Complexity Analysis and Playing Strategies for Ludo and its Variant Race Games

Faisal Alvi, *Member IEEE*, Moataz Ahmed

***Abstract*— Ludo is a 2-4 player non-deterministic race game with the objective of moving players' pieces through a designated circuit into a winning location, in accordance with die rolls. In this paper we evaluate the state-space complexity of Ludo, and propose and analyze strategies based on four basic moves. We also provide an experimental comparison of pure and mixed versions of these strategies. This research is aimed at enhancing the domain specific knowledge for Ludo and its variant race games, which can then be used for performance improvement in temporal difference learning networks or in evolutionary game analysis for race games.**

I. Introduction

Race games [1] are board games with the objective of being the first player to get one's pieces around a linear track and into a designated winning location, usually in accordance with die rolls. These range from the simple games such as snakes and ladders in which game-play depends entirely on chance, to the more complex ones such as backgammon which involves the use of several strategies.

Ludo [2], a derivative of Pachisi [3], is a non-deterministic race game with 2-4 players. Each player is represented by one of the colors Red, Green, Blue and Yellow, and has four pieces. The first player to circumnavigate all four pieces around the board and into the home area is the winner. Obstacles to this objective include shared paths with opponent pieces, unlucky die rolls, getting knocked off by opponent pieces and piece doubling. Several variants of the game exist, for example Parcheesi (United States), Ludo (Britain), Parqués (Columbia), Parchis (Spain) and Ludo (South Asia) [3].

Temporal Difference Learning [4] is one of the successful methodologies for developing game-playing agents in board games. Tesauro's successful application of the basic TD(λ) algorithm to produce TD-Gammon is the best known example [5]. However, application of the basic TD(λ) algorithm for other board games such as Chess [6] and Othello [7], has produced mixed results. Variations in learning modes such as learning by playing against an expert player and learning by observation have produced improved game-play [8], in contrast to the self-play learning approach used by Tesauro. Another promising approach for improved game-play in board games has been evolutionary computation [9], with the incorporation of domain-specific knowledge at various stages in game-play resulting in enhanced performance in checkers [10].

The authors are affiliated with the Information and Computer Science Department, King Fahd University of Petroleum and Minerals, Dhahran-31261, Saudi Arabia. email: `alvif@kfupm.edu.sa`, `moataz@kfupm.edu.sa`. The authors acknowledge the support of King Fahd University of Petroleum and Minerals for this research.

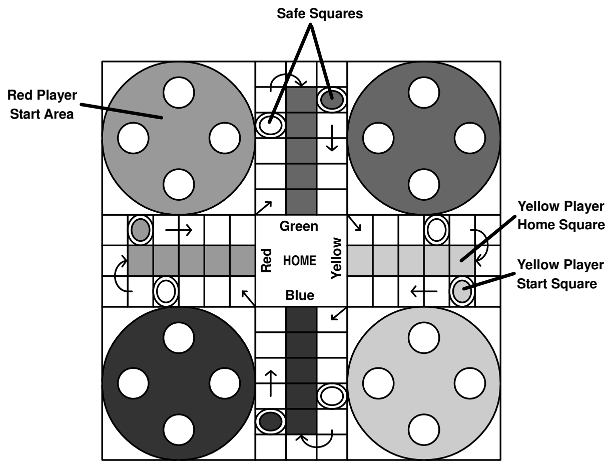


Fig. 1. Ludo Board with Safe Squares (South Asia)

Matthews et al. [11] have applied temporal difference learning to Parcheesi [12] (a variant of Ludo), using the basic TD(λ) algorithm. They have experimentally shown that, using derived smart features of the board, and training against heuristic Parcheesi players, initial learning can be achieved in a minimum of 5000 games of training, and performance level reaches a steady state at approximately 50,000 games of training. In the presence of a standard expert player, improved learning can be achieved using fewer trials. However, they state that no standard expert player exists, due to a lack of an in-depth study of Parcheesi [11].

In this work, we undertake a fundamental study of Ludo in order to gain a better understanding of the game and enhance its knowledge base. We analyze the state-space complexity of Ludo and show that it is comparable to the state-space complexity of Backgammon, which probably indicates that Ludo has some strategic variety and is not a trivial game [3]. Accordingly, we propose strategies based on four basic moves in the game; we give a comparative analysis of strategies based on expected number of moves. Experimental results of the comparison of pure and mixed versions of these strategies is also included to complement the theoretical results. The enhanced knowledge base resulting from this work can be used to generate standard players as a benchmark for TD-learning agents to compete against, thereby producing better players and reducing the required number of games for training. It can also be used for evolving better strategies in an evolutionary game-play approach for race games.

The rest of the paper is organized as follows: In Section 2, we evaluate the state-space complexity of Ludo. In Section 3, we propose strategies based on four basic move types in the game and in Section 4 we provide a comparative analysis of these strategies. We describe results of experimentation on strategies in Section 5. Section 6 concludes the paper by summarizing the achievements and highlighting future work directions.

II. State-Space Complexity

State-space complexity is defined as the number of legal game states reachable from the initial state of the game [13]. Estimated state-space complexities of some of the popular board games are: Chess (10^{50}) [13], Othello (10^{28}) [13], Backgammon (10^{20}) [14] and Checkers (10^{18}) [15]. The primary application of state-space complexity is to determine whether a perfect evaluation function can be constructed using table look-up by listing and evaluating every possible state of a game [13]. However, this reasoning may not be applicable to games for which a perfect algorithmic strategy has been discovered. The complexity class of generalization of a game is also used as an indication of game complexity, however this is dependent on the constructions used for generalization, for example in $n \times n$ chess [16].

