Fuzzeval: A Fuzzy Controller-Based Approach in Adaptive Learning for Backgammon Game

Mikael Heinze[1], Daniel Ortiz-Arroyo[2], Henrik Legind Larsen[2], and Francisco Rodriguez-Henriquez[3]

[1,2] Computer Science and Engineering Department, Aalborg University, Esbjerg Denmark
contact@gonex.dk, {do, legind}@cs.aaue.dk
[3] Computer Science Section, CINVESTAV, Mexico DF, Mexico
francisco@cs.cinvestav.mx

Abstract. In this paper we investigate the effectiveness of applying fuzzy controllers to create strong computer player programs in the domain of backgammon. *Fuzzeval,* our proposed mechanism, consists of a fuzzy controller that dynamically evaluates the perceived strength of the board configurations it receives. Fuzzeval employs an evaluation function that adjusts the membership functions linked to the linguistic variables employed in the knowledge base. The membership functions are aligned to the average crisp input that was successfully used in the past winning games. Fuzzeval mechanisms are adaptive and have the simplicity associated with fuzzy controllers. Our experiments show that Fuzzeval improves its performance up to 42% after a match of only one hundred backgammon games played against *Pubeval,* a strong intermediate level program.

Keywords: Fuzzy controller, machine learning, artificial neural networks, computer games, reinforcement learning.

1 Introduction

Researchers in AI have applied a wide variety of techniques [1,2,3,4,5,7,10] to create effective computer programs to play board games. These techniques vary from the database and brute-force approaches, to more sophisticated mechanisms based on board pattern classification and artificial neural networks (ANNs).

Board games, such as chess and backgammon, have been the subject of intense study by the AI community. In the domain of chess, it is feasible to create a very strong chess player by applying a combination of brute-force and opening/extended book databases, as was shown by Deep Blue's approach [5]. However, this methodology does not work in the domain of backgammon. Backgammon is a game of strategy and luck [9]. The use of dice in backgammon ensures stochastic variation during the play, producing a branching factor of such a scale, that a search is impossible beyond three moves.

A backgammon game can be characterized by two main stages: the *contact* stage where players' checkers are intermixed and the *race* stage, where there is no contact between players. The strategies applied at each game stage vary. At *contact* stage

A. Gelbukh, A. de Albornoz, and H. Terashima (Eds.): MICAI 2005, LNAI 3789, pp. 224 – 233, 2005.

blockades [9] may be used to delay an opponent's progress. However, at *race* stage what is important is removing most pieces as quickly as possible.

Reinforcement learning has been applied successfully in board games [1,4]. Reinforcement learning seeks to maximize a numerical reward signal in moves that turn out to be successful during a trial and error search in a game. Within this technique, each position where a mistake is made is remembered. The playing strategy changes when the mistaken position is encountered again. In reinforcement learning, evaluation functions are commonly used to estimate the performance quality of an attempted trial. A straightforward solution for learning the weights of an evaluation function is to train the program with example positions for which the exact values of the evaluation function are known. Then, the program learns to adjust the weights to minimize the error of the evaluation function for these positions. Typically this is done by a technique similar to the back-propagation training algorithm used in ANN [7].

Diverse techniques such as ANN, genetic algorithms and fuzzy logic may be used to build effective computer games. Fuzzy logic, a superset of Boolean logic, supports human type reasoning with vague and uncertain concepts. Fuzzy control, a technique derived from the application of fuzzy logic, has been used successfully in a wide variety of devices, from washing machines to industrial robots [6]. The main idea of fuzzy controllers is to build a model of an expert capable to synthesize, via interpolation, a control law. A fuzzy controller processes input signals, executes inferences, and calculates suitable control outputs. The inference mechanisms in a fuzzy controller employ a knowledge base consisting of linguistic expressions generated by a designer or expert. In spite of its conceptual simplicity, little research has explored the application of fuzzy logic-based techniques in board games [7,10].

