EvoTanks: Co-Evolutionary Development of Game-Playing Agents

Thomas Thompson, John Levine
Strathclyde Planning Group
Department of Computer & Information Sciences
University of Strathclyde
Glasgow, UK
(tommy@cis.strath.ac.uk, john.levine@cis.strath.ac.uk)

Gillian Hayes
Institute of Perception, Action and Behaviour (IPAB)
School of Informatics
University of Edinburgh
Edinburgh, UK
(gmh@inf.ed.ac.uk)

***Abstract* — This paper describes the EvoTanks research project, a continuing attempt to develop strong AI players for a primitive 'Combat' style video game using evolutionary computational methods with artificial neural networks. A small but challenging feat due to the necessity for agent's actions to rely heavily on opponent behaviour. Previous investigation has shown the agents are capable of developing high performance behaviours by evolving against scripted opponents; however these are local to the trained opponent. The focus of this paper shows results from the use of co-evolution on the same population. Results show agents no longer succumb to trappings of local maxima within the search space and are capable of converging on high fitness behaviours local to their population without the use of scripted opponents.**

Keywords: Genetic Algorithm, Games, Co-evolution, Neural Networks

I. Introduction

Despite the continuing advances in the development of intelligent agents across numerous applications, the field of video games is often considered unworthy for such methods. This notion is worth challenging, given that video games in general provide one of the best means to test and develop technologies in environments that are difficult and expensive to locate and generate. Video games provide researchers a means to create and control artificial environments of varying complexity; effectively an economic means to generate low risk testing scenarios. Applications such as autopilot systems for aircraft or ground vehicles can benefit from this, or the testing of robot controllers without the necessity to build the physical robot, a feat which could either have a significant price tag or not be possible given the current technology [1].

Such research has a hidden benefit; given that video games are a multi billion-dollar industry with millions of players playing a variety of games across the world, research can aid the development (and cost) of video games as well as enhance the playability of a particular game. The former provides a reason for game developers to show interest, while the latter is where such research is given consumer focus. For example *F.E.A.R. (First Encounter Assault Recon)* developed by Monolith Productions in 2005 was hailed by critics and gamers alike for the realistic environments and gameplay. One of the heavily contributing factors to *F.E.A.R* was the intuitive AI players that provided realism to the game that players craved. This acclaim aids in boosting the appeal of AI research for video games, highlighting the possibilities of intelligent agent research in powerful, realistic yet controllable environments.

At present however, there is still a large area for improvement, the majority of computer controlled agents in video games (referred to as *non-player-characters* or *NPCs*) are scripted i.e. their behaviour is controlled by a series of sequential actions which are typically performed in an infinite loop, thus ensuring that the agents are permanently active. Ultimately given sufficient time and effort made by the human player, any opponent can be defeated once the player has gained an understanding of how the NPC behaves. A video games' appeal will gradually wane due to the inability of an NPC to learn or adapt from previous games. This has been a drawback of video games for many years, if one were to take the likes of *Super Mario Bros.* released in 1985 and *Metal Gear Solid* in 1998, despite a difference of almost 15 years and an increase in the complexity of the enemy behaviour we still deal with predictable opponents [2].

Since computer games have now been a prominent entertainment medium for approximately 20 years, gamers in general are now more mature; either in age or their ability to deal with more complex problems in games. As a result it is required that games become more complex and engaging to maintain their ability to entertain. Intelligent NPC's can provide the means to keep games engaging.

We feel that the development of truly intelligent NPC agents requires 3 decision layers; a high-level goal directed layer that oversees the process of actions required to achieve

Manuscript received October 31st, 2006.

Thomas Thompson and John Levine are with the Strathclyde Planning Group, Department of Computer & Information Sciences, University of Strathclyde, Livingstone Tower, 26 Richmond Street, Glasgow G1 1XH, Scotland, UK

(E-mail: tommy@ cis.strath.ac.uk, john.levine@cis.strath.ac.uk).

Gillian Hayes, is with the Institute of Perception, Action and Behaviour (IPAB), School of Informatics, University of Edinburgh, James Clark Maxwell Building, King's Buildings Mayfield Road, Edinburgh, EH9 3JZ, Scotland, UK (E-mail: gmh@inf.ed.ac.uk).

goal conditions, a middle layer that deals with local goals and the breakdown of tasks into smaller actions and finally a low level component that deals with primitive and basic actions. Machine learning is one possible method for generating such low-level reactive mechanisms. A by-product of this is that we can generate successful yet unpredictable players.