It can be argued that a high value of state space complexity is not a definitive indicator of game complexity. For example, although the game of Nim (with n piles having n marbles each) has state-space complexity of the order of $4^n/\sqrt{n}$ which is huge for n = 140 [17], yet a simple polynomial-time perfect strategy exists for the game.

However, a smaller state-space complexity is a definitive indicator that a game may be solved using enumeration. As an example, although an extended generalization of tic-tac-toe is PSPACE-Complete [18] , yet a perfect evaluation function may be constructed for a 3×3 tic-tac-toe game since its state space complexity is bounded above by 10^3.

At present no attempts have been made to compute the complexity of a generalized version of Ludo; several variants of the game may provide some hints in that direction. In the absence of such a generalization, obtaining a reliable value of the state space complexity for Ludo is one way to rule out the possibility that Ludo can be easily solved by enumerating every possible state of the game.

A. Assumptions

A trivial upper bound of (*number of locations*)^{16} can be established by considering that each of the 16 pieces (4 players $\times$ 4 pieces/player) can be placed on any location on the board. For the board in fig. 1 this turns out to $57^{16} \approx 10^{28}$. However this estimate is overly simplistic as several of the game's rules are not taken into consideration. Below we state certain assumptions based on rules and observations within the game, which are used later in the evaluation of state-space complexity:

- We consider all pieces of a player to be identical, since unlike chess, there is no hierarchy between individual pieces of the same player. Hence a board configuration in which Red's pieces are on locations (3, 4, 2, 1) is identical to another configuration in which Red's pieces are on locations (4, 3, 2, 1).
- We classify the common locations on a Ludo board into two types: safe squares and non-safe squares (fig. 1). A *safe square* is defined to be a location on the game board where multiple pieces of different players can be placed simultaneously without being knocked off or sent back. A *non-safe square*, on the other hand, can accommodate only one piece at a time.
- The four start area locations for each player can be mapped into an imaginary single safe square for all pieces. Similarly, the five home squares for each player can be mapped to five common safe squares; likewise the final *home* location (fig. 1) can also be considered as a safe square. Therefore, the number of *computed* safe squares is defined to be equal to the sum of actual number of safe squares and these *mapped* safe squares.

B. Theorem

The following theorem establishes an upper bound on the state-space complexity of Ludo. This upper bound is almost-tight as it includes only one location which is not legal for each piece, i.e., the location right before the start square for each piece. We will also establish a lower-bound by excluding this location (square) from the board and show that the upper bound and the lower bound differ by only one order of magnitude for a typical game board.

Theorem 1: For a Ludo board, let
n_s = number of actual safe squares,
n_o = number of non-safe squares,
n_h = number of home squares for any player,
n_c = number of computed safe squares = n_s + n_h + 2,
r, g, y, b = number of pieces of Red, Green, Yellow, Blue, player on non-safe squares at any given instant,

Then, the number of states is bounded above by:

$$\begin{aligned} f(n_o, n_c) = \sum_{r,g,y,b=0}^{4} & C(n_o, r) \cdot C(n_c + 3 - r, 4 - r) \\ & \cdot C(n_o - r, g) \cdot C(n_c + 3 - g, 4 - g) \\ & \cdot C(n_o - (r+g), y) \cdot C(n_c + 3 - y, 4 - y) \\ & \cdot C(n_o - (r+g+y), b) \cdot C(n_c + 3 - b, 4 - b) \qquad (1) \end{aligned}$$

where $C(n, r) = \dfrac{n!}{r!(n-r)!}$

Proof: Initially, consider the locations for the Red Player. At any given instant let there be r pieces of the Red Player on any of the n_o non-safe squares of the board and the other $4-r$ pieces be on any of the n_c computed safe squares. These r pieces can be placed on any of the n_o squares of the board in $C(n_o, r)$ ways, and the other $4 - r$ pieces can be placed on the remaining n_c computed safe squares in $C(n_c + (4 - r) - 1, 4 - r) = C(n_c + 3 - r, 4 - r)$ ways [combinations with repetition] [19]. Then the number of ways in which Red's pieces can be placed is $\sum_{r=0}^{4} C(n_o, r) \cdot C(n_c+3-r, 4-r)$. For the next player (let's say Green), the number of available non-safe squares is reduced to $n_o - r$, however there is no change in the available n_c computed safe squares. Therefore, the number of possible locations for Green is given by $\sum_{g=0}^{4} C(n_o - r, g) \cdot C(n_c + 3 - g, 4 - g)$. Further application of these formulae for Yellow Player give $C(n_o - (r+g), y) \cdot C(n_c + 3 - y, 4 - y)$ ways and for Blue player gives $C(n_o - (r + g + y), b) \cdot C(n_c + 3 - b, 4 - b)$ ways. The resulting expression for $f(n_o, n_c)$ is the product of these using the product principle. ■

Corollary 1: The state-space complexity of a Ludo-game has a lower bound of $f(n_o - 4, n_c)$.