In this paper we investigate the effectiveness of applying fuzzy controllers in creating strong players in the domain of backgammon. Fuzzeval, our proposed mechanism, is an adaptive backgammon player program that combines a fuzzy controller with simple reinforcement learning mechanisms. To evaluate the effectiveness of Fuzzeval, three types of experiments were performed. First, a fuzzy controller built with a simple knowledge base was evaluated. Afterwards, an improved version of the same knowledge base was employed. Finally, a backgammon player designed with two ANNs was implemented. The performance of Fuzzeval and the ANNs was compared when both players were set to play against *Pubeval*, a strong intermediate level player benchmarking program. Our experimental results show that Fuzzeval learns quickly, competing well with Pubeval and the ANNs. Moreover, the techniques used in Fuzzeval may be applied in a broad variety of board games.

This paper is organized as follows. In next section we briefly describe previous related work on machine learning mechanisms for board games. A detailed description of Fuzzeval is presented in Section 3. Fuzzeval's experimental performance results are discussed in Section 4. Finally, we describe future work and provide our conclusions in Section 5.

2 Related Work

The checkers program written by A. Samuel in 1956 [1], featured the first successful attempt of doing automatic evaluation function tuning by the use of reinforcement

learning. Susan L. Epstein also employed reinforcement learning in *Hoyle* [4], a game system that is capable of learning a wide variety of games. In *Hoyle*, the positions encountered in a game are marked as either, *"significant"* or *"dangerous"*, depending on the result of a search for a better alternative on a loser player's positions. Positions identified as *"dangerous"* are avoided in subsequent games.

In the domain of backgammon, G. Tesauro [2] achieved a remarkable success with *TD-Gammon* [2]. The evaluation function built in TD-Gammon employs an ANN trained by the method of temporal-difference. This method, which is a variant of reinforcement learning, moves the evaluation of a played board closer to the evaluation of a subsequent played board. In the end, the evaluation of the final winning and losing boards are moved closer to one or zero, respectively. One drawback of this approach is that it requires long training times, e.g. TD-Gammon reached the master level after 1,500,000 games of self-playing. In spite of this fact, neural network-based backgammon programs are superior to any other known method to date. Tesauro also created *Pubeval*, a strong intermediate level backgammon player that is used as a benchmark to test other programs.

Chellapilla and Fogel explored the use of genetic algorithms to evolve a neural network that learned by self-play the checkers game. In the experiment described in [3], 30 neural networks were initialized in a random state and then set to play checkers against each other. In each generation, the neural nets played a series of games, and a fitness score was assigned according to their performance. The weights of the 15 best scoring neural networks were maintained as parents for the next generation. This process continued for 250 generations until a capable checkers player finally evolved.

C.T. Sun [7] used a combination of genetic algorithms and fuzzy sets in the domain of othello. In his experiments, a game was divided into several stages characterized by a fuzzy set. Afterwards, each stage was encoded into a chromosome structure.

Azaria and Sipper used genetic programming in the evolution of strategies for playing the game of backgammon. Using this technique, they report in [10] that most capable players evolved in 300 generations, after playing around 2,000,000 games.

Fuzzeval, the approach presented in this paper, differs from previous research work by including a combination of simple reinforcement learning mechanisms and a Fuzzy controller. In the next section we provide a detailed description of Fuzzeval.

