An Influence Map Model for Playing Ms. Pac-Man

Nathan Wirth and Marcus Gallagher, *Member, IEEE*

Abstract—In this paper we develop a Ms. Pac-Man playing agent based on an influence map model. The proposed model is as simple as possible while capturing the essentials of the game. Our model has three main parameters that have an intuitive relationship to the agent's behavior. Experimental results are presented exploring the model's performance over its parameter space using random and systematic global exploration and a greedy algorithm. The model parameters can be optimized without difficulty despite the noisy fitness function used. The performance of the optimized agents is comparable to the best published results for a Ms. Pac-Man playing agent. Nevertheless, some difficulties were observed in terms of the model and the software system.

I. INTRODUCTION

Pac-Man is a relatively simple and extremely well-known computer game. Although the original Pac-Man arcade game was released more than 25 years ago, the game is still played on various computing platforms, including mobile phones. In the game, the player controls the movement of the Pac-Man character around a sequence of mazes. The general aim of the game is to score as many points as possible, by eating dots initially distributed in a maze (see Figure 1 for a screen-shot of the game). Four ghost characters pursue Pac-Man around the maze and must be avoided otherwise Pac-Man loses one of his three lives (when all lives are lost the game is over). In addition, four power-pills are located in the corners of each maze. When Pac-Man eats a power-pill, he is then able to eat the ghost characters for a few seconds (who will retreat from rather than chase Pac-Man during this time).

Pac-Man style games have recently received some attention in Artificial/Computational Intelligence (AI/CI) research. This is because the game provides a sufficiently rich and useful platform for developing CI techniques in computer games. On one hand, Pac-Man is simple enough to permit reasonable understanding of its characteristics, requires relatively modest computational requirements and has a small code size. On the other hand, game-play based on "intelligent" strategies, planning and priority management is possible, as opposed to many other simple real-time computer games where success is based largely on speed and reaction-time. The predator-prey nature of Pac-Man provides significant challenges for using CI techniques to create intelligent agents in game-play.

Ms. Pac-Man is the successor to Pac-Man. The two games are very similar, but have a number of minor differences. In particular, the ghosts in Pac-Man behave deterministically. Assuming that the Pac-Man agent moves in precisely the

same way with the same timing over multiple games, the ghosts movements (which depend on the location of Pac-Man among other things) will also be repeated. While this is not a major factor in average human game play, it does make it possible to learn fixed, repeatable "optimal" paths for Pac-Man [1]. In Ms. Pac-Man however, the ghosts movement has an element of pseudo-randomness, making it impossible to learn paths and leading to a more challenging game. Ms. Pac-Man also features additional maze layouts compared to Pac-Man.

In this paper we apply influence maps, a widely-used technique in computer games, robotics and other areas, to the task of creating an artificially intelligent agent for playing Ms. Pac-Man. In the following Section, we review previous work on applications of AI and CI techniques to Pac-Man style games. Section III describes our influence map model and Section IV provides details of the software platform used in our implementation. Section V presents details and results of our experiments with the influence map agent on Ms. Pac-Man, including the task of finding suitable model-parameter values. Section VI gives some discussion and comparison of the results in this paper compared to the recent Congress on Evolutionary Computation (CEC'08) Ms. Pac-Man Competition, and Section VII provides a summary and conclusions.

II. AGENTS FOR PLAYING PAC-MAN AND MS. PAC-MAN

Previous research has utilized Pac-Man-type games as a test-bed for producing intelligent game-playing agents. Koza [2] and Rosca [3] use the general ideas of Pac-Man as a problem domain to study the effectiveness of genetic programming for task prioritization. Their approach relies on a set of predefined control primitives for perception, action and program control (e.g., advance the agent on the shortest path to the nearest uneaten power pill). The programs produced represent procedures that solve mazes of a given structure, resulting in a sequence of primitives that are followed.

Kalyanpur and Simon [4] use a genetic algorithm to try to improve the strategy of the ghosts in a Pac-Man-like game. Here the solution produced is also a list of directions to be traversed. A neural network is used to determine suitable crossover and mutation rates from experimental data. De Bonet and Stauffer [5] describe a project using reinforcement learning to develop strategies simultaneously for Pac-Man and the ghosts, by starting with a small, simple maze structure and gradually adding complexity.

