TD(λ) and Q-learning based Ludo players

Conference Paper · September 2012

DOI: 10.1109/CIG.2012.6374142

Faisal Alvi
King Fahd University of Petroleum and Minerals

TD(λ) and Q-Learning Based Ludo Players

Majed Alhajry, Faisal Alvi, *Member, IEEE* and Moataz Ahmed

Abstract— **Reinforcement learning is a popular machine learning technique whose inherent self-learning ability has made it the candidate of choice for game AI. In this work we propose an expert player based by further enhancing our proposed basic strategies on Ludo. We then implement a TD(λ)based Ludo player and use our expert player to train this player. We also implement a Q-learning based Ludo player using the knowledge obtained from building the expert player. Our results show that while our TD(λ) and Q-Learning based Ludo players outperform the expert player, they do so only slightly suggesting that our expert player is a tough opponent. Further improvements to our RL players may lead to the eventual development of a near-optimal player for Ludo.**

I. Introduction

Ludo [1] is a board game played by 2-4 players. Each player is assigned a specific color (usually Red, Green, Blue and Yellow), and given four pieces. The game objective is for players to race around the board by moving their pieces from start to finish. The winner is the first player who moves all his pieces to finish area. Pieces are moved according to die rolls, and they share the same track with opponents. Challenges arise when pieces knock other opponent pieces or form blockades. Figure 1 depicts Ludo game board and describes different areas and special squares.

In our previous work [2], we evaluated the state-space complexity of Ludo, and proposed and analyzed strategies based on four basic moves: aggressive, defensive, fast and random. We also provided an experimental comparison of pure and mixed versions of these strategies.

Reinforcement learning (RL) is an unsupervised machine learning technique in which the agent has to learn a task through trial and error, in an environment which might be unknown. The environment provides feedback to the agent in terms of numerical rewards [3]. TD(λ) learning is a RL method that is used for prediction by estimating the value function of each state [3]. Q-Learning is an RL method that is more suited for control by estimating the action-value quality function Q which is used to decide what action is best to take in each state [4]. Another method uses evolutionary algorithms (EA) to solve RL problems, as a policy space search approach [5].

Utilizing reinforcement learning techniques in board games AI is a popular trend. More specifically, the TD learning method is preferred for the board game genre because it can predict the value of the next board positions. The latter has been exemplified by the great success of Tesauro's TD-Gammon player for Backgammon [6], which competes with master human players. However, RL is not the silver-bullet for all board games, as there are many design considerations that need to be crafted carefully in order to obtain good results [7]. For example, RL application to Chess [8] and Go [9] yielded modest results compared to skilled human players and other AI techniques implementations.

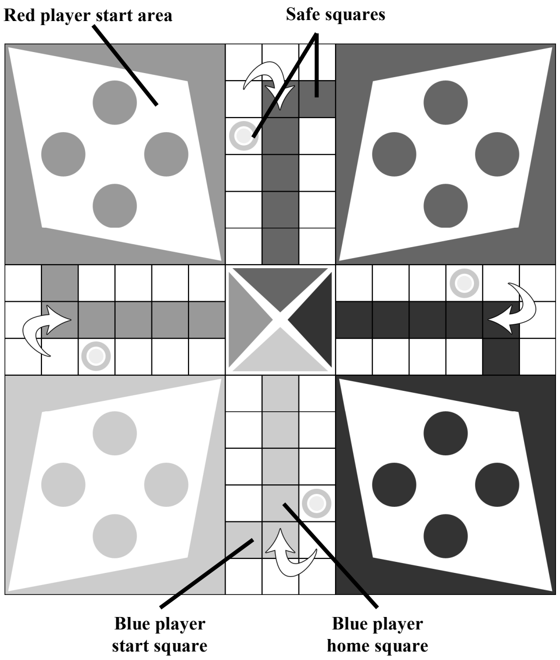


Fig. 1. Ludo Board

In a broader sense, RL has a growing popularity across video games genres since it helps cutting down development and testing costs, while reflecting more adaptive and challenging gameplay behaviors. Complexity in video games rises when AI players need to learn multiple skills or strategies (e.g. navigation, combat and item collection) which are usually learned separately before getting combined in one hierarchy, to help fine-tune the overall performance. RL has shown great success in learning some tasks like combat in first person shooter (FPS) games [10] and overtaking an opponent in a car-racing game [11], while producing less than satisfactory results for other tasks [10]. Evolutionary algorithms approach to RL was used in [12] to

The authors are affiliated with the Information and Computer Science Department, King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia. Email: {g201001880, alvif, moataz}@kfupm.edu.sa.
The authors acknowledge the support of King Fahd University of Petroleum and Minerals for this research, and Hadhramout Establishment for Human Development for support and travel grant.

978-1-4673-1194-6/12/$31.00 ©2012 IEEE

create a game where players teach different skills to a team of agents in the real-time strategy game NERO, by manually providing numerical rewards for a predefined set of behaviors.

