Pac-mAnt: Optimization Based on Ant Colonies Applied to Developing an Agent for Ms. Pac-Man

Martin Emilio, Martinez Moises, Recio Gustavo, Saez Yago *Member, IEEE*

Abstract— **This paper proposes the use of an optimization algorithm based on ant colonies for the development of competitive agents in the game environment in real time, specifically for the Ms. Pac-Man video game. Furthermore, a genetic algorithm is implemented to optimize the parameters of the artificial ants. The best agent obtained through experimentation will be sent to the competition of Ms. Pac-Man[1] organized by the IEEE, framed within the Computational Intelligence and Games 2010 (CIG2010).**

I. Introduction

Traditionally, games and video games have provided a framework for the study of Artificial Intelligence and Machine Learning. Evolutionary computation techniques can sometimes develop very competitive agents; in some cases, they can overcome agents hand-coded by human [17].

One of the most popular video games is Pac-Man. It was developed by Toru Iwatani in 1979 for the company Namco and quickly achieved worldwide success. In this game, the player controls Pac-Man through a maze. Throughout the maze are distributed a number of pills that Pac-Man must "eat" and that are worth 10 points each. To add difficulty to the game, there are four ghosts chasing Pac-Man. When a ghost meets Pac-Man in the same position, we say that Pac-Man was caught, and Pac-Man loses a life. The player starts the game with three lives and gets an extra life after reaching 10,000 points. There are also 4 pills, known as *"power pills,"* that are placed in each of the corners of the maze. Each of these pills is worth 50 points and allows Pac-Man to capture ghosts for a short period of time, which decreases as the player advances through the successive levels. For the first ghost captured, the player receives 200 points, and this number doubles with each additional ghost, so if the player manages to capture the four ghosts in a row, he or she will earn a total of 3,000 points (200 + 400 +800 +1600).

In 1981, Ms. Pac-Man was developed (see Fig. 1) as a successor to the original game. Both games share the performance characteristics discussed earlier, and the main difference between them lies in the behavior of the ghosts. In the original version, the movements of the ghosts are deterministic, which means that if a player repeats his movements over several games, the movements of the ghosts will also repeat. This determinism makes it possible to develop agents that learn optimal routes, as outlined in the work done in [9]. In Ms. Pac-Man, the movement pattern of the ghosts has a pseudo-random component, which prevents the learning of optimal routes and increases the difficulty of developing an agent. In addition, Ms. Pac-Man includes some additional mazes not included in the original version of Pac-Man.

The aim of this work is to verify whether the optimization method based on ant colonies can be applied to the development of a competitive agent in the environment of real-time video games, focusing the tests on the Ms. Pac-Man game. For testing purposes, the best agent obtained will be sent to the Ms. Pac-Man competition[1] organized by the IEEE framed within Computational Intelligence and Games 2010 (CIG2010).

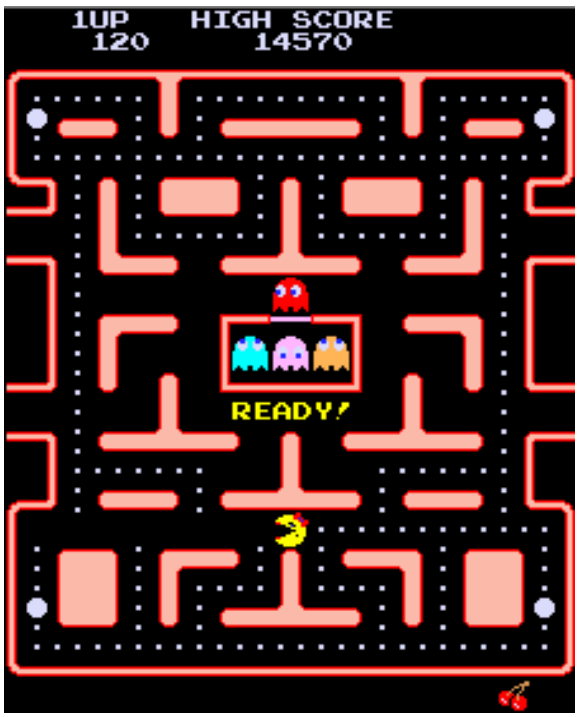

Fig 1. Screenshot from Ms. Pac-Man video game

The paper is organized as follows: Section II describes the various approaches proposed in other works for the development of agents both for Pac-Man and for Ms. Pac-Man.

