A Minimax Tutor for Learning to Play a Board Game

Dimitris Kalles[1] and **Panagiotis Kanellopoulos** [2]

Abstract. We investigate systematically the impact of a minimax tutor in the training of computer players in a strategy board game. In that game, computer players utilise reinforcement learning with neural networks for evolving their playing strategies. A traditionally slow learning speed during conventional self-play is substantially improved upon when minimax is employed; moreover, the ability of minimax itself to win games serves as a very powerful tutor for its opponents who must develop fast effective defensive strategies. Such strategies are eventually shown to be quite good when deployed against a player that cannot draw on minimax and must play utilising whatever it has learnt.

1 Introduction

Several machine learning concepts have been tested in game domains, since strategic games offer ample opportunities to automatically explore, develop and test winning strategies. The most widely publicised results occurred during the 1990s with the development of Deep Thought and Deep Blue by IBM but the seeds were planted as early as 1950 by Shannon [1] who studied value functions for chess playing by computers. This was followed by Samuel [2] who created a checkers program and, more recently, by Sutton [3] who formulated the TD(λ) method for temporal difference reinforcement learning. TD-Gammon [4, 5, 6] was the most successful early application of TD(λ) for the game of backgammon. Using reinforcement learning techniques and after training with 1.5 million self-playing games, a performance comparable to that demonstrated by backgammon world champions was achieved. Very recently, Schaeffer et al. [7] proved that the game of checkers is a draw with perfect play from both players, while the game of Go [8] has also been studied from an AI point of view.

Implementing a computer's strategy is the key point in strategy games. By the term strategy we broadly mean the selection of the computer's next move considering its current situation, the opponent's situation, consequences of that move and possible next moves of the opponent. In our research, we use a strategy game to gain insight into how we can develop (in the sense of evolving) game playing capabilities, as opposed to programming such capabilities (using mini-max, for example). Although the operational goal of achieving improvement (measured in a variety of ways) is usually achieved in several experimental settings [9, 10], the actual question of which training actions help realize this improvement is central if we attempt to devise an optimized training plan. The term *optimize* reflects the need to expend judiciously the training resources, be it computer power or human guidance.

[1] Hellenic Open University, Sachtouri 23, 26222 Patras, Greece. Email: dkalles@acm.org

[2] Research Academic Computer Technology Institute & Department of Computer Engineering and Informatics, University of Patras, 26500 Rio, Greece. Email: kanellop@ceid.upatras.gr

Previous work [11, 12, 13, 14] has shown that the strategy game under consideration in this paper is amenable to basic design verification using reinforcement learning and neural networks. The problem that we aim to highlight in this paper is that, even with the help of a sophisticated tutor, as implemented by a minimax algorithm, learning cannot be straightforward to automate without careful experimental design.

For this reason we have designed, carried out and analyzed several experimental sessions comprising in total about $25,000$ simple computer-vs.-computer games and about 500 complex computer-vs.-computer games. In complex games, one of the players was following the recommendations of a minimax algorithm, deployed at increasing levels of look-ahead (and incurring, accordingly, significantly increasing computational costs). We believe that the results are of interest as they indicate that increasing the look-ahead does not necessarily lead to increasing the quality of the learned behaviour and that a pendulum effect is present when two players compete and one of them is temporarily aided by a knowledgeable tutor.

The rest of this paper is organised in four subsequent sections. The next section presents the details of the game, including rules for legal pawn movements, a review of the machine learning context, which includes some reinforcement learning and neural network aspects, and a brief review of the to-date experimental findings. We then describe the experimental setup, presented in distinct sessions, each of which asks a specific question and presents data toward answering that question. We then discuss the impact and the limitations of our approach and identify recommended directions for future development. The concluding section summarises the work.

2 A board game in a nutshell

The game is played on a square board of size n, by two players. Two square bases of size α are located on opposite board corners. The lower left base belongs to the white player and the upper right base belongs to the black player. At game kick-off each player possesses β pawns. The goal is to move a pawn into the opponent's base.

