2008

Using NEAT for Continuous Adaptation and Teamwork Formation in Pacman

Markus Wittkamp

Luigi Barone

Philip Hingston
Edith Cowan University

This Conference Proceeding is posted at Research Online.
https://ro.ecu.edu.au/ecuworks/735

Using NEAT for Continuous Adaptation and Teamwork Formation in Pacman

Mark Wittkamp, Luigi Barone, *Member, IEEE*, and Philip Hingston, *Senior Member, IEEE*

Abstract— Despite games often being used as a testbed for new computational intelligence techniques, the majority of artificial intelligence in commercial games is scripted. This means that the computer agents are non-adaptive and often inherently exploitable because of it. In this paper, we describe a learning system designed for team strategy development in a real time multi-agent domain. We test our system in the game of Pacman, evolving adaptive strategies for the ghosts in simulated real time against a competent Pacman player. Our agents (the ghosts) are controlled by neural networks, whose weights and structure are incrementally evolved via an implementation of the NEAT (Neuro-Evolution of Augmenting Topologies) algorithm. We demonstrate the design and successful implementation of this system by evolving a number of interesting and complex team strategies that outperform the ghosts' strategies of the original arcade version of the game.

I. INTRODUCTION

Games are often used as a test-bed to further the development of artificial intelligence (AI) techniques. Games are suitable in this respect because they involve similar problems to those encountered in real life, but are simpler and more clearly defined. They have a finite number of rules and actions for players to make and there is some well understood goal. There are extra challenges facing players in video games, compared to traditional games (board and card games for example). Video games have a far greater number of actions available for players to make and these actions can have temporal significance.

Developing adaptive behaviour has been demonstrated using opponent modeling together with evolutionary algorithms [1], [2], but the problem becomes much more complicated when we add the requirement for this to be done in real time, during play. In a simple example, Quinn et al [3] have witnessed the real time evolution of cooperative and coordinated behaviour for a team of robots; achieving the goal of moving to a new location while staying within sensor range of each other.

Contrast real time learning with offline learning, where artificial players practice (generally by playing games) to become better players for future games. When an artificial player is able to learn a strategy offline, the amount of CPU time available is near limitless. The learning and fine tuning of artificial players can run continuously for days or weeks.

Mark Wittkamp and Luigi Barone are with the School of Computer Science & Software Engineering, The University of Western Australia, 35 Stirling Highway, Crawley, Australia (phone: +61-8-6488-1944; fax: +61-8-6488-1089; email: {wittkamp,luigi}@csse.uwa.edu.au). Philip Hingston is with the School of Computer and Information Science, Edith Cowan University, 2 Bradford Street, Mount Lawley, Australia (phone: +61-8-9370-6427; fax: +61-8-9370-6100; email: p.hingston@ecu.edu.au).

For real time adaptation, the time allowed for learning is much less. Not only must the learning yield results quickly enough so that adaptive behaviour can be achieved during play, but only a portion of the CPU time will be available due to the game's own running requirements. Further, computational intelligence techniques require many iterations and many more test cases for the evolution process to yield desirable results and so, speeding up of such techniques is of great importance.

II. LIMITATIONS IN VIDEO GAME AI

Despite a large amount of research in the field of video game AI, the majority of AI strategy in commercial games is scripted [4]. Developers resort to scripts because they are understandable, predictable, easy to modify and extend, and are usable by non-programmers [5]. While scripts can respond to the actions of human players, behaviours will be the same time and again. Game developers sometimes use adaptive learning techniques during development, but learning is rarely included in the released product [6]. This results in artificial opponents with weaknesses that, once discovered, are easily exploited. Predetermined behaviour also leads to repetitive and boring artificial players. While stochastic systems can be employed to add some variety into the behaviour of artificial players, they generally offer only slight variation to some predetermined strategy. Too much variation has the potential for seemingly random or irrational behaviour. *Appearing* random may not necessarily imply an ineffective strategy, but it can adversely affect the human player's immersion in the artificial environment.

Another limitation of current game AI is that the teams of opponents are often self-interested. While a good individual may be useful for a team, this is very different from team-interested individuals who prioritise the good of the team over personal good. Without team based learning, artificial players will in some respect always be "greedy". No matter how well the individual parts may be tuned, certain team strategies may never arise — a self-interested individual would not sacrifice himself to draw fire away from comrades, or to lead opponents into an ambush. Team based learning is also useful where the goal to be accomplished is too large or complex to be achieved by individuals without team coordination — RoboCup soccer [7] for instance.

This paper explores the use of computational intelligence techniques for real time learning in the game of Pacman. Focusing on team-work development, we examine how these techniques can be used to evolve strategies for the ghost agents in the game. Making use of continuous short-term

learning to regularly update ghost strategies, we introduce a novel framework that parallelises offline strategy learning with actual game play; constant adaptation over short time periods meaning the ghosts do not need to learn complex general strategies to be used in all game situations.

The rest of the paper is structured as follows. Sections III and IV provided background material that further motivate the problem and introduce the salient features of the underlying technologies used. Section V introduces our learning system for the game, discussing how our approach can be used for real time adaptation and team strategy development. Section VI details the AI used in the original arcade version of the game, reporting statistics that will be used as a baseline comparison for our work. Section VII then reports on experiments with a number of different fitness metrics; results indicating our system is able to evolve players that yield emergent team-work capable of superior performance to the AI used in the original arcade version. Finally, Section VIII summarises and concludes the work.