[1]http://cswww.essex.ac.uk/staff/sml/pacman/PacManContest.html

978-1-4244-6297-1/10/$26.00 ©2010 IEEE

Section III describes the software and settings used in the experiment. Section IV shows the design of agent-based optimization in ant colonies, and Section V presents the parameters of the artificial ants and their optimization using genetic algorithms. Finally, in Sections VI and VII, we present the results and conclusions of the findings.

II. Previous Works

Different research works on Pac-Man have used several versions of the game (original version, Ms. Pac-Man and custom simulators), which complicates the problem of comparing the different proposals. The works in this field can be classified into two main groups of techniques: those that make use of Artificial Intelligence, in whole or in part, and those techniques that implement hand-coded agents. Overall, the second approach is based on the application of rule sets and produces better results than the solutions given by the first group. The following is a summary of the most salient examples of each approach.

A. Approaches based on Artificial Intelligence techniques

One of the early works on Pac-Man was developed by Koza in [8], in which he presents results on the prioritization of tasks using Genetic Programming.

Remarkably, the author uses his own implementation of the game, and one of his most significant adjustments emphasizes the behavior of the ghosts. In the author's implementation the four ghosts have the same behavior, while in the real game, each one has its own pattern of movement. According to the authors Szita and Lorincz [6], the results published by Koza would be equivalent to approximately 5,000 points in his version of the game.

Bonet and Stauffer proposed reinforcement learning techniques in [7]. Using a network of neurons and temporal differences, they construct agents with basic policies of flight and pill collection. It should be emphasized that they use a very simplified version of Pac-Man, (there is only on ghost in the maze and there are no "power pills").

Using a version of the game similar to that of Bonet and Stauffer, Gallagher and Ryan proposed in [10] the design of an agent based on a finite state machine with a given set of rules. These rules contain a series of weights, which were optimized using an incremental learning algorithm based on populations (PBIL) [15]. With this approach a certain degree of learning was achieved, however the representation used has some drawbacks.

Lucas [16], proposed as the control algorithm a network of neurons. The network takes as input a vector of hand-coded features, most of which are distances between Pac-Man and relevant aspects of the game (ghost, nearest pill, closest intersection). This proposal achieved average results of 4,780 ± 116 points over 100 games. It is important to note this experiment also used a custom simulator, which, although it contains some differences, is a fairly reasonable approximation of the original game.

Recently, the author proposed another approach based on search trees, as detailed in [2]. This approach limits the depth of the search trees to a maximum of 40, evaluating each one of the possible paths with a heuristic function. With this approach, the maximum score of 40,000 points was achieved (using the simulator). In the original version of Ms. Pac-Man, obtaining information from the game via screenshots, the method obtained 9,630 ± 346 points.

The authors Szita and Lorincz [6] propose a different approach, namely to construct strategies based on rules. These rules are organized into modules, which have different priorities. The agent decides where to move according to the rules of the modules and their priority. To construct the different strategies, they make use of a crossed entropy maze (CEM). The results are comparable, on average, to those obtained by a set of five non-expert humans playing the same version of the game; the best result they obtained was 8.168 points.

Wirth and Gallagher in [13] propose to use influence maps. The proposed model to construct the maps is simple and uses three parameters, reflecting the most important features of the game. During the experiment these parameters are optimized, reaching an average of 6,848 points and a high of 19,490 points (in one run). However, in the Ms. Pac-Man competition held in the IEEE WCCI 2008 [12], their method only got 2,510 points on average and a maximum of 4,360. This difference in score is due to the system's lack of precise information on the state of the game because it is obtained through screenshots.

B. Hand coded strategies

The winner of the IEEE WCCI 2008 was the agent RAMP [1]. This agent is based on rules and has a set of nine conditions. It has four possible actions: eating nearest pill, eating ghost, running away from ghost and going toward "power pill." The mean score in the competition was 11,166 points and the maximum of 15,970 points.