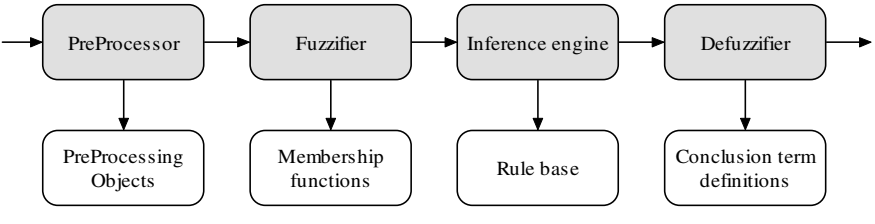

Fig. 1. Fuzzeval's fuzzy controller modules

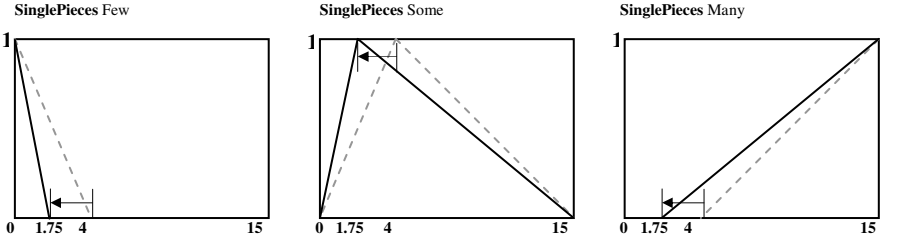

Fig. 2. Example of adjusted membership functions

3 Detailed Description of Fuzzeval

The core of Fuzzeval is a Mamdani type [7] fuzzy controller. This controller is responsible for grading the perceived strength of the board configurations it receives. The fuzzy controller consists of the four main modules illustrated in Figure 1. The *PreProcessor* module is responsible for reading the crisp linguistic input values describing a board's state. The *Fuzzifier* module employs *membership functions* to convert the crisp inputs into membership degrees for each linguistic input variable and term used. The membership degrees of all variables are then passed to the *InferenceEngine,* which calculates the fulfillment of each rule in the *rule base*. Finally, the *Defuzzifier* module generates the defuzzified output from the controller. Fuzzeval employs triangular-shaped membership functions, defined by its left, right base points, and its maximum central point. The membership functions declared as *variable* are tuned using a special function calibrator mechanism. The calibration mechanism calculates the average crisp inputs to the fuzzy controller for all the winning boards stored in a match's historic record. Subsequently, all the membership functions associated with the input values are aligned around these average values. To illustrate this process in detail, let us consider a crisp input for a typical linguistic variable such as *SinglePieces*. The range of this variable varies typically between 0 and 15. However, a player who owns more than four single checkers, is likely to be *hit* [9] by its opponent, an undesirable event that may delay his progress. A typical desirable average input value for the variable *SinglePieces* is around 1.75. The calibration procedure, applied to a membership function consists of moving the adjustment point, slightly closer to the target value matching the average crisp input. Figure 2 illustrates how Fuzzeval adjusts the membership functions, aligning them around the preferred value of 1.75. The amount of adjustment for a point in a membership function is obtained using equation 1.

$$AmountToAdjust = rate \cdot \left(\frac{\sum_{n} x_i}{n} - x_c \right) = rate \cdot \Delta x \qquad (1)$$

where Δx is the difference between the average value, obtained from the n winning boards, and the current location x_c of the adjustable point of a membership function. *Rate* is the learning rate constant, set to 0.1 in our tests.

Figure 3 shows the move selection mechanism employed in Fuzzeval. In a game, the *Board Generator* receives the current position of all pieces in a board together with the dice values. With this information, it generates all possible board configurations containing the valid moves that can be made. The *Fuzzy Controller* receives a board configuration as input and produces an output value that grades the perceived strength of that board. Using the *Match History Data Base,* the *Board Selector* module discards board configurations that have performed poorly in the past. From those that are left, the board receiving the highest score (assigned by the fuzzy controller) is selected for playing. To illustrate Fuzzeval's processing, let us consider a simple example consisting of the linguistic input/output variables and their associated terms presented in Table 1.

