Evolving a Neural Network Location Evaluator to Play Ms. Pac-Man

Simon M. Lucas
Department of Computer Science
University of Essex
Colchester, UK
sml@essex.ac.uk

Abstract

Ms. Pac-Man is a challenging, classic arcade game with a certain cult status. This paper reports attempts to evolve a Pac-Man player, where the control algorithm uses a neural network to evaluate the possible next moves. The evolved neural network takes a handcrafted feature vector based on a candidate maze location as input, and produces a score for that location as output. Results are reported on two simulated versions of the game: deterministic and non-deterministic. The results show that useful behaviours can be evolved that are frequently capable of clearing the first level, but are still susceptible to making poor decisions. Currently, the best evolved players play at the level of a reasonable human novice.

1 Introduction

Games have long been used in the study of computational intelligence and machine learning. Much of the early focus was on strategy games such as chess and checkers. There have been noteworthy successes for machine learning methods in Backgammon using reinforcement learning [12], or co-evolution [9], and also using co-evolution to find good position evaluation functions for checkers [3].

More recently, with the rise in popularity of real-time arcade and console games, there has been growing interest in applying computational intelligence to such games [7]. This paper investigates the use of Pac-Man style games, in particular Ms. Pac-Man as a challenge for computational intelligence. One of the motivations for this work is to gain insight into the kind of approaches are effective in this domain, where the large state space precludes the direct use of game-tree search. While this paper only evolves the Pac-Man controller, it provides a foundation for also evolving ghost behaviours, which might lead to Pac-Man variants that are even more fun to play than the original. Furthermore, the aim is to eventually gain insight into how hard these games are to play: are good human players simply reacting quickly in ways that would be obvious to some rudimentary form of computational intelligence, or does Ms. Pac-Man have chess-like complexity?

Pac-Man is a classic arcade game originally developed by Toru Iwatani for the Namco Company in 1980. The game rapidly achieved cult status, and various other versions followed. The best known of these is Ms. Pac-Man, released in 1981, which many see as being a significant improvement over the original. In the original game, the ghosts behaved deterministically, and a player could exploit this behaviour by using set patterns or routes. Provided the successful route was executed faithfully (in some routes precise pauses were necessary at certain points on the route), then the level could be cleared while achieving maximum points in the process. From a machine learning perspective, the learning of such routes is less interesting than the learning of more general intelligent behaviour, able to cope with novel circumstances. In Ms. Pac-Man, the ghosts have similar hunting abilities to the original game, but are non-deterministic, which makes the game much harder and more interesting. Ms. Pac-Man also uses four different mazes, which are revealed as the player progresses through levels, though in this paper we only use the first level. The rules and game setup are reasonably simple, at least compared to more modern arcade and console games, though close investigation reveals many interesting subtleties that might go unnoticed by the casual player. Many of these finer points are mentioned in sections 2 and 3 below.

From a computational intelligence perspective, there are many aspects of the game worthy of study. These include evolving Pac-Man playing agents and evolving ghost behaviours. The latter is a problem in multi-agent systems, since the ghosts should cooperate with each other to be maximally effective, though the game designer can impose various constraints on the ghosts, such as disallowing direct ghost communication. Both the Pac-Man and Ms. Pac-Man ghosts behave far from optimally, and this may add to the appeal of the game. However, it would also be interesting to experience playing against evolved 'optimal' ghosts, even if it turns out to be less enjoyable than playing against the standard sub-optimal variety.

1.1 Previous Research

There have been several previous efforts to evolve Pac-Man players. One of the earliest was that of Koza [6]. Koza used his own implementation of Pac-Man, based on the first screen of Ms. Pac-Man. He defined 15 functions: two conditionals and 13 primitives for controlling the Pac-Man. These included output controls such as "move towards to nearest pill along shortest path". Koza's implementation appears to be significantly different from the real game, and the ghost behaviours make the game quite easy for the Pac-Man. In his version, all four ghosts follow the same behaviour: pursue Pac-Man when he is in sight for twenty out of twenty-five time steps, and randomly change direction for five out of twenty-five time-steps. This is very different from the original Ms. Pac-Man ghosts, who each had distinctively different behaviours, would pursue the Pac-Man more aggressively, and would reverse direction much less often (more like once in two hundred time steps on average), and less predictably. More frequent ghost direction reversals make the game much easier for the Pac-Man, since each reversal can act as a lifesaver in an otherwise hopeless situation.