In the IEEE CEC 2009, the winning agent was ICE Pambush 2 [14], with a maximum score of 24,640 and an average of 13,059 points. This agent uses a set of seven decision rules. Also, two versions of the algorithm A* were used to find the minimum cost paths between Ms. Pac-Man and different target positions. In the IEEE CIG 2009, the same authors, using a modified version of ICE Pambush 2 (ICE Pambush 3), had a mean score of 17,102 and a maximum of 30,010, this being the highest ranking score achieved to date by an agent in Ms. Pac-Man.

Although the results obtained by these type of techniques have been very good, their fundamental problem is that they are not adaptive techniques, as they are developed "ad-hoc" for the Ms. Pac-Man video game and would not work for other variants.

III. CONFIGURATION AND SOFTWARE

To achieve the goals set out in Section I, it was necessary to construct some software components, some of them fully implemented from scratch, and others modified. During the experimental stage, both a simulator and an original version of the arcade game "Microsoft Revenge of Arcade" were used. The work can be divided into the following sub-tasks:

1) Implementation of a Ms. Pac-Man simulator that mimics the behavior of the original game.
2) Creation of an agent using an algorithm based on ant colonies to control Ms. Pac-Man.
3) Development of software to extract information about the state of the game from screenshots that enables the developed agent to control the agent in the real game.
4) Optimization of parameters for this algorithm using a genetic algorithm.

To perform the sub-tasks 1 and 3, a development kit made by Jonas Flensbak and Georgios N. Yannakakis for C# was used, which is a development kit for the Ms. Pac-Man competition [1]. Some modifications were made to this kit:

1) Modification of the simulator to be called by a genetic algorithm.
2) Modification of the simulator so that the speed of Pac-Man is decremented each time you eat a pill.
3) Changing the display to show certain characteristics of our algorithm, such is the level of pheromone (see Fig. 2).

The sub-tasks 2 and 4 were implemented in the same language from scratch; the implementation details of both are found in Sections IV and V.

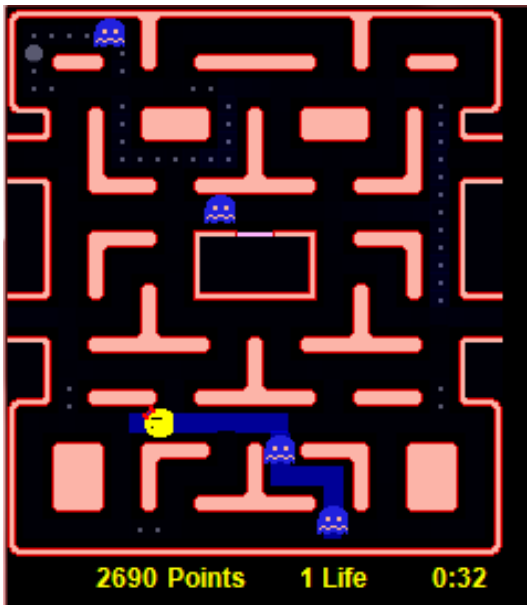

Fig 2. Screenshot from simulator showing the level pheromone for the best collector ant.

IV. AGENT DESIGN

Optimization algorithms inspired by ant colonies (Ant Colony Optimization, ACO) have been successfully applied to a wide variety of combinatorial optimization problems. One of the first issues discussed was the traveling salesman problem [3]. They have also been applied to routing problems in networks, both in fixed [4] and mobile [17] networks. All these problems can be formalized by graphs, where the objective is to minimize the cost of a route. In all cases the source and destination are known, while the costs may or may not change over time.

The problem of designing an agent for Ms. Pac-Man, shares certain characteristics with the problems raised in [3] [4] and [17]; however, it has some distinct characteristics that are worth mentioning:

1) The destination is not clearly-defined, unless this is set ad-hoc depending on the state of the game.
2) The costs of the nodes vary over time due to the pseudorandom motion of the ghosts.

Due to the particular nature of the problem mentioned above, the algorithms proposed in [3] [4] and [5] were not considered feasible, however, as they share many common features with the proposed agent, they were initially used as starting point.