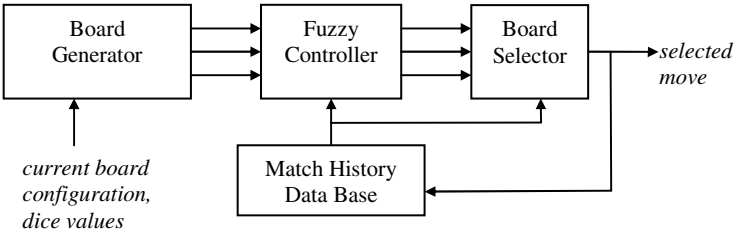

Fig. 3. Fuzzeval's move selection mechanism

Table 1. Linguistic input variables and terms used by the fuzzy controller

Linguistic variable	Linguistic terms	Base variable	Type
SinglePieces	Few, Many	0 to 15 pieces	Input
OpponentOnBar	Few, Many	0 to 15 pieces	Input
SinglePiecesInHome	Few, Many	0 to 6 pieces	Input
BoardStrength	Weak, Good	-1 and 1	Output

An example rule base defined with the previous linguistic variables is shown below:

1. If *SinglePieces* is *Many* then *BoardStrenght Weak*
2. If *SinglePieces* is *Few* then *BoardStrenght Good*
3. If *OpponentOnBar* is *Few* then *BoardStrenght Weak*
4. If *OpponentOnBar* is *Many* then *BoardStrenght Good*
5. If *SinglePiecesInHome* is *Many* then *BoardStrenght Weak*
6. If *SinglePiecesInHome* is *Few* then *BoardStrenght Good*
7. if *OpponentsOnBar* is *Many* and *SinglePiecesInHome* is *Many* then *BoardStrenght Weak*.

Finally, the membership functions associated with the input variables in Table 1 are expressed mathematically in Table 2.

Table 2. Example membership functions

SinglePieces	
$Few(x)=\begin{cases}1 & if\ NumPieces(x)<0\\ -\frac{NumPieces(x)}{4}+1 & if\ 0\le NumPieces(x)\le 4\\ 0 & if\ NumPieces(x)>4\end{cases}$	$Many(x)=\begin{cases}0 & if\ NumPieces(x)<1\\ \frac{NumPieces(x)-2}{13} & if\ 1\le NumPieces(x)\le 15\\ 1 & if\ NumPieces(x)>15\end{cases}$
OpponentOnBar	
$Few(x)=\begin{cases}1 & if\ NumPieces(x)<0\\ -\frac{NumPieces(x)}{3}+1 & if\ 0\le NumPieces(x)\le 3\\ 0 & if\ NumPieces(x)>3\end{cases}$	$Many(x)=\begin{cases}0 & if\ NumPieces(x)<1\\ \frac{NumPieces(x)-1}{14} & if\ 1\le NumPieces(x)\le 15\\ 1 & if\ NumPieces(x)>15\end{cases}$
SinglePiecesInHome	
$Few(x)=\begin{cases}1 & if\ NumPieces(x)<0\\ -\frac{NumPieces(x)}{4}+1 & if\ 0\le NumPieces(x)\le 4\\ 0 & if\ NumPieces(x)>4\end{cases}$	$Many(x)=\begin{cases}0 & if\ NumPieces(x)<1\\ \frac{NumPieces(x)-1}{5} & if\ 1\le NumPieces(x)\le 6\\ 1 & if\ NumPieces(x)>6\end{cases}$

The equations in Table 2 describe the fuzzy sets "*Many*" and "*Few*" pieces at certain positions on the board. Figure 4 shows a board configuration that we will use as an example to illustrate how Fuzzeval (playing the dark pieces) calculates the perceived strength of a board in a move that will *hit* two pieces of its opponent.