More recently, Gallagher and Ryan [5] evolved a Pac-Man player based on a finite-state-machine plus rule-set. In their approach they evolved the parameters of the rule set (85 in total), where the individual rules were hand-specified. However, the game simulator they used was a greatly simplified version of the original. Although the maze was a faithful reproduction of the original Pac-Man maze, only one ghost was used (instead of the usual 4), and no power pills were used, which misses one of the main scoring opportunities of the game.

De Bonet and Stouffer [2] describe a reinforcement learning approach to simple mazes that only involved one other ghost, and a 10×10 input window for the control algorithm centred on the Pac-Man's current location. They were able to learn basic pill pursuit and ghost avoidance behaviours. There has also been some work on learning routes for Pac-Man, but as explained above, this approach is not viable for Ms. Pac-Man.

2 Ms. Pac-Man Game Play

The player starts with three lives, and a single extra life is awarded at $10,000$ points. While it is never a good idea to sacrifice a life, it may be better to take more risks when there are lives to spare. There are 220 food pills, each worth 10 points. There are 4 Power Pills, each worth 50 points. The score for eating each ghost in succession immediately after a power pill starts at 200 and doubles each time. So, an optimally consumed power pill is worth 3050 $(= 50 + 200 + 400 + 800 + 1600)$. Note that if a second power pill is consumed while some ghosts remain edible from the first power pill consumption, then the ghost score is reset to 200.

Additionally, various types of fruit appear during the levels, with the value of the fruit increasing with each level. The fruit on level one is worth only 100 points, but this increases to many thousands of points in higher levels. It is not necessary to eat any fruit in order to clear a level.

While good Pac-Man players often make use of fixed routes, the non-determinism of Ms. Pac-Man makes this approach ineffective. Instead, short term planning and reactive skill is more important. An appreciation of ghost behaviour also plays a significant part, since although the ghosts are non-deterministic, they are still at least partially predictable. The reactive skill lies in judging the distance between the Pac-Man and the ghosts, and working out relatively safe routes to the pills and the power-pills. Appropriate consumption of the power pills is of critical importance both for gaining high-scores, and for clearing the levels. Expert players will often get the ghosts to form a closely-knit group, which chases the Pac-Man to the power pill, only to be eaten after consumption of the power pill. Indeed, expert players generally have a good understanding of how the ghosts will behave in any circumstances. Billy Mitchell, the world Pac-Man (but not Ms. Pac-Man) champion said the following:

> "I understand the behavior of the ghosts and am able to manipulate the ghosts into any corner of the board I choose. This allows me to clear the screen with no patterns. This was a more difficult method for the initial 18 screens. I chose to do it this way because I wanted to demonstrate the depths of my abilities. I wanted to raise the bar higher - to a level that no one else could match."

In both Pac-Man and Ms. Pac-Man, the ghost behaviour is far from optimal in several ways, in the sense that the ghosts do not try and eat the Pac-Man as quickly as possible. On leaving the nest, each ghost typically meanders for a while before pursuing the Pac-Man more actively. They then adopt different behaviours. The red ghost is the most predatory one, and is quite direct in seeking out the Pac-Man. The ghosts get less aggressive in order, through pink and blue down to the orange ghost, which behaves in a somewhat random way - and will often let Pac-Man escape from situations where the right moves would secure a certain kill. The game also appears to have a bug, in that sometimes the Pac-Man is able to escape what looks like a certain death situation by passing straight through one of the ghosts. Whether this is a bug (which can occur all too easily in multi-threaded systems due to synchronisation problems), or a deliberate feature is not clear. Either way, it does not detract from the quality of the game play, and perhaps even enhances it. For the current paper, only the opening maze of Ms. Pac-Man has been implemented, as the evolved players are still at an early stage.

