An Othello Evaluation Function Based on Temporal Difference Learning using Probability of Winning

Yasuhiro OSAKI Kazutomo SHIBAHARA Yasuhiro TAJIMA Yoshiyuki KOTANI

Abstract— **This paper presents a new reinforcement learning method, called Temporal Difference Learning with Monte Carlo simulation (TDMC), which uses a combination of Temporal Difference Learning (TD) and winning probability in each non-terminal position. Studies on self-teaching evaluation functions as applied to logic games have been conducted for many years, however few successful results of employing TD have been reported. This is perhaps due to the fact that the only reward observable in logic games is their final outcome, with no obvious rewards present in non-terminal positions. TDMC(λ) attempts to compensate this problem by introducing winning probabilities, obtained through Monte Carlo simulation, as substitute rewards. Using Othello as a testing environment, TDMC(λ), in comparison to TD(λ), has been seen to yield better learning results.**

I. INTRODUCTION

This paper presents a new method of optimizing an evaluation function, used by game playing programs to determine the best move. Evaluation functions are used to represent the importance of a given position in a numerical form, based on various characteristics derived from it. Recently, a progress in researches on reinforcement learning led to notable rise in accuracy of evaluation functions. This field is primarily represented by Temporal Difference learning method (TD)[1][2], which policy is to maximize the total sum of rewards provided by the environment. However, in case of logic games, the sole reward observed is the information of win/lose/draw in the terminal position, which makes it problematic to use self-play game records as a learning data for TD, as those records are too dependent on their final outcomes.

Y.Osaki is with Department of Computer and Information Sciences, Graduate school of Technology, Tokyo University of Agriculture and Technology, 2-24-16 Naka-cho, Koganei-shi, Tokyo, Japan (phone: +81-042-388-7492; fax: +81-042-388-7492; e-mail: osaki@fairy.ei.tuat.ac.jp).

K.Shibahara, Y.Tajima and Y.Kotani are with Department of Computer and Information Sciences, Tokyo University of Agriculture and Technology, 2-24-16 Naka-cho, Koganei-shi, Tokyo, Japan (phone and fax numbers are same above; e-mail: {k-shiba, ytajima, kotani@cc.tuat.ac.jp}).

This paper proposes a new idea in which the probabilities of winning, obtained through Monte Carlo simulations for each non-terminal position, is added to TD(λ) as *substitute rewards*. We called this method TDMC(λ) (Temporal Difference with Monte Carlo simulation). The advantage of Monte Carlo simulation is that it can produce approximate winning probability of a given position through random simulation, even if it is difficult for evaluation function to compute its value. It is important to note that this probability of winning is completely independent from the accuracy and structure of the evaluation function. The purpose of TDMC(λ) is to maximize the expected sum of substitute rewards, represented by those probabilities.

This paper compares the evaluation functions obtained through TD(λ) and TDMC(λ) in the game of Othello. The results of this comparison proved that TDMC(λ) produces significantly better evaluation functions than those produced by TD(λ).

II. REINFORCEMENT LEARNING

The basic concept behind reinforcement learning is, to put it simply, not *how to achieve*, but *what to achieve* [3][4]. Therefore, the only goal provided by the game environment to a learning agent is the information of win/lose/draw present in the terminal position. This goal is known as a *reward*. In case of simple logic games, where a game tree can be fully expanded, the final outcome can be propagated backwards to all the nodes. Those cases, however, are outside the scope of this paper. In more complex games, where the game tree cannot be fully expanded, non-terminal positions do not have obvious rewards. As stated earlier, the purpose of TD(λ) is to *maximize the total sum of rewards provided by the environment*. Thus, a final outcome of the game is treated as a reward and is assigned to all previous (non-terminal) positions, making them highly dependent on the result of a given game record. This leads to the conclusion that if two game records with different outcomes contain the same position, this position will be assigned different rewards, consequently producing a completely different learning result. As a way to overcome this problem, we tried an approach to substitute the reward in each non-terminal

978-1-4244-2974-5/08/$25.00 ©2008 IEEE

position with a probability of winning. This was inspired by recent incorporation of Monte Carlo simulations into computer Go, which led to great advancement in the field. We researched on a new algorithm which would be based on substitute rewards, as increasing the winning probability proved to be a successful policy in complex logic games.