Because the object is to get the highest score possible, in each time step the agent must decide what direction to take based on the state of the game, with the objective of maximizing the score while avoiding capture by the ghosts (security). As noted in Section 1, points are scored by eating pills and capturing ghosts. However, this property is not sufficient to determine a competitive agent, as it is also necessary to have policies to escape from the ghosts and find safe routes to transit. Both features are necessary and not exclusive. For example, when fleeing from the ghosts, if two different paths contain pills, you should choose the one that is more secure.

Following the previous approach, our agent uses an algorithm based on ant colonies. A different type of ant is used for each of the above objectives: one to find paths with points, where pills or ghosts can be captured, and another to find safe routes. The first ones will be called the *collector ants*, and the second ones *explorers*.

Because there is limited computing time in which to select the direction in which the agent moves, we should limit the maximum distance up to which an ant can explore. In addition, the algorithm introduces the dead ant metaphor, whose conditions are explained in the following paragraphs. The operation of the designed agent is shown in Algorithm 1,

```
ArrPosAdy ← adjacent positions.
for i to maxAnts do
   ArrAnts←sendAnts(ArrPosAdy)
end
updateGlobalPheromone (BestAnt)
directionNext←SelectDirection
```

Algorithm 1.Behavior of the agent

At every iteration, both types of ants are launched from all adjacent positions to the current position of Pac-Man. Selecting the next movement of the agent is done by a very simple decision rule: if the distance to a ghost is less than a certain distance (min_dist), the agent will choose the direction of the best explorer ant (the one with the biggest amount of pheromone), that means the agent will follow a safe path; otherwise, the agent will choose the direction of the best collector ant, going towards the path with maximum score. The behavior of each individual ant explained in Algorithm 2,

```
while no condition-stop do
    Select next node
    Record node information
    Record node as visited
    updateLocalPheromone
end
```

Algorithm 2. Behavior of each ant

As can be seen in Algorithm 2, an ant visits a node until it meets the halting condition. Each time a node is visited by k ants, it becomes part of the set of nodes S_K. The next node to visit is selected using a pseudo-random proportional rule, controlled by the parameters q_0^c and q_0^a, similar to that described in [11] and displayed in Equation(1) and Equation(2).

$$s^x = \begin{cases} \arg\max_{j \in N_k(i)} \left\{ \left[\tau^x(i,j)\right] \cdot \left[\varepsilon^x(i,j)\right]^\beta \right\} & \text{if } q > q_0^x \\ S & \text{if } q \le q_0^x \end{cases} \quad (1)$$

$$\mathrm{S} = \begin{cases} \dfrac{\left[\tau^x(i,j)\right] \cdot \left[\varepsilon^x(i,j)\right]^\beta}{\sum_{u \in N_k(i)} \left[\tau^x(i,u)\right] \cdot \left[\varepsilon^x(i,u)\right]^\beta} & \text{if } j \in N_k(i) \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

where x indicates the type of ant, $\tau^x(i,j)$ pheromone value in the node i, j, $\left[\varepsilon^x(i,j)\right]^\beta$ is the desirability of node i, j, $N_k(i)$ is the set of nodes reachable by ant k from node i, and q is a random number uniformly distributed in [0,1] and q_0^x corresponds to the parameters q_0^c y q_0^a.

The pheromone local update process is given by Equation (3),

$$\tau^x(i,j) \leftarrow (1-\rho_0^x) \cdot \tau^x(i,j) + \rho_0^x \cdot \tau_0^x \quad (3)$$

where τ_0^x is the minimum value of pheromone and ρ_0^x is the rate of pheromone dissipation. The pheromone global update process is given by Equation (4),

$$\tau^x(i,j) \leftarrow (1-\alpha_0^x) \cdot \tau^x(i,j) + \alpha_0^x \cdot \Delta\tau^x(i,j) \quad (4)$$

where α_0^x is the rate of pheromone deposition, and $\Delta\tau^x(i,j)$ is given by Equation (5),

$$\Delta\tau^x(i,j) = \begin{cases} Q_{ib}^x \text{ if (i,j)} \in S_k \wedge k \text{ is best iteration ant} \\ 0 \quad \text{otherwise} \end{cases} \quad (5)$$

where Q_{ib}^x is the quality of the solution found by best iteration ant, Q_k^x is specific to each type of ant, as specified in the following sections.