Machine learning is often used to assist in training agents prior to the game being played [3], with applications of different methods available across a host of games such as the use of evolution in Pac-man [4], co-evolution in Texas Hold'em Poker [5], Backgammon [6] and Checkers/Draughts [7], the application of reinforcement learning to Backgammon [8] and real-time evolution in the NERO video game project [9]. The EvoTanks project follows in a similar vain to some of the research mentioned above, focussing on the application of evolutionary methods to generate interesting and unique low-level reactive agents for a small combat based environment. EvoTanks provides a game where we are dealing with making primitive actions to solve a local goal. However the actions an agent makes relies heavily on the opponent's behaviour to generate its own actions. As a result, finding high performance behaviours is an interesting challenge.

Previous research using the EvoTanks game investigated the possibilities of agents learning behaviours based on focussed trials against one particular opponent using an evolutionary algorithm. The results generated from these experiments were positive, with agents learning competent, interesting (and occasionally unconventional) behaviours to defeat the chosen opponent [10]. However their competence ended at said opponent, as the majority of agents were unable to perform as effectively against different opponents. This was due to the evolutionary process and fitness function moving the majority of agents towards local maxima within the fitness space; as a result these agents were capable of competing against only one opponent. The purpose of the research expressed in this paper, was to investigate the application of co-evolution methods to move the agents away from these local maxima, with the intent of developing strong generic players, capable of playing against a variety of different agents competently.

This research is appropriate given the nature of the game and the environment that this game is used. Sub optimal global strategies are common in video games, where we have the most difficult opponents in a particular game designed to compete against even the more advanced human players regardless of their particular strategy. Not only do NPC's develop such behaviours, but also human players tend to move towards such behaviour, playing games using particular tactics to evaluate and react to any situation regardless of how difficult the opponent becomes.

This paper first describes the EvoTanks game, followed by a description of the implementation made and an analysis of the results generated.

I. The EvoTanks Game

EvoTanks is based loosely on the game of 'Combat' released on the Atari 2600 in 1978, composed of two tank agents viewed in a top-down fashion within a 600 x 600 arena encompassed by boundaries. Only two agents exist within the arena at any given time and are privy to a selection of actions; forward/backward movement, left/right rotation and to fire a shell from the cannon. The cannon is dependant on the direction in which the tank is facing.

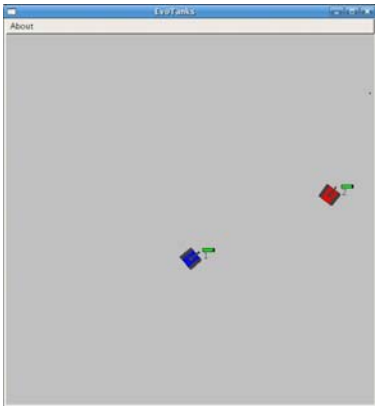

Fig. 1. The EvoTanks game with 2 agents competing with one another in the arena.

Each match between two tanks is given a time limit in which one tank must destroy the opponents 4 armour points with an unlimited amount of shells. A health point is deducted for every direct hit made by an enemy shell. Shells themselves will be destroyed if they come into contact with the boundaries of the arena. When a tank makes a move, each move results in a tank being moved a fixed distance across the arena, neither their own momentum nor the momentum of other tanks or shells have any effect on the tanks movement. Ultimately a game is complete once one tank has depleted all of the four hit points, at which point it explodes with the win given to the surviving tank. Should the timer reach zero and both tanks are still on the field, a draw is given regardless of the amount of health remaining on each tank.

The agents used in the learning process of the EvoTanks simulator use an unsupervised feed-forward artificial neural network (ANN) to control their autonomous reasoning. Each network is composed of 3 layers of neurons using a tanH transfer function. 3 normalised inputs from the domain inform the agent the difference in angle relative to the agent cannon from the enemy opponent and vice versa (hence 2 separate inputs) and finally the distance between the agent and the enemy. These 3 inputs help the agent to select one of 3 possible outputs, controlling movement, rotation and firing of the cannon. These agents were trained through the manipulation of the 27 connection weights contained within the neural network, with a genetic algorithm used to store the connection weight and evolving them using the EvoTanks simulator using an evolutionary algorithm.

