture that features both centralized control and autonomous agents with 'back doors' for external control.

Fuzzy logic is another intelligent technique that could be used to boost the performance of games. It handles complex control problems at low computational costs, without sacrificing the subtle details. While the paper discusses its applications within the context of agent-oriented game design, fuzzy control certainly applies to games of any architecture.

ACKNOWLEDGMENTS

Support provided by Alberta Science and Research Authority is gratefully acknowledged.

REFERENCES

[1] C. Crawford, *The Art of Computer Game Design*, Columbus: McGraw-Hill, 1984.

[2] M. Wooldridge and N. R. Jennings, "Intelligent agents: Theory and practice," *The Knowledge Engineering Review*, 10(2), pp.115-152, 1995.

[3] W. Pedrycz and F. Gomide, *An Introduction to Fuzzy Sets*, Cambridge: MIT Press, 1998.

[4] M. E. Bratman, D. Israel, M. Pollack, "Plans and Resource-Bounded Practical Reasoning," *Computational Intelligence*, 4:349-355, 1988

[5] M. P. Georgeff and A. L. Lansky, "Reactive Reasoning and Planning," Proceedings of the Sixth National Conference on Artificial Intelligence, volume 2, pp.677-682, Seattle, WA, 1987

[6] A. S. Rao and M. P. Georgeff, "Modeling Rational Agents within a BDI-architecture," Proceedings of Knowledge Representation and Reasoning, pp.473-484, 1991

[7] M. d'Inverno, D. Kinny, M. Luck, and M. Wooldridge, "A formal specification of dMARS," Intelligent Agents IV, M. P. Singh, A. S. Rao, and M. Wooldridge, eds, pp. 155--176. Springer Verlag, Berlin, 1998

[8] M. d'Inverno and M. Luck, "Engineering AgentSpeak(L): A formal computational model," Journal of Logic and Computation, 8(3), pp.233-260, 1998

[9] M. P. Georgeff and F. F. Ingrand, "Decision-Making in an Embedded Reasoning System," Proc. Int'l Joint Conf. on Artificial Intelligence, pp. 972--978, August 1989

[10] M. Ljungberg and A. Lucas, "The oasis air traffic management system," Proceedings of the Second Pacific Rim International Conference on Artificial Intelligence, 1992

[11] A. S. Rao, A. Lucas, D. Morley, M. Selvestrel, G. Murray. "Agent-oriented architecture for air-combat simulation," Technical Report Technical Note 42, The Australian Artificial Intelligence Institute, 1993

[12] M. E. Bratman, *Intentions, Plans and Practical Reasoning*, London: Harvard University Press, 1987

[13] A. S. Rao, "A Unified View of Plans as Recipes," Contemporary Action Theory, Editors Ghita Holmstrom-Hintikka and Raimo Tuomela, Kulver Academic Publishers, The Netherlands, 1997

[14] Y. Li and P. Musílek, "BDI.net: A Lightweight Framework," Proceedings of the Third ASERC Workshop on Quantitative and Soft Computing Based Software Engineering, pp. 49-53, 2003

[15] Microsoft Visual C# .net Home Page, http://msdn.microsoft.com/vcsharp/, 2002

[16] A.S. Rao, "AgentSpeak(L) : BDI agents speak out in a logical computable language," Proc. 7th European Workshop on Modeling Autonomous Agents in a Multi-Agent World, MAAMAW'96, LNAI-1038, Springer Pub., 1996

[17] J. Lee, M. J. Huber, E. H. Durfee, and P. G. Kenny, "UM-PRS: An Implementation of the Procedural Reasoning System for Multirobot Applications.," Proceedings of the AIAA/NASA Conference on Intelligent Robotics in Field, Factory, Service, and Space, pp.842-849, 1994

[18] P. Busetta, R. Rönnquist, A. Hodgson, and A. Lucas, "JACK Intelligent Agents – Components for Intelligent Agents in Java," AOS Technical Report 1, 1999

[19] M. Huber, "Jam: a bdi-theoretic mobile agent architecture," Proc. Intl. Conf. Autonomous Agents, pages 236–243, 1999

[20] H. Nwana, D. Ndumu, L. Lee, and J. Collis, "ZEUS: A tool-kit for building distributed multi-agent systems," *Applied Artificial Intelligence Journal*, 13(1):129-186, 1999

[21] Y. Labrou, T. Finin, and Y. Peng, "Agent Communication Languages: The Current Landscape," *IEEE Intelligent Systems*, 14(2):45-52, 1999

[22] Foundation for Intelligent Physical Agents Home Page, http://www.fipa.org/, 2003

[23] T. Finin et al, "Specification of the KQML Agent Communication Language," Technical Report, DARPA Knowledge Sharing Initiative, External Interfaces Working Group, 1993

[24] W. E. Combs, "The Combs Method for Rapid Inference," in *The Fuzzy Systems Handbook* E. Cox, Ed., 2nd Edition, New York: AP Professional, 1998, pp. 659--680

[25] W. E. Combs and J. E. Andrews, "Combinatorial Rule Explosion Eliminated by a Fuzzy Rule Configuration," *IEEE Trans. Fuzzy Systems*, vol. 6, pp. 1—11, Feb. 1998

[26] J. M. Mendel and Q. Liang, "Comments on 'Combinatorial Rule Explosion Eliminated by a Fuzzy Rule Configuration'," IEEE Transactions on Fuzzy Systems, vol. 7, no. 3., June 1999.

[27] S. Dick and A. Kandel, "Comment on 'Combinatorial Rule Explosion Eliminated by a Fuzzy Rule Configuration'," IEEE Transactions on Fuzzy Systems, vol. 7, no. 4., Aug. 1999.

[28] Y. Li, P. Musílek and L. Kurgan, "Battlecity Revived: Game Design with BDI.net," Proceedings of the Fourth ASERC Workshop on Quantitative and Soft Computing Based Software Engineering, pp. 24-28, 2004

[29] N. J. Nillson, *Problem-Solving Methods in Artificial Intelligence*, McGraw-Hill, 1971