Gallagher and Ryan [6] used a simple finite-state machine model to control the Pac-Man agent, with a set of rules governing movement based on the "turn type" at

Nathan Wirth and Marcus Gallagher are with The School of Information Technology and Electrical Engineering, University of Queensland 4072, Australia (phone: +61-7-3365-6197; fax: +61-7-3365-4999; email: marcusg@itee.uq.edu.au).

Pac-Man's current location (e.g corridor, T-junction). The rules contained weight parameters which were evolved from game-play using the Population-Based Incremental Learning (PBIL) algorithm [7]. This approach was able to achieve some degree of learning, however the representation used appeared to have a number of shortcomings.

Lucas [8] proposed evolving neural networks as move evaluators in a Ms. Pac-Man implementation. The neural networks evolved utilize a handcrafted input feature vector consisting of shortest path distances from the current location to each ghost, the nearest power pill and the nearest maze junction. A score is produced for each possible next location given Pac-Man's current location. Evolution strategies were used to evolve connection weights in networks of fixed topology. The results demonstrate that the networks were able to learn reasonably successful game-play as well as highlighting some of the key issues of the task (such as the impact of a noisy fitness function providing coarse information on performance. Gallagher and Ledwich [9] also evolved (multi-layer perceptron) neural networks, but used a more unprocessed representation of the game state based on raw screen data. The original Pac-Man game (simplified) was used and the performance achieved was modest. Nevertheless, the agents produced some elements of successful game-play from very limited information. This work showed at least the potential for a "pure" machine learning-based approach to producing a good Pac-Man playing agent.

Recently, Szita and Lorincz [10] proposed a different approach to playing Ms. Pac-Man. The aim is to develop a simple rule-based policy, where rules are organized into action modules and a decision about which direction to move is made based on priorities assigned to the modules in the agent. Policies are built using the cross-entropy optimization algorithm. The best performing agents were comparable to the performance of a set of human subjects tested on the same version of the game (note however that is not the original Ms. Pac-Man arcade game).

III. INFLUENCE MAP MODEL

Influence maps are a technique that have been used previously to facilitate decision-making in computer games [11]. In robotics, essentially the same idea is that of a potential field (see e.g.[12], [13]). An influence map is a function defined over the game world, representing the (un)desirability of a location according to some measure. Game objects, agents and other features typically exert an influence around their current location and the influence map becomes the sum of all such local influences. For example, locations containing food, health or point-scoring objects might exert a positive influence on their location and surrounding regions, while enemy agents or objects that need to be avoided might have a negative influence. On this basis, an agent could decide where to move by looking for positive regions of the influence map. Influence maps can be visualized as a landscape superimposed over the game world, with the height of the landscape at any position given by the value of the influence map. Influence maps are typically

discretized according to a grid defined over the game world for computational reasons.

Fig. 1. A screenshot of Ms. Pac-Man, with the value of an influence map model displayed as a "cloud" over the maze. Ghosts and dots (including power pills) contribute a localized component which combine additively to produce the overall influence map.

For Ms. Pac-Man, our influence map model was constructed by considering only dots, ghosts and edible ghosts (i.e. fruit and other features of the maze such as the tunnels and ghost cage were ignored). Power pills were considered as just another dot, though their effect is implicitly taken into account by considering edible ghosts (see below). Locations over the maze (including exterior walls) are defined over a 28 (wide) $\times$ 31 (high) grid, where each grid square contains an 8 $\times$ 8 square of pixels. Influence map values were then calculated for locations in this grid. An example of an influence map is shown in Figure 1 as a coloured cloud over a game maze.