To assist in the training and assessment of the agent tanks, we also have a collection of NPC's designed to provide means to build learning behaviours as well as evaluate how effective a particular agent is using a variety of strategies both defensive and offensive:

- *Sitting Duck:* A stationary agent designed to bring about basic homing and attack behaviours.
- *Lazy Tank:* Similar to the previous NPC with the exception of the cannon constantly firing.
- *Random Tank:* A tank that carries out movement, rotation or fire commands with equal probability.
- *Hunter:* An aggressive player that hunts its opponent down by constantly moving towards the opponent while firing continuously.
- *Turret:* A stationary player that can rotate the cannon and fire, providing a distant, strong offensive opponent that can be difficult to attack.
- *Sniper:* An evasive player that seeks to avoid its opponent by continuing to reverse away from the player whilst taking shots from a distance.

II. IMPLEMENTATION

A. Agent Representation

The agents are written in an object-oriented fashion in java, with each tank stored within an instance of a chromosome class, this data type contains the collection of network connection weights for the solutions controller. At the beginning of an evolutionary experiment, a population of chromosomes (and their genetic values) is generated randomly within a generational population model.

B. Fitness Assessment

Fitness evaluations are carried out in tournaments; a tournament consists of two teams of agents, each of whom must play all agents in the opposing team for a specified number of games. Each match is initialised with both agents in random positions facing random directions. Once each match is completed, the fitness of each agent in that match is calculated, with an average for their performance against a particular opponent made once the correct number of games is completed. An agent's overall performance is assessed by taking the average scores from competing against each player, generating what was considered to be a reasonable measure of the fitness. In testing, the number of tanks in the tournaments was modified to assess the performance of different sampling rates, i.e., the number of opponents an agent must face in order to be assessed for fitness.

The methods stated above assist in the evaluation of local fitness, i.e. the fitness a given chromosome has relative to the local population. However this value does not always reflect the agent's capabilities against scripted or human opponents. As a result, a supplementary evaluation was provided periodically throughout the evolution that selected each agent from the parent set of that generation to be evaluated against all NPC's equally. Thus allowing us to gain an understanding of how effective these agents were in the real game.

C. Fitness Function

Assessing the performance of a given agent was separated into two distinct areas, how efficiently an agent defeats an opponent and the amount of health remaining at the end of the battle:

$$F_{win} = \left(Win_{efficiency} \times 0.8\right) + \left(F_{P_a health} \times 0.2\right)$$
$$F_{lose} = \left(Lose_{efficiency} \times 0.8\right) + \left(F_{P_a health} \times 0.2\right)$$

1) Efficiency Component

Should an agent win a match against its opponent, the fitness is calculated by deducting a penalty for the number of time points taken (T_{game}) to complete the kill:

$$Win_{efficiency} = 1 - \left(\left(\frac{0.5}{T_{\max}}\right) \times T_{game}\right)$$

In a win scenario, an agent will always accrue a minimum 0.5 fitness for the efficiency component. This is only possible when the opponent takes the complete amount of time allocated to a match (T_{max}) to defeat the opponent. Consequently it is impossible for an agent to achieve an efficiency fitness of 1.0, ensuring that the agents are incapable of reaching the maximum fitness and cease exploring for better behaviours.

On the other hand, should the agent lose the match, a fitness value is measured as a bonus for each time point the agent managed to stay on the field:

$$Lose_{efficiency} = \left(\left(\frac{0.5}{T_{\max}}\right) \times T_{game}\right)$$

Hence the maximum fitness that could be attributed is 0.5 in the (unlikely) event the agent is killed at the very last time point.

In the event of a draw, the agent immediately receives a score of 0.5.

2) Health Component

The fitness component provides a 0.125 bonus for each of the agents 4 health points intact after a given match, plus a bonus for each of the 4 points deducted from successful shots on the enemy tank:

$$F_{P_a Health} = \left(P_a Health \times 0.125\right) + \left(\left(H_{\max} - P_b Health\right) \times 0.125\right)$$

This function allows for agents to gain strong scores for flawless victories against their opponents and also for agents who lose matches to gain some fitness if they were capable of damaging their opponent.

D. Evolutionary Structure

The evolution follows the canonical structure, however 2 veins of experimentation were conducted, one in which the selection of agents into the parent subset was dictated by a selection algorithm (tournament, roulette wheel and rank-based methods) or an alternative was a 'selection by evaluation' method. The latter filled the parent set by placing the agent with highest fitness from each tournament into the set until the parent quota has been filled.