Matthews et al. [13] have experimented with reinforcement learning in Pachisi [14] and Parcheesi [15] (two variants of Ludo), using TD learning with eligibility traces, TD(λ) algorithm. Their player works like TD-Gammon by evaluating next board positions and picking the most advantageous one. In their work, they implemented a heuristic player to act as an expert and an evaluation opponent for the RL based player.

In this work we begin with an enhanced mixed-strategy player that will serve as an expert to benchmark our RL player implementation. We then build two RL based players for Ludo: TD player that evaluates next board positions and another Q-Learning player which decides what basic strategy is best suited for a given state. Both our TD-based player and our Q-Learning Player outperform our expert player by learning a strategy. We also include a brief performance comparison of the two players.

The rest of this paper is organized as follows: Section II provides details about our proposed expert player. In Section III we provide a quick overview of reinforcement learning and its notation. In Section IV, we provide details about the TD(λ) player implementation and experiments,. Similarly, Section V provides details about the Q-Learning player. Section VI presents the conclusion by a brief discussion on the obtained results and performance, and finally, section VII concludes with guidelines for future work.

II. A Proposed Expert Player

In our previous implementation [2], we had proposed four basic strategies: defensive, aggressive, fast and random. We had also proposed a mixed or hybrid strategy based on an experimental performance of basic strategies in which strategies were prioritized as follows: defensive, aggressive, fast, and random.

A. Enhanced Strategies

In this work we continue to explore the performance of these strategies with the objective of formulating an expert player. However, to have a more realistic assessment of these strategies' performance, we implemented these basic strategies by applying the following game rules:

1) There are 4 pieces per player.
2) A player needs to roll 6 to release a piece.
3) Blockades are allowed.
4) At least 2 pieces are required to form a blockade.
5) Bonus die rolls are allowed when a player rolls a 6.
6) Releasing a piece is optional when a player rolls a 6.
7) First player is selected at random when the game is started.

Each of the above-mentioned rules had an effect on the performance of strategies. For example, in our previous implementation we did not take blockades into account, but the current experimentation included blockades in order to get more realistic results. Furthermore, we also introduced a 'prefer release move' to each strategy (i.e. a player prefers to release a piece on a die roll of 6). The rationale behind the introduction of this rule was to minimize the number of wasted moves incurred due to release allowed on a die roll of 6 only (Rule 2). We call these new strategies 'enhanced strategies with rules and prefer release' or simply 'enhanced strategies'. Similar to our previous experimentation, we found that the 'enhanced defensive' strategy performs the best among the basic strategies. Each enhanced strategy produced the results against All-Random Players as listed in Table I.

TABLE I
Performance Statistics of Basic Enhanced Strategies

Player 1 (% wins)	Player 2 (% wins)	Player 3 (% wins)	Player 4 (% wins)
Enhanced Defensive ($34.68 \pm 0.68\%$)	All Random ($21.75 \pm 0.49\%$)		
Enhanced Aggressive ($27.08 \pm 0.21\%$)	All Random ($24.3 \pm 0.65\%$)		
Enhanced Fast ($53.78 \pm 0.46\%$)	All Random ($15.4 \pm 0.38\%$)		
Enhanced Mixed ($61.09 \pm 0.41\%$)	All Random ($12.67 \pm 0.31\%$)		

B. The Expert Player

When these 'enhanced strategy players' played in all-strategy games, the performance of strategies was reported as in Table II. The results from Table II lead us to propose an '*expert player*' which is the '*enhanced mixed player*' since it won almost 50% of all the games played which is slightly less than the combined sum of wins of all three players. This expert also won against any individual player by at least twice as many games (48.6 % for Expert vs. 20.5% for Fast). Hence we propose that this expert player will serve as basis for benchmarking and training RL-based players in the forthcoming sections.

III. Reinforcement Learning Primer

As mentioned earlier, RL allows the agent to learn by experiencing with the environment. More formally, in RL the agent perceives the current state s_t at time t, and decides to take an action a_t that leads to a new state s_{t+1} (which might be undeterministic). The environment might provide the agent with an immediate reward r_{t+1} for the state-action-state triad (s_t, a_t, s_{t+1}). The agent keeps track of the value function, which is the long-run expected payoff for each

TABLE II
Performance Statistics of Enhanced Mixed Strategies Vs. Enhanced Basic Strategies

Player 1 (% wins)	Player 2 (% wins)	Player 3 (% wins)	Player 4 (% wins)
Mixed ($48.64 \pm 0.41\%$)	Defensive ($18.24 \pm 0.35\%$)	Aggressive ($12.6 \pm 0.23\%$)	Fast ($20.52 \pm 0.19\%$)

state $V(s_t)$ in order to evaluate its performance, with an objective to build a policy for deciding the best action to take in each state, so that it maximizes its future payoff, according to what has been learned from the previous experience.

