Evolving diverse Ms. Pac-Man playing agents using genetic programming

Conference Paper · October 2010

DOI: 10.1109/UKCI.2010.5625586 · Source: IEEE Xplore

2 authors, including:

Simon Lucas
University of Essex
303 PUBLICATIONS **6,916** CITATIONS

Evolving Diverse Ms. Pac-Man Playing Agents Using Genetic Programming

Atif M. Alhejali and Simon M. Lucas
Game Intelligence Group
School of Computer Science and Electronic Engineering
University of Essex, UK
{amalhe,sml}@essex.ac.uk

Abstract — This paper uses genetic programming (GP) to evolve a variety of reactive agents for a simulated version of the classic arcade game Ms. Pac-Man. A diverse set of behaviours were evolved using the same GP setup in three different versions of the game. The results show that GP is able to evolve controllers that are well-matched to the game used for evolution and, in some cases, also generalise well to previously unseen mazes. For comparison purposes, we also designed a controller manually using the same function set as GP. GP was able to significantly outperform this hand-designed controller. The best evolved controllers are competitive with the best reactive controllers reported for this problem.

Keywords: Pac-Man, Genetic Programming, Evolving Controllers.

I. INTRODUCTION

Artificial intelligence (AI) has a rich history of being applied to games, with some very notable successes such as Deep Blue, a chess playing machine that defeated world-champion Garry Kasparov in 1997[1]. Deep Blue was hand-programmed and serves as an outstanding example of what can be achieved with clever heuristic evaluation functions coupled with a highly efficient game-tree search. While this approach works well for chess, other games such as Go have proved to be far more elusive, although amazing progress has been made over the last few years with Monte Carlo tree search (MCTS) methods, which have also proved to be highly successful in general game playing (GGP).

The focus of this paper is on using video games as a test-bed for machine learning research. The aim is to gain a greater understanding of which methods work best for particular games and why. We choose Ms. Pac-Man because it is a classic, well-known arcade game for which it has proven to be very difficult to develop strong AI players. Pac-Man was introduced in 1980 by Namco[2]. In the next year, Ms. Pac-Man was released as a second version of the game with significant changes such as the behaviour of the ghost team. In the original game, the ghosts behave in a deterministic way, which means that the player can achieve high scores by following a set of learned routes. In Ms. Pac-Man, the ghosts behave non-deterministically, which makes the game more challenging and means that it cannot be played well by following fixed routes.

Our chosen computational evolution method is genetic programming (GP). Since its initial development by Cramer[3] and popularisation by Koza[4], GP has become a widely studied field, and GP research is regularly featured in human-competitive machine intelligence challenges[1]. It is, therefore, interesting to investigate how well GP performs in the challenging problem of developing an AI controller for Ms. Pac-Man.

II. PREVIOUS WORK

Pac-Man was one of the test-problems described in Koza's first GP book[4], but the results were obtained for a much simplified version of the game with a generous scoring system, and the evolved controllers were not really very smart. His work can be used as a foundation for the GP-based evolution of a Pac-Man controller because he used all of the game's basic components in his simulator and fundamental functionality in his GP. Gallagher and Ryan[5] used an even simpler version than Koza's with only one ghost and without any power pills. The agent was presented as a finite-state machine that was evolved using an evolutionary rule-based approach. Another work with a simplified simulator was conducted by De Bonet and Stauffer[6], who used reinforcement learning to evolve an agent. Thompson, McMillan, Levin, and Andrew[7] compared a greedy-look-ahead agent to a greedy-random agent, a random agent, and a human player. The human player scored the highest score, followed by the greedy-look-ahead agent. Their work proved that the non-deterministic ghost movements reduce the impact of the luck factor. Lucas[2] evolved a reactive neural network agent. The agent operated by using the neural network as a position evaluator to score the value of each of the neighbouring nodes to the agent's current position and choosing the neighbouring node with the highest value. He tested the system in both deterministic and non-deterministic simulated versions of the game. Developing a competitive agent for the non-deterministic game was more challenging than for the deterministic version. Gallagher and Ledwich[8] used a