1) Crossover

Results from previous EvoTanks research had shown that one-point crossover that blindly swapped subsets of weights was too disruptive to the neural networks to provide incremental improvement. An optional feature provided a new crossover method based on the implementation by Montana and Davis [11] that swapped the weights attributed to particular neurons provided they shared the same structure (i.e. same number of connections).

2) Mutation

Mutation was a mandatory component of the evolution process, using a random mutation algorithm that mutates the value of a particular gene within a ±1 range given a probability. A range of ±5 binds each weight and should the mutations result in weights exceeding these values they are immediately corrected to the closest value within bounds.

E. Neural Network Structure

Each agent uses a 12 neuron-network, with 3 neurons in both input and output layers followed by 2 hidden layers each containing 3 neurons, resulting in 27 connections across the entire network. This provides a small, manageable set of weights to evolve, with each weight bound within a ± 5 range.

Previous EvoTanks research opted for the use of a hyperbolic tangent (*tanH*) neuron transfer function due to the lack of bias nodes within the network, this function remained due to the successful results generated in previous experiments.

III. Results & Discussion

Two particular strains of research were investigated to see which could perform best. The first enforced the '*selection by evaluation*' method as previously discussed (*experiment A*), whilst the latter used traditional selection methods to generate the parent subset (*experiment B*). Initial results were disappointing, with a failure to generate a strong arms race dynamic which could push the population towards high fitness [2].

Further experimentation increased the sampling rate of the population to a maximum of 20 tanks (hence 20 tanks per team in a tournament), with results in *experiment A* using 10-tank sampling providing the best results. Showing a strong gradual increase in performance (showing in Fig. 2), whilst *experiment B* failed to reach the heights of its competitor with a much slower growth in fitness that failed to reach the same high fitness results given the number of evaluations permitted.

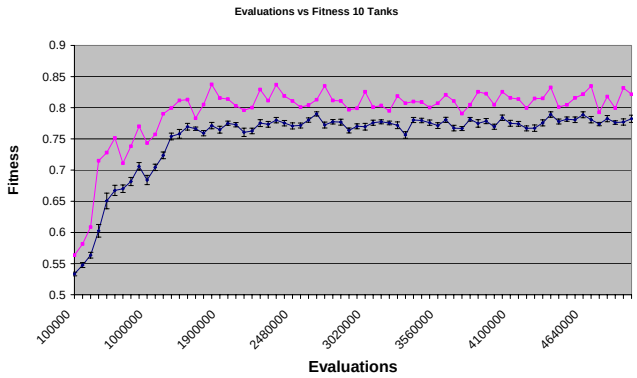

Fig. 2. The trends in best and average local fitness for the 10 tank sampling rates on *experiment A*. The agents initially continue to climb and then stabilize at a strong fitness on average greater than 0.75.

At this point further tests using *experiment A* were conducted, investigating the use of crossover, stable state population models and the modification of the size of parent set, population size and mutation probability. These experiments generated little difference from the initial results, with the exception of the steady state model that performed poorly in comparison due to the more gradual increase in fitness.

A final analysis compared the performance of the co-evolution simulation to 2 alternative methods. Firstly a 'ramped' evolutionary model, where a population of agents are evolved against all 6 NPC's in sequence. The first phase evolves against the sitting duck NPC, until 500,000 individual evaluations (i.e. games) have been performed. The evolution then switches in sequence to the lazy tank, the random tank, the hunter, the turret and the sniper, with 500,000 evaluations being performed for each NPC. Hence as evolution progresses we increase the difficulty of the competing NPC, with the intent of gradually evolving from a basic turn-and-shoot behaviour into something more aggressive. The second comparison measure was a direct hill climber using a 1 + 1 evolutionary strategy evaluating

against all opponents simultaneously, i.e. the fitness of the candidate is calculated by taking the average of the score gained against all 6 NPC's. Each method was given 3 million evaluations to generate their most effective agents.

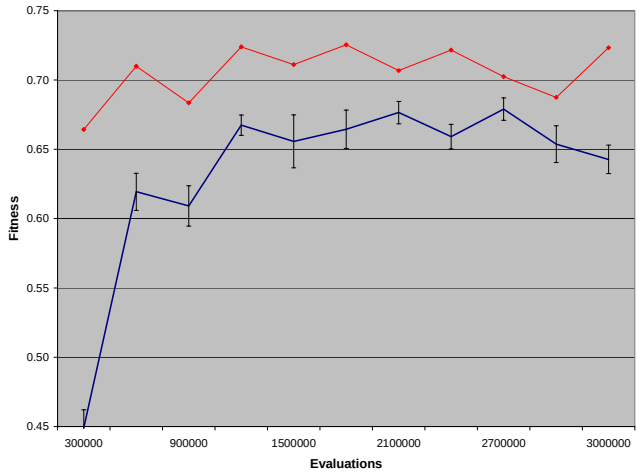

Fig. 3. This graph shows the trends in average and best fitness against NPC's throughout the final co-evolution run. With both showing reasonably strong values after 3 million games.