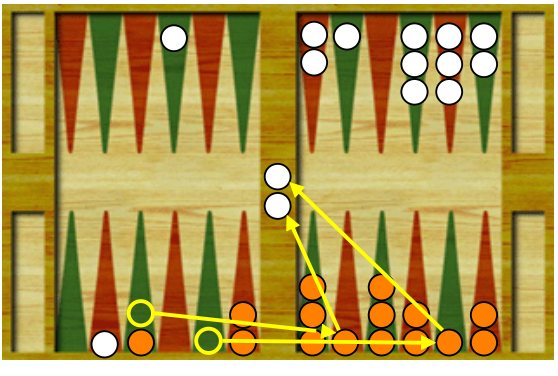

Fig. 4. Backgammon board example

Firstly the fuzzy controller's preprocessor reads the board state, consisting of the number and positions of its own pieces and those of the opponent. These values are the crisp inputs to the controller. Table 3 shows the crisp values obtained from the board shown in Figure 4.

Table 3. Crisp input for each linguistic variable

Linguistic variable	Crisp value
SinglePieces	3
OpponentOnBar	2
SinglePiecesInHome	2

After the preprocessing stage, the crisp inputs are converted into their associated fuzzy membership degrees. This step is performed for each different combination of linguistic variables and terms. The membership functions illustrated in Table 2 produce the membership degrees shown in Table 4.

Table 4. Fuzzified membership degrees for each linguistic input

Linguistic variable	Linguistic term	Membership
SinglePieces	Few	0.25
	Many	0.077
OpponentOnBar	Few	0.33
	Many	0.071
SinglePiecesInHome	Few	0.5
	Many	0.2

The degree of fulfillment is now calculated for each of the seven rules in the example. The result of this step is illustrated in Table 5.

Table 5. Fulfillment of the seven rules

Rule	Fulfillment
If *SinglePieces* is *Many* then *BoardStrenght Weak*	0.077
If *SinglePieces* is *Few* then *BoardStrenght Good*	0.25
If *OpponentOnBar* is *Few* then *BoardStrenght Weak*	0.33
If *OpponentOnBar* is *Many* then *BoardStrenght Good*	0.071
If *SinglePiecesInHome* is *Many* then *BoardStrenght Weak*	0.2
If *SinglePiecesInHome* is *Few* then *BoardStrenght Good*	0.5
If *OpponentsOnBar* is *Many* and *SinglePiecesInHome* is *Many* then *BoardStrenght Weak*	0.071

Finally, in the defuzzification step, the output of all rules in Table 5 is converted into a single value. The center of gravity is used for the purpose, as is indicated by equation 2.

$$u = \frac{\sum \mu(x_i)\ x_i}{\sum \mu(x_i)} \tag{2}$$

where x_i is the value of the output term and $\mu(x_i)$ is the degree of fulfillment of each rule. The last rule in Table 5 uses the T-norm *min* operator for the *and* condition. Using previous equation, jointly with Table 5 and the definition of output terms in Table 1 (*Weak=-1*, *Good=1)*, a final score of 0.095 is obtained for the example board configuration. Fuzzeval performs the evaluation procedure previously described to all boards received from the board generator module. The defuzzified values, which represent the strength of each board as perceived by the fuzzy controller, are then compared; the board receiving the highest score is selected as the best move to play.

4 Discussion of Results

To test Fuzzeval three experiments were performed. In the first test we employed a simple knowledge base similar to the one described in Section 3 but with a total of 15

rules. Using this knowledge base, Fuzzeval was able to learn effective strategies after playing only 100 games, achieving a winning rate of around 34% against Pubeval. In the second test, we extended the rule base to include a variety of situations that Fuzzeval may encounter in a game. Table 6 and Table 7, show some examples of the linguistic input variables and rules contained in the new knowledge base. Specifically, the rules shown in Table 7 are representative of the *contact* stage in backgammon.

Table 6. Some linguistic inputs used in Fuzzeval

Avoiding blots	
ToBeHitFactor	Probability of a checker being hit multiplied by a factor decided by its location on the board. Values for all single pieces are then summed together and then if result is greater than 50, rounded to 50.
SinglePiecesInHome	Amount of single pieces in one's own home board.
Maintaining blockades	
PointsOwned	Number of points occupied by two or more of one's own checkers.
PointsOwnedHome	Number of points occupied by two or more of one's own checkers within ones home board.
StrongestConsecutiveBlockade	Length of the strongest consecutive blockade.