A. Mathematical Model

A RL problem is a Markov decision process which is described as a 4-tuple $(S, A, P(\cdot,\cdot), R(\cdot,\cdot))$ as follows:

- S is the set of all environment states the agent can perceive.
- A is the set of all possible actions an agent can take.
- $P_a(s, s')$ is the transition model, which describes the probability of moving from state s to state s' given that action a was taken.
- $R_a(s, s')$ is the immediate reward received after transition to state s' from state s, given that action a was taken.

The ultimate goal of reinforcement learning is to devise a policy (or a plan) π which is a function that maps states to actions. The optimal policy π^* is the policy that produces the maximum cumulative rewards for all states:

$$\pi^* = \underset{\pi}{argmax}\, V^\pi(s), \forall s \in S$$

where $V^\pi(s)$ is the cumulative reward received (i.e. the value function) from state s using policy π.

Sometimes we might have a complete model for the environment. In this case, if the model was tractable then the problem becomes more of planning than learning, and can be solved analytically. When the model is incomplete (for example P or R are not adequately defined, rewards are not delivered instantly or the state space is too large), then the agent has to "experience" with the environment in order to estimate the value function for each state.

B. Finding Optimal Policy

There are two approaches to find the optimal policy: searching the policy space or searching the value function space. Policy space search methods maintain explicit representations of policies and modify them through a variety of search operators, while value function space search attempt learn the value function V^π.

Policy space search methods are particularly useful if the environment is not fully-observable. Depending on the attributes of the model, different methods can be used, like dynamic programming, policy iteration, simulated annealing or evolutionary algorithms.

Value function space search methods are concerned about estimating the value function for the optimal policy, through experience. They tend to learn the model of the environment better than policy state search. There are two methods to this approach: Monte-Carlo and Temporal-Difference (TD) learning, with the latter being dubbed as "the central and novel idea to reinforcement learning" [3].

For large state-action spaces, function approximation techniques (e.g. neural networks) can be used to approximate the policy and the value functions.

C. Temporal-Difference Learning

TD learning is a set of learning methods that combine Monte-Carlo methods by learning directly from experience with the environment, and dynamic programming by keeping estimates of value functions and updating them based on successor states (backing-up) [16]. Two of the most popular TD learning methods are TD(0) which uses prediction, and Q-Learning, which is a control method.

TD(0) [17] is the simplest state evaluation algorithm, in which value function is updated for each state according to the following formula:

$$V(s_t) = V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1} - V(s_t)]$$

where $0 < \alpha < 1$ is the learning rate and γ is the discount factor which models how much the future reward affect the current reward.

Q-learning [4] focuses on finding the optimal policy by estimating quality values for each state/action combination (known as Q-Values), and updates them in a manner similar to TD(0):

$$Q(s_t, a_t) = Q(s_t\,, a_t) \; + \; \alpha[r_{t+1} + \gamma \; \max_{a' \in A} Q(s_{t+1}\,, a') \\ - Q(s_t\,, a_t)]$$

Again, $0 < \alpha < 1$ is the learning rate and γ is the discount factor.

Q-learning is called an off-policy method, in which it uses a stationary policy for decision making, while improving another policy during the learning process. TD(0) is an example of searching the value function space, while Q-learning searches the policy space.

The algorithms discussed so far propagate rewards back for one time step only. A more general technique known as TD(λ) [18] which introduces a new "trace decay" parameter that reflects the proportion of credit from a reward that can be given to earlier state-action pairs for more steps, allowing more efficient learning from delayed rewards.

IV. TD(λ) Ludo Board Evaluator

In this section we discuss the design and implementation of a board evaluator for Ludo using TD(λ) algorithm. We use this evaluator to implement a player that picks the highest rated possible next move as dictated by the evaluator.

The board evaluator acts as an observer that learns by watching consecutive game boards from start to end (i.e. learning episodes), and attempting to evaluate each board. It receives feedback from the environment at the end of each episode in terms of actual evaluations.

A. Learning Objective

Similar to TD-Gammon [6], the evaluator's objective is to estimate the probability of winning for each player in any given state. More formally, the evaluator is a value function $v(\vec{s})$ that takes a state vector as an input $\vec{s} \in \mathbb{R}^n$ and returns a 4-dimensional vector $\vec{p} \in \mathbb{R}^4$, where each element in $\vec{p} = (p_1, p_2, p_3, p_4)$ corresponds to the probability of winning for the corresponding player given that state, or $p_i = P(\text{Player } i \text{ wins}|\vec{s})$.

A player based on this evaluator has to pick the move with the highest probability of winning, while making sure no other players may get advantage from that move. We define the utility of an evaluation vector $\vec{p}$ for player i as $u(\vec{p}, i) = p_i - \sum_{j \neq i} p_j$. The player has to pick the move with the maximum utility to guarantee optimal play.