The base is considered as a single square, therefore every pawn of the base can move at one step to any of the adjacent to the base free squares. A pawn can move to an empty square that is vertically or horizontally adjacent, provided that the maximum distance from its base is not decreased (so, backward moves are not allowed). Note that the distance from the base is measured as the maximum of the horizontal and the vertical distance from the base (and not as a sum of these quantities). A pawn that cannot move is lost (more than one pawn may be lost in one round). If some player runs out of pawns he loses.

In Figure 1 some examples and counterexamples of moves are presented. The upper board demonstrates a legal and an illegal move (for the pawn pointed to by the arrow - the illegal move is due to the rule

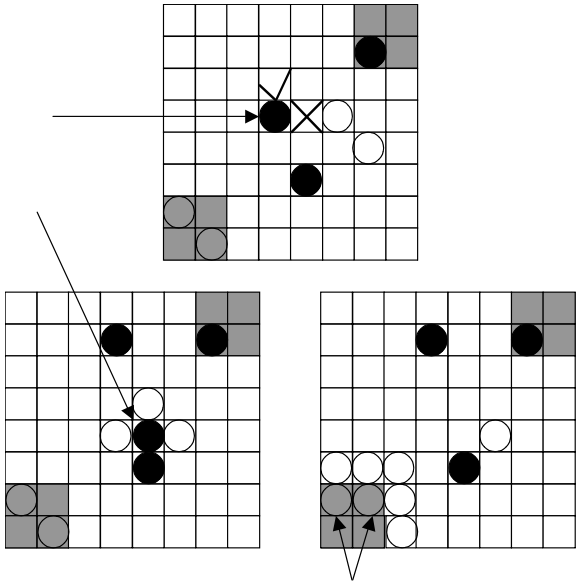

Figure 1. Examples and counterexamples of moves.

that does not allow decreasing the distance from the home base). The lower boards demonstrate the loss of pawns (with arrows showing pawn casualties), where a "trapped" pawn automatically draws away from the game. As a by-product of this rule, when there is no free square next to a base, the rest of the pawns of the base are lost.

The game is a discrete Markov procedure, since there are finite states and moves and each action depends only on the current configuration on the board and not on how this configuration was obtained; therefore, the *Markov property* (see for example [15], Section 3.5) is satisfied. The a *a priori* knowledge of the system consists of the rules only.

Reinforcement learning is quite good at helping explore the state space of such games when it comes to *learning* how to play (as opposed to being *instructed*), for example by observing a tutor or an opponent. In theory, the advantage of reinforcement learning to other learning methods is that the target system itself detects which actions to take via trial and error, with limited need for direct human involvement. The goal is to learn an optimal policy that will maximize the expected sum of rewards in a specific time, determining which action should be taken next given the current state of the environment.

Before moving on, we reiterate same basic nomenclature.

By *state* s we mean the condition of a physical system as specified by a set of appropriate variables. A *policy* determines which action should be performed in each state; a policy is a mapping from states to actions. *Reward* r is a scalar variable that communicates the change in the environment to the reinforcement learning system. For example, in a missile controller, the reinforcement signal might be the distance between the missile and the target (in which case, the RL system should learn to minimize reinforcement). The *value* $V(s)$ of a state is defined as the sum of the rewards received when starting in that state and following some fixed policy to a terminal state. The optimal policy would therefore be the mapping from states to actions that maximizes the sum of the rewards when starting in an arbitrary state and performing actions until a terminal state is reached. The *value function* is a mapping from states to state values and can be approximated using any type of function approximation (e.g., multilayered perceptron, radial basis functions, look-up table, etc.).

