Fourth International Workshop on Computational Intelligence & Applications
IEEE SMC Hiroshima Chapter, Hiroshima University, Japan, December 10 & 11, 2008

Constitution of Ms.PacMan Player with Critical-Situation Learning Mechanism

Hisashi Handa
Graduate School of Natural Science and Technology
Okayama University, Okayama, 700-8530, Japan
email: handa@sdc.it.okayama-u.ac.jp

***Abstract*— We previously proposed evolutionary fuzzy systems of playing Ms.PacMan for the competitions. As a consequence of the evolution, reflective action rules such that PacMan tries to eat pills effectively until ghosts come close to PacMan are acquired. Such rules works well. However, sometimes it is too reflective so that PacMan go toward ghosts by herself in longer corridors. In this paper, a critical situation learning module is combined with the evolved fuzzy systems, i.e., reflective action module. The critical situation learning module is composed of Q-learning with CMAC. Location information of surrounding ghosts and the existence of power-pills are given to PacMan as state. This module punishes if PacMan is caught by ghosts. Therefore, this module learning which pairs of (state, action) cause her death. By using learnt Q-value, PacMan tries to survive much longer. Experimental results on Ms.PacMan elucidate the proposed method is promising since it can capture critical situations well. However, as a consequence of the large amount of memory required by CMAC, real time responses tend to be lost.**

I. Introduction

Recently, as benchmark problems to realize intelligent software, several game competitions have been carried out in the international conferences on Computational Intelligences, such as CIG2007, FUZZ-IEEE2007, CEC2007, and WCCI2008 [1]-[4]. The most distinguished points of these game competitions in comparison with conventional AI game studies are that 1) they are not limited to board games, 2) they often require computer programs to promptly response in the case of action games, and 3) available information on them occasionally includes noise.

The Ms.PacMan competition is one of such game competitions. PacMan is one of the most popular video games played in all over the world. Ms.PacMan, a mutant of PacMan, employs probabilistic transitions of ghosts. Thus, there is no deterministic (surefire) routes to solve for each stage in Ms.PacMan. That is, real time control mechanisms are needed to realize auto-play for Ms.PacMan by computers. In the previous study by us, evolutionary fuzzy systems for playing Ms.PacMan was proposed. As a consequence of the evolution, reflective action rules such that PacMan tries to eat pills effectively until ghosts come close to PacMan are acquired. Such rules works well. However, sometimes it is too reflective. In other words, the scopes where PacMan make attention are too narrow.

In this paper, a critical situation learning module is combined with the evolved fuzzy systems, i.e., reflective action module. The critical situation learning module is composed of Q-learning with CMAC. Location information of surrounding ghosts and the existence of power-pills are given to PacMan as state. This module punishes if PacMan is caught by ghosts. Therefore, this module learns which pairs of (state, action) cause her death. By using learnt Q-value, PacMan tries to survive much longer.

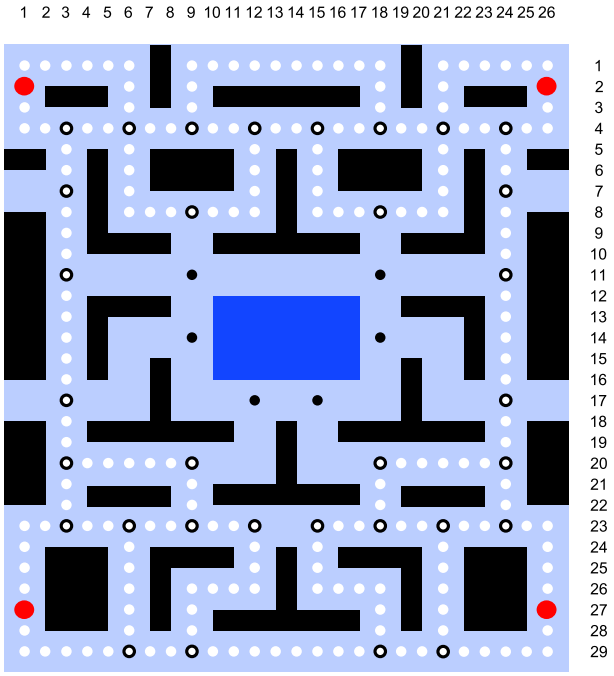

Fig. 1. A map of the maze of Ms.PacMan (Stages 1-2)