3 Experimental Setup

While the long-term aim of this work is to evolve controllers for the original Ms. Pac-Man game, there are many other worthy objectives as well, such as evolving new ghost behaviours, and experimenting with the effects of various changes to the rules. For these purposes, and also to enable much faster fitness function evaluation, we implemented our own Ms. Pac-Man simulator. The implementation was designed to be a reasonable approximation of the original game, at least at the functional if not cosmetic level, but note that there are several important differences:

- The speed of both the Pac-Man and the ghosts are identical, and the Pac-Man does not slow down to eat pills.
- Our Pac-Man cannot cut corners, and so has no speed advantage over the ghosts when turning a corner.
- The ghost behaviours are different (ours are arguably more aggressive, apart from the random one).
- Our ghosts do not slow down in the tunnels. This comes as a bit of a shock when playing the game!
- In our maze, there is no fruit. The main aim at present is to learn the basic game. Fruit only plays a minor part until the higher levels when it becomes more valuable, and can add significantly to the score.
- No additional life at $10,000$ points. This would have been easy to implement, but has little bearing on evolved strategies.
- All ghosts start immediately at the centre of the Maze, and are immediately in play: there is no nest for them to rest in.
- A ghost once eaten returns instantly to the centre of the maze, and resumes play immediately.
- Currently, our ghosts do not consider the locations of the other ghosts; they could improve their hunting if they utilised such information to 'spread-out' more when hunting the Pac-Man.
- Our game lacks the nice cosmetic finishes of the original. Game play (for a human player) is surprisingly diminished as a result of being chased by a silent rectangle, instead of a wailing ghost with watchful eyes!

Some of these are due to lack of time available in implementing the game, but replicating the exact ghost behaviours is hard. While it is possible that the underlying behaviours may be reducible to a few simple rules, it is difficult to determine these from observation alone, though clearly the expert human players have done a good job of this, at least implicitly. Indeed, learning the ghost behaviours by observing game-play would be a challenging machine learning project in itself.

3.1 The Implementation

The implementation is written in object-oriented style in Java and is reasonably clean and simple. The maze is modelled as a graph, with each node in the graph being connected to its immediate neighbours. Each node has two, three or four neighbours depending on whether it is in a corridor, a T-junction, or a crossroads. Each graph node is separated by two screen pixels - this gives very smooth movement. After the maze has been created, a simple efficient algorithm is run to compute the shortest-path distance between every node in the maze and every other node in the maze. These distances are stored in a look-up-table, and allow fast computation of the various controller-algorithm input features listed below. During evolution, each game lasts for between a few hundred and a few thousand time-steps, depending on luck and on the quality of the controller. Typically, around 33 games per second can be played (on a 2.4ghz PC) when evolving a perceptron. This slows to around 24 games per second for an MLP with 20 hidden units. The cost of simulating the game and computing the feature vector for each node under consideration is greater than the cost of simulating the neural networks.

For a given maze, game play is defined by the ghost behaviours and a few other parameters, listed in Table 1. The ghosts operate by scoring each option and choosing the best one, when more than one option exists, which is only the case at junction nodes (recall that ghosts are not allowed to turn back on themselves, except during a reversal event). Ghost 1 pursues the most aggressive strategy, always taking the shortest path to the Pac-Man. Ghosts 2 and 3 operate under less effective distance measures, while ghost 4 makes random choices, except when playing deterministically, in which case it chooses the option that minimises the y distance to the Pac-Man. All ghosts have small pseudo-random noise added to their evaluations in order to break ties randomly.

The author tested the implementation to assess the difficulty of the game. For a human player, of course, this is influenced very significantly by the speed at which the game is played. When played at approximately the same speed as the original Ms. Pac-Man, the game seemed harder than the original, with the author's best score after ten games be-

Feature	Description
Ghost 1	Minimise shortest-path dist. to Pac-Man
Ghost 2	Minimise Euclidean dist. to Pac-Man
Ghost 3	Minimise Manhattan dist. to Pac-Man
Ghost 4	Random (Or minimise y-dist. if non-rand.)
Edible	Ghosts are edible for 100 time steps
Reversal	Ghosts reverse every 250 time-steps
Ed. Spd	Ghosts move at half-speed when edible

Table 1. *The Simulated Game Setup.*

ing only $12,890$, compared to over $30,000$ on the original Ms. Pac-Man. When slowed down to $1/3$ speed, however, a score of over $25,000$ was achieved before boredom set in. Since playing the original game at $1/3$ speed was not possible, it is hard to make a direct comparison beyond this.