Proof: As explained in the introduction to *Theorem* 1, the computation of the expression for upper bound includes one illegal non-safe location for each piece. By excluding all such locations (4 in total), we come up with the expression for lower bound. ■

Remark 1: Using equation (1), we find that the state-space complexity of Ludo using the board as shown in fig. 1 is 10^{22}. This can be obtained by substituting the values, $n_o = 44$, $n_h = 5$, $n_s = 8$, $\Rightarrow n_c = 5+8+2 = 15$ in equation (1), and getting a resulting value of $\Rightarrow f(n_o, n_c) = 1.6 \times 10^{22}$. Similarly a lower bound of $f(n_o - 4, n_c) = 5 \times 10^{21}$ is also computed. It is clear that the lower bound differs from the upper bound by a factor of 10 only.

A state-space size of 10^{22} is slightly larger than that of Backgammon, hence it may not be possible to create a perfect evaluation function using table look-up. It may be pertinent to mention here that Ludo is played with some variations in rules and boards worldwide. Equation (1) can be used to find state-space complexity for a number of variants of the game.

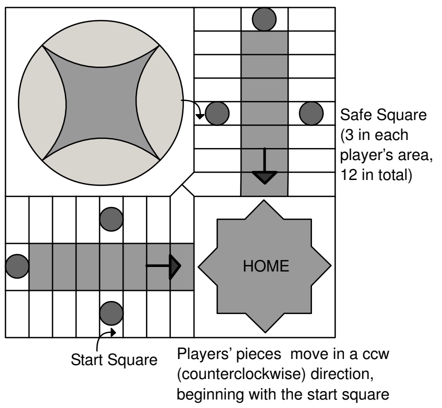

Safe Square (3 in each player's area, 12 in total)
HOME
Start Square
Players' pieces move in a ccw (counterclockwise) direction, beginning with the start square

Fig. 2. Parcheesi Board (Top-left and Center Shown)

Remark 2: For Parcheesi (Fig. 2), a popular race game played in the US, the game circuit for a player consists of 72 squares. However, there are 12 safe squares on the board, 56 non-safe squares and 7 home squares for each player. Out of the 56 non-safe squares, four locations are illegal for each player. Since game rules prohibit placing more than one piece on a safe square, we substitute n_o = 56 + 12 = 68, n_c = 0 + 7 + 2 = 9 in equation (1) to get an upper bound of $f(n_o, n_c) \approx 10^{24}$ and a lower bound of $f(n_o - 16, n_c) \approx 10^{22}$. As compared to Ludo, the higher value of both the lower bound and upper bound is due to the larger number of squares on a Parcheesi board.

C. Blockades

A blockade is defined as two pieces of the same player being placed on a non-safe square, which cannot be knocked off or passed over. In our computation for the state-space complexity, we did not take blockades into account. However, it can be argued that inclusion of game-states with blockades will not significantly affect the overall state-space complexity of Ludo. The primary reason is that a blockade is equivalent to three pieces of a player being placed on the board, with the fourth piece deterministically taking one of the three possible locations of the blockade. Hence, states with blockades form a subset of the states presented by equation (1). Furthermore, since the pieces are considered to be identical for each player, the number of game-boards with blockades is at most $3 \times 64 \times g(n_o, n_c)$. (Reason: Each player can have at least 0 and at most 2 blockades, hence $4^3 = 64$ blockades $\times$ 3 possible blockade locations). Here the function $g(n_o, n_c)$ is strictly less than $f(n_o, n_c)$, since in $g(n_o, n_c)$ at least one summation for the variables r, g, y and b is reduced to $\sum_{r,g,y,b=0}^{3}$ instead of $\sum_{r,g,y,b=0}^{4}$.

III. Strategies

The game-tree branching factor for Ludo is 24 corresponding to 4 pieces $\times$ 6 possible die rolls for each player. After a player has rolled a die in his move, he has four options to move his pieces. In this section we classify basic types of moves that a player may choose during game-play. Then, we outline various strategies based on these types of moves.

A. Random

In a *random* move, a player chooses to play one of his pieces completely at random. Such a move may be undertaken when a player may see no advantage in moving a particular piece.

A *random strategy* is based on random moves during the entire game. Although a random strategy has little usefulness for winning games, it can be used as a benchmark to compare performance of other strategies.

B. Aggressive

In an *aggressive* move, a player prefers to move a piece which can knock out or eliminate the piece of another player, based on the die roll outcome. An aggressive move incurs an additional overhead on the attacked player in the sense that the attacked player now has to play his eliminated piece over the entire game board again. Thus, an aggressive move gives a definite advantage to the aggressor over its victim.

An *aggressive strategy* is based on aggressive moves throughout the game, whenever possible. However, a player may use another type of move (e.g. a random move), when there is no chance of attacking another player's piece.