Having substituted the actual outcome of the game with probability of winning gained from simulation, the new purpose of TDMC(λ) is to *maximize the total sum of substitute rewards provided by the environment*. This difference in approaches contributed to the significant differences in learning results.

Probability of winning, as obtained through Monte Carlo simulation, is characterized by the following points:

- It is independent of previous moves.
- It is independent of the accuracy and structure of the evaluation function.
- The closer it is to the terminal position, the closer the probability is to the actual outcome.

The next section provides a detailed explanation of TDMC(λ) algorithm.

III. The TDMC(λ) Algorithm

Let M denote the total number of feature weights in an evaluation function. Let $w=(w_1,w_2,...,w_M)$ denote the vector containing the weights of those features, and $x_t=(x_{t1},x_{t2},...,x_{tM})$ denote the vector containing the value x_i of i-th characteristic in position P_t, where $t\in[1,T]$. The value of the evaluation function $V(x_t,w)$ in position P_t is as follows:

$$V(x_t,w) := x_t \cdot w.$$

TD(λ) updated the evaluation $V(x_t,w)$ of present position by a reward r_t gained in time step t and the evaluation $V(x_{t+1},w)$ of the next position:

$$V(x_t,w) := V(x_t,w)+\alpha[r_t+\gamma V(x_{t+1},w)-V(x_t,w)]$$

where $\gamma\in[0,1]$ is a discount rate and $\alpha\in[0,1]$ is a learning rate. In logic games, however, since there is no reward r_t in non-terminal positions, the learning depends on the accuracy of $V(x_{t+1},w)$, which propagates the final outcome backwards.

TDMC(λ) substitutes the reward r_t in position P_t with a result of running Monte Carlo simulation. The *return* R_t in P_t becomes

$$R_t := \gamma^0 r_t+\gamma^1 r_{t+1}+\gamma^2 r_{t+2}+...+\gamma^{T-t}r_T=\sum_{i=t}^{T}\gamma^{j-t}r_i. \quad (1)$$

The larger γ is, the more R_t is influenced by substitute rewards of further positions. On the other hand, smaller γ makes R_t take strongly near positions into account.

There exists a n-step return $R_t^{(n)}$ based on n time steps, as opposed to all time steps from t to T in R_t. $R_t^{(n)}$ uses a value of evaluation function in P_{t+n} in place of all subsequent substitute rewards after $t+n$:

$$R_t^{(n)} := \gamma^0 r_t+\gamma^1 r_{t+1}+...+\gamma^{n-1}r_{t+n-1}+\gamma^n V(x_{t+n},w). \quad (2)$$

Using eligibility rate $\lambda\in[0,1]$ and incorporating eqs. (1) and (2), one can obtain a λ-return R_t^λ:

$$\begin{aligned} R_t^\lambda &:= \lambda^0 R_t^{(1)}+\lambda^1 R_t^{(2)}+...+\lambda^{T-t-2}R_t^{(T-t-1)}+\lambda^{T-t-1}R_t \\ &= \sum_{n=1}^{T-t-1}\lambda^{n-1}R_t^{(n)}+\lambda^{T-t-1}R_t. \end{aligned} \quad (3)$$

R_t^λ from eq. (3) is defined as a *target value*, and as such, it is approximated by the evaluation value $V(x_t,w)$ in position P_t.