Table 7. Some of the fuzzy rules used in Fuzzeval

Contact
If *SinglePiecesInHome* is *Few* and *Contact* is True then *BoardStrenght Good* if *SinglePiecesInHome* is *Many* and *Contact* is True then *BoardStrenght Weak*
If *PointsOwned* is *Few* and *Contact* is True then *BoardStrenght Weak* if *PointsOwned* is *Many* and *Contact* is True then *BoardStrenght Good*
If *PointsOwnedHome* is *Few* and *Contact* is True then *BoardStrenght Weak* if *PointsOwnedHome* is *Many* and *Contact* is True then *BoardStrenght Good*
If *StrongestConsecutiveBlockade* is *Short* and *Contact* is True then *BoardStrenght Weak* if *StrongestConsecutiveBlockade* is *Long* and *Contact* is True then *BoardStrenght Good*

Using the refined rule base, Fuzzeval was able to play more effectively when the conditions specified in the rule base occurred in a game. The full refined knowledge base consists of 15 linguistic variables and 30 rules. With the new knowledge base, Fuzzeval improved its performance by an additional 8%, achieving a winning rate of approximately 42% against Pubeval in a match of 100 games.

In our third experiment, a new backgammon player was created, implemented this time with ANNs. The purpose of this experiment was to determine the learning rate of the ANNs and the effect of training in their performance when the ANNs were set to play against Pubeval. The ANNs were trained by a combination of temporal difference and back-propagation algorithms, similarly as it is done in TD-Gammon [2]. Two ANN were used in this experiment: one trained for the *contact* stage, and one for the *race* stage. Both networks consist of an input layer of 196 units, a hidden layer of 80 units and an output layer of 5 units. Training was done by self-play, adjusting the network weights after the completion of each game. First, the final winning board was adjusted, then the final losing board, then the second last winning board, and the second last losing board etc., until the first played board was adjusted as the last one.

During training, the temporal difference mechanism was used to calculate the desired target output for a board pattern. The back propagation algorithm made the actual adjustment of the network weights so that the actual and desired outputs matched. In this experiment the step-size parameter in the temporal difference calculation was set to 0.1, and the networks were adjusted at training stage to get a maximum error of between 0.0035 and -0.0035 from the desired target output.

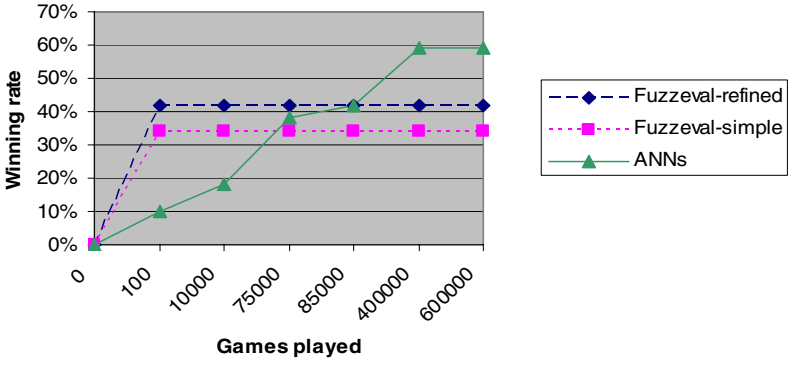

Fig. 5. Performance of Fuzzeval and ANNs

Figure 5 shows the percentage of winning games achieved by Fuzzeval, while playing against Pubeval using a simple and refined rule base. Fuzzeval was trained at same time as it was playing against Pubeval. Figure 5 also shows the performance achieved by the ANNs while playing with Pubeval. For this last case, the figure indicates how many games of self-playing training were required by the ANN to achieve a specific winning rate against Pubeval. As is shown in the figure, 75,000 and 85,000 games of training are required by the ANNs to reach the maximum playing level of Fuzzeval using a simple and a refined rule base, respectively. Figure 5 also shows that at 400,000 training games the ANNs were beating 59% of time to Pubeval and no further improvement was noted after that point.