III. Neuro-Evolution of Augmenting Topologies

Evolutionary algorithms (EAs) are a powerful tool for designing and training neural networks. Recently, Stanley and Miikkulainen developed a new evolutionary algorithm called *Neuro-Evolution Through Augmenting Topologies* (NEAT) [8], which incrementally evolves the topology and weights of neural networks simultaneously.

The NEAT algorithm has been applied to a number of interesting problem domains, yielding some very impressive results. For example, the application of NEAT to the game of Othello saw the development of the sophisticated mobility strategy as a required step towards defeating alpha-beta search [9]. The development of this strategy, even at an intermediate level, suggests that NEAT has the ability to seek out weaknesses in opponents and develop strategies to exploit them.

NEAT has a number of features making it an attractive choice for our desired application area — that is, real time learning, where learning speed is important. While NEAT has been applied to real time learning in games in the past [10], most previous work has focussed on self-interested individuals and not team-based strategy development that we seek for this work.

NEAT utilises *speciation* to avoid premature convergence to suboptimal solutions. Potential innovations are protected, giving newer structures a better opportunity to develop rather than being discarded early on in favour of existing, more developed, structures. This is done by allowing individuals to compete primarily against other members of their species rather than with the entire population. The number of offspring allowed per species is proportional to the average fitness of that species.

Bloat is the name given to the problem that results from individuals' structures becoming unnecessarily large (i.e. without an increase in performance), leading to slower execution of the EA and individuals that can not be further optimized. NEAT has a number of features to avoid bloat.

It has a historical marking mechanism avoids the crossing of sections arbitrarily from highly varying structures. NEAT builds up from an initial population consisting of minimal networks without hidden nodes and structural complexity grows incrementally via mutation; complexity is only added if it yields a fitness advantage. NEAT's use of speciation/niching also aids in ensuring smaller structures are kept in the population as long as they are competitive. Should a species' members become bloated, these members will split off and form a new species. Only if this new species exhibits superior performance will the original species begin to die off.

IV. The Game of Pacman

The human player's goal in the video game Pacman [11] is to navigate *Pacman* through a maze and progress to the next level by collecting (*eating*) all the *pellets* and *power-pills* in the maze (see Figure 1 for the default Pacman level map). There are four opposing *ghosts*, who try to stop Pacman by chasing him down and eating him. In the centre of the maze is a *hideout* area that Pacman is unable to enter. At the beginning of each level, one ghost begins just above this hideout while the remaining three venture out one after another every two seconds.
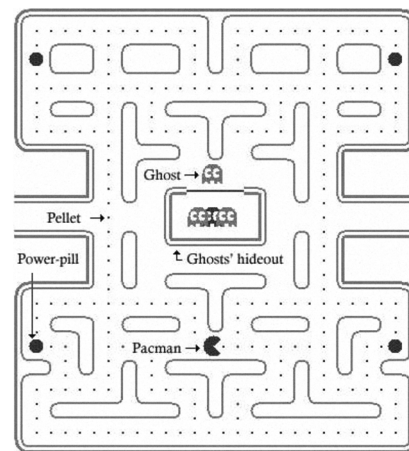

Fig. 1.   Default Pacman level map

Pacman begins the game with three lives and loses one each time he is eaten by a ghost. The game is over when Pacman runs out of lives.

Typically, the ghosts chase Pacman throughout the maze in an attempt to restrict his progress with the threat of consuming him and taking one of his lives. When Pacman eats a power-pill however, the situation is reversed and for a limited amount of time, Pacman is able to eat the ghosts and the ghosts cannot harm him. When eaten by Pacman, the ghost is consumed and loses its body, at which point it can not be eaten again nor pose any threat to Pacman until it is *restored*. The ghost must make a trip back to the hideout in order to restore itself. We define these three different ghost states as: *chasing* (pursuing Pacman), *fleeing* (evading Pacman when Pacman has consumed a power-pill), and *returning* (returning back to the hideout to be restored).

Other than a game of survival, there is also a point scoring system to Pacman. The consumption of pellets, power-pills, and the bonus items that appear on occasion throughout the game all give Pacman points. Eating ghosts also rewards Pacman with points, increasing exponentially for each ghost eaten while still under the effects of that power-pill. When Pacman reaches certain scores, he is awarded an extra life. Despite its simplicity, the game of Pacman provides an interesting environment for potentially very complex team strategy development as the ghosts need to cooperate together to "trap" Pacman in order to kill him. In this paper, we work with a modified version of Pacman (in part, to be more true to the original arcade) based upon a Java applet version by Ben Chow [12].

V. A TEAM LEARNING SYSTEM FOR PACMAN

In this work, we aim to construct an environment where a number of ghosts are able to learn as a team in real time to exploit Pacman's weaknesses. The learning scheme for the ghosts is designed to work in parallel with the execution of the Pacman game, however, for this paper, we run it in "simulated real time" where we simply pause the game in progress and allow NEAT to take over. The work reported here is intended to be a proof of concept for a complete real time implementation of our system.

The representation we use for ghosts is a feed-forward neural network which we evolve through the use of the NEAT algorithm — we have disallowed NEAT from producing recurrent links. Note that we are using the original NEAT algorithm, not the rtNEAT extension. We actually run four separate instances of NEAT with separate populations, only the best of which ever becoming active participants in the real Pacman game.

For the experiments reported in this work, we hand-coded a *pacbot* to play the Pacman game and act as a training partner to our team of ghosts. The pacbot is a decent player, capable of completing the first level almost every time when faced against the default ghost strategies (described in Section VI).

