Aritificial Intelligence Q-Learning based Ludo Agent

Muhammad, Faraz Ali, and Ronit Kumar Kataria
Dhanani School of Science and Engineering
Habib University
Gulistan-e-Jauhar, University Avenue 75290, Karachi, Pakistan

Dr. Syeda Saleha Raza
Dhanani School of Science and Engineering
Habib University
Gulistan-e-Jauhar, University Avenue 75290, Karachi, Pakistan

***Abstract*—Q-Learning is a model free, value based reinforcement learning algorithm that is used to learn the optimal action-selection policy for an agent in a Markov decision process (MDP). In this work we propose an Ludo Agent that will learn and formulate the ludo strategies making use of Reinforcement Learning and Q-Learning to make an action selection policy. This paper will describe the implementation process of the algorithm, the efficacy of the algorithm, and the winning ratio of the RL Agent.**

***Index Terms*—Ludo board, Q-Learning, Q-Table, State Space, Action Table, Q-Values, RL, Agent.**

I. INTRODUCTION

Ludo is a popular board game played by people all over the world. The exact origin of this game comes from historic Ellora Caves present in Maharashtra. It is a two-to-four player game where the objective is to move all the player's pieces to the home quadrant before the other players do the same.

It is a classic board game that has been enjoyed by people of all ages for generations. With the rise of artificial intelligence and machine learning, there has been a growing interest in developing AI agents that can play board games as well as, or even better than, human players. In this project, we aim to create a ludo game that puts human players against an AI agent based on the Q-learning approach. We present a human vs AI ludo game, where the AI is based on the Q-learning approach. Ludo is a popular board game that requires players to move their pieces around the board, with the aim of getting all their pieces to the home area before their opponents. Our AI agent uses Q-learning to learn the optimal strategy for playing ludo. Q-learning is a reinforcement learning algorithm that allows the agent to learn from experience by estimating the value of different actions in different states. In our implementation, we use 57 states to represent the board state and the position of the player's pieces. We evaluate the performance of our RL agent against human players by measuring the win rate and the average number of moves taken to win. Our results show that the Q-learning based RL agent is competitive with human players and can provide an engaging and challenging game experience.

The Ludo board is a square shaped board divided into four coloured quadrants with colors: blue, red, green and yellow. As seen in Figure 1 the board has a cross-shaped pattern in the center, which forms the home area for each player. Each quadrant consists of 13 spaces or squares, with a start zone and a home zone.

Ludo game presents an environment to the agent which is unknown, and the agent has to learn the task through trial and error. An ideal policy is needed for the agent for which it can adhere to while making decisions when playing in a real life environment. In this paper we will introduce an AI Ludo Agent, that will make use of Q-Learning, which is a Reinforcement Learning method.

One of the popular techniques used for creating game-playing agents is Q-Learning. Q-Learning is a Reinforcement Learning algorithm that is used to learn a policy that maximizes the expected cumulative reward. In this paper, we propose a Ludo game-playing agent that learns to play the game using Q-Learning and Bellman equation. The proposed agent will learn from the game's state-action-reward sequences and adjust its policy to maximize its expected reward.

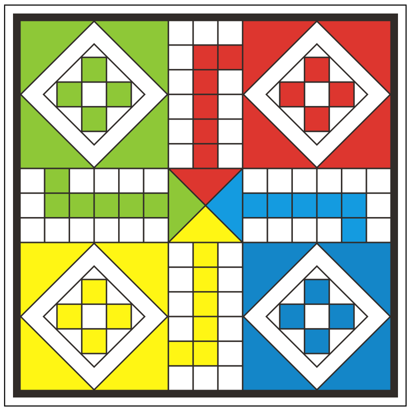

Fig. 1. Ludo Board

Reinforcement Learning (RL) has gained immense popularity in recent years for designing agents to play games, including Ludo. RL algorithms, such as Q-Learning and

TD, provide a framework for the agent to learn by interacting with the environment and receiving rewards based on its actions. RL agents can learn to play games at a superhuman level and have been successful in various domains, including board games, video games, and robotics. In the context of Ludo, RL agents have been developed that use Q-Learning to learn to play the game and outperform human players. RL-based Ludo agents have been shown to be effective in learning the optimal policy for the game and can adapt to changes in the game's rules or environment. Therefore, RL is a popular choice for designing Ludo agents that can learn to play the game and improve their performance through experience.

In this work we propose an expert ludo agent that will train itself against a random player. A policy will be stored for that agent and then that agent will compete against a human player. We will also include a performance comparison of our expert ludo agent against random and human player.

The entire simulation of our game, the state space and the ludo board comes from an open-source python library developed by Simon Soerensen, named LudoPy

The rest of this paper is organized as follows: Section 2 describes our proposed Ludo game-playing agent. Section 3 presents the Reinforcement Learning algorithms used to train the player, and Section 4 presents our experimental setup and results. Section 5 discusses the results and concludes the paper.