3.2 Location Features

Our Pac-Man control algorithm works by evaluating each possible next location (node), given the current node. Note that each node is two screen pixels away from it's neighbour, which leads to smooth movement, similar to the arcade game, but also makes the maze-graph rather large, with $1,302$ nodes in total.

The control algorithm chooses the move direction that will move it to the best-rated location. From an evolutionary design viewpoint, the most satisfying approach would be to use all the information available in the state of the game as inputs to the node evaluation neural network. This would include the maze layout, the position of each pill and power pill, and the position of each ghost (and whether the ghost was currently edible, and how much longer it was due to remain edible for). This would involve processing a large amount of information, however. For the current experiments, a less satisfactory, but more tractable approach has been taken. Features were chosen on the basis of their being potentially useful, and efficient to calculate. These are listed in Table 2. These features are unlikely to be optimal, and almost certainly place significant limitations on the ultimate performance that such a controller can achieve. Whether the networks evolved for this paper have reached these limits is another question. Note that the use of maze shortest-path distances in the features naturally helps the Pac-Man avoid getting stuck in corridor corners; something that could otherwise happen if the inputs were based on geometric distance measures (such as Euclidean). Note that two of the ghosts do use geometric distance measures, but since ghosts are not normally allowed to turn back on themselves, they cannot become stuck in this way.

Input	Description
$g_1 \dots g_4$	distance to each predatory ghost
$e_1 \dots e_4$	distance to each edible ghost
x, y	location of current node
pill	distance to nearest pill
power pill	distance to nearest power pill
junction	distance to nearest junction

Table 2. *The feature vector. Each input is the shortest-path distance to the specified object, or set to the maximum possible distance if that object does not currently exist in the maze.*

3.3 Evolutionary Algorithm

Our Evolutionary Algorithm (EA) is an $(N + N)$ evolutionary strategy [1]. Experiments were done with $N = 1$ and $N = 10$. The case of $N = 1$ results in a simple random mutation hill-climber. These hill-climbers can outperform more complex EAs in some cases [8], but they have to be applied carefully in this domain, due to the high-levels of noise present in making fitness evaluations. For this reason, we investigate the effects of playing multiple games per fitness evaluation, which reduces noise. The ES was also run with $N = 10$, and this was found to give better evolutionary progress, with the larger population also quenching the effects of noise. Using an ES removes the need to define a crossover operator. This would not be a problem for the single-layer perceptrons, but naively defined crossover operators are typically destructive when evolving MLPs, due to the problem of competing conventions [11].

The overall objective was to evolve the best player possible, where quality of play is measured by average score over a significant number of games (e.g. 100). The fitness function used by the EA was the average score over a number of games, where the number of games was either 1, 5, or 50.

3.4 Neural Networks to be Evolved

The neural networks were implemented in object-oriented style with the basic class being a Layer, consisting of a weight matrix mapping the inputs (plus bias unit) to the output units, and a *tanh* non-linearity being applied to the net activation of each output. A single layer perceptron then consisted of just one such layer, a multi-layer perceptron consisted of two or more such layers, though experiments in this paper were limited to a maximum of two layers (hence one hidden layer in standard terminology).

For initial members of the population, all weights were drawn from a Gaussian distribution with zero mean and standard deviation equal to the reciprocal of the square root of the fan-in of the corresponding node. A mutated copy of a network was made by first creating an identical copy of the network and then perturbing the weights with noise from the same distribution. Weights to be mutated were selected in one of four ways as follows (probabilities in parentheses): all the weights in the network (0.1), all the weights in a randomly selected layer (0.27), all the weights going into a randomly selected node (0.27), or a single randomly selected weight (0.36). While these parameters could have been adapted (or indeed, the noise distribution for each weight could have been adapted), these adaptations can also have detrimental effects on the convergence of the algorithm when high levels of noise are present, as is the case here.

4 Results

Results are presented for evolving neural network location evaluators, first for the deterministic game, then for the non-deterministic game.