The general idea of our influence map encodes basic intuition about playing Ms. Pac-Man; eat dots, avoid ghosts and eat edible ghosts. The actual influence map function used is:

$$\begin{aligned} influence(i_{(x,y)}) = & \sum_{k=0}^{n_d} \frac{p_{dots} \cdot \overline{n_d}}{d(i_{(x,y)}, k_{(x,y)})} \\ & - \sum_{k=0}^{n_r} \frac{p_{run} \cdot n_d}{d(i_{(x,y)}, k_{(x,y)})} \\ & + \sum_{k=0}^{n_c} \frac{p_{chase}}{d(i_{(x,y)}, k_{(x,y)})} \end{aligned} \quad (1)$$

where $i_{(x,y)}$ is the evaluation position in the maze with coordinates x and y . The first summation term on the right-hand side produces a positive contribution to the influence

map for the current number of uneaten dots, n_d (including power pills) in the maze. p_{dots} is a parameter controlling the weight of this term in the equation. $d(i_{(x,y)}, k_{(x,y)})$ is the distance between an uneaten dot (located at $k_{(x,y)}$) and the evaluation position (we have used Euclidean distance in this paper). The influence of a dot is therefore inversely proportional to distance and decays geometrically. Finally, $\bar{n}_d$ is the number of dots that have currently been *eaten* (for the first maze in Ms. Pac-Man, $n_d + \bar{n}_d = 224$). Adding this to the numerator has the effect of increasing the influence of each remaining dot in the maze as the number of dots decreases. This is useful because without such a term, one or two isolated dots that are distant from Ms. Pac-Man in the maze will have very little influence on the agent. Consequently, Ms. Pac-Man will be unable to finish a maze and performance will drop as dots become scarce.

The second term is similar in form to the first, but this time produces a negative contribution to the influence map for (inedible) ghosts in the maze at the current time instant. The sum is over the number of inedible ghosts currently in the maze, n_r . The p_{run} parameter controls the weight of this term in the equation. Each term in the sum is scaled by n_d so that, as the game progresses (and n_d decreases) the overall influence of inedible ghosts also decreases. The third term in the equation is also similar to the previous but adds a positive influence for edible (blue) ghosts. The p_{chase} parameter controls the weight of this term, and no other scaling factor is used.

When playing Ms. Pac-Man, the agent needs to provide input to control the direction of movement of the Ms. Pac-Man character (i.e. up, down, left or right). To produce this controller in our agent (at each time instant), the value of the influence map is calculated at each of the (up to¹ 4) adjacent feasible locations that Ms. Pac-Man can move to from her current position in the maze. The location with the maximum influence map value (and corresponding move direction) is then selected and output from the controller.

The influence map controller has the advantage of being simple, with a small number of parameters that need to be set. Furthermore, the influence map model is reasonably intuitive in its design and the parameters have clear intentions. In a sense, the model is one way of encoding human knowledge and heuristics for playing Ms. Pac-Man into a model that can be used to control an agent. Influence values can be calculated reasonably efficiently, making the technique applicable in a real-time game setting.

IV. METHODOLOGY AND SOFTWARE

A. Setting Model Parameters

The influence map model gives rise to a 3-dimensional optimization problem over the model parameters (p_{dots} , p_{run} , p_{chase}). Our objective/fitness function for this optimization problem was average performance (score) over a number of games. Note that even when averaged over many games, this

objective function is highly stochastic because of the nature of the game.

The terms in Equation 1 each involve (Euclidean) distance measures between grid locations in the maze. The dimensions of a maze (28×31) mean that these distance values range approximately from 0 to 40. Given this, values for the influence map model parameters were constrained to be in the range $[0, 100]$. This simply ensures that the variables in Equation 1 have similar magnitudes: if this were not the case then the effect of any single variable might be insignificant or completely dominate the value of the influence map. Furthermore, the additive nature of the terms in the influence map suggests that the model is not likely to be highly sensitive to small variations in model parameter values in terms of overall game play performance. This notion was supported by preliminary experiments. Hence, only integer values in this range were allowed (effectively limiting precision). In this case, the size of the parameter (search) space is $100^3 = 10^6$.

B. Software Details

Our software platform (C++) is based on the Pac-Man Instructional Emulator (PIE) [14] as the main software component, utilizing the ROMs from the original Ms. Pac-Man game. However, experimentation revealed that the Z80 processor emulator within PIE did not seem to produce the stochastic behavior of Ms. Pac-Man: we found that game-play was deterministic given identical agent/player behavior. We replaced this with the Z80 emulator within TICKLE [15]. Finally, the software made available by Manuel Loth (available from [16]) was used for extracting information about the state of the game world. The influence map agent was then implemented on this platform as a game controller.