Temporal difference (TD) learning is an approach to RL, based on Monte Carlo and dynamic programming. TD methods update estimates based in part on other learned estimates, without waiting for a final outcome (*bootstrapping*). Whereas Monte Carlo methods must wait until the end of the episode to determine the increment to $V(s)$ (only then is the reward known), TD methods need only wait until the next time step. Eligibility traces are one of the basic mechanisms of reinforcement learning. They can be seen as a temporary record of the occurrence of an event, such as the visiting of a state. When a TD error occurs, only the eligible states or actions are assigned credit or blame for the error. Values are backed up according to the following equation: $V(s)_{new} = V(s)_{old} + \alpha e(s)[r + V(s') - V(s)]$, where s is the state-position, $V(s)$ its value, $e(s)$ the eligibility trace, r the reward from the transition, α the learning rate and s' the resulting state-position.

We now proceed to discuss some implementation issues concerning the actual learning procedure. Each player approximates the value function on its state space with a neural network. The input layer nodes are the board positions for the next possible move plus some flags depending on the number of surviving pawns and on the adjacency to an enemy base. The hidden layer consists of half as many hidden nodes and there is just one output node; it serves as the degree to which we would like to make a specific move. At the beginning all states have the same value except for the final states. After each move the values are updated through TD(λ) [15]. The algorithm used for the training was "vanilla" backpropagation, with $\gamma = 0.95$ and $\lambda = 0.5$. By using $\gamma \neq 1$, we favour quick victories, as the reward decreases over time. Network weights constitute a vector $\vec{\theta}$ where updates occur according to $\vec{\theta}_{t+1} = \vec{\theta}_t + \alpha\delta_t\vec{e}_t$, where δ_t is the TD error, $\delta_t = r_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t)$ and $\vec{e}_t$ is a vector of eligibility traces, one for each component of $\vec{\theta}_t$, updated by $\vec{e}_t = \gamma\lambda\vec{e}_{t-1} + \nabla_{\vec{\theta}_t} V_t(s_t)$ with $\vec{e}_0 = \vec{0}$. All nodes use the nonlinear sigmoid function $h(j) = \frac{1}{1+e^{-\sum_i w_{ij}\phi(i)}}$, of which the values range from 0 to 1. In order to avoid local minima and encourage exploration of the state space, a commonly used starting ϵ-greedy policy with $\epsilon = 0.9$ was adopted, i.e., the system chooses the best-valued action with a probability of 0.9 and a random action with a probability of 0.1.

Note that, drawing on the above and the game description, we may conclude that we cannot effectively learn a deterministic optimal policy. Such a policy does exist for the game [16], however the use of an approximation effectively rules out such learning. Of course, even if that was not the case, it does not follow that converging to such a policy is computationally tractable [17].

Earlier experimentation [11] initially demonstrated that, when trained with self-playing games, both players had nearly equal opportunities to win and neither player enjoyed a pole position advantage. Follow-up research [12] furnished preliminary results that suggested a computer playing against itself would achieve weaker performance when compared to a computer playing against a human player. More recently that line of research focused on the measurable detection of improvement in automatic game playing, by constraining the moves of the human (training player), while experimenting with different options in the reward policies [13] and with varying game workflows [14].

3 The experimental setup

To investigate the effect of employing a tutor at several sophistication levels, we devised a set of experiments along the following stages and

associated objectives:

1. One session of $1,000$ computer-vs.-computer (CC) games. The objective was to generate a baseline reference.
2. Five sessions of 100 computer-vs.-computer (MC) games each, where the white player used minimax to determine its next move. Each session involved a different look-ahead; we experimented with look-aheads $1, 3, 5, 7$ and 9 (note that a look-ahead of $2n+1$, denoted by MC_{2n+1}, indicates $n+1$ moves for the white player and n moves for the black player). The objective was to train the white players by tutors of increased sophistication.
3. Five sessions of $1,000$ CC games each, where each session was based on one of the previous stage (for example, the MC_3 session was followed by a $1,000$ CC session). The objective was to examine how the white player did when the tutor was absent, as well as how the black player reacted when its opponent lost expert support.
4. A tournament between all MC variants. A comparison between variants X and Y is done in two steps of $1,000$ CC games each where, in the first step the white player of the Xth batch plays again the black player of the Yth batch and in the second step, the white player of the Yth batch plays again the black player of the Xth batch. The objective was to measure the quality of deep look-ahead, which is an expensive undertaking.