4.1 Evolving Players for Deterministic Play

As might be expected, evolving strategies against deterministic ghosts is easier than when the ghosts exhibit a small amount of non-determinism. Effective ghosts chase aggressively most of the time, while occasionally making a surprise move in order to disrupt any patterns. To test this we experimented with evolving players against deterministic ghosts. Figure 1 shows the fitness evolution of a perceptron against deterministic ghosts. Since, for a given neural network, every run of the game is identical, only one game run per fitness evaluation is required. The maximum fitness (score) attained was 9,200. Note that when played in the non-deterministic game, this player did not have especially high fitness, with an average score (over 100 games) of only 1,915 (Table 3).

4.2 Simple Ghost Avoidance

Before evolving neural networks for the non-deterministic game, tests were run to see how well a simple ghost avoidance strategy would perform. To this end, a hard coded controller was implemented, which scored a node as the shortest path distance to the closest ghost, and then chose the node that maximised that score. This strategy performed poorly, however, as shown in Table 3 (Simp-Avoid), with an average score of only 980. This was thought to be a worthwhile test in order to demonstrate that the neural networks are learning something more than this simple, if somewhat flawed, ghost avoidance technique.

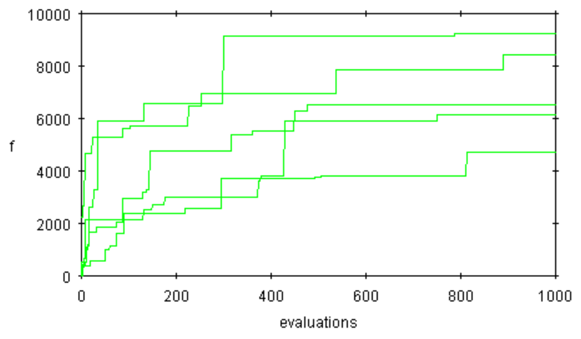

Figure 1. *Fitness evolution of a perceptron for the deterministic game.*

4.3 Evolving for Non-Deterministic Play

In the non-deterministic game, the aim is to learn general behaviours, but the noise makes progress difficult. Increasing the number of games per fitness evaluation improves the reliability of the fitness estimate, but means that fewer fitness evaluations can be made within the same time.

Figure 2 shows the effects that the games per fitness evaluation has on evolving an MLP with twenty hidden units, using a $(1+1)$ ES. The graph shows plots for 1, 5 and 50

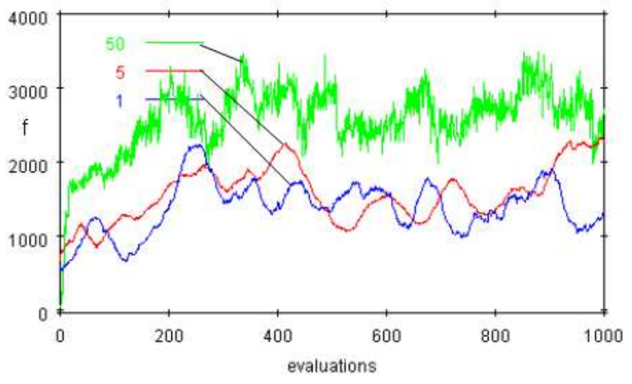

Figure 2. *Evolving an MLP (20 hidden units) with a* $1+1$ *ES, using 1, 5, and 50 games per fitness evaluation.*

games per fitness evaluation, and indicates that a large number is needed in order for evolution to make progress, with best performance being obtained when we run 50 games per evaluation. Note that the graphs for 1 and 5 games have been averaged using a smoothing window of 25 samples either side of the centre. This is to present a better picture of the average fitness at each point on the graph. We ran the same $(1+1)$ experiments for a perceptron, and found that this was able to significantly outperform the MLP, but that the overall trend was the same, in that making many fitness evaluations per game (e.g. 50) was essential in order to make good evolutionary progress.

An alternative that seems to work better, is to increase the population size instead. Figure 3 plots average fitness versus generation when using a $10 + 10$ ES to evolve an MLP with 20 hidden units. Error bars are shown at $(+/-)$ one standard error from the mean, to give an idea of the range of performance present in each generation. Experiments were also made with 5 and 10 hidden units, but 20 seemed to perform best, though this was not thoroughly analysed. For this graph, 5 games per fitness evaluation were used, and therefore the total number of games played was identical to the 50 game plot of Figure 2. Unlike the $1 + 1$ ES, the larger population is able to smooth the effects of noise. A similar graph is observed when evolving a single layer perceptron using the $10 + 10$ ES (not shown), but it was consistently found that the best evolved MLP out-performed the best evolved perceptron. While each evolutionary run produces neural networks that have different behaviours, and even a particular network can behave rather differently on each game run, there are some interesting trends that can nonetheless be observed, and these are further discussed in Section 4.4.