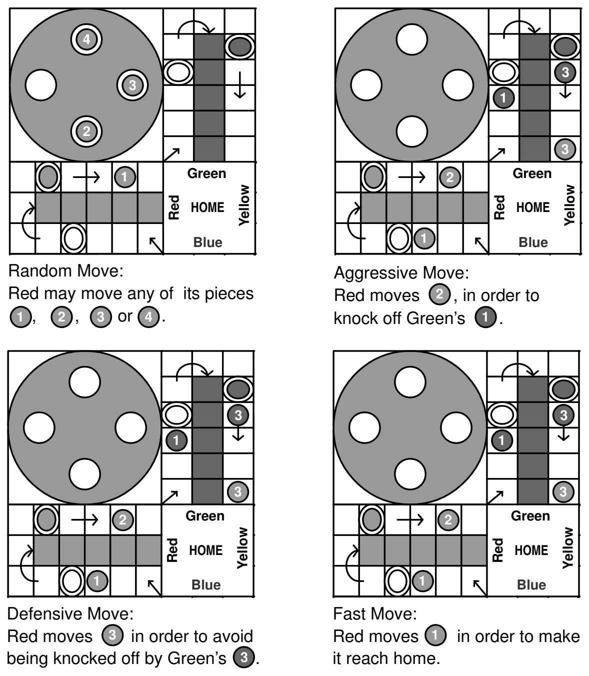

Green
Red
HOME
Yellow
Blue
Random Move:
Red may move any of its pieces 1, 2, 3 or 4.
Aggressive Move:
Red moves 2, in order to knock off Green's 1.
Defensive Move:
Red moves 3 in order to avoid being knocked off by Green's 3.
Fast Move:
Red moves 1 in order to make it reach home.

Fig. 3. Basic Move Types

C. Defensive

In a *defensive move*, a player always defends his pieces against the danger of an impending attack. However, the notion of being threatened (or being in danger of elimination) needs elaboration. We propose that a piece is in danger of being eliminated when it is less than a distance of a single die roll of another piece, i.e., it is 1-6 squares away from another piece. This distance can be defined as a *knocking range*. Therefore, in a defensive move, a player moves his piece if it is within the knocking range of another player's pieces. A defensive move if successful, gives an advantage to the defender against all players, in the sense that the defending player has avoided a loss against all other players by saving his piece.

A *defensive strategy* is based on a preference for defensive moves whenever possible.

D. Fast

In a *fast* move, a player chooses to play the piece which has moved the maximum distance in its circuit around the board. A fast move is based on the idea that the loss of a piece that has advanced the most in the game (i.e. the *fastest* piece) would be the most expensive for a player (in terms of the additional number of moves required), hence it must be preferably moved first and sent to the final home location.

A fast move reorders the movement of a player's pieces within the game and therefore does not offer any comparative advantage over other players. However the risk of piece elimination is somewhat reduced since a player's only one piece is active (moving) at any instant. In this sense it can be termed as a *depth-first game-play*, since a player always moves the most advanced piece, which in turns makes that piece more likely to be played in the next move.

A *fast strategy* is based upon a preference for fast moves.

E. Mixed

It is possible, and in fact advantageous that a player may choose to play different types of moves at different stages in the game. For example, a player may play any combination of defensive, aggressive, fast and random moves, which may give rise to a mixed or hybrid strategy.

We emphasize that the types of moves and strategies presented here are by no means exhaustive and several strategies or types of moves may be discovered at various stages in the game.

IV. Analysis

In this section we present a theoretical analysis of the strategies presented. Using expected values of die rolls and average piece movement over a large number of games, we analyze the comparative advantage of some of the proposed strategies over others.

A. Definitions

Here we formally define a few terms required for the analysis, which have also been earlier used in the paper.

- A *move* is defined as the movement of a player's piece resulting in from a single die roll. We assume that each player plays a single die roll during his *turn* in the game.
- The *distance* moved by a piece is measured in terms of the number of squares. For example, on a single move, a piece moves a distance of 1-6 squares. The length of the track over the entire board for each piece is represented by d_{board}.
- The *winner* is defined as the player who sends his pieces to the final home position in the least number of moves.

B. Expected number of moves

Although Ludo is a non-deterministic game unlike chess and checkers, typical game progress can be modeled based on the expected number of moves over a large number of games. Since the expected value of a single die roll is $E[die] = 3.5$ [19], the expected minimum number of moves taken by a piece to move a distance of d squares is given by

$$E_{min}[moves] = \frac{d}{E[die]} = \frac{d}{3.5}$$

This expected minimum value is based on the assumption that the piece does not get knocked off by another piece. To complete an entire circuit around the board (fig. 1) a piece will require an expected minimum number of $d_{board}/3.5 = 57/3.5 \approx 16.3$ moves. Similarly, a player will take an expected minimum number of $4 \times 16.3 \approx 65$ moves to win. Therefore, the expected-minimum ply-length of a game is $4 \times 65 = 260$.

However in an actual Ludo game, a piece may get knocked off several times. Let us assume that piece i of a player p gets knocked off n times and let d_{ik} represent the distance covered by piece i from its initial position when knocked off for the k^{th} time. Then, the expected number of moves piece i needs to reach home is given by:

$$E_i[m] = [\frac{d_{board} + \sum_{k=1}^{n}(d_{ik})}{3.5}] \quad (2)$$

Therefore, a player p will require

$$E_p[m] = \sum_{i=1}^{4} E_i[m]$$

expected moves to complete his pieces' circuit around the board. The player p with the minimum value of $E_p[m]$ would be the expected winner.

C. Comparison of Strategies

It can be observed from equation (2) that the quantity $[\frac{d_{board}}{3.5}]$ is a constant. Hence to minimize $E_p[m]$ a player needs to do one of the following: (a) minimize the quantity $\sum_{k=1}^{n}(d_{ik})$ for his own pieces by following a defensive strategy, or (b) increase other players $\sum_{k=1}^{n}(d_{ik})$ by following an aggressive strategy.