1) Collector ants.

This type of ants is aimed at collecting points. The halting condition is satisfied in the following four cases:

1) The maximum distance is reached.
2) The ant is in a loop. All nodes adjacent to the current node have already been visited.
3) Dead ant. This type of ant is "dead" when it has reached its maximum distance and has not scored any points.

The calculation of the heuristic function for a node i,j is given by Equation (6),

$$\eta_{ij}^c = \begin{cases} \dfrac{1}{pill_{act}/pill_{tot}} & if\ node_{ij} = Pill \\ \dfrac{\chi}{pill_{act}/pill_{tot}} & if\ node_{ij} = Edible\ ghost \end{cases} \quad (6)$$

where χ are a system parameter, $pill_{tot}$ and $pill_{act}$ are the total number of pills at the beginning of the map and current number of pills. As can be seen in Equation (6), the heuristic will have a nonzero value if the node contains a pill or contains a ghost that can be caught. Similarly the quality of a solution found by k collector ant, Q_k^c is calculated according to Equation (7),

$$Q_k^c = \sum \frac{\eta_{ij}^c}{d(P,n_{ij})} + \sigma \cdot \tau^e(i,j) \quad (i,j) \in S_K \quad (7)$$

where $d(P,n_{ij})$ is the distance between Pac-Man and the node i,j, σ is a system parameter and $\tau^e(i,j)$ is the value of the pheromone (explorer ants) for the node i, j. The quality of solution found by collector ant is influenced by the level of explorer ant pheromone.

2) Explorer ants.

The role of this type of ant is to find safe paths and prevent Ms. Pac-Man from being caught. As for the

collecting ants, the halting condition is given by the following conditions:

1) The maximum distance is reached. In this case, the ant cannot stop at just any node, but it must stop at a node with connectivity greater than or equal to three. The goal of this condition is to try to avoid paths being marked as safe alleys.
2) The ant is in a loop. That is, adjacent nodes to the current one have been visited.
3) Dead ant. This type of ant is "dead" when there is a ghost who can get to the current node before Ms. Pac-Man can, if Ms. Pac-Man has not previously passed some node in which there is a "power pill".

The calculation of the heuristic function for a nod takes into account the distance from the nearest ghost to that node, as shown in Equation (8),

$$\eta_{ij}^{e} = \frac{d(G, n_{ij})}{d(P, n_{ij})^2} \tag{8}$$

where $d(G, n_{ij})$ is the distance of the closest ghost to node i,j. Similarly, the quality of a solution found by k explorer ant, Q_k^e is calculated according to Equation (9),

$$Q_k^e = \sum \eta_{ij}^e + \omega \cdot \tau^c(i,j) \quad (i,j) \in S_K \tag{9}$$

where ω is a system parameter and $\tau^c(i,j)$ the value of pheromone (collector ants) for the node i, j.

The halting conditions set out above, together with the restrictive conditions to consider a *dead* ant, determine that in certain situations (see Fig. 3) there is no safe way to traverse the desired distance, meaning that all the explorer ants launched to a maximum distance are *dead*. In such a situation, that ant's pheromone only updates the value of whichever $|S_K|$ is greater. For example, in Fig.3, the algorithm would choose the path A.

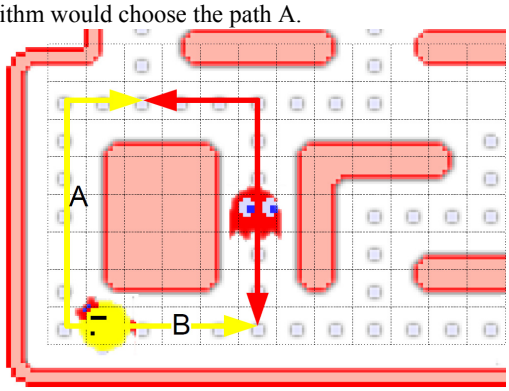

Fig. 3. Game location without a safe path for Ms. Pac-Man.

As for the collector ants, the maximum distance of the explorer ants is modified dynamically throughout the game.

V. Parameter Optimization.

To optimize the parameters of the artificial ants, we use a genetic algorithm (GA), where the system parameters to optimize are listed in Table I.