A. Precomputed information

Pacman levels are made up of a number of adjacent cells, each of which is either a pathway or a wall. Pacman, as well as the ghosts, pellets, power-pills, and bonus items may occupy these pathways.

We wish to concentrate on developing high level game play and avoid complicating the problem with lower level tasks such as navigating around walls, finding intersections, and so on. We are aware of a study that evolved a Pacman playing agent using a very small set of raw inputs and was able to produce a basic player, but its ultimate ability was hampered by having to learn how to avoid walls [13]. Our interest lies in the team-work and game strategies that can be evolved, rather than evolving ghosts from raw game data or minimal information.

Ghosts process the level map as a graph made up of a series of interconnected nodes that correspond to the pathway cells that the ghosts are able to reach. The initial positions of all pellets and power-pills are stored and updated in the environment model as the game progresses. We also precompute an all-pairs shortest-path table for every node in the level map, but we store only the length of these paths. Finally, we also allow our ghosts access to a precomputed table of shortest-path lengths from each cell to its nearest intersection — that is, any node that has more than two connecting nodes.

B. Learning structure

We use a neural network as the representation of an individual ghost strategy, which we evolve by running a series of simulated games in between the progression of the real game. We aim to have real time learning which learns only the specific team behaviours necessary to do well in the short-term, allowing for a high level of adaptivity which we hope will overcome the lack of a more generalised game playing strategy. Using evolutionary selection pressure that rewards an individual ghost based on the performance of the team, we use NEAT to train our ghost strategies.

To allow for heterogeneous strategy development, we keep a number of populations in memory, divided up amongst the four separate ghosts. A simplistic approach might be to allow each ghost to have its own population, however, as relative distance to Pacman is more likely to decide what strategy to employ than a ghost's arbitrary colour, we use a population scheme based on a ghost's proximity ranking to Pacman instead.

In each time slice, a ghost is marked for training (cycling through the four different ghosts in turn) and then allowed to evolve from the population corresponding to its proximity to Pacman (i.e. closest, second closest and so on). A time slice is about as long as it takes for a ghost to travel a distance of 15 cells (approximately three to four seconds of game time). Section VII report experiments with different fitness schemes; results showing that the fitness scheme has a major impact on the strategies evolved by the learning system.

We begin by initialising a number of randomly generated neural network populations, each corresponding to ghosts classified according to their distance to Pacman. Let G_n be the state of the game at the beginning of time slice n. During these time slices and at their boundaries, a number of tasks must be performed. The general ghost training algorithm is as follows (see also Figure 2):

1) Mark a ghost for learning during current time slice, beginning at G_n.
2) Look ahead (based on our models of the other ghosts and Pacman) and store the game state as we expect it to be like at the beginning of the next slice through simulated play (eG_{n+1}). This will be the starting state for the NEAT simulation runs.
3) The fitness of a ghost strategy is determined by evaluating the game state that we expect to reach when the strategy is used in place of the marked ghost (eG_{n+2}). This evaluation is an evaluation of the end state, and we experiment with various fitness schemes.

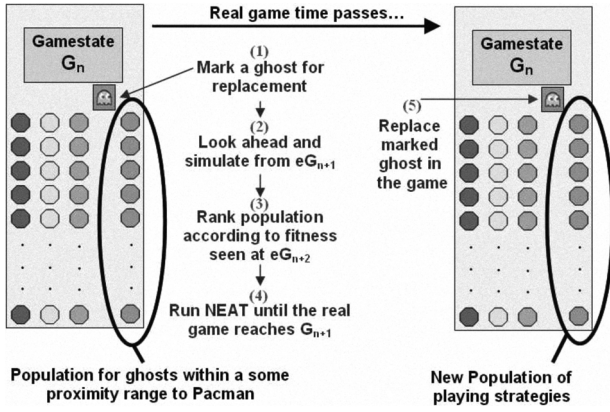*2008 IEEE Symposium on Computational Intelligence and Games (CIG'08)*

Fig. 2. Pictorial representation of our learning system

4) In parallel to the running of the actual game, run NEAT until the actual game reaches G_{n+1}.
5) The best individual from the simulations is substituted into the game, replacing the marked ghost.
6) Repeat the process for the next ghost in turn.

Our aim in this work is to determine the feasibility of the proposed approach for learning in Pacman. Instead of restricting NEAT to a limited amount of time (point 4 above), in this work, we allow the marked ghost to learn by simulating play with a perfect Pacman player model for a definite number of generations (20 generations). We maintain four separate populations (each of size 10) to allow for variation between ghost strategies. Our eventual aim is for this learning to occur in parallel, and hence it will be limited by the time available while the game is in play (the time slice). For the moment however, we simply pause the game's progress and allow the simulations to run sequentially.

The accuracy of the estimated next game state depend on the accuracy of the Pacman player model. The estimates for these experiments will be close to perfect because we are providing NEAT with a perfect model (the game itself has slight variation due to processor load and interleaving of events). The ghosts are deterministic and have perfect knowledge of each others' behaviour and so will not introduce any error into game state estimates.

C. Ghost neural network

We aim to evolve a neural network that acts as a move evaluator for the cells adjacent to a ghost. This approach has been successfully tried before, albeit to control Pacman rather than ghosts [14]. Our network is applied up to four times (once for each adjacent cell); the cell with the highest evaluated score is where the ghost will move to next. The neural network we construct will therefore only have one output value — the score of a cell. The score provides a measure of desirability of the resulting game state if the training ghost moved to the cell under consideration. An alternative approach uses a neural network with four outputs, representing the value of moving up, down, left or right in each situation, as in [15]. The advantage of our approach is

that the network must only learn to output a single value.