In this context, a question arises that which one of the two options is more advantageous? More specifically, would a player be better off by defending his own pieces or by attacking other players' pieces? Intuitively, it can be argued that a defensive move is more advantageous than an aggressive move for a given player, since a single save would give a player an advantage (or not cause a deficit) against other three players by not increasing n, which in turn minimizes $\sum_{k=1}^{n}(d_{ik})$ for his own pieces. On the other hand, an attack would increase $\sum_{k=1}^{n}(d_{ik})$ for the attacked player only, thereby still leaving two other players to compete against.

It can be inferred that in a typical ludo game, the number of successful defenses (where a successful defense is defined as moving a piece out of the knocking range of one or more attacking pieces) is probably greater than the number of successful attacks. A detailed proof-outline appears in the Appendix. Using this argument, we may infer that a defensive strategy may be more advantageous than an aggressive strategy, as not only a defensive move is likely advantageous over an aggressive move as outlined previously, but the number of successful defenses is probably greater than the number of successful attacks in a typical game.

It may be relevant to mention here that the position at which a piece gets knocked off is important. The loss of an *advanced* piece (i.e., a piece which has covered a greater distance and is at the final stages of its circuit) is more expensive in terms of the additional required number of moves as compared to the loss of a *new* piece (i.e., a piece in the beginning stages of its circuit). This explains the utility of a fast strategy since a fast strategy aims to minimize exposure to ripe pieces, by prioritizing their movement.

A random strategy is no strategy at all, since players completely move their pieces at random and hence is the most disadvantageous. However it can be used as a benchmark against which to compare other strategies.

In summary it can be inferred that,

$$\text{DEFENSIVE} \succeq \text{AGGRESSIVE} \succeq \text{RANDOM},$$
$$\text{FAST} \succeq \text{RANDOM},$$

where the operator $\succeq$ = 'is at least as advantageous as'.

V. Experimentation and Results

In this section we describe the experiments conducted on the proposed strategies and the results obtained. These tests were conducted on (a) each basic-strategy player playing against all random players and (b) each basic-strategy player playing against different basic-strategy players. A mixed strategy was also formulated based on the obtained results and was tested against these basic strategies.

A. Setup

The game was set up by designing classes representing entities involved in a typical Ludo game. Initially we tested our game setup by running numerous games with all four players selecting the random strategy. This was done to ensure that our game setup is correct and that each random player wins approximately equal number of times. We found that each random player wins 25.0 $\pm$ 1.0% of the games and that this performance stabilized at nearly 5000 games. Running a higher number of games did not reduce the variation in results significantly, suggesting that 5000 games was an adequate number to observe reliable trends in strategy evaluation. These results were also unchanged when the player order was changed suggesting that there was no significant bias towards the player taking the first turn. The tests were also repeated for each strategy separately and identical results were obtained.

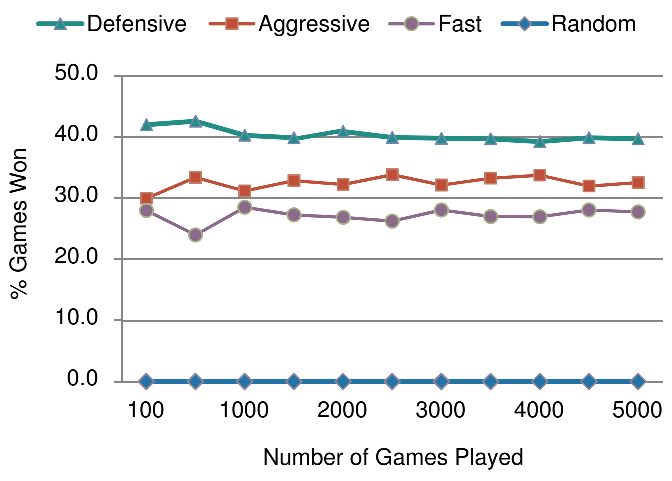
B. Testing Performance of Strategies

In the next phase we performed tests on each one of the proposed strategies. This phase consisted of two types of tests:

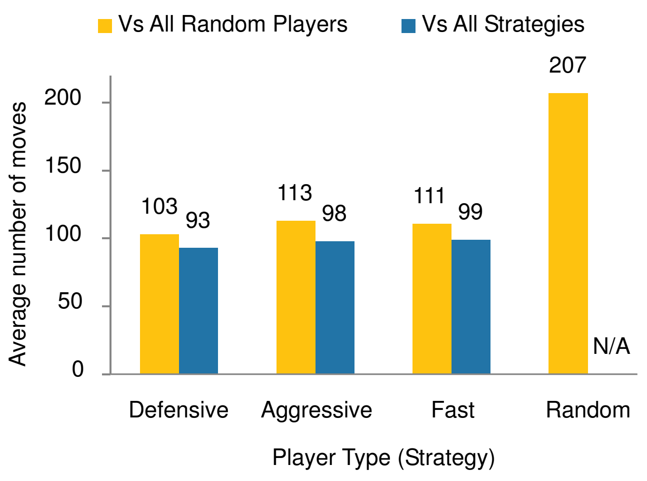
- Testing each basic-strategy player individually against three random players to observe the individual performance data of that strategy.
- Testing all four basic-strategy players against each other in a game to observe the relative performance data of each strategy.