To achieve the learning objective stated, we chose the TD(λ) because the actual feedback a learning agent receives is provided at the end of each episode (i.e. game over state), and TD(λ) is more efficient at handling delayed rewards.

As per our previous work [2], we found the Ludo state-space complexity to be approximately 10^{22} which is slighly larger than that of Backgammon. Thus, storing the value function in tabular format is infeasible. Hence, we utilized function approximation technique in the form of artificial neural network to estimate the value function.

B. State Representation

Ludo's board circular track has 52 squares, and each player has an additional 5 home squares to pass. Besides, pieces might be at the start area square or finish. These sum up to 59 squares available for each player pieces to occupy, with a maximum of 4 pieces per square per player.

Instead of using the famous truncated unary representation utilized in TD-Gammon [6], we opted in for a simpler representation we refer to as raw representation: For each player, we represent each square with a real number that indicates the number of pieces occupying the square for that player; normalized by division by 4 (i.e. the value 1 indicates 4 pieces). In addition, we added 4 unary inputs that indicate the current player turn. Thus, the total number of inputs using this representation is $(59 \times 4) + 4 = 240$.

The reason behind using raw representation is that it uses fewer inputs, which means shorter training time for the player, while preserving representation smoothness [7]. The representation adheres to an objective perspective, where it is viewed by a neutral observer who doesn't play the game [19].

C. Rewards

At the end of each episode, the game can easily distinguish the winner, and provide a reward vector that corresponds to the actual winning probabilities for each player. Let $\vec{r} = \{r_1, r_2, r_3, r_4\}$ be the reward vector, and the winner is i, then all $r_j = 0; i \neq j$ and $r_i = 1$.

D. Learning Process

For the learning process, we experimented with 3 different settings of player combinations, in order to test the effect of learning opponents:

1) 4 TD-based players (self-play).
2) 2 TD-based players and 2 expert players.
3) 2 TD-based players and 2 random players.

Other combinations that do not include TD-players are possible, but these should suffice to give a clear idea about the performance gain/loss introduced by altering player combinations.

For each setting, the evaluator is trained 100000 times, using a neural network of 20 hidden layers. The learning parameter α is set to 0.2 to balance between exploitation and exploration with less noise, and the trace decay λ is set to 0.7.

E. Experiment Design

We evaluated each setting of learning player combinations and number of training episodes against 3 random players and 3 heuristic players, using winning rate as performance measure. Each test is executed 4 times, with 5000 game plays per test.

It should be noted that because we fixed the player positions during training, the evaluator may output incorrect evaluations for certain players, specifically, those which were trained using inexpert players. The reason lies in the fact that the evaluator does not explore enough states for inexpert players. Hence, we circulate the TD player to play with different color in each test (4 tests accommodate all 4 colors) to add more credibility to the results.

We recorded the mean, minimum and maximum winning rate for each 2500 learning episodes. The mean winning rate provides a realistic performance measurement because in real life application, an intelligent player should perform well regardless of each side of the table it sits on.

F. Results

1) 4 TD Players (self-play):

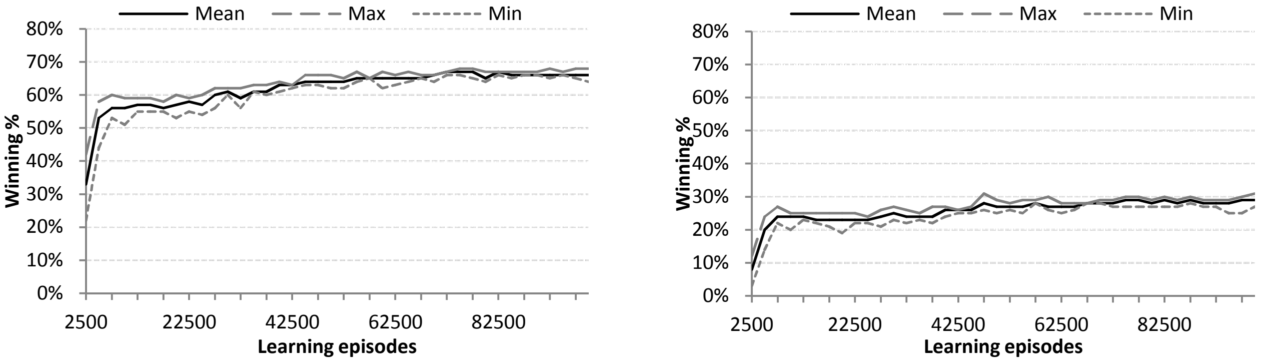
Fig. 2 (a) shows the winning rate of self-play trained player against 3 random players. After 30000 learning episodes, its performance starts to match that of an expert player (which wins 61% against random players), steadily increasing to outperform it. After 80000 training episodes, the performance stabilizes at around 66%, and does not show any significant enhancement afterwards.