A square error OF_{TDMC} between target and evaluation values in each time step t is denoted as:

$$OF_{TDMC}=\sum_{t=1}^{T}[R_t^\lambda-V(x_t,w)]^2. \quad (4)$$

Basing on eq. (4), the gradient of each parameter can be calculated:

$$\Delta w := \frac{\partial OF_{TDMC}}{\partial w}=-2\sum_{t=1}^{T}[R_t^\lambda-V(x_t,w)]\nabla_w V(x_t,w)$$

$$\nabla_w V(x_t,w)=\left(\frac{\partial V(x_t,w)}{\partial w_1},\frac{\partial V(x_t,w)}{\partial w_2},...,\frac{\partial V(x_t,w)}{\partial w_M}\right).$$

The method of steepest descent is used to adjust the parameters w so as to reduce the square error from eq. (4).

$$w := w+\alpha\Delta w. \quad (5)$$

With the aid of eq. (5), an evaluation value $V(x_t,w)$ is updated to approximate the target value R_t^λ in each position. The convergence of the evaluated value to the target value marks the end of the learning process, and then, selecting the highest evaluated value from the available child positions corresponds to maximizing the expected total sum of substitute rewards. In an Othello game, a reward r_t is treated as an average of values returned by Monte Carlo simulations for a position P_t (+1 for a win, 0 for a draw and -1 for a loss). An evaluated value $V(x_t,w)$ and target value R_t^λ are normalized by passing them through the hyperbolic tangent function for updating the feature weights.

The TDMC(λ) algorithm is summarized in Fig.1.

- Let $w=(w_1,w_2,...,w_M)$ denote the vector containing feature weights of the evaluation function.

- Let $x_t=(x_{t1},x_{t2},...,x_{tM})$ denote the vector containing features of position P_t, $t\in[1,T]$.

- For $t=1,2,...,T-1$, compute the value of evaluation function $V(x_t,w)$:

$$V(x_t,w) \coloneqq x_t \cdot w .$$

- For $t=1,2,...,T-1$, compute the gradient of w:

$$\nabla_w V(x_t,w)=\left(\frac{\partial V(x_t,w)}{\partial w_1},\frac{\partial V(x_t,w)}{\partial w_2},...,\frac{\partial V(x_t,w)}{\partial w_M}\right).$$

- For $t=1,2,...,T-1$, compute the return R_t:

$$R_t \coloneqq \gamma^0 r_t+\gamma^1 r_{t+1}+\gamma^2 r_{t+2}+...+\gamma^{T-t} r_T=\sum_{i=t}^{T}\gamma^{i-t} r_i$$

where discount rate $\gamma\in[0,1]$, and r_t denotes the probability of winning in P_t obtained through Monte Carlo simulation.

- For $t=1,2,...,T-1$, compute the n-step return $R_t^{(n)}$:

$$R_t^{(n)} \coloneqq \gamma^0 r_t+\gamma^1 r_{t+1}+...+\gamma^{n-1} r_{t+n-1}+\gamma^n V(x_{t+n},w).$$

- For $t=1,2,...,T-1$, compute the λ-return R_t^{λ} using R_t and $R_t^{(n)}$:

$$R_t^{\lambda} \coloneqq \lambda^0 R_t^{(1)}+\lambda^1 R_t^{(2)}+...+\lambda^{T-t-2} R_t^{(T-t-1)}+\lambda^{T-t-1} R_t$$

where eligibility rate $\lambda\in[0,1]$.

- Update each feature weight w_j according to the method of steepest descent:

$$w_j \coloneqq w_j+\alpha\sum_{t=1}^{T-1}[R_t^{\lambda}-V(x_t,w)]\nabla_{w_j} V(x_t,w)$$

where learning rate $\alpha\in(0,1]$.

Fig.1. The TDMC(λ) algorithm

IV. Learning Environment

Section IV-A explains the 15 features utilized in the evaluation function. In section IV-B, an explanation is given on different feature weight vectors used depending on the game stage. Finally, section IV-C describes the learning process employed.

A. Evaluation features

$V(x_t, w)$ uses 15 evaluation features, listed in Table I.

TABLE I
Features of the evaluation function

Feature	Quantity	Description
Squares	10	10 squares reflected through 8-way symmetry
Legal moves	1	Number of possible moves
Mobility	1	Number of opponent's moves resulting in flipping own discs
Stable discs	1	Number of discs that cannot be flipped by opponent
Own discs	1	Number of own discs on the board
Player turn	1	Represents presently moving player

The Player Turn feature takes on a value of +1 when it is one's own turn, with 0 for the opponent's turn.