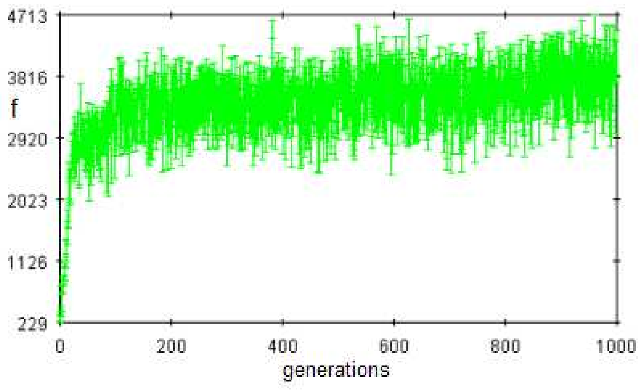

Figure 3. *Evolving an MLP (20 hidden unit) with a* $10 + 10$ *ES, using* 5 *games per fitness evaluation.*

Table 3 shows the performance statistics for three evolved controllers and one hand-designed controller. MLP-20 (20 hidden units) is one of the best MLPs found with the $10 + 10$ ES. This is significantly better than the evolved perceptron (Percep-LC) - where LC stands for Level-Clearer - as explained below. Next we have the perceptron evolved on the deterministic game, but tested on the non-deterministic game (Percep-Det), and finally the hand-designed weights, and Simp-Avoid, both of which perform very poorly.

Controller	min	max	Mean	s.e.
MLP-20	2590	9200	4781	116
Percep-LC	1670	8870	4376	154
Percep-Det	440	5140	1915	90
Hand-Coded	400	3540	1030	58
Simp-Avoid	480	1320	980	82

Table 3. *Controller performance statistics (measured over 100 runs of the non-deterministic game).*

4.4 Evolved Behaviours

One of the most surprising behaviours that sometimes evolves is the ghost chaser strategy. This exploits the fact that when a ghost reaches a junction, it is not normally allowed to retrace its path (unless all ghosts are reversing). Therefore, especially when the ghosts are bunched together, the Pac-Man can safely chase them for a while - until all the ghosts suddenly reverse, at which point, the Pac-Man should make a hasty retreat, but this particular evolved strategy fails to do this and causes the Pac-Man to be immediately eaten instead.

A more successful strategy has been dubbed the Level-Clearer (Percep-LC), owing to its ability to regularly clear the first level of the game, although it usually loses a few lives in the process, and therefore does not usually clear the second level (even though all levels are currently identical). Also, it is rather poor at eating ghosts, and so does not achieve very high scores relative to its progress through the game. The best evolved controller, MLP-20, on the other hand is not as good at clearing a level, but much better at eating ghosts.

4.5 Interpreting the Evolved Neural Network

Recall that the evolved perceptron assigns a value to each possible neighbour of the Pac-Man's current location. To gain more insight into the kind of judgements it is making we can plot the score for every node in the network, which is independent of the current Pac-Man location. The intensity gradients then show the immediate route the Pac-Man will follow, and show the current ratings of the rest of the maze locations. Note that in many cases, all ratings fall into a small range, but only the order of the nodes matters, not their absolute values. For display purposes, we map the range of scores into the full range of grey levels from black to white.