For the first type, we tested each basic-strategy player individually against all random players. We found that each of the basic-strategy players (*defensive*, *aggressive* or *fast*) wins at least 98% of the games against all random players. We state the results of these tests in Table I. This test clearly demonstrates that having any basic strategy is better than playing randomly.

For the second type of test, we tested all basic-strategy players against each other in several game runs. Figure 4(a) gives a graphical view of test results. It can be seen from Figure 4(a) that the defensive strategy always outperformed other strategies on average (40% wins), and the aggressive strategy (32% wins) performed better than the fast strategy



(a) % of Games Won in All-Strategy Games



(b) Average Number of Moves in Wins

Fig. 4. Performance Graphs of Basic Strategies

TABLE I
PERFORMANCE STATISTICS OF BASIC STRATEGIES

Player 1 (% wins)	Player 2 (% wins)	Player 3 (% wins)	Player 4 (% wins)
Random (25.0±1.0%)	All Random (25.0±1.0% for each)		
Defensive (99.3±0.3%)	All Random (< 1% for each)		
Aggressive (99.4±0.2%)	All Random (< 1% for each)		
Fast (98.7±0.4%)	All Random (< 1% for each)		
Defensive (40.3±1.0%)	Aggressive (32.5±1.0%)	Fast (27.2±1.1%)	Random (≈ 0%)

TABLE II
PERFORMANCE STATISTICS OF MIXED STRATEGY

Player 1 (% wins)	Player 2 (% wins)	Player 3 (% wins)	Player 4 (% wins)
Mixed (100.0%)	All Random (≈0% for each)		
Mixed (91.4±1.1%)	Defensive (3.3±0.4%)	Aggressive (2.7±0.5%)	Fast (2.6±0.5%)

(27% wins). However, the random strategy performed poorly (0% wins). These results enhance our theoretical analysis presented in Section 4. These trends were also independent of the number of games played because, although the individual number of wins varied in a particular run of games, the comparative ordering of strategies in terms of the percentage of wins remained the same.

Fig 4(b) illustrates the number of moves taken by each basic-strategy player on average for winning games. It can be observed that the defensive player took comparatively less number of moves to win against other-strategy players, despite it winning a higher number of times. The aggressive and fast strategies took an almost equal number of moves. It can also be observed that each player took more moves against all-random players than against players with strategies, which at first seems counter-intuitive. However this should not be surprising since players won more than 98% of the time against all-random players which may have included longer games as well.

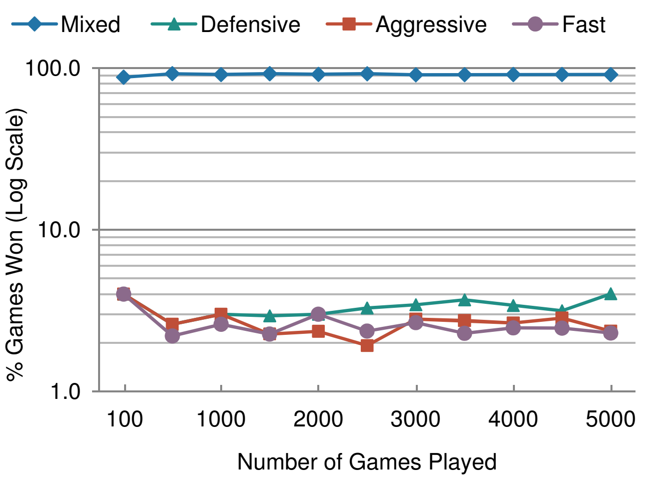
C. Mixed Strategy

Based on the comparative performance of strategies, we formulated a *mixed* or a *hybrid* strategy stated as follows:

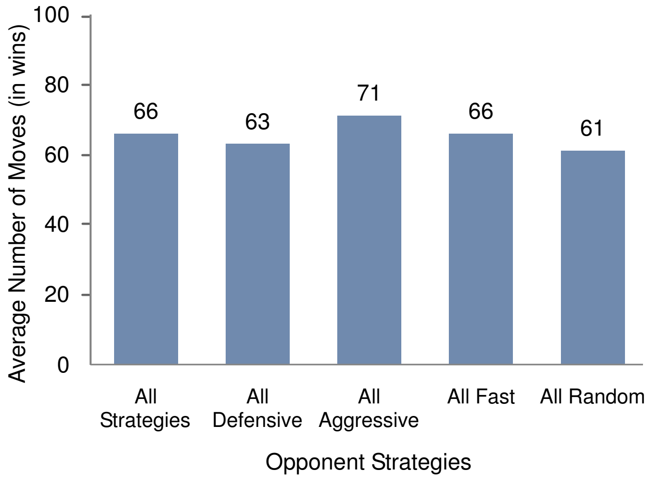
1) *At his turn, a player should play a defensive move;*
2) *if not possible play an aggressive move;*
3) *if not possible play a fast move;*
4) *else play a random move.*

We conducted tests for the mixed strategy similar to the ones done for the basic strategies. Fig 5(a), 5(b) and Table II highlight the performance of the mixed strategy. It can be seen that the mixed-strategy player wins at least 90% games against basic-strategy players suggesting that this strategy is far superior than any basic strategy taken alone. We can also observe from Fig 5(b) that the average number of moves taken by the mixed-strategy player against other basic-strategy players is close to 65, which is the expected minimum number of moves required to win (as outlined in Section 4). This implies that the mixed strategy is close to being an *optimal* strategy against the basic-strategy players, since it takes close to an expected minimum number of moves to win a game on average.