[1] www.genetic-programming.org/hc2009/cfe2009.html

neural network to create an agent in several adapted versions of the original game with from one to four ghosts that behaved deterministically and non-deterministically. Wirth and Gallagher[9] used an influence map model to create a controller for Ms. Pac-Man. Their controller achieved a higher maximum score than the human player they tested, while the average score was lower. Robles and Lucas[10] used a simple tree search method to create a Ms. Pac-Man agent. They were able to achieve an average score of 9,630 in the original game using a screen capture-based controller that played on the original arcade version but over 40,000 on the simulated version of the game used in this paper.

Some researchers tackled Pac-Man from another perspective; Luke and Spector[11] and Wittkamp, Barone, and Hingston[12] used genetic programming and the neural network algorithm NEAT (neuro-evolution of augmenting topologies), respectively, to create a team of ghosts.

Many other researchers used Pac-Man as a tool to study the various phenomena related to computational intelligence. Rosca[13] studied generality versus size in genetic programming as applied to his experiment on Pac-Man. Ohno and Ogasawara[14] aimed to provide a cognitive model that could estimate human performance in a computer operation task while emphasising the bidirectional interactive (BDI) tasks, which are the tasks by which a computer interacts with the user by changing the environment visually and audibly. DeLooze and Viner[15] combined fuzzy logic with reinforcement learning in a method known as fuzzy q-learning to develop an intelligent agent in a non-deterministic environment represented as Ms. Pac-Man.

Genetic programming has also been used with many other games. One of the major games that have been tested with GP is football. Bajurnow and Ciesielski[16] used layered learning to evolve a goal-scoring behaviour. They compared the layered learning GP to the standard GP in their paper. Luke[17] used GP to evolve soccer-playing agents for a RoboCup97 competition. Langdon and Poli[18] used GP to create a controller for solo Pong. Their experiment included a human player and hand-coded algorithms, including an optimal player and evolved players. The best scores came from a combination of the optimal player and GP. Ebner and Tiede[19] used GP to evolve driving controllers for the racing game TORCS (The Open Racing Car Simulator).

III. THE GAME

In this section, we give a brief description of the game and, in particular, explain the key points where our simulation differs from the original. The simulation we used in this paper was similar to the one used by [2] but with some extensions, the main difference being the inclusion of all four original Ms. Pac-Man mazes in the current version. The simulator is available from the *Ms. Pac-Man Agent Versus Ghost Team Competition*[2] Web site.

From a competitive point of view, the aim of the game is to get the highest score given a series of games. Based on the results of several Ms. Pac-Man competitions[3], it usually happens that the winner of the competition (defined as the highest score given ten games) is usually also the player with the best average score over those games. The simulator offers a choice of ghost teams, but all of the results in this paper used the "legacy team"[2].

The game unfolds as a series of levels. Each level is defined by the following characteristics:

The points scoring scheme is as follows:

- Food pill: 10 points. There are around 200 food pills in each maze, e.g. the first maze has 222.
- Power pill: 50 points. Each maze has 4 power pills.
- Edible ghosts: after each power pill is consumed, the ghosts are edible for some time and are worth 200, 400, 800, and 1600 points for each one consumed. Each time a pill is consumed, the next ghost eaten is worth 200 points. After a ghost is eaten, it then goes back to the lair to regenerate in its normal predatory state. If all four ghosts are eaten after a power pill, the score for these totals 3,000. Hence, 12,000 points per level are available for eating all of the ghosts after each power pill.

The simulator also differs from the original game in some other ways. The speed of the ghosts is the same throughout the different levels and the edibility time is also the same. The speed of Pac-Man and the ghosts do not change eating (pills, ghosts, or Pac-Man), cutting corners, or crossing tunnels. The simulation used here does not contain the fruit that appears several times during the game allowing the agent to receive bonus points for eating it.