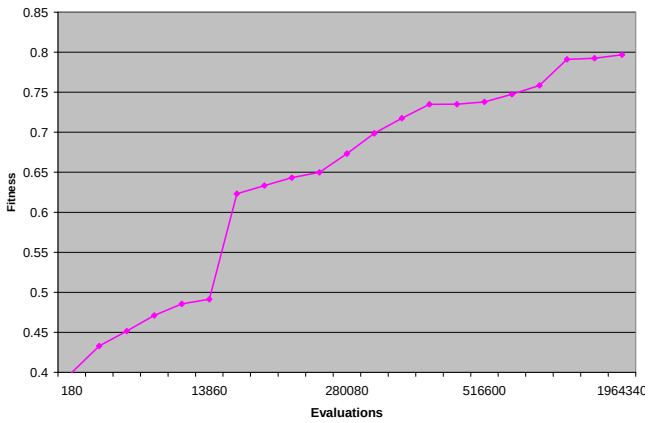

Fig. 4. The fitness trend of the hill climber, which developed a high performance agent within less than 2 million evaluations. These results scored even better than the co-evolution, resulting in a more efficient agent in less time.

As is shown in Table 1, the ramped evolution performed incredibly poorly, whilst the co-evolution (fig. 3) generated strong fitness values against the NPC's. Surprisingly the hill climber (fig. 4) was still capable of surpassing the performance of the co-evolution run, generating a final fitness that was almost 0.1 stronger than the co-evolution.

It was surprising to see the hill climber does so well in these tests and at this point we paused to consider why the hill climber performs so well for this domain. It is important to consider that the hill climber is a more direct method to assess an agent's position within the search space of behaviours. The search space presented may not be difficult to traverse but requires a lot of behavioural analysis to assess where any given agent exists. This analysis is provided by the performance against the NPC's, allowing us to gain a very strong understanding of where the agent is in the search space and how fit it is.

One must then consider whether it is worth using the co-evolution at all and continue onward using the hill climber? Statistically the hill climber performs better, with higher fitness results in a smaller period of time. We feel that relying solely on hill climbing would be ill advised for numerous reasons, primarily since a hill climber has a strong dependence on NPC agents. Hill climbers use the NPC's to evaluate the performance of the agent; as a result we are required to present a range of opponents that provide a strong coverage of that which the agent may face. In this experiment we have been fortunate in providing NPC's that facilitate this particular problem. Should the problem change and require new coverage, we cannot guarantee the appropriate behaviours to facilitate this.

The co-evolution can generate opponents of almost the same quality without the necessity of NPC's, allowing us to generate high quality opponents using only a randomly instantiated population. When one considers the impact the co-evolved population made, the co-evolution performs exceptionally well given that they have no mapping to the actual fitness space. Instead they continue to improve based upon a local fitness relative to the population. The assessment against NPC's provided a means to assess how well agents perform outside of the population.

We consider the final score of the hill climber to provide an upper bound on the fitness that can be achieved in this problem. The results from the co-evolution are very positive given their environment; since the ability to defeat NPC's was neither the focus of the co-evolution nor the means of assessing agent fitness. Despite this the best result from the population was only 0.1 from the upper bound.

Table 1. A table representing the best actual fitness values (assessed by running agents against the NPC's) after the final experiments. It is clear from this table that the evolution was incapable of generating any high fitness behaviours. The co-evolution performs well, with actual fitness values reaching a maximum of greater than 0.7 and a strong performance from the population altogether when compared against the hill climber, that provides a fitness upper bound slightly greater than 0.8.

Method	Best	Mean	S.D.	S.E.
Evolution	0.3153	0.3107	0.003228258	0.00072
Co-Evolution	0.711	0.642	0.0217	0.0048
Hill Climber	0.8103	N/A	N/A	N/A

The majority of high performance agents evolved competent behaviours ranging from highly aggressive strategies to more defensive tactics. One example of aggressive behaviour includes an agent that evolved the exact same properties as *Hunter* NPCs, attacking the opponent outright with little chance to evade or counter. A *Hunter* often wastes the first shot since they are still not positioned correctly to challenge their opponent, however these agents' behaviours are more tailored and as a result

carry out a much more efficient job than their NPC counterparts. One interesting feature was a defensive capability that backed away should it come into contact against another aggressive player. These tactics worked well against all opponents, especially the *Hunters* themselves, since the agents developed effectively a more efficient *Hunter*.