The initial Othello board is shown on Fig.2. A Black disc on a square is represented as +1 by the corresponding feature, with -1 for White disc, and 0 for empty. Instead of information on all 8x8 squares, only 10 features are used, which are then reflected by horizontal, vertical and 2 diagonal axes (8-way symmetry). Thus, from algorithmical perspective, corresponding coordinates (e.g. (b,1), (g,1), (a,2), (h,2), (a,7), (h,7), (b,8), (g,8)) all bear the same meaning.

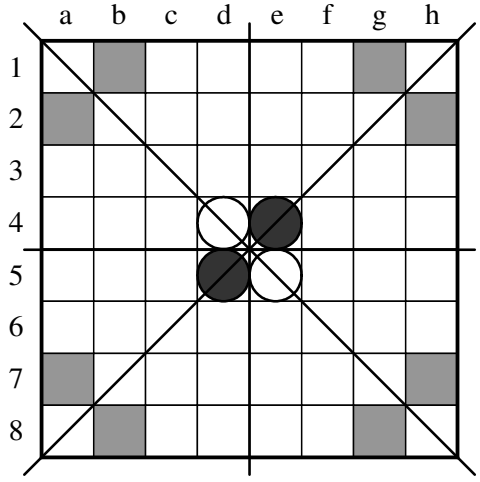

Fig.2. The initial Othello board. The 4 axes are shown, along with an example of reflection

B. Game stage

Once a disc is put on board it cannot be removed, thus a game of Othello ends in maximum 60 moves (not including passes). By analyzing the situation on board, we can assign it to one of the 3 stages of the game — opening, middle, or ending. A separate feature weights vector w is used for each of those stages, as shown in Table II.

TABLE II
Characterizing factors for 3 game stages

Stage	Characterizing factor
Opening	Initial stage
Middle	At least one disc of any color is present on the edges
Ending	At least 2 discs of the same color are present in the corners

As the game progresses into a new stage, w is swapped with a vector corresponding to that stage. This is to avoid the situation in which the learning results from opening stage give unnecessary influence to the strategy employed in the ending stage.

C. Collection of game records through self-play

The learning process takes place in cycles: a game record is obtained through self-play, which is then used as learning data for adjusting the weights of the evaluation function. The new function is used in the next self-play, producing a new game record for adjusting the evaluation function's weights again. This cycle is repeated a predefined number of times.

A self-play is based on ε-greedy policy: a program chooses the move with the highest evaluated value, while random moves are forced with ε rate. In case there is more than one move with the highest evaluated value, the choice between them is done randomly. Small ε leads to high dependency of the policy on the evaluation function, whereas large ε tends to decrease the effect that learning results have on the learning data for the next cycle.

The learning process is summarized in Fig.3.

1. Initialize all feature weights with 0.
2. Obtain game record through self-play based on ε-greedy policy.
3. Update evaluation function's feature weights using TDMC(λ).
4. Repeat 2. and 3. for a predefined number of times.

Fig.3. The steps performed in the learning process

V. Experimental Results

This section gives results of experimental games between trained and untrained programs. First, two programs were trained, one by TD(λ) and the other by TDMC(λ), and then were tested against an untrained program, to confirm the methods' learning performance (section V-A). Then, both trained programs were tested against each other (section V-B). Finally, section V-C gives results of experimenting with different values of ε in the learning stage, to explore the dependency of both learning methods on game records (see section VI for further explanation). The values of discount rate γ and eligibility rate λ where decided on 0.98 through preliminary experiments. The learning rate α was initially set on 0.5 and was gradually decreased as the learning progressed.

A. *Results of games against untrained function*

The self-play training was done 1000 times, with ε set to 0.03. In the learning phase, the Monte Carlo simulation was also done 1000 times for each position, producing corresponding substitute rewards. Game records with substitute rewards were then used for updating the feature weights, by employing TDMC(λ).