An approach where we evaluate only a ghost's adjacent cells may at first seem to be a very localised and "greedy" choice. However, when we consider that the inputs to the neural network contain precomputed high-level information as described in Section V-A, it becomes clear that the few cells explicitly under consideration are capable of encompassing information from many other cells.

Allowing our neural networks to receive their inputs as higher-level information allows complex behaviour to be very simply represented by the ghosts' neural network. For example, consider a neural network with a single input: the shortest-path distance to Pacman. For a ghost to chase Pacman, all that would be required is a positive weight connecting this single input to the output.

We selected neural inputs to give the ghosts sufficient information to be used as fundamental building blocks for general ghost strategy development. There are 19 inputs in total, which we hope will be sufficient to allow for complex and diverse team strategies to develop. Many of these 19 neural network inputs are based on information we found useful when constructing the pacbot player.

The full list of inputs is listed below. To keep the list concise, we first introduce some terminology. The inputs are written from the point of view of the ghost using the neural network as if that ghost was positioned on the cell that is currently being evaluated. We use the word *this* to refer to the current ghost (the one whom the neural network is being used to control). Recalling the three different ghost states (chasing, fleeing, or returning), we define a function called $status(g)$ that returns $\{-1, 0, 1\}$ respectively depending on the state of ghost g. As the ghosts need information about their relative positions to key objects in the game, we use a function called $closest(t, o)$ that returns the closest object of a particular type t to the object o (e.g. $closest(Powerpill, Pacman)$ represents the closest power-pill to Pacman). We refer to the length of the (Manhattan) shortest path from points a and b as $dist(a, b)$.

Each evolving ghost strategy has access to the following information in the form of its neural network inputs:

1) $status(this)$
2) $status(closest(Ghost, this))$
3) $status(closest(Ghost, Pacman))$
4) $dist(this, Pacman)$
5) $dist(this, closest(Ghost, this))$
6) $dist(Pacman, closest(Ghost, this))$
7) $dist(Pacman, closest(Ghost, Pacman))$
8) $dist(this, closest(Powerpill, this)$
9) $dist(Pacman, closest(Powerpill, this))$
10) $dist(this, closest(Powerpill, Pacman))$
11) $dist(Pacman, closest(Powerpill, Pacman))$
12) $dist(this, closest(Pellet, this))$
13) $dist(Pacman, closest(Pellet, this))$
14) $dist(this, closest(Pellet, Pacman))$
15) $dist(Pacman, closest(Pellet, Pacman))$
16) $dist(this, closest(Intersection, this))$
17) $dist(this, closest(Intersection, Pacman))$
18) $dist(Pacman, closest(Intersection, this))$
19) $dist(Pacman, closest(Intersection, Pacman))$

D. Performance assessment

After a simulated game run has finished, the performance of the ghosts is evaluated as a whole and this group score is used to evaluate the variable component during that simulation — i.e. the ghost currently in training.

Ideally, we would like the fitness to be measured by some high-level means (for example, the amount of time Pacman managed to stay alive, or Pacman's achieved score). This would minimise the bias in our evaluation metric. The problem with a high-level evaluation metric is that most of the time there would be very little, if any, selection pressure for the evolutionary algorithm to use. If only the death of Pacman is given as positive feedback, a mostly flat fitness landscape results, with little search gradient to guide the evolutionary search. Useful feedback will only be forthcoming if a strategy is found that is good (or lucky) enough to eat Pacman in the first place — but it does not provide a means of comparing all the group behaviours that *failed* to eat Pacman.

The aim of a fitness function is to guide the evolutionary algorithm toward learning effective team strategies. We tried a number of fitness functions, as outlined in the corresponding experiments in Section VII. All are designed to reward individual ghosts based on the performance of the team they are a part of (global reinforcement), as opposed to directly evaluating an individual solely by their own performance (local reinforcement).

VI. ORIGINAL GHOST AI IN PACMAN

In order to show the benefits of our learning system for evolving good ghost strategies, we compare the performance of strategies evolved using our approach to the strategies used in the original arcade game of Pacman. Here we describe the main elements of the AI used to control the ghosts in the original version.

In the original Pacman game, the ghosts alternate between the two modes of play (*attack* and *scatter*) several times until eventually remaining in a permanent state of attack [16]. Each of the four ghosts has its own unique attacking strategy, and each heads toward their own "home" corner when scattering.

When a ghost reaches an intersection, it decides upon a *target cell*. The ghost then moves in the direction to minimise its horizontal or vertical distance to that target (whichever is greater). When attacking, the red ghost's target is the current position of Pacman. The pink ghost's target is the cell four positions in front of Pacman. The blue ghost's target is the position such that the cell two positions in front of Pacman is the midpoint between Pacman and the red ghost's target. The orange ghost will target Pacman when it is far away, but will target its home corner instead when within a vertical and horizontal distance of eight cells to Pacman. Ghosts revert to a pseudo-random behaviour when under the effect of a power-pill [17].

TABLE I
PERFORMANCE OF THE PACBOT VERSUS THE ORIGINAL GHOST AI

Score at end of first level:	4808.4
Lives lost during first level:	1.44

A. Performance of the Original Ghost AI

To measure the success of our learning system, we compare the performance of our evolved strategies against a simulation of the original Pacman ghost AI described above. Table I reports this performance, averaged over 25 runs to compensate for the non-determinism in the game. We report the score at the end of the first level, and the number of lives lost in completing the level — when either the pacbot has eaten all pellets and power-pills or has died three times.