The minimum and maximum performance measures are very tight around the mean, suggesting that this evaluator does not suffer from noisy evaluations caused by inexperienced players.

Fig. 2 (b) illustrates direct comparison against 3 expert players. After 30000 learning episodes, the player begins to match the expert players (25% wins), continuing to increase until it manages to outperform expert player with 30% wins, after 80000 episodes.

2) 2 TD Players and 2 Expert Players:

Using an evaluator trained by observing 2 TD-based players and 2 expert players, Fig 3 (a) shows the results for playing against 3 random players. The player outperforms basic strategy players at around 7500 learning episodes (54% wins). After 80000 episodes, it slightly outperforms expert player performance against random players (62% wins), with no significant increase in performance



(a) Winning rate against 3 random players

(b) Winning rate against 3 expert players

Fig. 2. Performance graphs for TD-player trained using 4 TD-players

afterwards. This becomes more elaborate in Fig 3 (b), where results show that the player hardly outperforms 3 expert players with 26% wins after 85000 learning episodes.

We observe that introducing an expert player to the mix did not increase the average performance of the player, simply because observing the expert player adds more exploitation to the learning process. However, the other TD-based players push more exploration, yielding results that are not far degraded from the self-learning player.

We also observe the increase in the gap between minimum and maximum performance, because the evaluator has developed more noisy evaluations for expert players.

3) 2 TD Players and 2 Random Players:

Learning by observing random players introduces more exploration with unmatched exploitation during the learning process. Fig. 4 (a) and (b) illustrates the degraded mean performance of this player against 3 random and 3 expert players, respectively.

The low minimum performance reflects incorrect board evaluations caused when the player plays on sides trained by random players. The maximum performance, however, is still on par with the previous results.

V. Q-LEARNING BASED LUDO PLAYER

In this section we discuss the design and implementation of a Q-learning (QL) Ludo player. In each game state, the player has to select one of the basic strategies in a way that maximizes future rewards. We base our implementation on the knowledge we obtained from expert player's behavior.

A. Learning objective

A Q-learning Ludo player's objective is to select the best applicable action for a given state. The set of actions A is defined a list of basic moves we proposed earlier, i.e. $\mathrm{A} = \{\mathrm{defensive}, \mathrm{aggressive}, \mathrm{fast}, \mathrm{random}, \mathrm{preferRelease}\}$.

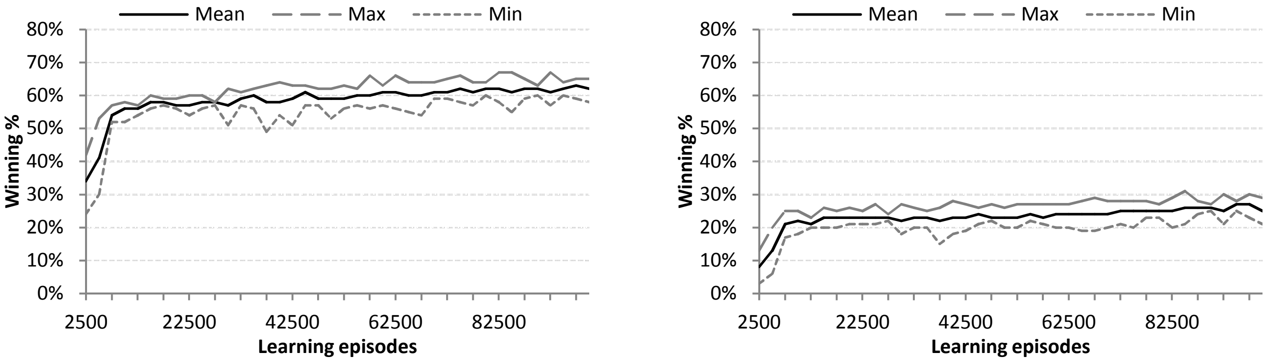
Since this is a control problem, and the rewards are provided immediately after selecting the move, we chose Q-Learning (without trace decay parameter) to train this player.

Similar to TD-player, we used artificial neural network to estimate the Q function.

B. State representation

We use a representation similar to that of section 5.2 with two differences:

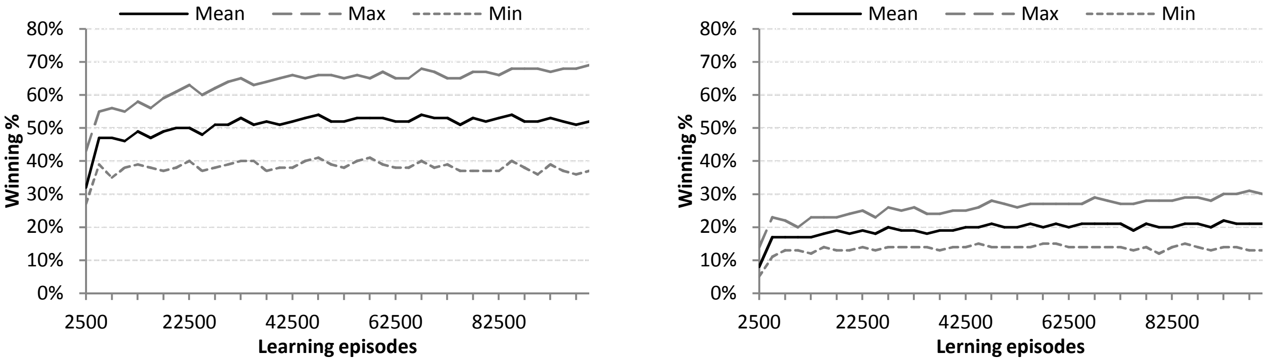
- The representation is subjective [19], i.e. it represents the board as seen by the current player.
- No turn indicators added, because the subjective representation already includes that information.



(a) Winning rate against 3 random players

(b) Winning rate against 3 expert players

Fig. 3. Performance graphs for TD-player trained using 2 TD-players and 2 expert players



(a) Winning rate against 3 random players

(b) Winning rate against 3 expert players

Fig. 4. Performance graphs for TD-player trained using 2 TD-players and 2 random players

Hence, the total number of inputs using this representation is $59 \times 4 = 236$.

C. Rewards

The rewards were designed to encourage the player to pick moves that achieve a combination of the following objectives (in descending order):

- Win the game
- Release a piece.
- Defend a vulnerable piece.
- Knock an opponent piece.
- Move pieces closest to home.
- Form a blockade.

On the other hand, the agent is penalized in the following situations:

- Getting one of its pieces knocked in the next turn.
- Losing the game.

The rewards are cumulated in order to encourage the player to achieve more than one objective of higher values.

D. Learning process

We experimented with 3 different settings of player combinations, in order to test the effect of learning opponents:

1) 4 QL-based players (self-play).
2) 2 QL-based players and 2 expert players.
3) 2 QL-based players and 2 random players.

All QL-based players share and update the same neural network for faster learning. Each setting is trained 100000 times, using a neural network of 20 hidden layers. To boost the learning process, we set the learning parameter $\alpha = 0.5$, discount factor $\gamma = 0.95$. We used an ε-greedy [18] policy which selects an action at random with probability $1 - \varepsilon$ to encourage exploration. The value of ε is set to 0.9 and decayed linearly to reach 0 after 30000 learning episodes.

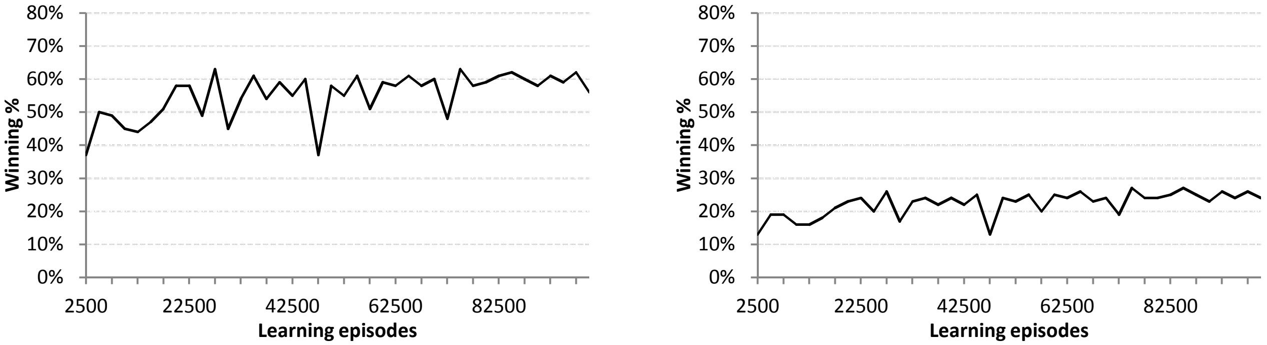
The rewards and penalties are defined as following:

- 0.25 for releasing a piece.
- 0.2 for defending a vulnerable piece.
- 0.15 for knocking an opponent piece.
- 0.1 for moving the piece closest to home.
- 0.05 for forming a blockade.
- 1.0 for winning the game.
- -0.25 for getting one of its pieces knocked.
- -1 for losing the game.

These values are directly influenced by the knowledge we obtained from building the expert player.