IV. EXPERIMENTAL SETUP

A. *The game setup*

Three versions of the game were used here depending on the number and type of mazes.

1. Single level: the first experiment was with one level only using the first maze from the game. The game terminates when all of the pills have been eaten or when the agent is eaten. This configuration should favour agents who are highly skilled at eating all of the ghosts, since agents who consume only the pills will inevitably achieve low scores. The maximum score achievable is 14420.

[2].http://csee.essex.ac.uk/staff/sml/pacman/kit/AgentVersusGhosts.html

[3].http://csee.essex.ac.uk/staff/sml/pacman/PacManContest.html

2. Four mazes: the next experiment used all four mazes from the original game but terminated the game when all four had been cleared. The maximum score for this setup is 58,160.

3. Unlimited levels: the third experiment used the first maze but repeated it each time it was cleared. Hence, an agent could achieve an infinitely high score with this setup.

In all of the games, the Ms. Pac-Man agent had only one life.

The function set used was created based on John Koza's experiment[4] with some major amendments. All of the functions and terminals that he used were included in addition to the extensions that he recommended and a few more designed by the author. Many of the features use shortest-path distance calculations, but as explained previously, the simulator pre-calculates these.

The GP system used is an extended version of simple GP (SGP) by the second author. Simple GP is a minimal Java implementation of GP that may soon be made freely available and was written with the aim of having a simple but easily extensible structure. To give an idea of how minimal the system is, the size of the zip file containing all of the source code together with a graphical tree-rendering application and a toy test problem is only 16k. While this is twice the size of TinyGP, it is significantly easier to extend with new functions. The extension used for this paper is to make it strongly-typed[20]. Each node has a defined output type and a defined type for each of its children. For example, some of the functions require a comparison between two children to choose the target branch. This comparison cannot be done between two terminals that should take the agent to a target. They need to be between two special nodes (or sub-trees) that return information about the current status of the game.

Thus, the function set was divided into three groups: functions, data-terminals, and action-terminals (terminals). The design of these is still a work in progress, and the current set of functions could be justly criticised for being too specific to particular items of data rather than taking possible data items as parameters. Each function has its arity (number of arguments) in parentheses.

A. *Functions*:

Iflte (4): "If-Less-than-Else" takes four arguments (children) and compares the first two. If the first is less than the second, then the third child will be run; otherwise, the fourth branch is the choice. The first two arguments have to be data-terminals, while the last two must be action-terminals.

The rest of the functions are IF conditions. Each function runs the required condition and applies the first child if it returns TRUE or the second if FALSE.

IsEdible (2): a power pill has been recently eaten and some of the ghosts are still edible.

IsInDanger (2): compares the current position of Pac-Man to the current position of the ghosts. If the distance between Pac-Man and one or more ghosts is less than a specific number (10 in this test) then Pac-Man will be considered to be in a dangerous position. Note that this definition of being in danger is flawed, and true danger depends, of course, not only on the distances to the ghosts but also their relative positions.

IsEnergizersCleared (2): all of the energizers in the maze have been eaten.

IsToEnergizerSafe (2): this function checks every node in the shortest path between Pac-Man and the closest power pill. If the distance of any of these nodes to the nearest ghost is less than a specified amount, then the path is deemed to be unsafe.

B. *List of terminals*:

1) *Data-terminals*:

Most of the data terminals return the current distance of a component from the agent using the shortest path. The list of the terminals includes DISPill, DISEnergizer, DISGhost, DIS2ndGhost, DIS3rdGhost, DISEdibleGhost, and DISUnedibleGhost.

Constant: gives a constant random number between 0 and the largest distance between two nodes in the maze. This number can be used in distance comparisons. Once generated, it remains constant (also called ephemeral random constant).