Figure 4 shows these scores during a sample game run for an evolved perceptron controller. The Pac-Man is following a gradient towards some pills and an edible ghost. In this game the Pac-Man next proceeds to eat the two nearest edible ghosts, but is then eaten by the red ghost (which has

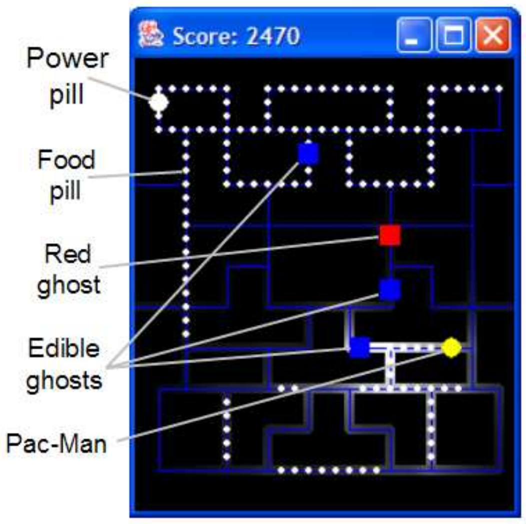

Figure 4. *The node scores overlaid on the Maze as background intensity.*

just itself been eaten by the Pac-Man, and is on the chase again having been reset to the centre of the maze).

Since the single layer perceptron is a linear weighting of the input features, we can inspect these to gain insight into what the system has learned. Table 4 shows the weights for the evolved level clearing perceptron, together with a set of hand-designed weights. One of the most surprising observations is that the evolved controller actually makes Pac-Man *attracted* to the strongest ghost (g_1), as well as the random ghost (g_4). The strongest attraction is towards pills (-15), while unexpectedly, the Pac-Man is mildly repelled by junctions. This was unexpected, since junctions have more escape routes than corridors and are therefore safer places to be. Note that the Pac-Man is attracted to only three out of four edible ghosts - again a surprise. The evolved controllers do vary from run to run, but many display these counter-intuitive features. When observing the game-play, it became clear why there is sometimes an advantage in being attracted to ghosts - it enables the Pac-Man to lure the ghost to a power-pill - and subsequently eat the ghost. This is a dangerous policy, however, and inevitably leads the Pac-Man to unnecessary death on many other occasions.

With these considerations in mind, a hand-designed controller was constructed, with the weights also shown in Table 4. This controller performed poorly, however, as can be seen from the statistics in Table 3.

Parameter	Designed	Evolved
g_1 (red)	1.0	-0.29875
g_2 (pink)	0.8	2.88190
g_3 (blue)	0.6	2.60610
g_4 (orange)	0.4	-0.31166
e_1	-1.0	-1.72596
e_2	-1.0	-0.66969
e_3	-1.0	-1.36305
e_4	-1.0	0.21379
x	0.0	-0.35274
y	0.0	3.48056
pill	-0.1	-15.18330
power	-1.0	-3.95439
junction	-1.0	1.31463

Table 4. *Designed and evolved perceptron weights (see Table 2 for description of parameters).*

5 Discussion and Future Directions

Perhaps the least satisfactory aspect of the current work is the fact that the input vector has been hand-designed. This was done in order to enable fast evolution of behaviours, but also places limits on how good the evolved controllers ultimately become. One possibility would be to feed the raw 2D array of pixels to the network, and see what it can learn from that, although the answer may well be ‘not very much’. Note however, that this approach has been applied with some success in [4], where they evolved a car controller given the visual scene generated by a 3D car simulator as input to their evolved active vision neural network. In that case, the main thing it learned was to follow the edge of the pavement. Learning a good Pac-Man strategy, however, is a rather different kind of challenge.

A promising direction currently being investigated is using the connectivity of the nodes in the maze as the basis for a modular recurrent neural network. The idea is that the current location of the objects (pills, power-pills, ghosts and Pac-Man) will propagate a signal-vector through the neural maze-model, where the features of the vector and its propagation characteristics would be evolved.

Another interesting issue is that of the best space for evolution to work in. It is not clear that neural networks are the most appropriate structures for evolving good behaviours. It may be that expression trees or function graphs involving logical, comparator and arithmetic functions are a better space to work in, since much of the reasoning behind location evaluation might be reducible to making distance comparisons between various objects in the maze and acting accordingly on the basis of those comparisons.

There is currently a strong trend in evolving neural network game players where fitness is a function of game performance. By contrast, reinforcement learning techniques such as temporal difference learning utilise information available during the game to update the network parameters, which can ideally lead to faster and more effective learning than evolution [10].

Regarding ghost behaviours, the current ghosts tend to crowd together too much, making the game too easy for a human player, at least when slowed down. A future version will use ghosts that repel each other, in order to spread out more. Also, a very natural development would be to co-evolve ghost strategies, where the fitness is assigned to a team of ghosts whose objective is to minimise the score that the Pac-Man can obtain.