E. Experiment design

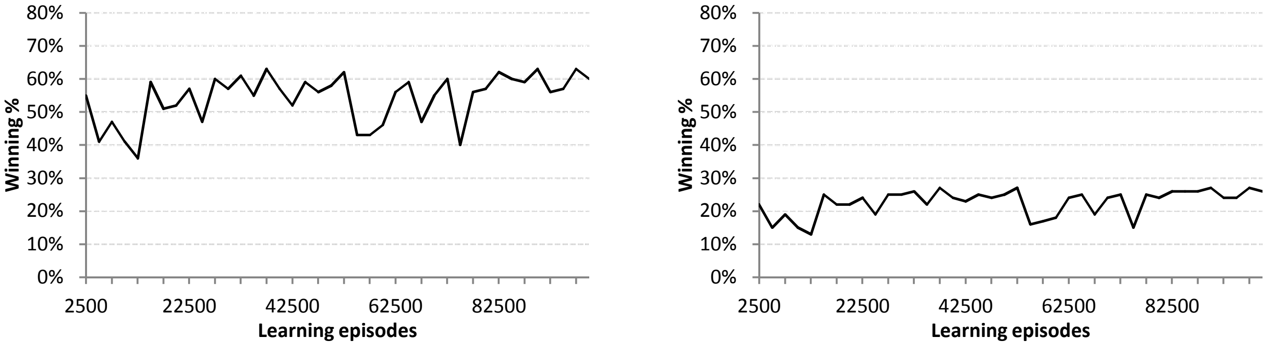
We used a testing setup similar to section 5.5 for each player combination. Since we're using subjective representation, the QL players do not suffer from the effect



(a) Winning rate against 3 random players

(b) Winning rate against 3 expert players

Fig. 5. Performance graphs for QL-player trained using 4 QL-players (self-play)



(a) Winning rate against 3 random players

(b) Winning rate against 3 expert players

Fig. 6. Performance graphs for QL-player trained using 2 QL-players and 2 expert players

of changing sides during game play, so we opted to record the mean performance only.

F. Results

1) 4 QL-players (self-play):

Figure 5 (a) shows the winning rate of self-play trained QL-player against 3 random players. We observe the noisy learning curve due to the high value of learning parameter α. However, the player still manages to learn good game play after 75000 episodes ($63 \pm 1\%$ wins), and the performance relatively stabilizes afterwards.

Figure 5 (b) illustrates direct comparison against 3 expert players. The player manages to slightly outperform 3 expert players at $27 \pm 1\%$ wins after 75000 episodes.

2) 2 QL and 2 Expert Players

Figure 6 (a) and (b) shows the winning rate of this player against 3 random players and 3 expert players, respectively. The results did not indicate any advantage when learning against expert player, in terms of winning rate. They show, however, faster learning than self-play. Maximum performance against 3 expert players is still $27 \pm 1\%$.

3) 2 QL-players and 2 Random Players

Once again, the player managed to slightly outperform the expert player.as seen in figures 7 (a) and (b). The results did not indicate any disadvantage when learning against random player, in terms of winning rate, but the negative performance strikes increased. The maximum winning rate is also $27 \pm 1\%$ against 3 expert players.

G. TD-players against QL-players

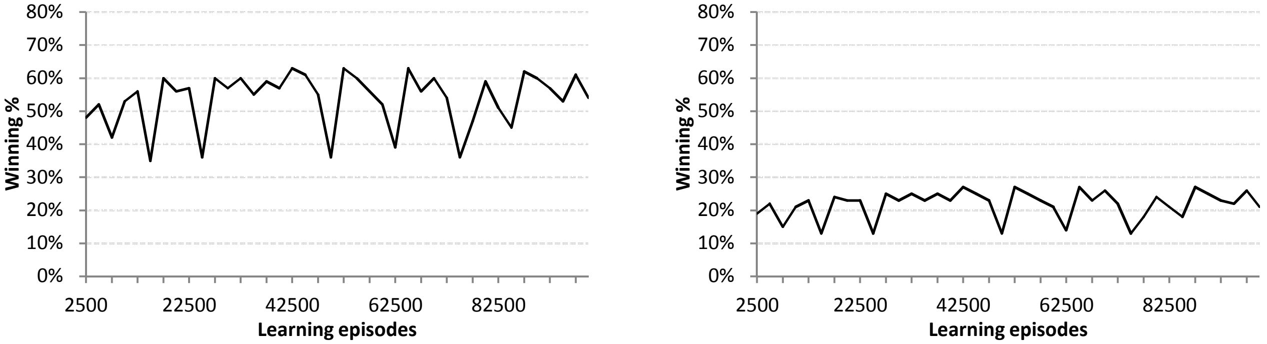
We performed one final test for the best 2 TD-players against the best 2 QL-players. The results are summarized in Table III below. We observe that the TD-player outperforms the QL-player by a margin of 5% (27.3% wins for TD vs. 22.3% wins for QL), which suggests that the TD-player is somewhat better than the QL-player.

TABLE III
PERFORMANCE STATISTICS OF TD-PLAYER VS. QL-PLAYER

Player 1 (% wins)	Player 2 (% wins)	Player 3 (% wins)	Player 4 (% wins)
Both TD-Players ($27.37 \pm 0.22\%$)		Both QL-Players ($22.37 \pm 0.22\%$)	

VI. CONCLUSIONS

In this work, we built an expert Ludo player by enhancing the basic strategies we proposed earlier. We used this expert player for training and evaluation of two RL-based players. which were based on TD-Learning and Q-Learning.