These results confirm the competence of the pacbot, which successfully completed the first level 22 out of 25 runs. The score obtained by the pacbot in completing the first level (4808.4) is significantly greater than the "baseline" score of 2700 (the score obtainable by just eating pellets and power-pills), indicating that the pacbot uses the power-pills to eat ghosts to maximise its score. If we continue the game to completion (repeating the level until the pacbot has lost all three of its lives), the pacbot obtains on average a final score of 7930.8, roughly equivalent to that of a novice human player. This pacbot was designed to be a "safe" player — its primary goal is to stay alive and pass the level; achieving a high score is a secondary concern.

VII. EXPERIMENTAL FRAMEWORK AND RESULTS

In this section, we report experiments with various fitness functions used in our evolutionary learning system and discuss our observations.

A. Experiment 1 — Chasing and Evading Pacman

In our first experiment, we aim to learn a simple chasing and evading strategy for the ghosts. We use an evaluation metric that rewards killing Pacman and then a secondary reward for minimising the distance of chasing ghosts to Pacman and maximising the distance of fleeing ghosts to Pacman. We use the secondary measure to resolve ties between solutions with the same performance on the first measure. The metric we use is:

$$\text{rank } 1: \quad \text{Pacman's number of lives}$$
$$\text{rank } 2: \quad \Sigma_{i=1}^{n} dist(Ghost_i^c, Pacman)$$
$$-\Sigma_{i=1}^{n} dist(Ghost_i^f, Pacman)$$

where $Ghost^c$ denotes a chasing ghost and $Ghost^f$ denotes a fleeing ghost.

A comparison of the performance of the pacbot when played against our learning system using this evaluation function and the original ghost AI is shown in Table II. As for all the experiments, results are averaged over 25 independent runs. Italicised entries are statistically significantly different

TABLE II

PERFORMANCE OF THE PACBOT VERSUS THE LEARNING SYSTEM FOR
EXPERIMENT 1

	Score	Lives lost
Original AI	4808.4	1.44
Experiment 1	4127.6	1.12

(using a two-tailed student t-test at the 0.05 level) to the original AI.

These results show that the pacbot obtained a statistically significant lower score than against the original AI, but not a statistically significant difference in the number of lives lost. While the ghosts have seemingly learned to successfully evade Pacman (thus reducing the score the pacbot was able to obtain from eating ghosts), we did not observe the increase in the kill-rate of the opponent ghosts we sought, thus suggesting some other mechanism to reward killing Pacman is needed in the fitness function.

Inspection of the phenotypic behaviour of the resulting collection of ghost strategies demonstrates how the collection behaves as a team. We observe that each ghost quickly learns to chase Pacman, vigorously pursuing the pacbot as it moves through the maze. We do note however that the ghosts often cluster together, typically within a few cells of each other, and often approach Pacman from the same direction. This clustering behaviour reduces the overall performance of the team, effectively reducing the team of four ghosts to a team of one or two ghosts, reducing the ability of the team to "trap" Pacman by approaching from different sides (which is ultimately what is needed for a successful kill). An example of this behaviour is shown in Figure 3.
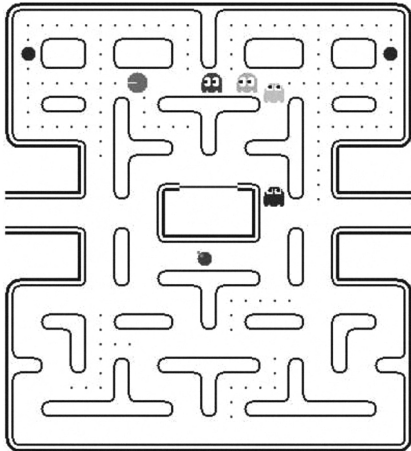

Fig. 3. Clustering of ghosts as they chase Pacman

This clustering behaviour may explain the reduced score obtained by Pacman — the pacbot is not surrounded by ghosts and hence is less likely to find a nearby ghost to eat after eating a power-pill (it must always proceed in the one correct direction to kill the ghosts) We also observe that being so clustered together, the ghosts have a reduced ability to kill Pacman (fewer effective ghosts makes trapping Pacman less likely). This seems to indicate that the system would

benefit from a more directed fitness function that promotes dispersion among the ghosts.

Examination of the resultant ghost behaviours also highlights a beneficial feature of our approach. By using continuous short-term learning, the system does not need to learn complex general game-playing strategies that will be used in all situations in the game, and instead need only learn "simple" strategies suitable for the time slice of use. Indeed, since the fitness function evaluates the behaviour (phenotype) of the ghosts rather than the decision network (genotype) that results in this behaviour, any strategy that manifests with good behaviour during that time slice will suffice. That is, the search for a rewarding behaviour is made easier by the many-to-one relationship that exists between genotype and phenotype over a time slice.