All experiments were made on 8×8 boards with 2×2 bases, with 10 pawns for each player.

3.1 The *tabula rasa* case

The first stage delivered 490 games won by the white player and 510 games won by the black player. However, the white player needed an average of 630 moves to win each game, whereas the black player only required 438 moves on average.

On closer inspection of intermediate steps (by examining the first $1/10$-th and then the fifth $1/10$-th of experiments), we observed that the balance of games won was never really disturbed, whereas the average number of moves per game won fluctuated widely.

It is reasonable to declare that session a draw - and a good reference point. It also confirmed earlier similar findings [11].

3.2 The *minimax* case: early and standard

When deploying a minimax tutor, a certain level of look-ahead is required.

Note that the white player's network is always updated. This reflects that the white player attempts to build a playing model based on its (minimax) tutor. That model is also used whenever minimax examines a leaf state (in the minimax tree) that is not also a final game; we use that state's value as a surrogate for the minimax value which we cannot compute.

When we examine MC experiments, we expect that the white player's performance will be improved the longer we allow experimentation to carry on. There is a simple reason for that: the white player is better situated to observe winning states and then update its learning data structures accordingly, for those states that lead to a win but have not been yet reached via minimax. The results are shown in Table 1.

Indeed, we observe that as we experiment with 100 MC rounds up from 10 MC rounds, the percentage of games won by the white player does not decrease. Also, a consistent observation is that the average number of moves, per won game of the white player, decreases.

Look-ahead	Games Won				Average # of Moves			
	10 MC		100 MC		10 MC		100 MC	
	W	B	W	B	W	B	W	B
1	7	3	93	7	30	67	34	110
3	6	4	93	7	19	31	17	45
5	9	1	91	9	21	19	17	73
7	3	7	82	18	62	162	54	181
9	10	0	89	11	17		21	14

Table 1. The evolution of minimax tutoring.

This indicates that the white player actually improves its playing, taking less time to achieve a win. This is consistent with using the neural network as a surrogate for the minimax value; a high value steers the white player towards a promising path whenever minimax cannot observe a final state. It is also interesting to observe that the sessions where the black player wins have also become increasingly lengthier.

The MC_9 experiments are quite interesting. While the 10 - 0 score of the first 10 experiments can be attributed to a twist of luck, the rather short duration of games even for the games won by the black player are intriguing. However, this observation can lead to two extreme explanations: a very efficient black learner (suggesting that the overall session at 100 games may be too small to call it significant) or a performance degradation for both players.

3.3 Judging a tutor by the impact of absence

When the minimax tutor is absent, the black player has a learned behaviour that can be effectively deployed. The white player, however, may be able to better deal with end-games; therein look-ahead is easier to deliver a win and subsequently update the neural network (with credit, however, only marginally being able to flow back to states that correspond to game openings).

There are two easily identifiable alternatives for game development given that the two players have roughly similar learning capabilities (see section 3.1).

One option is that the relative distribution of wins will not change much from the numbers reported in the relevant columns (MC 100) of Table 1.

Another option, however, is that the black player, which has had to sustain a battering by an unusually effective opponent (the minimax player), has also had to improve itself as much as possible due to such harsh circumstances. In that case, we would expect that the black player should have developed quite an effective defence. Since both players are now not really *adept* at offence, it should be that both could converge to an equal number of games being won by each side in the long run and, as a side effect, games should also take longer to complete.

The results are shown in Table 2.

We observe, that with the notable exception of MC_1 experiments, where the black player actually becomes faster at winning when the CC session takes longer, actually allowing the CC session to evolve makes both players slower.

It is very interesting that the first part of the CC session, which consists of the first 100 games, invariably shows a dramatic increase in the games won by black. That increase is dramatic throughout, and not just for MC_1 experiments, where one might be tempted to say that no real minimax is actually employed.