The rest of this paper is organized as follows: Section II introduces rules and game states of Ms.PacMan. Section III explains the previous study by us, i.e., reflective action module by Evolutionary Fuzzy Systems [12]. Section IV describes the critical situation learning module by using Q-learning with CMAC by referring to the definition of states and how to use Q-value in this module. Following these sections, experimental results are shown in section V. Finally, we conclude this paper in Section VI.

II. Ms.PacMan

A. Game rule

Fig. 1 depicts the maze for stages 1 and 2 in Ms.PacMan. Numbers above and right of the map mean coordinate values.

There are four warp zones in the side of this maze, where two horizontal warps are connected with each other, i.e., PacMan can go through from the upper right warp zone to the upper left one. Black circles indicate intersections. Smaller white and bigger red[1] circles denote pills and power-pills, respectively. Double circles denote that an intersection and a pill exist at corresponding location.

The task subjected to PacMan is to eat up all the pills and power-pills. The scores of eating a pill and a power-pill are 10 and 50 points, respectively. After eating a power-pill, the state of ghosts is changed into edible one during a certain period. The period varies from one second to 15 seconds, based on the number of stages. PacMan can eat ghosts in the edible state. The score of eating ghosts depends how many ghosts have been ate for a power-pill eating period: The score is doubled after eating other ghost, i.e., for the first eating, score is 200 points, and then, 400 points are given to Ms.PacMan for the second eating. 800 and 1,600 points are awarded for the third and fourth eating. The power-pill eating period, where ghosts are in edible state, is not so long that Ms.PacMan should take a risk to go to eat edible-stated ghosts. On the other hand, if she concentrates to eat pills only, she may be able to go the next stage easily while she get fewer points.

B. Preprocess of captured image

The image process of captured image is carried out at every cycle. As a consequence of the image process, we obtain the following information:

- Current location of PacMan,
- Location of and distance to each ghost,
- Location of and distance to each edible ghost,
- Position of the closest uneaten pill, and
- Distance to the nearest intersection from PacMan or Ghost,

where the metric for calculating distance, including the "closest" pill, does not indicate Euclid one. The distance used here is calculated by taking account into the structure of the maze: In order to calculate the distance on the maze, the distances between any two points are calculated by using Dijkstra method in advance.

Moreover, the prepossess can detect if the game play is end and if the PacMan is caught. Hence, this detection allows us to use learning methods since auto-replay can be done by java.awt.robot.

III. Reflective action module

A. Overview

Reflective action module was previously studied by us in [12]. In this module, fuzzy logic is used to decide the action of Ms.PacMan at each time step. The structure of fuzzy rules is predefined while parameters in the rules are evolved by (1+1) ES. As a result of evolution, this module acquires reflective action rules such that PacMan aggressively eats pills until ghosts come close to PacMan. This section briefly introduces it.

As depicted in Fig. 2, three types of rules are employed in the reflective action module: Avoidance, Chase, and Go-Through. These rules are examined in turn according to the priority as depicted in Fig. 2. The avoidance rules are examined for each ghost at first. An action is chosen if some rules are activated. In this case, when activated values of the avoidance rules are tie, tie-breaker is executed, where an action is randomly chosen among tied activated rules. Secondly, the chase rules are examined if no avoidance rule is activated. The way to examine the chase rules is the same as the avoidance one. Finally, eat rule is examined. The eat rule eat is not fuzzy rule. It is designed to eat closet pill. The following subsections explain avoidance and chase rules.