The resulting evaluation function was tested against the untrained function (with all weights set to initial 0), to assess the learning results. During play, in case there was more than one legal move with the highest evaluated value, the trained program chose one randomly. Untrained program played in essentially random manner. Table III shows the results of 1000 games played.

TABLE III
Results of games between TDMC(λ) and untrained player

TDMC(λ) Player color	TDMC(λ) Player wins	Draws	Untrained Player wins
Black	481	1	18
White	483	6	11
TOTAL	964	7	29

As Table III shows, the program trained with TDMC(λ) is of notably better performance than the untrained one. Also, it can be seen there is no bias related to player color.

A similar experiment was done using TD(λ). In this case, the learning process was equivalent to the one presented in Fig.3, with ε set to 0.03 as well. However, since the learning time for TD(λ) is 1/5 the time for TDMC(λ), TD(λ)'s learning was allowed to be performed 5000 times (as opposed to 1000 for TDMC(λ)). Lastly, the trained program was tested against the untrained one, similarly to the case above. Table IV shows its results.

TABLE IV
Results of games between TD(λ) and untrained player

TD(λ) Player color	TD(λ) Player wins	Draws	Untrained Player wins
Black	479	2	19
White	478	3	19
TOTAL	956	5	38

As expected, the program trained with TD(λ) clearly outperformed the untrained one and has no bias related to player color, similarly to the case of TDMC(λ).

B. *Results of games between TD(λ) and TDMC(λ)*

Subsequently, TDMC(λ) was confronted against TD(λ). Similarly to section V-A, both programs made the choice randomly if there was more than one move with the best evaluated value. However, for first *d* moves, both programs made random moves regardless of the learning results, to diversify the experimental data and thus expand the basis for methods assessment.

Table V shows the results for different values of *d*.

TABLE V
Results of games between TD(λ) and TDMC(λ) with different values of *d*

d	TDMC(λ) Player wins	Draws	TD(λ) Player wins
4	880	0	120
6	817	10	173
8	794	20	186
10	782	18	200

Table V shows that TDMC(λ) outperformed TD(λ) for all values *d=4,6,8,10*. Furthermore, experiments have shown that without any random moves (*d=0*), TDMC(λ) outperformed TD(λ) as well, regardless of player color.

Tables VI, VII and VIII illustrate the weights obtained through TDMC(λ) for each of the 3 game stages. Darker colors represent higher weights.

Looking at the table for opening stage (Table VI), it can be seen that rows *1* and *8* and columns *a* and *h* have values of 0, unchanged from initialization. This is because the opening stage is characterized by no discs on any of the edges, therefore at this stage those squares are outside of the scope of learning. Squares (c,3), (c,6), (f,3), (f,6) have high weights, reflecting the high importance of corner squares at later stages (putting discs on those squares will likely result in opponent's discs put on (b,2), (b,7), (g,2), (g,7), which would open the way to the corners). On the other hand, putting discs on (b,2), (b,7), (g,2), (g,7) would result in 'giving out' corners to the opponent, therefore lowering the winning probability. This fact is reflected in negative weights assigned to those squares.

Table VII shows the weights in the middle stage. There, the corner weights have significantly higher values than for the rest of the board, while the squares surrounding the corners have the lowest values. This corresponds to the weights for the previous stage and follows the common sense in Othello strategy, where situation at the corners contributes greatly to the winning probability.

In Table VIII, we can see the corners still hold high weights, but at the same time weights at the center of the board, which previously showed relatively low importance, have high values assigned. This illustrates the fact that in the ending stage, the presence of own discs at the center of the board gives more possibilities to flip opponent's discs. In addition, all the weights outside the diagonal axes have taken on negative values.

C. *Results of games with different values of ε*

While *d* is used to randomize the games in the playing stage, ε controls the degree of randomness in the learning stage. This section experiments with influence of ε on TD(λ) and TDMC(λ)'s performance. First, TDMC(λ) was trained under different values of ε, with TD(λ) trained in an unchanged manner (*ε=0.03*). Table IX shows results of 1000 games played with *d* set to 10.