V. EXPERIMENTAL DETAILS AND RESULTS

The initial set of experiments was a uniform random exploration of the 3D parameter space. This was done to gain general insight into the relationship between performance and the parameter space and to look for general trends. 787 sample points within this search space were evaluated over 40 trial runs of the game (3 lives per game).

Figure 2 shows the results of these experiments, with points in the parameter space shaded according to the average score attained over the 40 runs. A clear global trend is evident, with the highest performing results clustered in a triangular prism/wedge-shaped region at the bottom foreground in the figure. This performance trend also appears to be largely invariant to the value of the p_{chase} parameter.

Motivated by the above results, a more systematic search was performed in the seemingly most promising region of the search space. The p_{chase} parameter was arbitrarily set constant at an intermediate value of 50, while the value of p_{dots} was varied as $1, 2, \dots, 62$ and p_{run} was varied as $4, 8, 12, \dots, 100$. This is a total of 1575 parameter combinations, again with 40 trial runs conducted at each point.

The results of this second set of experiments are shown in Figure 3. The trend noted from Figure 2 is more clearly

¹Depending on maze walls.

Fig. 2. Performance results on Ms. Pac-Man with different values for the influence map parameters (three axes on the graph). Random integer parameter values were explored within a given range. Points in the graph are shaded according to average game performance over 40 trials (see scale to the left of the graph).

observable here (this surface effectively being a vertical “slice” through the space shown in Figure 2). In fact, the top 10% of parameter configurations (again according to average score) are well-described by the region between the lines $p_{\text{dots}} = \frac{1}{20}p_{\text{run}}$ and $p_{\text{dots}} = \frac{3}{10}p_{\text{run}}$. To take this further, the top 1% of parameter configurations is between the lines $p_{\text{dots}} = \frac{1}{10}p_{\text{run}}$ and $p_{\text{dots}} = \frac{1}{5}p_{\text{run}}$.

Within this influence map controller, the results suggest the following intuition. Good performance is achieved by running away from ghosts and by collecting dots. For best performance, running from ghosts should be 5 to 10 times more important than collecting dots. Giving priority to chasing blue ghosts has little effect on performance within this controller. This is possibly due to the relatively small amount of game time where this influence is relevant, together with the high degree of uncertainty about successfully eating one or more ghosts given the opportunity.

We were also interested in testing the ability of a simple iterative optimization technique to locate a high-performing region of the parameter space. A hill-climbing/greedy algorithm was implemented which took steps in random directions in the (integer) parameter space, with the step size gradually reduced over the period of the search. A sample run (i.e. single trial) is shown in Figures 4 (parameter values) and 5 (performance). It was observed that over this two-dimensional parameter space, a simple hill-climbing optimizer was able to discover a high-performing parameter combination without difficulty. Such performance was not particularly sensitive to the particular nature of the hill-climber implementation (e.g. step size values). Multiple hill-climbing runs produced different parameter value combinations within the wedge-like area identified from the systematic experiments described above.

Also evident from the above results above is the high degree of noise in the fitness function (i.e. average score).

Fig. 3. Results over a systematic search of the $p_{\text{dots}} - p_{\text{run}}$ space, with p_{chase} held constant. The height of the surface indicates average performance over 40 trials. A “wedge” of high-performing parameter combinations is clearly visible towards the right-hand side of the surface.

Fig. 4. An example run (single trial) using a greedy search to select values for the p_{dots} and p_{run} parameters (parameter value evolution shown). Despite the highly noisy fitness function, the algorithm is able to find the high-performing region of the (2-D) parameter space without major difficulty.

This is due to the stochastic nature of the game and presents significant challenges when trying to assess the performance of a given agent, since any agent will produce highly variable results when playing over several games.