For example, when learning continuously, a ghost never actually needs to evolve a specific genotype that explicitly encodes a "chase Pacman" strategy; all that is required is a genotype that (for the current time slice) dictates the ghost moves towards Pacman. An example of this behaviour is shown in Figures 4 and 5.
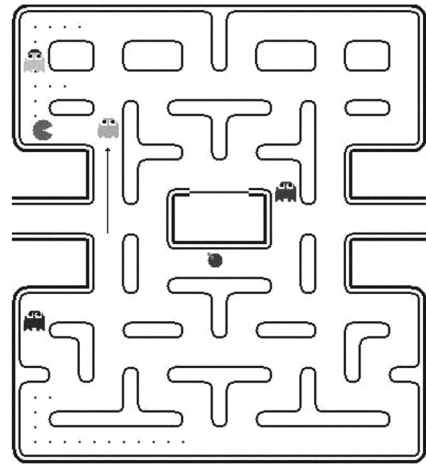

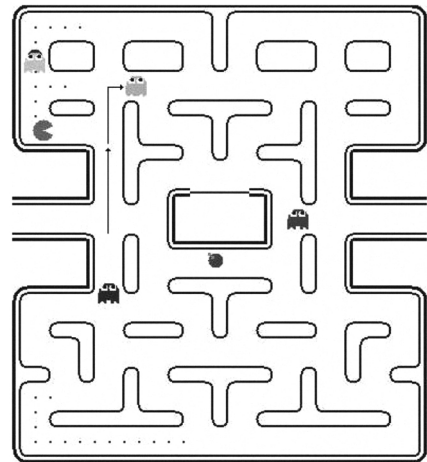Fig. 4. A ghost chasing Pacman during its time slice

Fig. 5. Continued game play of the ghost strategy from Figure 4

Figure 4 plots the trajectory of one ghost during a time

	Score	Lives lost
Original AI	4808.4	1.44
Experiment 1	*4127.6*	1.12
Experiment 2	4930.8	1.52

slice. At first sight, the ghost appears to be chasing Pacman. However, when we extend play beyond that time slice, it becomes evident in Figure 5 that the behaviour encoded by the strategy is not one of chasing Pacman, but some other strategy entirely. So, while the system may not be able to learn good general strategies for the game, through the use of continual adaptation, the system is able to "simulate" a general strategy without the need for offline learning.

B. Experiment 2 — Remaining Dispersed

In Experiment 1, we found that ghosts often cluster together, limiting their influence over the maze. To overcome this problem, we extend the performance evaluation metric to consider the dispersion of ghosts. As before, we use an evaluation function with a tiered approach over a number of separate metrics:

$$rank\ 1:\quad Pacman's\ number\ of\ lives$$
$$rank\ 2:\quad min_{i=1}^{n}\, dist(Ghost_i^c, Pacman)$$
$$-max_{i=1}^{n}\, dist(Ghost_i^f, Pacman)$$
$$rank\ 3:\quad -\Sigma_{i=1}^{n}\, dist(Ghost_{i-1}^c, Ghost_i^c)$$
$$-\Sigma_{i=1}^{n}\, dist(Ghost_{i-1}^f, Ghost_i^f)$$

where $Ghost^c$ denotes a chasing ghost and $Ghost^f$ denotes a fleeing ghost.

The first tier evaluation is used to reward ghosts that kill Pacman. The second tier rewards how close the *closest* chasing ghost is to Pacman, and how far the *furthest* fleeing ghost is from Pacman. Note that this differs from Experiment 1, since a reward for minimising the distance to Pacman for *all* ghosts would counter-act our desire for dispersion. The third tier promotes dispersion directly by rewarding ghosts that are more distant from other ghosts of the same status than closer ghosts.

A comparison of the performance of the pacbot when played against our learning system using this evaluation function and the original ghost AI is shown in Table III.

Although the difference is not statistically significant, these results suggest that this evaluation function produces a team of ghosts that does not perform as well as those produced in Experiment 1.

As mentioned in the previous section, a network of dispersed ghosts is easier for the pacbot to score against (consume) after eating a power-pill. This may seem counter-intuitive to those familiar with game (human players often attempt to force the ghosts to bunch together and allow for easier mass-consumption), however, the pacbot does not view a bunch of ghosts as a hugely profitable target, but a minor goal relative to completing the level. With more dispersed ghosts, it is more likely that the pacbot will encounter a ghost while collecting pellets and then proceed to consume a nearby ghost. In contrast, a tight cluster of ghosts require the pacbot to choose one of a limited number of "critical" paths that intersect with the cluster. Of course, should the pacbot successfully intersect the cluster, he is richly rewarded, however experimental results indicate that the "conservative" strategy scores better on average than the "higher-risk-higher-reward" strategy.

As expected, the ghosts in this experiment evolved a behaviour leading them to organise themselves in a dispersed manner, while minimising the shortest distance of the closest chasing ghosts to Pacman. As the fitness function only rewards chasing behaviour if a ghost is able to become the closest ghost, what tends to evolve is team behaviour in which the separate ghosts "sit" in very dispersed locations (the ghosts actually oscillate back and forth on the spot due to the game restriction that ghosts must remain in motion), with the single closest ghost actively pursuing Pacman through the maze.