B. Avoidance

The rule description of "Avoidance" is:

IF a ghost IS close THEN PacMan go to avoiding directions.

This Fuzzy rule is examined for each ghost. Fig. 3 illustrates how this rule works: Suppose that PacMan is now at an intersection, and Blinky and Pinky are approaching to PacMan from upper side and lower side, respectively. For Blinky, the Fuzzy rule "Avoidance" infers that Ms.PacMan should go down or right. The Fuzzy rule "Avoidance" is also applied for Pinky so that it recommends that Ms.PacMan should go up or right. Therefore, as a consequence of these inference, Fuzzy Systems chooses the action "right."

The membership function for this rule can be illustrated as shown in Fig. 4. The x axis denotes the distance from a ghost to Ms.PacMan. The y axis is corresponding membership value. The parameters x_1^a and x_2^a are designed by using Evolutionary Computation in [12].

In the avoidance rule set, another rule set, i.e., "Go through", is also examined simultaneously. With only the fuzzy rule "Avoidance," PacMan tends not to pass through intersections if ghost is around there. Thus, a fuzzy rule "Go-Through" is defined as follows:

IF the distance of Ms.PacMan to the nearest intersection on the direction of movements is smaller than the distance

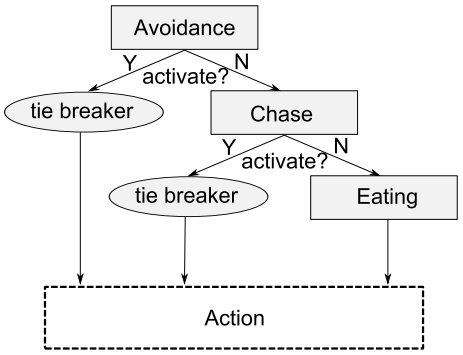

Fig. 2. A depiction of the priority of fuzzy rules

[1]Black and White prints, light gray circles

of a ghost to the intersection THEN Ms.PacMan go through the intersection.

Therefore, the membership function for the fuzzy rule "Go-Through" are defined in the Fig. 6. The x axis denotes α in Fig. 5, the difference of the distances between Ms.PacMan and the nearest intersection, and between a Ghost and the intersection. The y axis is corresponding membership value.

C. Chase

The rule description of "Chase" is: IF an edible ghost IS close THEN PacMan go to the ghost.

The mechanism of this fuzzy rule is similar to the one of "Avoidance." However, this rule is only applied to edible ghosts. This fuzzy rule recommends PacMan to go to the direction where near edible ghosts exist.

D. Problems in the reflective action module

As described in [12], evolutionary fuzzy systems works well. However, the resultant rules are reflective one so that sometimes it causes problems at tie-breaker. Fig. 7 explains the problem caused by the reflective action module. Suppose that PacMan is now chased by Blinky. In addition, Pinky is also approaching to PacMan, but it is too far to activate the avoidance rule. In this case, the tie-breaker decide an action either right or left. Obviously, the action right is wrong choice since Pinky will be able to catch up PacMan. In this paper, modified tie-breaker using Q-Learning with CMAC which learns critical situations is proposed. The next section explain it.

IV. Critical situation learning module

In order to settle problems described in subsection III-D, critical situation learning module is introduced in this paper. This module is a modified tie-breaker for avoidance rule sets, which receives broader information about surrounding ghosts and power-pills. The module learns critical situation causing PacMan's death by means of reinforcement learning. In the next subsection, reinforcement learning algorithms employed in this paper, i.e., Q-Learning with CMAC, is explained. Subsections IV-B and IV-C introduce the definition of states and how to use learn Q-value in this module, respectively.

A. Q-Learning with CMAC

CMACs have been widely known as function approximators for Reinforcement Learning Algorithms. They have been used for control problems with continuous inputs. They involve multiple overlapping tilings of the state space [2]. Suppose that there are two tilings for given state space. In CMACs, such tilings are overlapped but they are assigned different offsets as depicted in Fig. 8. Here, $c_i(s, a))$ denotes a cell index in tiling i for corresponding state-action pair

[2]Technically speaking, hash is employed to realize tilings in order to avoid "the curse of dimensionality." That is, all the cells are not prepared in memory.