In terms of performance, the distribution of results over the 1575 parameter configurations from Figure 3 is shown in Figure 6. For each parameter combination, this graph shows the mean and standard deviation over the 40 trials, with the results sorted by mean value. In addition, the maximum performance values of the top 50 runs are shown. It is clear that the standard deviation of the results tends to shrink as average performance improves. But perhaps most striking is how far the highest performing results lie from the mean

Fig. 5. Fitness progress for the greedy search trial shown in Fig.4.

over trials. Typically, the distribution of performance over runs has a long tail towards higher scores than the mean, leading to such outlying maximum values.

In the above results, the highest average score is 6848 points. The highest score achieved in any single run is 19490, with many of the best scores around 15000 points. This performance is roughly comparable to a novice human player. Anecdotaly, many people find it difficult to achieve a score around 15000-20000.

Fig. 6. Distribution of performance results for the systematic search experiments (Fig.3). Results are sorted left-to-right by mean (solid line) for each configuration. Standard deviations of all configurations are shown, as well as the performance values of the 50 configurations with the highest score on a single trial.

Qualitatively speaking, the game-play of the best-performing influence map agents tends to be fairly systematic and methodical. Ms. Pac-Man focuses on the task of clearing dots while trying to keep a safe distance from the ghosts. The third (p_{chase}) term in Equation.1 leads to active pursuit of the (blue) ghosts after a power pill is eaten, which is quite successful when the ghosts happen to be in close vicinity to

Ms. Pac-Man at the time. However, there is nothing in the model that attempts to “sets up” such an opportunity, nor does the model have any representation of the knowledge that eating a power pill triggers blue ghosts to be present, and that this is a time-limited event from that trigger (a fact that any human player is aware of).

On the whole, the agents seem to behave sensibly. The main exception observed was that occasionally Ms. Pac-Man would sit at a fixed position and oscillate between two intended directions to move. This happens when the influence map has very similar values in both directions. While this situation usually resolves itself dynamically during play (moving ghosts change the influence map), it is at a disadvantage to Ms. Pac-Man and is sometimes fatal. This and other problems (oscillation and local minima) are in fact well-known when using potential field maps for navigation in robotics [12], [13].

VI. CEC'08 COMPETITION ENTRY

An agent was entered into the 2008 Congress on Evolutionary Computation (CEC) Ms. Pac-Man competition [16] based on the influence map model described in this paper. For this entry, the agent was ported to Java to work with the toolkit provided for the competition. The competition entry used Ms. Pac-man available online² and played through a web browser.

The performance of the agent in the competition was significantly worse than the results presented in this paper. Over 10 runs prior to the competition, the agent was only able to achieve an average score of 2510 and a maximum score of 4360 [16], in contrast to the highest average score reported above (6848) and the highest scoring single run (19490). The highest performing single game in the competition (from the entry by Fitzgerald et. al) was 15970.

The major difficulty we have observed in the competition/screen-capture version of our agent was that it was unable to navigate the maze with precision. For example, Ms. Pac-Man would be travelling from left to right and intend to travel up a tunnel, but would “miss” the grid point at the turn. The agent would then turn around (heading right to left) and repeat this a few times before successfully moving into the upwards tunnel. This is mainly due to the delay present when the agent’s decision is made on the basis of a slightly delayed screen capture: by the time the decision has been made, the agent is no longer in the position it thought it was. While it may be possible to implement an allowance to fix this, we were not able to do this in time for the competition. It is also possible that the time taken to perform the agent calculations was greater in the Java version than the C++ version, leading to another delay. Finally, screen position of the game window is very important in the screen capture version: this may have an impact on the accuracy of the information available to the agent controller. These results highlight the importance of the software platform when conducting research with

²WebPacMan: <http://webpacman.com>

AI in real-time computer games. Issues of replicability of experimental results and the influence of system-specific factors are likely to require particular attention as the field develops.

VII. CONCLUSION/SUMMARY

This paper describes an influence-map model for producing an intelligent agent/controller to play Ms. Pac-Man. The model is relatively simple and intuitive and has relatively few user parameters that require tuning. Our experimental results show that the parameters of the model interact in a fairly simple way and that (with some computational resources) it is reasonably straight-forward to optimize the parameters to maximize game playing performance. Average results of the controller show comparable performance to a novice human player. Performance results vary considerably due to the stochastic nature of the game, but the highest scores achieved by the agent in single games is at least comparable to the best reported results in the literature and in the recent CEC'08 Ms. Pac-Man competition.