2) *Action-terminals*:

Each action terminal will move Pac-Man one step toward the target, which is a component in the maze. At each time step, the whole situation is evaluated and a new node may be targeted depending on the current status. The action terminals include ToPill, ToEnergizer, ToEdibleGhost, FromGhost, FromEnergizer, and ToSafety**.**

ToSafety: this terminal was hand-coded to give Pac-Man the ability to attempt to escape from a deadly situation. It uses the available information to navigate through all of the possible routes starting from the current agent position. At each junction, the current route is duplicated and each new route will continue in a different direction. The algorithm keeps a record of the visited nodes so it does not retreat on itself. This process continues along each route. For each route, every node is tested to make sure that it is safe for Pac-Man to use this route. If the positions of the ghosts compromise a route, the navigation process will stop in this route. The route size will be restricted to a maximum number of steps. If a route reaches its maximum size, the navigation process will stop as well. This restriction is to prevent the process from being too

expensive. When all of the routes are tested, they will be sorted depending on their size to make sure that Pac-Man can choose the route with the least interference from the ghosts or at least the one that can give it the longest time before facing a ghost. If more than one route is rated equal in this way, then they are sorted on the average distance between the furthest point in the route and the four ghosts, and then if more than one route has the same rank, they will be sorted depending on the their minimum distance from the closest ghost.

All of the experiments were done using the same GP parameters. The population size was set to 1000 and run for 50 generations. The crossover rate was 0.8, which means that 80% of the new offspring were created using crossover, while the other 20% were created using mutation. The parent selection method is a tournament with the selected parent chosen to be the individual with the highest fitness out of 3 that were randomly selected.

The fitness was the average score of the controller running in the simulator five times. Due to the non-determinism of the game, fitness evaluation is extremely noisy, and the value of five is chosen as a compromise between robustness and CPU time.

To guarantee that the best possible agent is chosen in each run, a score is selected for each stage depending on pre-runs to be an evaluator value. The fitness of each individual will be compared to this evaluator. All of the individuals that can pass this evaluation will be retested in the simulation to calculate a new fitness, which is the average of 100 runs. The best agent after this re-evaluation phase will be presented as the best agent of the whole run.

For comparison purposes a hand-coded agent was designed using the same function set. This agent starts by determining whether Pac-Man is in danger or not. The aim is to escape or gain points. If it is in danger, then, depending on its distance from the nearest power pill and safety of the route to this pill, Pac-Man will go toward this energizer or call the ToSafety method to escape from the current dangerous situation. If Pac-Man is not in immediate danger, then it will chase the nearest edible ghost if there is one and there are no ghosts in the route between the Pac-Man and this ghost. If there is a danger in this route, then Pac-Man will go to eat the nearest pill. If the ghosts are inedible, Pac-Man will eat the food pills until it gets very close to a power pill. If this is the case, then Pac-Man will move one step away from this pill. This enables the agent to loiter by a power pill waiting to eat it and pounce on the ghosts when they are sufficiently close.

V. THE RESULTS

In each experiment, the individual that scored the best average in the re-evaluation phase will be retested again in the testing phase. All three controllers and the hand-coded controller were tested with 4 mazes and four lives (3 lives from the beginning of the game and a fourth one when it scores over 10,000 points). The results were taken over 1,000 runs of each agent.

A) *The one-maze experiment*:

The evolution process in this experiment was the most stable among the three experiments, and the evolutionary progress is shown in Figure 1. In each run, there were one or more controllers that scored the maximum score on at least one of the five games in their fitness evaluation.

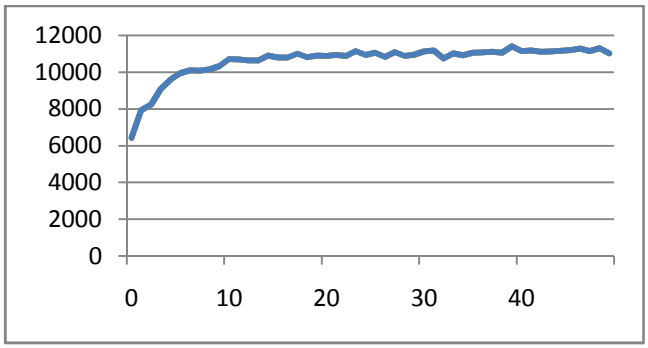

Figure 1. The average of the evolution process of best agents in the one maze experiment.