It is also very interesting that a deeper look-ahead is associated

Look-ahead	Games Won				Average # of Moves			
	100 CC		1000 CC		100 CC		1000 CC	
	W	B	W	B	W	B	W	B
1	72	28	556	444	32	28	36	49
3	58	42	390	610	679	628	742	377
5	55	45	649	351	341	235	697	429
7	42	58	576	424	284	199	441	430
9	23	77	300	700	47	20	733	437

Table 2. The evolution of post-minimax self play.

with a more dramatic decrease in how white does; however, we note that for a look-ahead value of more than 5, that trend is eventually reversed. Maybe, this signifies that the white player initially pays the price for the absence of its tutor, yet loses so many games that it is also forced to update that part of its learning structure that deals with effective defence and eventually manages to counter the attack. Where this eventually is not possible (see MC_3 and MC_9 experiments), games take the longest to conclude among the observed experiments; we believe that this is a signal that a learning stalemate is being reached. That may be particularly true for the MC_9 experiments, where the increase in the length of the CC session does not really affect the percentage of games won by each player.

3.4 Judging a tutor by a student tournament

When the minimax tutor is present, the white player does usually win; however, the black player also learns quite a bit by losing too often against an opponent that does have a strategy (losing against an opponent by luck does not allow one to really learn anything).

Deciding what level of look-ahead to choose is a computationally sensitive issue that can easily lead to an exponential explosion. It is therefore natural to ask whether a certain level of look-ahead is worth the price we pay for it. A straightforward way to look for an answer is to organize a tournament among all learned behaviours.

A tournament game involves measuring the relative effectiveness of the learning policies that delivered the model for each one of any learning batches X and Y. Each comparison is done in two steps of $1,000$ CC games each. In the first step the white player of the Xth batch plays again the black player of the Yth batch; in the second step the white player of the Yth batch plays again the black player of the Xth batch. Sample results are presented in Table 3.

	Games Won		Average # of Moves	
	W	B	W	B
White$_X$ - Black$_Y$	715	285	291	397
White$_Y$ - Black$_X$	530	470	445	314

Table 3. Comparing learning batches X and Y.

Now, for each MC session, we can pit it against the other look-ahead sessions (we remind the reader, that even during comparison, the players actually evolve), by opposing white to black players. Thus, each white player plays against all available black players; for each such tournament game we report below, in Table 4, the surplus of wins for the white player (a negative value indicates more wins for the black side; -S- denotes the sum of these values and -R- denotes the rank of the player compared to its peers).

		Black						
		1	3	5	7	9	-S-	-R-
White	1		−142	−138	−198	−88	−566	4
	3	−48		430	−46	−204	132	2
	5	80	56		−138	−84	−86	3
	7	44	−258	216		186	188	1
	9	−524	−400	−68	148		−844	5
	-S-	−448	−744	440	−234	−190		
	-R-	2	1	5	3	4		

Table 4. Tournament results (on $1,000$ games).

The results are quite interesting. The MC_3 session did surprisingly well, both for the white player (132 more wins) and the black player (744 more wins; note the negative sign for black players). Moreover, its white player is just a close second to the white player of MC_7. MC_9 is plain bad.

It is instructing to compare the above results to their snapshot at only 100 CC games, as shown in Table 5.

		Black						
		1	3	5	7	9	-S-	-R-
White	1		−6	94	−30	−64	−6	1
	3	−24		2	−72	2	−92	4
	5	18	−84		−54	−82	−202	5
	7	12	−24	8		−42	−46	2
	9	−22	−32	−22	18		−58	3
	-S-	−16	−146	82	−138	−186		
	-R-	4	2	5	3	1		

Table 5. Tournament results (on 100 games).

The overall ordering has changed quite dramatically. We believe that this confirms the findings of section 3.3, where we noted that the disappearance of the tutor has a very profound effect on the white player in the short term.