TABLE VI
Feature weights of board squares in opening stage

Row \ Col	a	b	c	d	e	f	g	h
1	0	0	0	0	0	0	0	0
2	0	-0.02231	0.05583	0.02004	0.02004	0.05583	-0.02231	0
3	0	0.05583	0.10126	-0.10927	-0.10927	0.10126	0.05583	0
4	0	0.02004	-0.10927	-0.10155	-0.10155	-0.10927	0.02004	0
5	0	0.02004	-0.10927	-0.10155	-0.10155	-0.10927	0.02004	0
6	0	0.05583	0.10126	-0.10927	-0.10927	0.10126	0.05583	0
7	0	-0.02231	0.05583	0.02004	0.02004	0.05583	-0.02231	0
8	0	0	0	0	0	0	0	0

Visualizations of TABLE VI-VIII

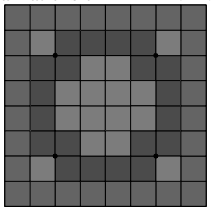

TABLE VII
Feature weights of board squares in middle stage

Row \ Col	a	b	c	d	e	f	g	h
1	6.32711	-3.32813	0.33907	-2.00512	-2.00512	0.33907	-3.32813	6.32711
2	-3.32813	-1.52928	-1.87550	-0.18176	-0.18176	-1.87550	-1.52928	-3.32813
3	0.33907	-1.87550	1.06939	0.62415	0.62415	1.06939	-1.87550	0.33907
4	-2.00512	-0.18176	0.62415	0.10539	0.10539	0.62415	-0.18176	-2.00512
5	-2.00512	-0.18176	0.62415	0.10539	0.10539	0.62415	-0.18176	-2.00512
6	0.33907	-1.87550	1.06939	0.62415	0.62415	1.06939	-1.87550	0.33907
7	-3.32813	-1.52928	-1.87550	-0.18176	-0.18176	-1.87550	-1.52928	-3.32813
8	6.32711	-3.32813	0.33907	-2.00512	-2.00512	0.33907	-3.32813	6.32711

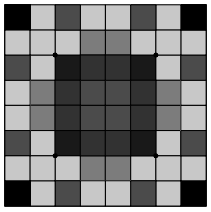

TABLE VIII
Feature weights of board squares in ending stage

Row \ Col	a	b	c	d	e	f	g	h
1	5.50062	-0.17812	-2.58948	-0.59007	-0.59007	-2.58948	-0.17812	5.50062
2	-0.17812	0.96804	-2.16084	-2.01723	-2.01723	-2.16084	0.96804	-0.17812
3	-2.58948	-2.16084	0.49062	-1.07055	-1.07055	0.49062	-2.16084	-2.58948
4	-0.59007	-2.01723	-1.07055	0.73486	0.73486	-1.07055	-2.01723	-0.59007
5	-0.59007	-2.01723	-1.07055	0.73486	0.73486	-1.07055	-2.01723	-0.59007
6	-2.58948	-2.16084	0.49062	-1.07055	-1.07055	0.49062	-2.16084	-2.58948
7	-0.17812	0.96804	-2.16084	-2.01723	-2.01723	-2.16084	0.96804	-0.17812
8	5.50062	-0.17812	-2.58948	-0.59007	-0.59007	-2.58948	-0.17812	5.50062

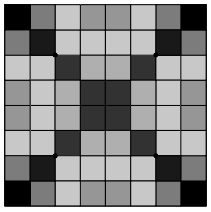

TABLE IX
Results of using different ε in TDMC(λ) Player's learning

TDMC(λ)'s ε	TDMC(λ) Player wins	Draws	TD(λ) Player wins
0.01	781	11	208
0.03	782	18	200
0.05	757	13	230
0.10	767	12	221
0.30	721	14	265
1.00	712	12	276

Table IX shows that the lower the value of ε, the more TDMC(λ) outperforms TD(λ). This is a consequence of the fact that if a move P_t was chosen randomly, it makes accuracy of λ-return R_t^{λ} drop heavily at previous positions, especially P_{t-1}.