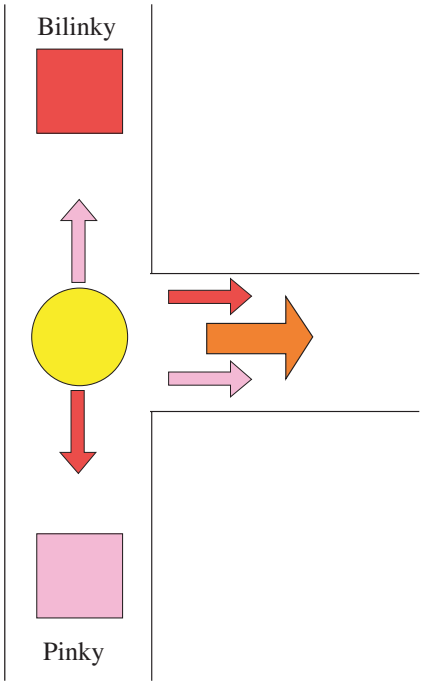

Fig. 3. A depiction of avoidance action

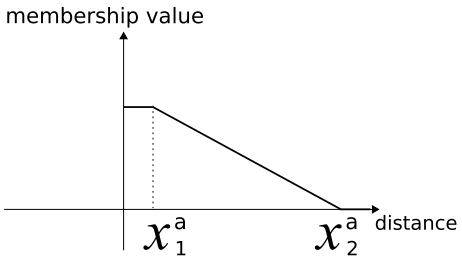

Fig. 4. The membership function of fuzzy rule "Avoidance"

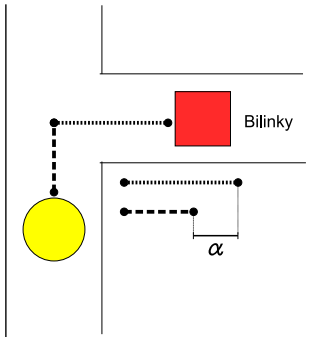

Fig. 5. A depiction of Go-Through action

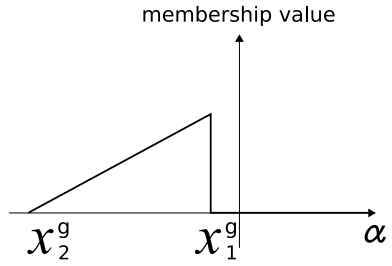

Fig. 6. The membership function of fuzzy rule "Go-Through"

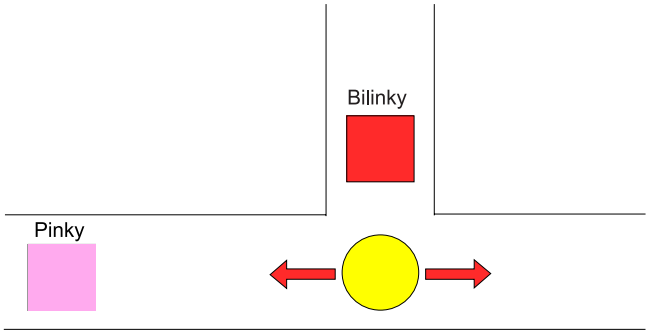

Fig. 7. An example of problems of the reflective action module

(s, a). For each cell, weight value $w(c_i(s, a))$ is associated. The state action value $Q(s, a)$ is calculated as follows:

$$Q(s, a) = \sum_{i}^{l} w(c_i(s, a)),$$

where l denotes the number of tilings.

Suppose that PacMan takes an action a_t in a state s_t at time step t, and then at time step $t + 1$, as a result of the action a_t, she recognize a new state s_{t+1}. The state action value $Q(s_t, a_t)$ is updated as follows:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\{r + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\},$$

where r indicates the amount of reward, α and γ denote the learning rate and the discount rate, respectively.