Another example is the more common behaviour of distant shooting. Often agents will keep a distance from their opponent and take shots due to their more precise aiming abilities. It also allows them to play a more evasive match, where the agent can maintain a distance even if the opponent does make a move. There were many variations on this tactic, some which would eventually move towards their opponent should the opponent lose sight of the agent, or even back away further if the opponent locked onto the agent.

One interesting point to note was that the hill climber agents tended to become the former, aggressive agents, whilst those using co-evolution developed the latter more distant approach. At this point we must consider which is more favourable given the environment we wish to place these agents in. Given a small amount of human testing against these behaviours, the aggressive opponents are extremely difficult to compete against due to the kamikaze nature of its behaviour; the user requires extensive practice at playing the EvoTanks game to be able to defeat the agent. While the latter behaviours tend to have more variety while maintaining a high level of quality. They provide a means for the player to move around and mount a defence against the agent.

I. Conclusion

This paper has described one approach for a primitive tank game using neural network controllers and genetic algorithms through co-evolutionary simulation. We now have results showing strong capable agents in what is at present a rather simple environment. Evolved populations can competently react to varying strains of NPC behaviour and counteract them with a range of strategies.

There is more room for improvement, with numerous ways in which the EvoTanks game can be expanded. One possibility is the introduction of obstacles within the environment, allowing for agents to be able to navigate more complex arenas and environments.

Further research could also investigate team based play for multiple agents to fight co-operatively, the expansion of the agent's sensors to respond to objects or *power-ups* within the environment, as well as the natural evolution of the tank controller to allow for a separate turret control, or multiple objectives.

Ultimately, the natural evolution of EvoTanks is to create the most complex and immersive environment that can lead to natural play, either for machines to play, or for humans to play for entertainment. After all, it is a video game.

References

[1] J. E. Laird, M. van Lent "*Human-level AI's Killer Application: Interactive Computer Games*," in AAAI Fall Symposium Technical Report, 2000, pp. 80-97.

[2] T. Thompson, "*EvoTanks II: Co-evolutionary Development of Game Playing Agents*", Masters Thesis, Department of Informatics, University of Edinburgh, Edinburgh, Scotland, 2006, unpublished.

[3] B. Geisler, "*An empirical study of machine learning algorithms applied to a modelling player behaviour in a 'first person shooter' video game*", Master's Thesis, Department of Computer Sciences, University of Wisconsin-Madison, Madison, WI, USA.

[4] S. M. Lucas, "*Evolving a Neural Network Location Emulator to Play Ms. Pac-man*", in Proceedings of the IEEE Symposium on Computational Intelligence and Games, pp. 203-210. 2005.

[5] J. Noble, "*Finding robust Texas Hold'em poker strategies using Pareto co evolution and deterministic crowding*," in *ICMLA*, pp. 233-239. 2002.

[6] J. B. Pollack, A. D. Blair, and M. Land, "*Coevolution of a backgammon player*," in Proceedings of Artificial Life V (C. G. Langton, ed.), (Cambridge, MA), MIT Press, 1996.

[7] K. Chellapilla and D. B. Fogel, "*Anaconda defeats Hoyle 6-0: A case study competing an evolved checkers program against commercially available software*", in Proceedings of the 2000 Congress on Evolutionary Computation CEC00, (La Jolla Marriott Hotel La Jolla, California, USA), pp. 857-863, IEEE Press, 6-9 July 2000.

[8] G. Tesauro, "*Temporal Difference Learning and TD-Gammon,*" in Communications of the ACM Vol. 3, pp. 58-58/ 1995.

[9] K.O. Stanley, B. D. Bryant, I. Karpov, R. Miikkulainen, "*Real-Time Evolution of Neural Networks in the NERO Video Game*", To Appear in the Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI-2006, Boston, MA), 2006.

[10] T. Thompson, "*EvoTanks II*", Honours Project Thesis, Department of Computer and Information Sciences, Glasgow, Scotland, 2005, unpublished.

[11] M. Mitchell, "*An Introduction to Genetic Algorithms,*" MIT Press, 1996.