5 Future Work and Conclusions

The experiments reported in this paper show that fuzzy controllers may be used effectively to create strong players in the domain of backgammon. Fuzzeval's approach differs from other proposed mechanisms [2,3,7] in a number of ways. Firstly, reinforcement learning and fuzzy controllers are combined in Fuzzeval to create a simple and effective backgammon player. Secondly, conversely to methods based on genetic algorithms or ANN [2,3,10], Fuzzeval approach is adaptive, i.e. Fuzzeval may be retrained after playing a few games. Thirdly, Fuzzeval mechanisms are simple comparatively with the approaches described in [7, 10], based on the application of genetic algorithms and genetic programming in the domain of othello and backgammon respectively.

In our experiments, Fuzzeval was able to adjust its playing style, to beat around 42% of the time to Pubeval. Fuzzeval reached its best performance after playing only 100 games. Comparatively, a more complex mechanism based on ANNs requires playing around 85,000 games to reach the effectiveness of Fuzzeval, and 400,000 games to exceed Fuzeval's performance by 17%. Our experiments also show that further improvement in Fuzzeval is possible with a more refined knowledge base. While the current performance of Fuzzeval is lower than the one obtained with ANNs, its combined simplicity and effectiveness makes this approach attractive. Moreover, contrary to approaches based on ANN and evolutionary techniques, Fuzzeval has the advantage of allowing humans learn why certain moves were executed.

One drawback of the fuzzy controller-based approach presented in this paper is obtaining the optimum knowledge base. As it is shown by our experiments, a few simple rules are likely to provide very good performance. However, to obtain the best performance a refined knowledge base may be more difficult to design. To overcome this problem, we are currently exploring diverse methods to generate automatically the fuzzy rules. Our preliminary experiments have shown that there are correlations between pairs of fuzzy rules. Furthermore, previous performance history of a game may be used to select, keep, or discard certain rules. Finally, for comparison purposes, we are exploring the use of approaches based on ANN and genetic algorithms to automatically learn the fuzzy rules.

References

1. A. L. Samuel, "Some studies in machine learning using the game of checkers". *IBM Journal of research and development* 44(1/2): 206-226, 2000.
2. G. Tesauro, "Temporal Difference Learning and TD-Gammon." *Communications of the ACM* 38(3): 58-68, 1995.
3. K. Chellapilla, D. B. Fogel, "Evolving neural networks to play checkers without relaying on expert knowledge." *IEEE Transactions on Neural Networks* 10(6): 1382-1391, 1999.
4. S. L. Epstein, "Learning to play expertly: A tutorial on Hoyle." *Machines that Learn to Play Games* 153-178, 2001.
5. Y. Seirawan, H. A. Simon, T. Munakata, "The implications of Kasparov vs. Deep Blue." *Communications of the ACM* 40(8): 21-25, 1997
6. L.A. Zadeh, "Making computers think like people," *IEEE. Spectrum*, 8/1984, pp. 26-32.
7. J.S. Jang, C.-T. Sun, and E. Mizutani, *Neuro-fuzzy and Soft Computing.*, Ed. Prentice Hall, New Jersey 1997
8. Backgammon Rules. http://www.bkgm.com/rules.html
9. Key Concepts in Backgammon. http://www.redtopbg.com/keyconcepts.htm
10. J.P. Azaria and M. Sipper, "GP-Gammon: Using Genetic Programming to Evolve Backgammon Players", *Proceedings of 8th European Conference, EuroGP 2005*, Lausanne, Switzerland, March 30 - April 1, 2005.