TABLE I
SYSTEM PARAMETERS

Parameter	Range	Precision
maxAnts	[5-30]	1
min_dist	[5-20]	1
χ	[5-50]	1
p_0^c	[0-1]	0.01
p_0^e	[0-1]	0.01
α_0^c	[0-1]	0.01
α_0^e	[0-1]	0.01
σ	[0-1]	0.01
ω	[0-1]	0.01
q_0^c	[0-1]	0.01
q_0^e	[0-1]	0.01

The evaluation function is calculated based on the mean score for each individual over the total games played. The total number of individuals in the population is 50; the selection operator is based on a tournament technique with tournament size 3, and the type of crossover implemented is multipoint and positional.

VI. Results

In order to validate the proposed method, several experiments were performed on both, the original version of the arcade game "Microsoft Revenge of Arcade". In both cases, a simple GA was used to find the optimal operating parameters for different levels of the game.

A. Simulation environment

For these experiments, we have used the game simulator discussed in Section 3. Some significant changes have been made to the original version of the game: the initial speed of the ghosts is matched to that of Ms. Pac-Man, and the maze changes on a cyclical basis every time you complete a level (increasing the speed of the ghosts at the same time.) That way, the speed of the ghosts is not reduced by moving up to the next level. To produce the parameters explained in Section 5, it was necessary to use a simple AG, with 50 individuals iterated for 100 generations (see Fig. 4).

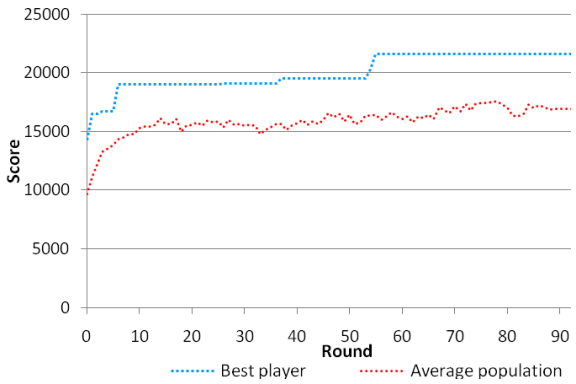

Fig. 4. Evolution of maximum and average scores of the population over 100 rounds of execution.

As shown in the figure above, there is much variability between the results of on round and the next one. As explained in Section 1, to capture the four ghosts consecutively is worth a total of 3,000 points because there are four "power pills" and a total of 220 pills. For one level, the maximum possible score is 14,200 points, of which 84.5% corresponds to capturing ghosts. This, together with the non-deterministic behavior of the ghosts, causes the evaluation function, to contain noise, so it is not the most appropriate to guide the evolution of the GA. A possible alternative to reduce this noise is to increase the number of assessments used to obtain the mean score for the first level of the game.

B. *Real environment*

For the experiments performed over the actual game, the best parameters obtained through simulations have been proved in a real environment. The result of 20 games can be seen in Fig. 6.

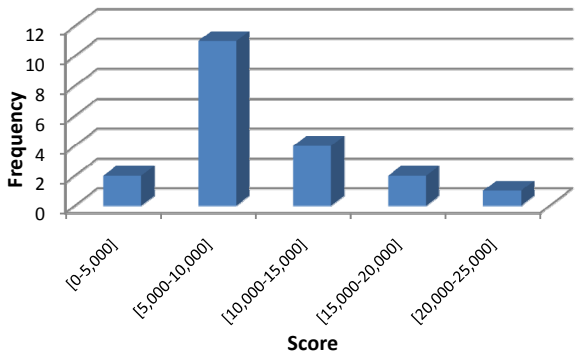

Fig. 6. Results of 20 games in original game.

The mean score was 10,120 points, and the best individual score achieved in a single game was 20,850 points. The lack of accuracy in the process used to capture the state of the game makes the results presented in fig 6 differ from those of the simulations. Due to the high computational cost required in real environments, using a GA was not practical. This issue will be tackled in future works.

VII. Conclusions

We have presented a new approach to designing videogame agents based on optimization algorithms based on ant colonies. This approach has been tested in the game of Ms. Pac-Man.