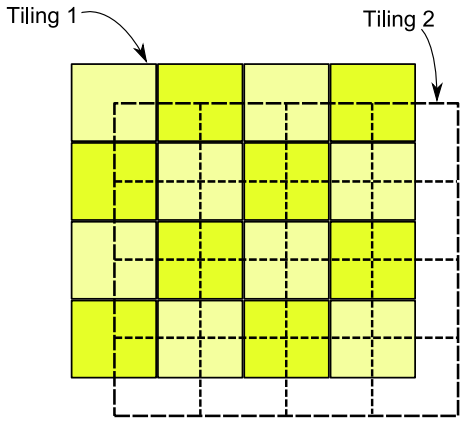

Fig. 8. A depiction of tilings in CMAC

B. Definition of States

The tilings of CMAC are individually prepared for each intersection. That is, states are individually defined at each intersection, which is denoted by black circles in Fig. 1. In other words, PacMan refers state-action values only at the intersections. In corridors, she employs the reflective action module to survive, which is mentioned in the previous section. The time step t in the previous subsection does not correspond to actual time. It is incremented whenever PacMan reaches one of any intersections.

At each intersection, PacMan perceives the position of ghosts, and the existence of the power pills. Such information is mapped into one of cell in CMAC. The size of each cell is set to be 6×6. The size of tiling is set to be 5×5 or 6×6, where it is varied with the offset of tiling. Three tilings whose offsets are set to be -2, 0, and 2 are used. In addition of this, "no show" cell such that corresponding ghost is either of in the base, in edible state, or out of scope, is added to each of tilings. The scope is defined for each intersection. As delineated in Fig. fig:stateInputs, the position is perceived to PacMan if the distance on map between a ghost and corresponding intersection is less than 16. For instance, two scopes for certain two intersections indicated by black circles are depicted in this figure: dashed (green) line and solid (red) line. In the case of intersection around the center of the map, the area surrounded by dashed line-segments means that PacMan perceive the position of any ghosts if they are in the area. Otherwise, the "no show" cell is activated. Therefore, all the cells in the tiling are not always used: In the case of dashed lines, a fifth cells are not be used. On the other hand, in the case of solid lines, more than two third cells are not used.

The key of hash is calculated as follows: c_l^b, c_l^p, c_l^i, and c_l^s denote cell ID for each ghost in the case of tiling l. Note that the "no show" cell is also assigned an ID. pp takes 0 or 1, which indicates if power pill is in the scope. The key of hash k_l for the tiling l is defined as

$$k_l = (((c_l^b n_{ID} + c_l^p) n_{ID} + c_l^i) n_{ID} + c_l^s) \cdot 2 + pp,$$

where n_{ID} denotes the number of IDs in the tilings, including the "no show" cell. The hash is prepared for each tiling, and returns a vector whose element corresponds the state-action value for each action.

C. Utilization of Q-value

All the Q-value is initialized to 0. Only negative reward, i.e., punishment, is given to PacMan if she died. Thus, as a consequence of learning, negative Q-values mean that corresponding action may cause to her death.

Therefore, at tie-breaking for the same activation values in avoidance rule sets, the action which has higher Q-Value is chosen. If there are actions with the same activation values and the same Q-values, then an action is randomly chosen among such actions.