We can now probably formulate the following explanation: a minimax tutor for the white player actually trains the black one by forcing it to lose; when the tutor goes, however, the black player overwhelms the white one, which has to adapt itself due to black pressure as fast as possible. This is the pendulum effect that we spoke about in our introductory section.

4 Discussion

We definitely need more experiments if we are to train (and not program) computer players to a level comparable to that of a human player. The options considered to-date [13, 14] range from experimentation with a wealth of parameters of the reinforcement learning and neural network parameters, with the input-output representation of the neural network, or with alternative reward types or expert playing policies.

We have developed our paper along the last recommendation, but we believe that this may affect a decision on how to deal with the other options as well.

In particular, we note that in the experiments detailed in earlier studies of this game [11, 12, 13, 14], no indication was observed of the pendulum effect; indeed, any interesting patterns of behaviour eventually surfaced after the number of self-play games greatly exceeded the number actually used in this paper.

A possible explanation for this behaviour is that, indeed, the parameters of both the reinforcement learning and neural network infrastructure are inadequately specified to capture training on behalf of the white player. However, even observing the pendulum effect and relating it to an explanation that is not outside the realm of human-to-human tutoring (namely, that one learns to do well when facing a competent player), is a finding that, at the very least, justifies the term "artificial intelligence".

When viewed from the *pendulum effect* viewpoint, the finding that increasing the look-ahead brings about quite a disturbance in the winning patterns of CC games is less surprising. To be able to home on a more precise explanation, we must scale experimentation up to at least MC_{19} experiments, since we need at least 10 moves to move a pawn out of its home base and into the enemy base (we say at least, because an attacking pawn may have to move around a defending one and such a manoeuvre could increase the overall path to the target base).

The above development directions notwithstanding, there is a key technical development that merits close attention. This is the implementation of the actual minimax algorithm; a brute force approach, as is currently employed, is quite expensive and its scaling leaves a lot to be desired.

Overall, we believe that an interesting direction for future work is determining whether the pendulum effect is due to the introduction of the minimax tutor or if it relates to the experimental setup (i.e., number of games, board size, learning parameters, etc.). Having said that, the recommendations set out in previous treatments of this game [13, 14] are still valid. A meta-experimentation engine [18] that would attempt to calculate (most probably, via evolution) good reinforcement learning and neural network parameters, as well as design a series of minimax-based training games, seems quite promising. Yet again, however, it is interactive evolution that seems to hold the most potential. While intuitively appealing, earlier experimental workflows [14] had to be customized for the requirements of this paper. This was a very human-intensive effort, costing well beyond what the actual experiments cost. Yet, it is exactly the design of experimental sessions that could help uncover interesting learning patterns, such as the pendulum effect. While a meta-experimentation game could in theory deliver an excellent computer player, it would obviously subtract from our research effort some of the drive that is associated with the discovery of interesting learning phenomena.

5 Conclusion

This paper focused on the presentation of carefully designed experiments, at a large scale, to support the claim that expert tutoring can measurably improve the performance of computer players in a board game. In our context, the expert role is assumed by a minimax player.

After elaborating on the experimental setup, we presented the results which are centered on two key statistics: number of games won at the beginning and at the end of a session.

The computation of these statistics is a trivial task, but the key challenge is how to associate them with the actual depth of the tutoring expertise. As minimax offers a straightforward way to control such depth via its look-ahead parameter, it is tempting to consider such task as an easy one, however we have found that the quality of the training did not necessarily increase with increasing look-ahead.

The AI toolbox is full of techniques that can be applied to the problem of co-evolutionary gaming and beyond [19]. Still, however, streamlining the experimentation process in game analysis is more of an engineering issue. As such, it calls for productivity enhancing tools, especially so if we also attempt to shed some light into the dynamics of intelligent systems and how they relate to identifiable traits of human cognition.