The parameters of the artificial ants were optimized using a GA, and in spite of having a noisy evaluation function, it produced promising results compared with those obtained by other agents [12].

The system parameters were optimized for all levels of the game; given the changing conditions on the original videogame, a possible improvement would be to optimize them for each level individually. Another goal for future work is to increase the number of generations tested in the real environment. Based on the results obtained so far, we can say that the implementation of systems based on ant colonies for the creation of intelligent game controllers is worth further research.

Further improvements on the performance of the agent for the Ms. Pac-Man competition could be achieved by applying elitism to the process of global pheromone update, so that the updating would be done only by the best global ant instead of the best ant of each iteration. And also by including some rules allowing the agent to eat a power pill only if there is a ghost nearby.

Acknowledgements

This research has been co-financed by the project Mstar TIN2008-06491-C04-04 and the project eInkPlusPlus TSI-020110-2009-137.

References

[1] A. Fitzgerald, P. Kemeraitis, and C. B. Congdon, “Ramp: A rulebased agent for ms. pac-man,” IEEE Congress on Evolutionary Computation, pp. 2646-2653, 2009.

[2] David Robles and Simon M. Lucas, Senior Member, IEEE, “A Simple Tree Search Method for Playing Ms. Pac-Man” IEEE Symposium on Computational Intelligence and Games, pp. 249-255, 2009.

[3] Dorigo,M. “Optimization, Learning and Natural Algorithms”. Tesis doctoral. Dipartamento di Elettronica, Politecnico di Milano, 1992.

[4] G. Di Caro and M. Dorigo “AntNet: Distributed stigmergetic control for communications networks”. Journal of Artificial Intelligence Research, vol. 9, pp. 317–365, 1998.

[5] G. Di Caro, F. Ducatelle “AntHocNet: An adaptive nature-inspired algorithm for routing in mobile ad hoc networks”. European Transactions on Telecommunications, vol. 16, pp. 443-455, 2005.

[6] I. Szita and A. Lorincz, “Learning to play using low-complexity rulebased policies,” Journal of Artificial Intelligence Research, vol. 30, no. 1, pp. 659–684, 2007.

[7] J. S. De Bonet and C. P. Stauffer. “Learning to play pacman using incremental reinforcement learning”. Disponible en http://www.debonet.com/Research/Learning/PPacMa/, (accessed 12/04/2010).

[8] John R. Koza, “Genetic programming: on the programming of computers by means of natural selection”, MIT Press, Cambridge, MA, 1992.

[9] K. Uston, “Mastering Pac-Man”, Macdonald and Co, 1981.

[10] M. Gallagher, A. Ryan, “Learning to play Pac-Man: An evolutionary, Rule-based approach,” in IEEE Symposium on Computational Intelligence and Games, pp. 2462–2469, 2003.
[11] M. Dorigo, Gambardella L. “Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem”, IEEE Transactions on Evolutionary Computation, vol. 1, pp. 53-66, 1997.
[12] Ms.Pac-Man competition IEEE WCCI 2008 Results.http://cswww.essex.ac.uk/staff/sml/pacman/WCCI2008Results.html (accessed 12/ 04/ 2010).
[13] N. Wirth and M. Gallagher, “An influence map model for playing Ms. Pac-Man,” IEEE Symposium on Computational Intelligence and Games, pp. 228-233,2008.
[14] R. T. Hiroshi Matsumoto, Chota Tokuyama, “Ice Pambush2” http://cswww.essex.ac.uk/staff/sml/pacman/cec2009/ICEPambush2.pdf (accessed 12/ 04/ 2010).
[15] S. Baluja, “Population-Based Incremental Learning: A Method for Integrating Genetic Search Based Function Optimization and Competitive Learning”, Technical Report CMU-CS-94-163, Universidad Carnigie Mellon, 1994.
[16] S. M. Lucas, “Evolving a neural network location evaluator to play Ms. Pac-Man,” IEEE Symposium on Computational Intelligence and Games, pp. 203–210, 2005.
[17] S.M Lucas and G. Kendall, “Evolutionary computation and games” IEEE Computational Intelligence Magazine, vol. 1, pp. 10-18, 2006.