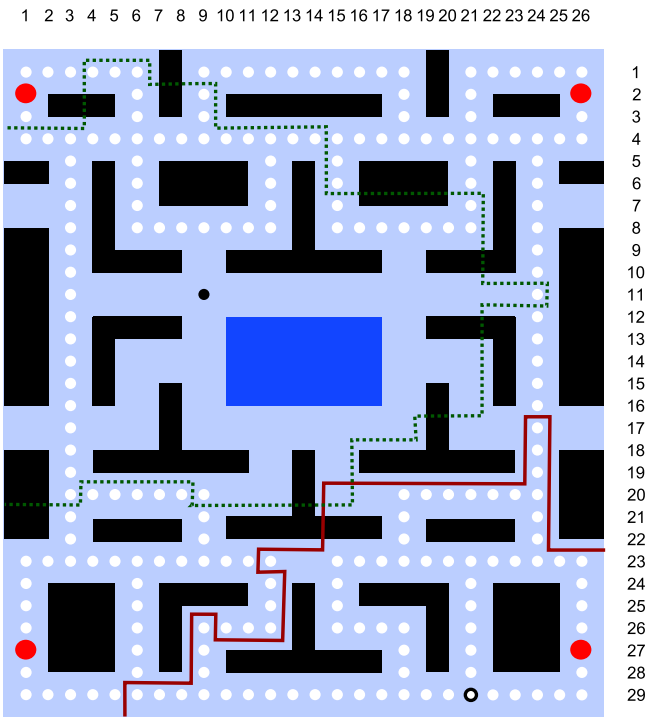

Fig. 9. Input areas for CMAC

V. Experimental Results

The proposed method, i.e., critical situation learning mechanism by using Q-learning with CMAC, is examined on Ms.PacMan game. The reward (punishment) is set to be -10. This is given to PacMan if she died. The learning parameters for Q-learning α and γ are set to 0.3 and 0.9, respectively. 10 runs are examined in experiments. Each run is composed of 200 games.

Figs. 10 and 11 show the experimental results on the changes of the moving average of scores and the number of loops, respectively. The number of loops means how long PacMan was able to survive. The plotted line is moving average over last 20 games. Moreover, these lines are averaged over 10 runs. These graphs denotes experiments during learning period of Q-Learning with CMAC. In this paper, the reflective action module is not evolved. We employ the result in the previous work in [12]. The horizontal line in each graph denotes the corresponding performance of the reflective action module only. Score is improved during learning period by using the critical situation learning module. Sometime, it is worse than the reflective action module only. The number of loops has a greater tendency to be worse than original one than score. One reason of this is that the score and the number of loops in the case of Ms.PacMan are significantly affected by the movements of ghosts. The proposed method tries to memorize past critical situations so that there is no way to infer for unseen situations.

Fig. 12 shows the number of opportunities that critical situation learning mechanism affects their actions. That is, this affection means that 1) there are a few actions suggested

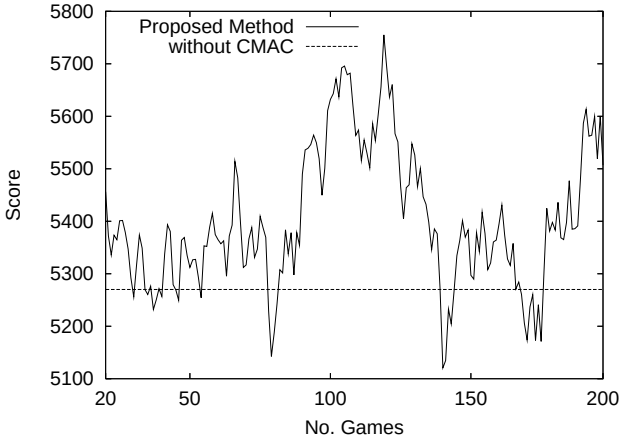

Fig. 10. The changes of the moving average of scores. The plotted line is averaged value over 10 runs