Studying the behaviour of the individuals chosen by the GP shows a lot of similarity between them. Pac-Man starts by pursuing the nearest energizer. After eating this energizer, it will start chasing and eating the edible ghosts. When it finishes eating all of these ghosts, it will start looking for the next power pill to eat and then chase the ghosts again. Once all of the power pills were eaten and all of the ghosts became inedible again, most of the controllers would lose interest in the game; some of them did not even pursue any target at all and just moved around the maze until they were killed. The most successful agents had a slightly better strategy. The selected controller that scored one of the best scores in the testing phase (15709.31 on average and a maximum of 44,000) had a different behaviour. It compared the distances between Pac-Man and the nearest edible and inedible ghost if there were any. If the edible ghost was closer or there were no inedible ghosts, it would chase this ghost to eat it. In the other side of the tree, which will be reached if the inedible ghost is closer or there are not any edible ghosts in the maze, the agent will check to determine if there is any immediate danger and focuses on eating the food pills if there are not any. If there are ghosts in threatening positions, the agent will go directly to the nearest power pill if the route is safe or start looking for a safe place using the ToSafety method.

B) *The four-maze experiments*:

In this stage, the agents continue to evolve until the last generation. All of the agents chosen by the GP in this stage were similar to the best agents proposed by the first experiment (chasing the ghosts when they were edible, clearing the maze when they were safe, and escaping to a safe route when it was in danger,

including going to the nearest power pill). Clearing the maze gave the agent the opportunity to increase its score in the next maze(s). The evolution of fitness is shown in Figure 2.

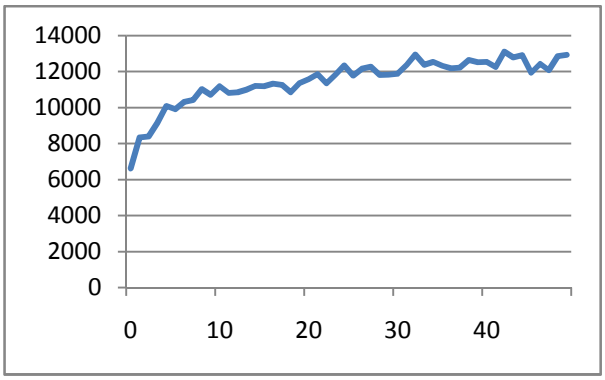

Figure 2. The average of the evolution process of best agents in the four-maze experiment.

The best agent chosen in this stage essentially has the previous behaviour. It checks to determine whether it is in danger at the beginning. If it is in danger, then it will go to safety if the path to the energizer is blocked by a ghost or to the energizer if the route is clear. If Pac-Man is not in danger, then it chases the edible ghosts if there are any, or it will start eating the food pills. The agent does not check the safety of the route when it chases an edible ghost but because the first question in the tree is whether Pac-Man is in danger, when a ghost becomes very close to Pac-Man, the state will change from chasing an edible ghost into looking for safety or an energizer. This behaviour causes the agent to risk its life when it is following an edible ghost.

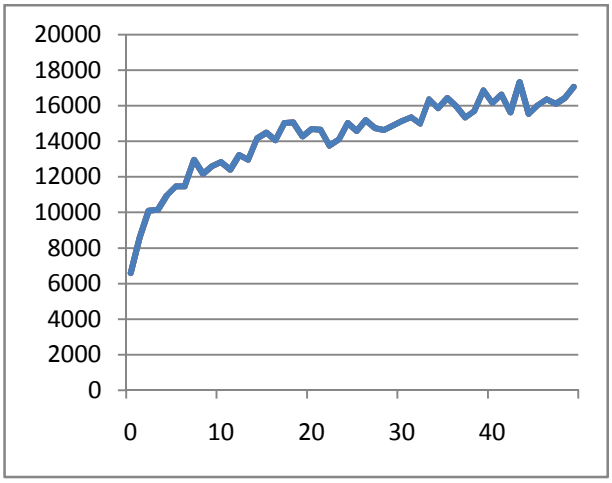

Figure 3. The average of the evolution process of best agents in the unlimited-maze experiment.