The above experimental analysis is applicable to Parcheesi as well as other Ludo variants, since these variants differ by only the number of squares on the game-board and a few game rules; however the basic game structure is the same.



(a) **Mixed Strategy** - % Games Won in All -Strategy Games



(b) **Mixed Strategy** - Average Number of Moves in Wins

Fig. 5. Performance Graphs of Mixed Strategy

VI. Conclusions and Future Work

In this research we conducted a fundamental study of Ludo (a 2-4 non-deterministic race game based on Pachisi). We found the Ludo state-space complexity to be approximately 10^{22} which is slighly larger than that of Backgammon, suggesting that the game is not solvable using current computational resources. Accordingly, we identified four basic moves and developed a number of playing strategies using these moves: *defensive*, *aggressive* and *fast* playing strategy. Theoretical and experimental results show that the defensive strategy performs better than the other strategies based on the percentage of wins. We also formulated a *mixed* strategy and found it to be far superior to each basic strategy. Furthermore, our analysis both for state-space complexity and playing strategies is applicable to several variants of Ludo.

For future work, the analysis of strategies presented here can be used as input for TD-learning networks for Ludo in order to obtain better players in lesser number of trials. It can also be incorporated in an evolutionary game analysis for Ludo, resulting in the discovery of better strategies and improved game-play. One way of achieving this goal is by formulating one or more evaluation functions, which assign weights to each type of move based on the type of game-play required and the comparison of strategies presented in this work. For example,

- a defensive move may be assigned a higher weight as compared to an aggressive move in an evaluation function.
- an aggressive move on an advanced piece may be given a higher value over an attack a new piece.

Using reinforcement learning, the evaluation function(s) may be adjusted after several game runs. Evolutionary algorithms may also be used to improve the evaluation function(s) after successive generations, possibly leading to the discovery of better strategies for improved game-play.

Appendix

Here we give a proof-outline of the statement in Section 4: *In a typical ludo game, the number of successful defenses is probably greater than the number of successful attacks*. The proof-outline that follows involves a case-by-case evaluation of the probabilities of a successful attack and defense for different types of piece configurations. We then complete our argument by suggesting that the piece configurations for which $p(Defense)$ is higher may have a greater chance of occurrence in a game than piece configurations for which $p(Attack)$ is higher.

The statement as well as the proof-outline presented here are informal in nature, since a formal theorem requires a more precise definition of successful defense and attack; likewise a rigorous proof requires an exact count of the number of different piece configurations involved. We avoid both of these requirements in order to keep our argument simple, yet plausible. We now proceed to a case-by-case enumeration and analysis of each possibility:

Case 1: 2-Piece Configuration

A defending piece lies within the knocking range of exactly one attacking piece: For this piece layout (Fig. 6), the probability that the attacking piece successfully knocks off the defending piece is $p(Attack) = 1/6$ since an exact die throw is required for a knock to be successful. To find $p(Defense)$ we consider the distance d between the two pieces. If the distance between the two pieces is d squares, then the defending piece requires a die roll of $7-d$ or higher to move away from the knocking range of the attacking piece. Assuming all 6 possibilities of distance separation to be equally likely, we find that: $p(Defense) = 1/6 \cdot \sum_{i=1}^{6} i/6 = 21/36 = 7/12$

We emphasize here that these values for $p(Attack)$ and $p(Defense)$ depend upon player order i.e. which player makes the first move. Also, these values may not represent exact probabilities in all situations; for example in case when the defending piece is already placed on a safe square, the defense probability is 1 (i.e., the piece is not under threat) which is greater than $7/12$. What is important however, is the relative ordering of these two values, i.e. $p(Defense) > p(Attack)$ for a 2-piece configuration.

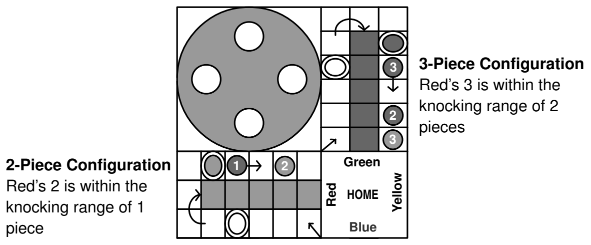

3-Piece Configuration
Red's 3 is within the knocking range of 2 pieces
2-Piece Configuration
Red's 2 is within the knocking range of 1 piece
Green
Red
HOME
Yellow
Blue
1
2
3
2
3

Fig. 6. Various Piece Configurations

Case 2: 3-Piece Configuration

A defending piece lies within the knocking range of exactly two attacking pieces: Similar to the previous case, we find that $p(Defense) = 1/5 \cdot \sum_{d=1}^{5} i/6 = 15/30 = 1/2$. Similarly, it can be shown that $p(Attack) = 1/6 + 1/6 - 1/36 = 11/36$, using the Inclusion-Exclusion Principle. Again, similar to case 2, we see that $p(Defense) > p(Attack)$. Here we assume that the two attacking pieces are of different colors. If the attacking pieces belong to the same player, then $p(Attack) = 1/3$. However, this does not affect the overall inequality i.e., $p(Defense) > p(Attack)$.