The next experiment involved applying varying ε for both TDMC(λ) and TD(λ). The rest of the conditions were set as previously: 1000 plays with *d=10*.

TABLE X
Results of games between TDMC(λ) and TD(λ) with different values of ε

ε	TDMC(λ) Player wins	Draws	TD(λ) Player wins
0.01	816	7	177
0.03	782	18	200
0.05	758	19	223
0.10	700	15	285
0.30	719	16	265
1.00	781	10	209

As Table X shows, TDMC(λ) Player outperformed TD(λ) Player in at least 70% of games, regardless of ε rate applied. The highest winning rate was observed under *ε=0.01*, with 816 TDMC(λ) Player wins.

Experiments in sections V-B and V-C explored the influence of randomness, both in learning and playing stages, on TDMC(λ)'s performance, but results have shown visible superiority of TDMC(λ) over TD(λ) in all test cases. The following section gives analysis as to its sources.

VI. Analysis

Results of experimental games, presented in Table X, show noticeable outperformance of TDMC(λ) over TD(λ) even at *ε=1.00*, in which case a game record is constructed in a completely random manner.

The reason behind this is the way TDMC(λ) assigns the target values: even supposing that, at some point, choosing a random move leads to a lost game, TDMC(λ) still bases its learning on winning probability obtained in non-terminal positions, which makes it far less dependent on game records, as compared to TD(λ). This can lead to the conclusion that TDMC(λ) is a preferred method in case of cycles-based learning where initial feature weights are updated through self-play game records.

Although TDMC(λ) and TD(λ) rely their learning on different target values, both values tend to be similar in the later parts of the game. This is because Monte Carlo simulations give approximations closer to the actual outcome as the terminal position approaches. Thus, one can presume that in the ending stage TDMC(λ) and TD(λ) hold similar feature weights.

Fig.4 and Fig.5 show weights of TDMC(λ) and TD(λ) normalized on the basis of Player Turn's weight (see Table I)

at the middle and ending stages.

Fig.5 clearly shows the relative similarity of weights between TD(λ) and TDMC(λ) in the ending stage, while this similarity, as Fig.4 illustrates, is hardly observable in the middle stage. Weights of Square B*, Square D*, Square G* and Mobility for TD(λ) hold values over 3 times larger than for TDMC(λ), while the weight of Square F* is even of different sign for both functions. Moreover, two of the highest weights, namely Square A* and Stable Discs, show as much as 150% difference in the middle stage, in favor of TDMC(λ). This leads to the conclusion that the outcomes of games between the two programs are greatly influenced by differences in weights in the middle stage.