C) *The unlimited-maze experiment*:

This experiment produced agents that behave completely differently to the other two. Instead of focusing on chasing the ghosts, the GP found it more beneficial in this stage to clear as many mazes as possible. The behaviour that was noticed in all of the top scoring agents was based on clearing the pills while not in danger. When a ghost compromises the agent's safety, the agent seeks the nearest power pill if it is safe or goes to a safe path if it is not. After eating the power pill the Pac-Man will not use the edible time to eat ghosts but instead chooses to eat more pills in a safe environment. The evolution of fitness for this experiment is shown in Figure .

D) *The hand-coded agent*:

As explained previously, this agent was built to compare its result to the evolved figures. The average score of this controller during the testing phase was 14,607 with a maximum of 41,830 out of 58,160.

E) Summary of results:

Table I shows the scores that the best agents from each experiment achieved in the testing phase (average of 1,000 trials). All of the averages are significantly different (using a two-tailed un-paired *t*-test with $p > 0.99$) with the exception of the leading two controllers (1-maze and 4-maze). Table II shows the percentage breakdown of the level reached by each controller in the testing phase. It shows that the unlimited-maze controller (evolved in stage 2) has a significantly different behaviour from those of the others.

TABLE I

The scores of the four agents in the testing phase.

Agent	Min Score	Max Score	Average Score	Standard Deviation
1 maze	3,250	44,000	15,709.31	8,230.39
4 mazes	4,080	44,560	16,014.21	8,357.24
∞ mazes	2,300	20,760	11,143.31	3,710.14
HCoded	2,400	42,060	14,647.90	7,846.88

TABLE II

The numbers of agents that reached and died in each maze.

Agent	Maze 1	Maze 2	Maze 3	Maze 4	none Cleared all of the mazes
1 maze	50.3%	34%	13.8%	1.6%	0.3%
4 maze	48.4%	34.8%	13.6%	2.6%	0.6%
∞ mazes	1.4%	20.1%	30%	18.2%	30.3%
HCoded	51.4%	30.6%	15.4%	1.8%	0.8%

VI. DISCUSSION AND CONCLUSIONS

Although the first experiment (one maze) and the second experiment (four mazes) produced similar final agents, the four mazes gave agents with the same behaviour and close results in each run. The one-maze experiment produced this agent only once during the ten runs of the GP, while the rest of the runs evolved agents that were not able to clear the first maze because they did not have sufficient interest in the food pills. In the first maze, the pills are worth only 2,220 points, while the ghosts are worth up to 12,000 out of the

possible 14,220. This was not the case in the second stage, when clearing the maze was an essential condition to move on to the next maze, resulting in a significant impact on the final score. A test of the second-best individual produced by the one-maze experiment scored an average of only 9,566 and a maximum score of 20,680 in the testing phase.

The unlimited-mazes test was an example of the ability of GP to choose the best general behaviour for the problem at hand despite the noisy evaluation process. Because there was no limit to the mazes and the score and the mazes did not become harder at the advanced levels, the best behaviour was to focus on making the environment (maze) safe to allow Pac-Man to clear and then go to the next one.

Ms. Pac-Man continues to provide a rich test-bed for research into machine learning and games, and as far as we are aware, the best software agents tested on the screen-capture version of the game are all hand-programmed. The GP system reported in this paper is one of the highest-performing controllers that we are aware of that does not make extensive use of tree-search, but in the future, we will need to test this on the screen-capture version of the game.