6 Conclusions

This paper described an approach to evolving Pac-Man playing agents based on evaluating a feature vector for each possible next location given the current location of the Pac-Man. The simulation of the game retains most of the features of the original game, and where there are differences, our version is in some ways harder (such as the ghosts traversing the tunnels at full speed).

As expected, whether the ghosts behave deterministically or not has a significant effect on the type of player that evolves, and on the typical score that players achieve. The non-deterministic version is much harder, since a strategy must be learned which is generally good under lots of different circumstances.

On the other hand, one of the features of simulated evolution is its ability to exploiting any loopholes in the way a system has been configured. Evolution does indeed exploit the deterministic game in this way. The neural network weights, together with the maze and the ghost behaviours, dictate in a complex and indirect way the exact route that the Pac-Man will take, and hence the ultimate score obtained. While it is impossible to foresee the effects that even minor changes in neural network weights will have on the way the game unfolds, without actually simulating it, the fitness function cares only about the final score, and exploits any changes that happen to be advantageous.

While it is harder to evolve controllers for the non-deterministic game, it is a more interesting domain to study, since evolved controllers should now encode generally good strategies, rather than particular routes. However, the non-determinism causes high-levels of noise, which makes evolutionary progress more problematic. This would be less of a problem if the game consisted only of eating pills, but eating multiple ghosts can cause major differences in the score that could be due either to differences in strategy, or just luck.

Ms. Pac-Man is an interesting and challenging game to evolve controllers for, and much work remains to be done in this area. This paper explored the use of a purely reactive neural network for controlling the Pac-Man, but there are many other possible approaches, and it would also be interesting to investigate how game-tree search techniques could be adapted to cope with the large state space of the game.

Acknowledgements

I thank the anonymous reviewers and the members of the Natural and Evolutionary Computation group at the University of Essex, UK, for their helpful comments on this work.

References

[1] H.-G. Beyer. Toward a theory of evolution strategies: The (mu, lambda)-theory. *Evolutionary Computation*, 2(4):381–407, 1994.

[2] J. S. D. Bonet and C. P. Stauffer. Learning to play Pac-Man using incremental reinforcement learning., 1999. `http://www.ai.mit.edu/people/stauffer/Projects/PacMan/`.

[3] K. Chellapilla and D. Fogel. Evolving an expert checkers playing program without using human expertise. *IEEE Transactions on Evolutionary Computation*, 5:422 – 428, (2001).

[4] D. Floreano, T. Kato, D. Marocco, and E. Sauser. Coevolution of active vision and feature selection. *Biological Cybernetics*, 90:218 – 228, 2004.

[5] M. Gallagher and A. Ryan. When will a genetic algorithm outperform hill climbing? In *Proceedings of IEEE Congress on Evolutionary Computation*, pages 2462 – 2469. 2003.

[6] J. Koza. *Genetic Programming: on the programming of computers by means of natural selection*. MIT Press, Cambridge, MA, (1992).

[7] J. Laird. Using a computer game to develop advanced AI. *Computer*, 34:70 – 75, 2001.

[8] M. Mitchell, J. Holland, and S. Forrest. When will a genetic algorithm outperform hill climbing? In J. Cowan, G. Tesauro, and J. Alspector, editors, *Advances in Neural Information Processing Systems 6*, pages 51 – 58. Morgan Kaufman, San Mateo, CA, 1994.

[9] J. B. Pollack and A. D. Blair. Co-evolution in the successful learning of backgammon strategy. *Machine Learning*, 32:225–240, 1998.

[10] T. Runarsson and S. Lucas. Co-evolution versus self-play temporal difference learning for acquiring position evaluation in small-board Go. *IEEE Transactions on Evolutionary Computation*, page Submitted August 2004.

[11] J. Schaffer, D. Whitley, and L. Eshelman. Combinations of genetic algorithms and neural networks: a survey of the state of the art. In *Proceedings of COGANN-92 – IEEE International Workshop on Combinations of Genetic Algorithms and Neural Networks*, pages 1 – 37. IEEE, Baltimore, (1992).

[12] G. Tesauro. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68, 1995.