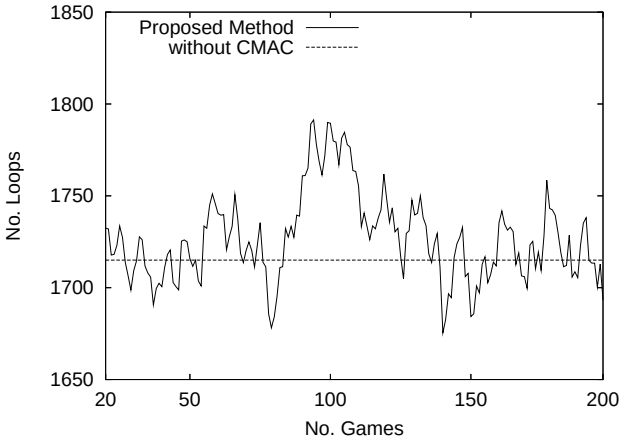

Fig. 11. The changes of the moving average of the number of loops. The plotted line is averaged value over 10 runs

by the avoidance rules in the reflective action module, and 2) by referring to Q-values , critical actions are not chosen. The leftest point in the graph indicate the average of the first 20 games. Remarkable point here is that even if the first 20 runs, critical situation is detected by the proposed method less than once but close to once (around 0.9). Another point is that, around 90 games, the line in the graph is raised. At the same time, the scores and the number of opportunities in Figs. 10 and 11 also shows in rising trend.

Table I shows the performances by initial PacMan, i.e., the reflective action module only, and by the proposed method at 50, 100, 150, and 200 games. These results are acquired by additional runs of these separate from the runs during learning period. These results are averaged one over 50 games. Unfortunately, these data shows no statistical significant difference due to variances of these data are quite big. Such performance strongly depend on the movements of ghosts. However, these data shows similar tendency to graphs in Figs. 10, 11, and 12, except for that the number of games is 150. Another reason why the proposed methods at 200 games

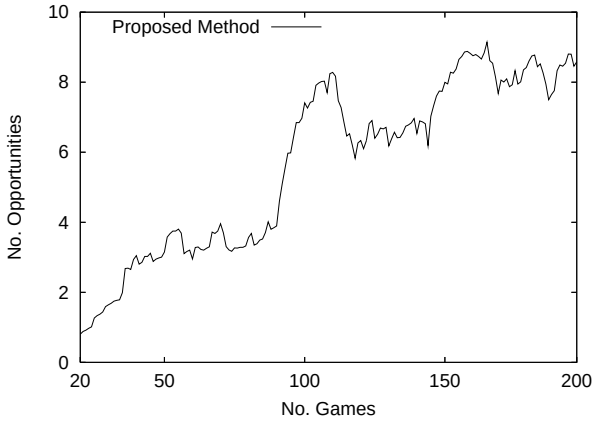

Fig. 12. The changes of the moving average of the number of opportunities that critical situation learning mechanism affects actions. The plotted line is averaged value over 10 runs

TABLE I
THE PERFORMANCE BY INITIAL PACMAN AND BY THE PROPOSED METHOD AT 50, 100, 150, AND 200 GAMES

No. Games	initial	50	100	150	200
Scores	5270	5739	5533	5515	5037
Loops	1715	1780	1731	1838	1648
Opportunities	n/a	1.8	7.6	7.7	8.8

and at 150 games during learning period are not so good is due to the search time of the hash in CMAC. That is, the search time is too much so that it causes the delay to control PacMan. Although we cannot show well such phenomena as data, we were able to recognize such delay by our eyes. Table II summarizes the total file size of weight data for CMAC. As you can see, as increasing the number of games, the total file size is also growing. In the case of Table I, Q-Learning with CMAC is not carried out. Thus, CMAC is only used for the reference of Q-values. On the other hand, during learning period, CMAC is used not only for the reference but also the update of Q-values. Hence, the number of accesses to the hash in CMAC during learning period is at least double in the case of Table tab:offlinePerformance.

VI. CONCLUSIONS

In this paper, a critical situation learning mechanism by using Q-learning with CMAC was proposed. This mechanism learns which situation and action pairs cause to be caught by ghosts by using reinforcement learning scheme. This approach is promising in the sense that the proposed method can capture critical situations well. However, as a consequence of the large amount of memory required by CMAC, real time responses tend to be lost. This problem must be settled.