(a) Winning rate against 3 random players

(b) Winning rate against 3 expert players

Fig. 7. Performance graphs for QL-player trained using 2 QL-players and 2 random players

Both TD and QL players exhibited better game play against a team of 3 expert players. The TD player showed slightly better performance (30% winning rate against 3 experts) than QL player (27% winning rate against 3 experts), most probably due to the learning parameters we used to train QL player.

An important conclusion we draw from the obtained results is that both RL players have learned a strategy that is somewhat an improved version of the expert player's one, because both players did not gain significant improvement against the expert player. The TD player's self-learning capabilities support this argument since it did not improve that much under different settings, suggesting that the expert player serves as an excellent training and evaluation opponent for RL applications to Ludo.

VII. Future Work

Several directions of further work arise from this research. One way in which we can improve performance of TD player is by implementing deeper game tree search like TD-Leaf [8]. We may also improve QL-player further by optimizing the reward function. This can be achieved by experimenting with graduating values used for rewards and penalties and achieving optimization by producing the highest winning rate. Another possible direction to explore is the analysis of RL player moves to enhance the expert player behavior. These optimizations may lead to improve game-play eventually resulting in the development of a near-optimal player for Ludo.

References

[1] "Ludo (Board Game)," (Accessed: 16-Feb-2011). [Online]. Available: http://en.wikipedia.org/wiki/Ludo (board game)

[2] F. Alvi and M. Ahmed, "Complexity Analysis and Playing Strategies for Ludo and its Variant Race Games," in *IEEE Conference on Computational Intelligence and Games (CIG)*, 2011, pp. 134-141.

[3] R. S. Sutton and A. G. Barto, *Reinforcement Learning – An Introduction*, MIT Press, 1998.

[4] C. Watkins and P. Dayan, "Q-Learning," *Machine Learning*, vol. 8, no. 3-4, 1992.

[5] D. Moriarty , A. Schultz and J. Grefenstette, "Evolutionary Algorithms for Reinforcement Learning," *Journal of Artificial Intelligence Research*, vol. 11, pp. 241-276, 1999.

[6] G. Tesauro, "Practical Issues in Temporal Difference Learning," *Machine Learning*, vol. 8, pp. 257–277, 1992.

[7] I. Ghory, "Reinforcement Learning in Board Games," Depart. of Computer Science, Univ. of Bristol, Tech. Rep., May 2004.

[8] J. Baxter, A. Tridgell and L. Weaver, "TD-leaf(λ): Combining Temporal Difference Learning With Game-tree Search," in *Proceedings of the 9th Australian Conference on Neural Networks (ACNN'98)*, 1998, pp. 168-172.

[9] D. Silver, R. Sutton and M Müller, "Reinforcement Learning of Local Shape in the Game of Go," in *Proceedings of the 20th international joint conference on Artificial intelligence (IJCAI'07)*, 2007, pp. 1053-1058.

[10] M. McPartland and M. Gallagher, "Reinforcement Learning in First Person Shooter Games," *IEEE Transactions On Computational Intelligence And AI In Games*, vol. 3, no. 1, pp. 43-56, 2011.

[11] D. Loiacono, A. Prete, P. Lanzi, and L. Cardamone, "Learning to Overtake in TORCS Using Simple Reinforcement Learning," in *IEEE Congress on Evolutionary Computation (CEC)*, 2010, pp. 1-8.

[12] K. O. Stanley, B. D. Bryant, and R Miikkulainen, "Real-time Neuroevolution in the NERO Video Game," IEEE Transactions on Evolutionary Computation, vol. 9, no. 6, pp. 653-668, 2005.

[13] G. F. Matthews and K. Rasheed, "Temporal Difference Learning for Nondeterministic Board Games," in *Intl. Conf. on Machine Learning: Models, Technologies and Apps. (MLMTA'08)*, 2008, pp. 800–806.

[14] V. K. Petersen, "Pachisi & Ludo," (Accessed: 16-Feb-2012). [Online]. Available: http://pachisi.vegard2.no/index.html

[15] "Parcheesi," (Accessed: 16-Feb-2012). [Online]. Available: http://en.wikipedia.org/wiki/Parcheesi

[16] S. Zhioua, "Stochastic Systems Divergence Through Reinforcement Learning," PhD thesis, Univ. of Laval, Quebec, Canada, 2008.

[17] R. Sutton, "Learning to predict by the methods of temporal differences," *Machine Learning*, pp. 9-44, 1988.

[18] C. Watkins, "Learning from delayed rewards," PhD thesis, Cambridge Univ., Cambridge, England, 1989.

[19] G. F. Matthews, "Using Temporal Difference Learning to Train Players of Nondeterministic Board Games," M.S. thesis, Georgia Institute of Technology, Athens, GA, 2004.