Recall from the discussion of Experiment 1, we outlined a problem with evolved strategies being kept in play for longer than their intended time slice. A side effect of this leads to desirable behaviour in this experiment — Pacman often becomes trapped by two ghosts approaching from opposite sides due to the "lag" in updating the different ghosts. An example of this behaviour is shown in Figure 6.
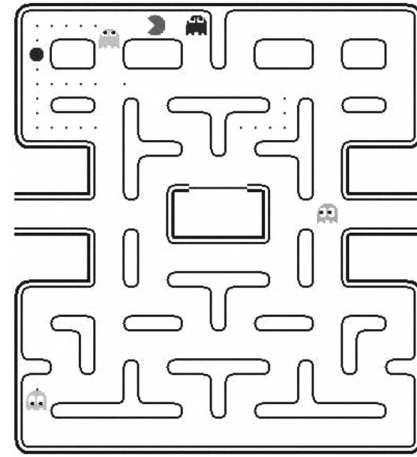

Fig. 6.   Ghosts trapping Pacman

C. Experiment 3 — Protection Behaviour

The aim of this experiment is to see if we can evolve strategies where ghosts protect either a vulnerable team mate or a power-pill/pellet. To achieve this, we alter the evaluation metric for this experiment by rewarding strategies that minimise both the number of ghosts that have been eaten and the number of vulnerable (fleeing) ghosts:

TABLE IV

PERFORMANCE OF THE PACBOT VERSUS THE LEARNING SYSTEM FOR
EXPERIMENT 3

	Score	Lives lost
Original AI	4808.4	1.44
Experiment 1	*4127.6*	1.12
Experiment 2	4930.8	1.52
Experiment 3	4271.6	1.64

TABLE V

PERFORMANCE OF THE PACBOT VERSUS THE LEARNING SYSTEM FOR
EXPERIMENT 4

	Score	Lives lost
Original AI	4808.4	1.44
Experiment 1	*4127.6*	1.12
Experiment 2	4930.8	1.52
Experiment 3	4271.6	1.64
Experiment 4	4494.4	*1.96*

rank 1 : Pacman's number of lives

rank 2 : $count(Ghost^r)$

rank 3 : $count(Ghost^f)$

rank 4 : $min_{i=1}^n dist(Ghost_i^c, Pacman)$
$-max_{i=1}^n dist(Ghost_i^f, Pacman)$

rank 5 : $-\Sigma_{i=1}^n dist(Ghost_{i-1}^c, Ghost_i^c)$
$-\Sigma_{i=1}^n dist(Ghost_{i-1}^f, Ghost_i^f)$

where $Ghost^c$ denotes a chasing ghost, $Ghost^f$ denotes a fleeing ghost, $Ghost^r$ denotes a returning ghost, and *count* is a function that returns the number of objects of a particular type.

A comparison of the performance of the pacbot when played against our learning system using this evaluation function and the original ghost AI is shown in Table IV.

The performance results reported in Table IV suggests that this evaluation function goes some way toward promoting successful team-work that leads to killing Pacman, though the difference from Experiment 2 is not statistically significant.

We have witnessed three examples of different protection behaviour emerging from the system, although they do not commonly occur due to the specific situations required for such an opportunity to arise:

1) A chasing ghost moving towards a vulnerable ghost to protect it from Pacman.
2) A vulnerable ghost moving towards a chasing ghost to protect itself from Pacman.
3) A chasing ghost moving toward a power-pill to prevent Pacman from eating it.

An interesting (and obviously unplanned) emergent behaviour that we observed was that of suicidal behaviour, where in certain situations, ghosts would run straight into Pacman! This occurred because the vulnerable ghosts learned that killing themselves could yield a reward if they were able to make it back to the hideout before the time slice was up. Similarly, we saw some strange shepherding behaviours where a training ghost would learn to behave in a way that would cause suicides in the other (fixed) ghost strategies, who obviously had their actions influenced by the actions of the ghost in training. This was an exciting result as it demonstrates the level of complex team behaviour that we hoped we would see.

D. Experiment 4 — Ambushing Pacman

Furthering the idea of trapping Pacman, in our final experiment, we define a fitness function that attempts to directly encode this concept. To achieve this, we include the number of maze intersections that are "controlled" by Pacman — controlled in the sense that Pacman can reach the intersection before any chasing ghost can. We also only consider intersections within a threshold distance of 20 from Pacman, intersections further away than this are ignored. This controlled-intersection value captures the "freedom" of Pacman; the lower the number, the lower the number of options Pacman has in order to escape from being killed.

Using the evaluation function of Experiment 2, we add as a second tier metric a measure that explicitly captures this value, thus driving the evolutionary process to reward strategies that limit the number of escape routes available to Pacman. We also add a fifth measure to the evaluation function that captures Pacman's score, attempting to drive Pacman to the less valuable areas of the maze, prolonging the game and increasing the likelihood of forcing Pacman into a trap. The evaluation function we use in this final experiment is:

rank 1 : Pacman's number of lives

rank 2 : Intersections "controlled" by Pacman

rank 3 : $min_{i=1}^n dist(Ghost_i^c, Pacman)$
$-max_{i=1}^n dist(Ghost_i^f, Pacman)$

rank 4 : $-\Sigma_{i=1}^n dist(Ghost_{i-1}^c, Ghost_i^c)$
$-\Sigma_{i=1}^n dist(Ghost_{i-1}^f, Ghost_i^f)$

rank 5 : Pacman's score

where $Ghost^c$ denotes a chasing ghost and $Ghost^f$ denotes a fleeing ghost.

A comparison of the performance of the pacbot when played against our learning system using this evaluation function and the original ghost AI is shown in Table V. The results clearly indicate that this evaluation function is strong in maximising the kill-rate of Pacman, doing better than all previous evaluation functions and the original ghost AI on this measure.

We also continued the runs of Experiment 4 to game end (repeating the level until Pacman dies three times), recording an average final score of 4853.2 for the pacbot in play against our learning system using this evaluation function compared to 7930.8 against the original ghost AI; results statistically

significantly different using a two-tailed student t-test at the 0.05 level. This further demonstrates the superiority of our learning system over the original ghost AI, producing more difficult opponents than the original ghost AI, at least for our hand-coded pacbot player.