TABLE II
THE TOTAL FILE SIZE OF WEIGHTS IN CMAC

No. Games	50	100	150	200
Total File Size (KB)	216.5	360.0	494.3	635.6

One of solution for this could be a combination use of the proposed method with other function approximator, such as neural networks, support vector machines, Fuzzy Logic and so on. The reason why we do not want to discard Q-learning with CMAC is that the approach is quite effective way for reinforcement learning (Sutton does not recommend reinforcement learning with neural networks in FAQ of his book). Moreover, reinforcement learning is needed to estimate which situations are critical one. Hence, possible combination method could be a two-phase method: In the first phase, learning data is collected, and then, Q-learning with CMAC is used to estimate off-line which situation is critical. In the second phase, knowledge transfer from CMAC to other function approximators is carried out. Weight values in hash can be used as training instances for such function approximators. Go back to the first phase with learnt function approximator. Such phases are iterated until PacMan learns critical situations well.

REFERENCES

[1] CIG2007 Competitions home page, http://csapps.essex.ac.uk/cig/2007/index.jsp?page=competitions.html
[2] FUZZ-IEEE2007 Competitions home page, http://cswww.essex.ac.uk/staff/lucas/fuzzieee/FuzzyRace.html
[3] CEC2007 Competitions home page, http://cswww.essex.ac.uk/staff/sml/cec2007/competitions.html
[4] WCCI2008 Competitions home page, http://www.wcci2008.org/competitions.htm
[5] Simon M. Lucas, Evolving a Neural Network Location Evaluator to Play Ms. Pac-Man, Proceedings of IEEE Symposium on Computational Intelligence and Games (2005) pp. 203–210
[6] Simon M. Lucas and Graham Kendall, Evolutionary Computation and Games, IEEE Computational Intelligence Magazine (2006) Vol. 1, pp. 10–18
[7] Julian Togelius, Simon M. Lucas, and Renzo de Nardi, Computational Intelligence in Racing Games, Advanced Intelligent Paradigms in Computer Games, Norio Baba, Lakhmi C. Jain, Hisashi Handa editors, Springer Series on Studies in Computational Intelligence (2007) Vol. 71, pp. 39–69
[8] The Handbook of Fuzzy Systems and Soft Computing, Kyoritsu (1999, in Japanese)
[9] Evolutionary Computation 1: Basic Algorithms and Operators, Thomas Baeck, D.B Fogel, Z Michalewicz Editors, IOP press (2000)
[10] Marcus Gallagher and Amanda Ryan, Learning to Play Pac-Man: An Evolutionary, Rule-based Approach, Proceedings of the 20003 IEEE Congress on Evolutionary Computation (CEC), (2003) pp 2462–2469
[11] Marcus Gallagher and Mark Ledwich, Evolving Pac-Man Players: Can We Learn from Raw Input?, Proceedings of the 2007 IEEE Symposium on Computational Intelligence and Games (2007) pp. 282–287
[12] Hisashi Handa and Maiko Isozaki, Evolutionary Fuzzy Systems for Generating Better Ms.PacMan Players, Proceedings of the 2008 International Conference on Fuzzy Systems (FUZZ-IEEE08 in WCCI08, 2008) pp. 2182-2185
[13] I. Szita and A. Lorincz, 'Learning to Play Using Low-Complexity Rule-Based Policies: Illustrations through Ms. Pac-Man, Vol. 30 (2007) pp. 659-684
[14] Y. Hirashima, Selective Generalization of CMAC for Q-Learning and its Application to Layout Planning of Chemical Plants, Proceedings of 2007 International Conference on Machine Learning and Cybernetics, Vol. 4 (2007) pp. 2071-2076
[15] R.S. Sutton, Generalization in reinforcement learning: Successful examples using sparse coarse coding, Advances in Neural Information Processing Systems 8 (1996) pp. 1038–1044
[16] Reinforcement Learning FAQ, http://www.cs.ualberta.ca/ sutton/RL-FAQ.html