Case 3: 4-Piece Configuration

A defending piece lies within the knocking range of exactly three attacking pieces: Here $p(Attack) = \sum_{r=1}^{3}(-1)^{r+1}C(3,r)/6^r = 91/216$, $p(Defense) = 1/4 \cdot \sum_{d=1}^{4} i/6 = 10/24 = 5/12$. Contrary to the previous two results, here $p(Defense) < p(Attack)$.

Case 4: 5-Piece Configuration

A defending piece lies within the knocking range of exactly four attacking pieces: Here $p(Attack) = 671/1296$, $p(Defense) = 1/3$, hence $p(Defense) < p(Attack)$.

Case 5: 6-Piece Configuration

A defending piece lies within the knocking range of exactly five attacking pieces: Here $p(Attack) = 4651/7776$, $p(Defense) = 1/4$, hence $p(Defense) < p(Attack)$.

Given the number of pieces and squares in ludo and the fact that game states may be repeated during game progress, it is almost infeasible to estimate the actual number of n-piece configurations in a game. Instead we give an argument based on the maximum possible number of (non-overlapping) n-piece configurations on a ludo-board at any given instant.

If x_n represents the number (non-overlapping) n-piece configurations on ludo-board at any instant, where x_1 specifically represents a single piece not defending itself or attacking any piece, then

$$x_1 + 2x_2 + 3x_3 + 4x_4 + 5x_5 + 6x_6 = 16 \qquad (3)$$

Using equation (3), we see that x_2 (which represents the number of 2-piece configurations at any given instant) can have a maximum value of $16/2$ = 8. Likewise, the maximum value of x_3 = 5, x_4 = 4, x_5 = 3 and x_6 = 2.

We state these results in table 3. From this table we observe that $p(Defense) > p(Attack)$ for the first two cases which may have a higher chance of occurrence based on the maximum number of states for that piece-configuration. However, the cases for which $p(Defense) < p(Attack)$ may occur less frequently. Based on this table it may be inferred that the expected value of $p(Defense)$ in a typical ludo game will probably be higher than the expected value of $p(Attack)$, thus implying that the number of successful defenses will probably be higher on average.

TABLE III
$p(Defense)$ AND $p(Attack)$ FOR VARIOUS CONFIGURATIONS

Piece Config	$p(Defence)$	$p(Attack)$	$p(Defence) > p(Attack)$	Max No of States
2	7/12	1/6	Yes	8
3	1/2	11/36	Yes	5
4	5/12	91/216	Nearly Equal	4
5	1/3	671/1296	No	3
6	1/4	4651/7776	No	2

REFERENCES

[1] D. Parlett, *The Oxford History of Board Games*. Oxford Univ. Press, 1999.

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

[3] V. K. Petersen, "Pachisi & Ludo ," (Accessed: 12-Mar-2011). [Online]. Available: http://pachisi.vegard2.no/index.html

[4] R. S. Sutton, "Learning to Predict by the Method of Temporal Differences," *Machine Learning*, vol. 3, pp. 9–44, 1988.

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

[6] S. Thrun, "Learning to Play the Game of Chess," in *Advances in Neural Information Processing Systems (NIPS) 7*, 1995, pp. 1069–1076.

[7] A. Leouski, "Learning of Position Evaluation in the Game of Othello," Master's thesis, Univ. of Massachusetts, 1995.

[8] M. Wiering, J. P. Patist, and H. Mannen, "Learning to Play Board Games using Temporal Difference Methods," Utrecht Univ., Tech. Rep. UU-CS-2005-048, 2005.

[9] Y. Jin, *Knowledge Incorporation in Evolutionary Computation*. New York: Springer-Verlag, 2004.

[10] K.-J. Kim and S.-B. Cho, "Systematically Incorporating Domain-Specific Knowledge Into Evolutionary Speciated Checkers Players," *Machine Learning*, vol. 9, no. 6, pp. 615–627, 2005.

[11] 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.

[12] "Parcheesi," (Accessed: 21-Feb-2011). [Online]. Available: http://en.wikipedia.org/wiki/Parcheesi

[13] V. Allis, "Searching for Solutions in Games and Artificial Intelligence," Ph.D. dissertation, Univ. of Limburg, The Netherlands, 1994.

[14] G. Tesauro, "Temporal Difference Learning and TD-Gammon," *Communications of the ACM*, vol. 38, no. 3, 1995.

[15] J. Schaeffer, J. Culberson, N. Treloar, B. Knight, P. Lu, and D. Szafron, "Reviving the Game of Checkers," in *Heuristic Prog. in Artificial Intelligence 2: The 2nd Comp. Olympiad*, 1991, pp. 119–136.

[16] A. S. Fraenkel and D. Lichtenstein, "Computing a Perfect Strategy for n x n Chess Requires Time Exponential in n," *Journal of Combinatorial Theory, Series A*, vol. 31, no. 2, pp. 199 – 214, 1981.

[17] A. S. Fraenkel, "Nim is Easy, Chess is Hard - But Why??" *Intl. Comp. Games Association Journal*, vol. 29, no. 4, pp. 203–206, 2006.

[18] T. J. Schaefer, "On the Complexity of Some Two-Person Perfect-Information Games," *Journal of Comp. and System Science*, vol. 16, no. 2, pp. 185–225, 1978.

[19] K. Rosen, *Discrete Mathematics and its Applications*, 6th ed. Boston: McGraw-Hill, 2007.