Our main aim in this work was to develop an agent that performed well on Ms. Pac-Man, with the implication that the techniques used in such an agent might be potentially useful in other real time computer games. While we have focused on Ms. Pac-Man, the influence-map model could also be readily applied for other games, since it is really a functional representation of (un)desirable objects/locations in a game world. Although a simple influence map model shows good novice ability on Ms. Pac-Man, the model has some inherent limitations that have been recognized in similar potential field models in robotics. Future work could investigate the ways that such problems have been addressed in robotics to see if they can be usefully translated to the games domain. The influence map model described above could readily be expanded upon to include more detailed information about the game (e.g. to distinguish power pills, recognize fruit, etc.). We predict that adding such features would need to be combined with addressing the previously mentioned difficulties to lead to significant improvements in performance. Another promising direction for future could be to consider hybrid approaches: using an influence map to represent low-level intuitions about a game in combination with a different system to capture "higher-level" strategies for game-play. Another obvious limitation of our work was in the use of the Euclidean distance metric. A more accurate measure of actual path distance is likely to improve performance.

REFERENCES

- [1] K. Uston, *Mastering Pac-Man*. Macdonald and Co, 1981.
- [2] J. Koza, *Genetic Programming: On the Programming of Computer by Means of Natural Selection*. MIT Press, 1992.
- [3] J. P. Rosca, "Generality versus size in genetic programming," in *Genetic Programming (GP96) Conference*, J. K. et. al., Ed. MIT Press, 1996, pp. 381–387.
- [4] A. Kalyanpur and M. Simon, "Pacman using genetic algorithms and neural networks," Retrieved from <http://www.ece.umd.edu/~adityak/Pacman.pdf> (19/06/03), 2001.
- [5] J. S. De Bonet and C. P. Stauffer, "Learning to play pacman using incremental reinforcement learning," Retrieved from <http://www.debonet.com/Research/Learning/PacMan/> (20/10/06).

- [6] M. Gallagher and A. Ryan, "Learning to play Pac-Man: An evolutionary, rule-based approach," in *Congress on Evolutionary Computation (CEC)*, R. S. et. al., Ed., 2003, pp. 2462–2469.
- [7] S. Baluja, "Population-Based Incremental Learning: A method for integrating genetic search based function optimization and competitive learning," School of Computer Science, Carnegie Mellon University, Tech. Rep. CMU-CS-94-163, 1994.
- [8] S. M. Lucas, "Evolving a neural network location evaluator to play Ms. Pac-Man," in *IEEE Symposium on Computational Intelligence and Games*, 2005, pp. 203–210.
- [9] M. Gallagher and M. Ledwich, "Evolving pac-man players: Can we learn from raw input?" in *IEEE Symposium on Computational Intelligence and Games (CIG'07)*, 2007, pp. 282–287.
- [10] I. Szita and A. Lőrincz, "Learning to play using low-complexity rule-based policies: Illustrations through ms. pac-man," *Journal of Artificial Intelligence Research*, vol. 30, pp. 659–684, 2007.
- [11] P. Tozour, "Influence Mapping," *Game Programming Gems 2*, 2001.
- [12] Y. Koren and J. Borenstein, "Potential field methods and their inherent limitations for mobilerobot navigation," *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*, pp. 1398–1404, 1991.
- [13] J. Barraquand, B. Langlois, and J. Latombe, "Numerical potential field techniques for robot path planning," *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 22, no. 2, pp. 224–241, 1992.
- [14] A. Scotti, "PIE - Pacman Instructional Emulator," Available at <http://www.ascotti.org/programming/pie/pie.htm>, (accessed 06/08/08).
- [15] —, "TICKLE (arcade machine emulator)," Available at <http://www.ascotti.org/programming/tickle/tickle.htm>, (accessed 06/08/08).
- [16] S. Lucas, "Ms Pac-Man Competition," Available at <http://cswww.essex.ac.uk/staff/sml/pacman/PacManContest.html>, (accessed 11/08/08).