From examination of the evolved strategies, we observe that the ghosts tend to attack as a group (thus limiting the number of intersections controlled by Pacman), and importantly attack from different directions (again as a consequence of the desire to restrict escape routes, but also to ensure dispersion). Attacks are not as "directed" as in Experiment 1 (the ghosts are driven to remain dispersed and therefore do not simply head straight for Pacman), but they appear more "structured", surrounding Pacman to limit escape routes before proceeding to ambush Pacman.

The evolution of structured team-work is indeed an impressive result. Recall, the system was not provided with any expert guidance to help organise the ghosts to work effectively together. This emergent behaviour is exactly what is sought in this work — a system capable of learning and adapting to a player's style without direction from a human designer.

VIII. CONCLUSIONS

This paper has proposed a novel system for real time team strategy development using computational intelligence techniques for the game of Pacman. Our approach reduces the problem to be solved by the ghosts' neural network to one of game-state valuation — each ghost using their neural network to decide on the worth of each adjacent cell in order to select the next best cell to move to. This is made possible, in part, through the precomputed high-level data we make available to the neural networks.

As a proof of concept, we have implemented the system in simulated real time, with learning occurring in (pseudo) parallel to actual game play. Using continuous short-term learning, each ghost in the team is subjected to evolutionary selection pressure in order to learn a good strategy for use in the next time slice. Time slices are deliberately kept short, thus allowing the system to learn simple strategies that suffice for a short period of time and not complex general game-playing strategies to be used in all game situations.

Our approach of using multiple evolving populations depending on the proximity of the ghost to Pacman has allowed us to evolve very interesting and complex team behaviours with noticeable role development. Results show that our system is able to evolve players that yield emergent team-work capable of superior performance compared to the AI used in the original arcade version of the game.

Furthermore, our system has developed this team behaviour using a small population and a small number of generations for learning per time slice. This gives us confidence that our goal of implementing the system in true real time will show similar successes.

It is expected that this work will lead to a fully functional implementation capable of delivering real time team strategy development in the near future. Future work will also examine the importance of the accuracy of the opponent model in learning, and explore ways of using a multi-objective approach for simultaneously evolving different strategies that satisfy the different metrics of interest without an explicitly stated ordering of such metrics.

REFERENCES

[1] M. Wittkamp and L. Barone, "Evolving adaptive play for the game of spoof using genetic programming," in *Proceedings of the 2006 IEEE Symposium on Computational Intelligence and Games*. IEEE Publications, 2006.

[2] M. Wittkamp, L. Barone, and L. While, "A comparison of genetic programming and look-up table learning for the game of spoof," in *Proceedings of the 2006 IEEE Symposium on Computational Intelligence and Games*. IEEE Publications, 2006.

[3] M. Quinn, L. Smith, G. Mayley, and P. Husband, "Evolving teamwork and role allocation with real robots," in *In Proceedings of the 8th International Conference on The Simulation and Synthesis of Living Systems (Artificial Life VIII)*, 2002.

[4] L. Berger, "Scripting: overview and code generation," in *AI Game Programming Wisdom*. MIT Press, 2002, vol. 1, pp. 505–510.

[5] P. Tozour, *The Perils of AI Scripting*. Charles River Media, Inc., 2002.

[6] D. Charles, C. Fyfe, D. Livingstone, and S. McGlinchey, *Biologically Inspired Artificial Intelligence for Computer Games*. Medical Information Science Reference, 2007.

[7] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, and E. Osawa, "RoboCup: the robot world cup initiative," in *Proceedings of the First International Conference on Autonomous Agents (Agents'97)*. ACM Press, 5–8, 1997, pp. 340–347.

[8] K. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary Computation*, vol. 10, no. 2, pp. 99–127, 2002.

[9] T. Andersen, K. Stanley, and R. Miikkulainen, "Neuro-evolution through augmenting topologies applied to evolving neural networks to play Othello," Department of Computer Sciences, University of Texas at Austin, Tech. Rep.

[10] K. Stanley, B. Bryant, and R. Miikkulainen, "Real-time neuroevolution in the NERO video game," *IEEE Transactions of Evolutionary Computation*, vol. 9, no. 6, pp. 653–668, 2005.

[11] T. Iwatani, *Pac-man*. Namco, 1980, http://en.wikipedia.org/wiki/Pac-Man.

[12] B. Chow, "Java Pac-man implementation," http://www.bennychow.com/play_pacman.shtml.

[13] M. Gallagher and M. Lewdich, "Evolving Pac-man players: can we learn from raw input?" in *Proceedings of the 2007 IEEE Symposium on Computational Intelligence and Games*. IEEE Publications, 2007.

[14] S. Lucas, "Evolving a neural network location evaluator to play Ms. Pacman," in *Proceedings of the 2005 IEEE Symposium on Computational Intelligence and Games*. IEEE Publications, 2005.

[15] G. Yannakakis and J. Hallam, "Evolving opponents for interesting interactive computer games," in *Proceedings of the 8th International Conference on the Simulation of Adaptive Behavior (SAB'04); From Animals to Animats 8*. MIT Press, 2004, pp. 499–508.

[16] M. Hanshew, "Pac-man FAQ/strategy guide," July 2006, http://www.gamefaqs.com/coinop/arcade/file/589548/439591.

[17] Twin Galaxies Forums, "Pac-man ghost behavior revealed," August 2008, http://www.twingalaxies.com/forums/viewtopic.php?t=12231.