ACKNOWLEDGEMENTS

This paper and the results reported herein have not been submitted elsewhere. All previous related work by the same authors has been referenced and properly acknowledged. The code is available on demand for personal academic research purposes, as well as the complete design of the experimental sequences and the related results. Finally, the authors wish to thank the reviewers for useful comments and suggestions.

REFERENCES

[1] C. Shannon. "Programming a Computer for Playing Chess", *Philosophical Magazine*, Vol. 41 (4), pp. 265-275, 1950.

[2] A. Samuel. "Some Studies in Machine Learning Using the Game of Checkers", *IBM Journal of Research and Development*, Vol. 3, pp. 210-229, 1959.

[3] R.S. Sutton. "Learning to Predict by the Methods of Temporal Differences", *Machine Learning*, Vol. 3, pp. 9-44, 1988.

[4] G. Tesauro. "Practical Issues in Temporal Difference Learning", *Machine Learning*, Vol. 8, No. 3-4, pp. 257-277, 1992.

[5] G. Tesauro. "Temporal Difference Learning and TD-Gammon", *Communications of the ACM*, Vol. 38, No 3, pp. 58-68, 1995.

[6] G. Tesauro. "Programming Backgammon Using Self-teaching Neural Nets", *Artificial Intelligence*, 134(3), pp. 181-199, 2002.

[7] J. Schaeffer, N. Burch, Y. Bjornsson, A. Kishimoto, M. Muller, R. Lake, P. Lu and S. Sutphen. "Checkers is Solved", *Science*, Vol. 317, pp. 1518-1522, 2007.

[8] H. Yoshimoto, K. Yoshizoe, T. Kaneko, A. Kishimoto, and K. Taura. "Monte Carlo Go Has a Way to Go". *Proceedings of AAAI Conference on Artificial Intelligence*, 2006.

[9] I. Ghory. "Reinforcement Learning in Board Games", Technical report CSTR-04-004, Department of Computer Science, University of Bristol, 2004.

[10] D. Osman, J. Mańdziuk. "TD-GAC: Machine Learning Experiment with Give-Away Checkers", *Issues in Intelligent Systems. Models and Techniques*, M. Dramiński et al. (eds.), EXIT, pp. 131-145, 2005.

[11] D. Kalles and P. Kanellopoulos. "On Verifying Game Design and Playing Strategies using Reinforcement Learning", *ACM Symposium on Applied Computing*, special track on *Artificial Intelligence and Computation Logic*, Las Vegas, pp. 6-11, March 2001.

[12] D. Kalles, E. Ntoutsi. "Interactive Verification of Game Design and Playing Strategies", *Proceedings of IEEE International Conference on Tools with Artificial Intelligence*, pp. 425-430, Washington D.C., 2002.

[13] D. Kalles. "Measuring Expert Impact on Learning how to Play a Board Game", *Proceedings of 4th IFIP Conference on Artificial Intelligence Applications and Innovations*, Athens, Greece, September 2007

[14] D. Kalles. "Player Co-modeling in a Strategy Board Game: Discovering How to Play Fast", *Cybernetics and Systems*, Vol. 39, No. 1, pp. 1 - 18, 2008.

[15] R. Sutton and A. Barto. "*Reinforcement Learning - An Introduction*", MIT Press, Cambridge, Massachusetts, 1998.

[16] M.L. Littman. "Markov Games as a Framework for Multi-Agent Reinforcement Learning", *Proceedings of 11th International Conference on Machine Learning*, San Francisco, pp 157-163, 1994.

[17] A. Condon. "The Complexity of Stochastic Games", *Information and Computation*, 96, pp. 203-224, 1992.

[18] I. Partalas, G. Tsoumakas, I. Katakis and I. Vlahavas. "Ensembe Pruning Using Reinforcement Learning", *Proceedings of the 4th Panhellenic conference on Artificial Intelligence*, Heraklion, Greece, Springer LNCS 3955, pp. 301-310, 2006.

[19] C.D. Rosin. "*Co-evolutionary Search Among Adversaries*". Ph.D. Thesis, University of California at San Diego, 1997.