In common with the vast majority of papers on GP, the "programs" evolved by GP in this paper were all constant-time expression trees with no iteration – a restriction imposed by our chosen function set and GP implementation. This significantly limits the level of innovation that the system can achieve, and future work will consider less-restricted versions of GP.

VIII. REFERENCES

1. Lucas, S. and G. Kendall, *Evolutionary computation and games.* IEEE Computational Intelligence Magazine, 2006. **1**(1): p. 10.
2. Lucas, S. *Evolving a neural network location evaluator to play Ms. Pac-Man*. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games*. 2005. Essex, UK: IEEE.
3. Cramer, N. *A Representation for the adaptive generation of simple sequential programs*, Proc. of an Intl. Conf. on Genetic Algorithms and their Applications, Carnegie-Mellon University, July 24-26, 1985.
4. Koza, J., *Genetic programming: on the programming of computers by means of natural selection*. 1992, Cambridge, MA: The MIT press.
5. Gallagher, M. and A. Ryan. *Learning to play Pac-Man: An evolutionary, rule-based approach*. In *The 2003 Congress on Evolutionary Computation*. 2003. Piscataway, NJ: IEEE.
6. Bonet, J. and C. Stauffer. *Learning to play Pac-Man using incremental reinforcement learning*. In *the Congress on Evolutionary Computation*. 1999.
7. Thompson, J., L. MacMillan, and A. Andrew. *An evaluation of the benefits of look-ahead in Ms. Pac-Man*. In *IEEE Symposium on Computational Intelligence and Games*. 2008. Perth, Australia.
8. Gallagher, M. and M. Ledwich. *Evolving Ms. Pac-Man players: Can we learn from raw input?* In *IEEE Symposium on Computational Intelligence and Games*. 2007: IEEE.
9. Wirth, N. and M. Gallagher. *An influence map model for playing Ms. Pac-Man*. In *IEEE Symposium on Computational Intelligence and Games (CIG'08)*. 2008. Perth, Australia: IEEE.
10. Robles, D. and S. Lucas. *A simple tree search method for playing Ms. Pac-Man*. In *The IEEE Symposium on Computational Intelligence and Games (cig'09)*. 2009: IEEE.
11. Luke, S. and L. Spector. *Evolving teamwork and coordination with genetic programming*. In *the First Annual Conference on Genetic Programming*. 1996: MIT Press.
12. Wittkamp, M., L. Barone, and P. Hingston. *Using NEAT for continuous adaptation and teamwork formation in pacman*. In *IEEE Symposium on Computational Intelligence and Games*. 2008: IEEE.
13. Rosca, J. *Generality versus size in genetic programming*. In *the First Annual Conference on Genetic Programming*. 1996: MIT Press.
14. Ohno, T. and H. Ogasawara. *Information acquisition model of highly interactive tasks*. In *ICCS/JCSS*. 1999. Charlotte, North Carolina, USA: Association for Information Systems.
15. DeLooze, L. and W. Viner, *Fuzzy q-learning in a nondeterministic environment: eeveloping an intelligent Ms. Pac-Man agent.*
16. Bajurnow, A. and V. Ciesielski. *Layered learning for evolving goal scoring behavior in soccer players*. In *Proceedings of the 2004 Congress on Evolutionary Computation (CEC2004)*. 2004: IEEE.
17. Luke, S., *Genetic programming produced competitive soccer softbot teams for RoboCup97*. Genetic Programming, 1998: p. 214-222.
18. Langdon, W. and R. Poli. *Evolutionary solo Pong players*. In *IEEE Congress on Evolutionary Computation*. 2005.
19. Ebner, M. and T. Tiede. *Evolving driving controllers using genetic programming*. In *IEEE Symposium on Computational Intelligence and Games*. 2009: IEEE.
20. Montana, D.J., *Strongly typed genetic programming*, Evolutionary Computation, vol 3, p. 199-230, 1999.