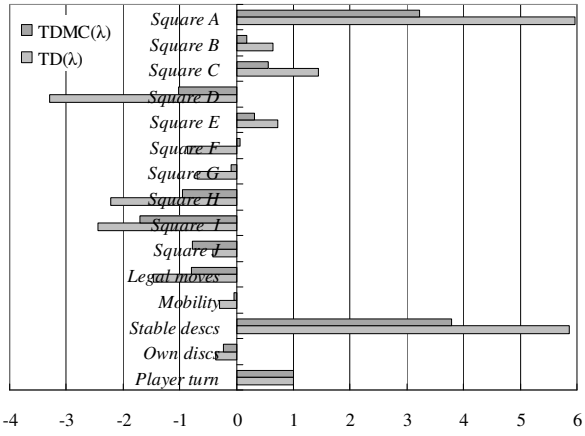

Fig.4. Normalized feature weights in the middle stage

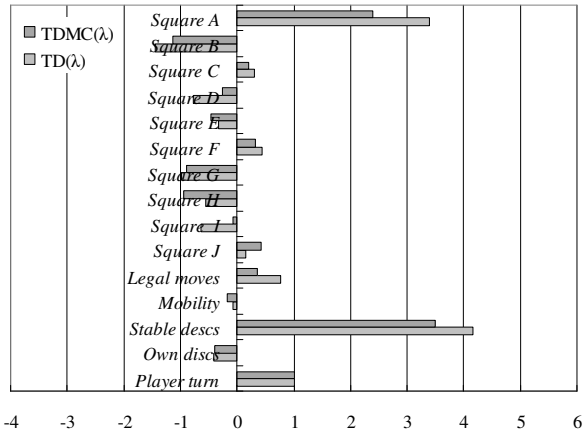

Fig.5. Normalized feature weights in the ending stage

* Squares A to J are distributed as follows:

	a	b	c	d	e	f	g	h
1	A	I	B	D	D	B	I	A
2	I	J	H	G	G	H	J	I
3	B	H	C	E	E	C	H	B
4	D	G	E	F	F	E	G	D
5	D	G	E	F	F	E	G	D
6	B	H	C	E	E	C	H	B
7	I	J	H	G	G	H	J	I
8	A	I	B	D	D	B	I	A

VII. Conclusions and Future Work

In this paper, we proposed TDMC(λ), a new method of reinforcement learning using winning probability as substitute rewards in non-terminal positions. Experimental results have shown that TDMC(λ) outperforms TD(λ) in all test cases. It is thought that the primary source of this superiority is compensating TD(λ)'s dependence on final outcome of self-plays, with the help of probabilities of winning. Perhaps the most significant conclusion of this paper is, however, that maximizing the expected total sum of substitute rewards is a successful learning policy in case of complex logic games.

An example of further work would be using TDMC(λ) to train an evaluation function using 64 board squares as feature weights (in the likes of those present in CEC2006COMPETITION [5][6]), and testing it against functions trained by TD(λ) and Co-Evolutionary Learning (CEL). Another possibility is to train an evaluation function similar to the one present in LOGISTELLO [7], to assess TDMC(λ)'s performance in case of large amount of parameters involved. Furthermore, TDMC(λ) could be compared against TDLeaf(λ) [2], by basing the training on grandmaster game records. Finally, another room for future work would be incorporating Upper Confidence Bounds applied to Trees (UCT) [8][9] instead of Monte Carlo simulation, and use it to compute the substitute rewards for TD(λ), which could perhaps yield further improvement in learning results.

References

[1] Gerald Tesauro,"Temporal difference learning and TD-gammon", *Communications of the ACM*, 38(3):pp.58-68,(1994).

[2] Jonathan Baxter, Andrew Tridgell, Lex Weaver,"Experiments in Parameter Learning Using Temporal Differences", *ICGA Journal* (June):pp.84-99,(1998).

[3] Richard Sutton, Andrew Barto,"Introduction to Reinforcement Learning",*MIT Press*,1998.

[4] Richard Sutton,"Learning to predict by the methods of temporal differences",*Machine Learning*(3):pp.9-44,(1988).

[5] Edward P.Manning,"Temporal Difference Learning of an Othello Evaluation Function for a Small Neural Network with Shared Weights", *IEEE Symposium on Computational Intelligence and Games* (2007): pp.216-223.

[6] Simon M.Lucas, Thomas P.Runarsson,"Temporal Difference Learning Versus Co-Evolution for Acquiring Othello Position Evaluation", *IEEE Symposium on Computational Intelligence and Games* (2006): pp,52-59.

[7] M.buro, "LOGISTELLO-a strong learning othello program" http://www.cs.ualbeta.ca/~mburo/ps/log-overview.ps.gz.

[8] Sylvain Gelly, Yizao Wang, Remi Munos, Olivier Teytaud, "Modification of UCT with Patterns in Monte-Carlo Go", *RR-6062-INRIA* (2006):pp.1-19.

[9] Remi Coulom,"Computing Elo Ratings of Move Patterns in the Game of Go", *Proceeding. of the 6th International Conference on Computers and Games* (2007):pp.113-124.