II. Proposed Expert Ludo Agent

A. Rules

The Rules that are defined for our Ludo game comes from the same *ludopy* library that was discussed earlier. The rules are as follows:

1) There are 4 pieces per player.
2) A player needs to roll 6 to release a piece.
3) If a piece stops at a star on the board, it is leaped to the next star position.
4) Releasing a piece is optional when a player rolls a 6.
5) Globe states are Safe states where the pieces can't be killed

The main strategy of the agent is to quickly move all its pieces around the board, and if there is any opponent's piece in front of it, always stay back until it can attack it. In case of no other option, the agent may make a random move.

B. Expert Agent

Our expert ludo agent developed in this paper was made through the Reinforcement Learning method of Q-Learning. Here the agent takes the action by estimating the value of each possible action in the given state. These values are stored in the q-table that is an attribute of the Agent's class.

An action table, and a Q-Table is associated with each agent as its attribute in the agent's class. The agent choose its next action making use of those tables by multiplying them. ϵ is used to control the exploitation and exploration of the agent. As the agent learns and the Q-Values are updated, the agent starts to take more greedy action.

The agent is trained by competing against a random player and then tested against human.

III. Reinforcement Learning Method

A. Mathematical Model

A RL problem is a Markov decision process which is described as a 4-tuple (S, A, P(·,·),R(·,·)) 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 s' given the 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 plan or a policy π which is a function that maps states to actions. The optimal policy π^* is the policy that produces cumulative rewards for all states.

B. Learning Methods

The main Reinforcement Learning algorithm used to train the Ludo agent is the QLearning Algorithm. It is a model-free algorithm in which the agent tries to perform adaptively without understanding it. In Q-learning, the agent learns to estimate the Q-value of a state-action pair, which represents the expected total reward that the agent can obtain by taking a particular action from a particular state and following a particular policy thereafter. The Q-function is updated iteratively using the Bellman equation, which is a recursive formula that expresses the value of a state in terms of the values of its successor states. The Q-learning algorithm works by starting with an initial estimate of the Q-function, and then iteratively improving this estimate by exploring the environment and updating the Q-values based on the rewards obtained. At each step, the agent selects an action based on the current estimate of the Q-function, and then observes the resulting state and reward. It then updates the Q-value for the current state-action pair based on the observed reward and the expected future rewards.

The equation used to update the Q-values is given as:

$$Q(s,a) = Q(s,a) + \alpha[R + \gamma(max_{a'}Q(s',a')) - Q(s,a)]$$

Here, $Q(s,a)$ is the Q-value for the current state and action pair. s' and a' are the new state and action the agent will lend upon after taking the action a.

C. Action Selection

The action selection criteria used by our agent during the QLearning process is *epsilon-greedy*. It is a simple but effective way of encouraging the agent to explore different actions while still taking advantage of its current knowledge. In epsilon-greedy action selection, the agent selects the best action based on its current estimate of the Q-values with probability (1 - epsilon), and selects a random action with probability epsilon.

In order to create a good balance between exploration and exploitation and ensure effective convergence, we used the policy with a decaying epsilon value. What this means is that we set a hyper-parameter called *decayRate* which was the rate at which the epsilon value was decayed at each episode. The equation used was as follows:

$$\epsilon \times e^{d \times ep}$$

Here d refers to the *decayRate* and ep refers to the episode number.

D. State Space and Action Table

The state space was divided into two parts. A Local State Space and a Global State Space. Altogether, there are 57 possible states a single piece could move to. These 57 states are present in local space as well as global space. A simple modulus equation was used to map all of these 57 states from local to the global state space.

The equation used was as follows:

$$\text{localposition} + (13 * \text{playeridx}))\ \%\ 52$$

The reason behind using the constant 13 was that for each quarter of the board, there were 13 states outside of the home states.

1) Local State Space:: Local, as the name itself suggests, is the relative state space for each player. The first star state that is nearest to the home for each player constituted as being the 1st state in the local space for each player.

2) Global State Space:: Global Space is where all the calculation and learning takes place. States within the global space is divided into three major categories:

1) SAFE
2) UNSAFE
3) HOME

The HOME states are when the pieces are inside the home or at the very first star state near the home. SAFE states are states in which the token has reached the goal. UNSAFE states are when the token is out but hasn't reached the goal yet.

The diagram *fig 2* at the end of the section depicts how the states are numbered. It shows the Global space and the local space for index = 0.

At each state, the actions that a token can take are

1) MoveOut: Moving token out of start
2) MoveDice: Moving eyes of dice
3) Goal: Move into goal position
4) Star: Move into star position
5) Globe: Move into globe position
6) Protect: Move to same token as yourself
7) Kill: Kill opponent

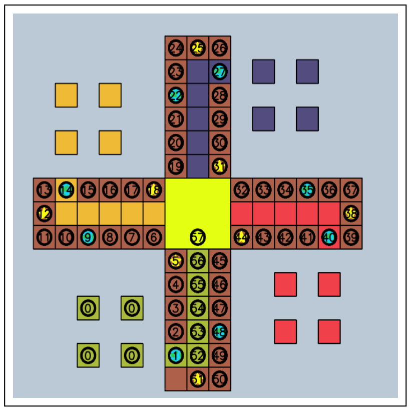
Fig. 2. Global States

8) Die: Move to a field where opponent has 2 or more pieces
9) GoalZone: Move into goal zone

IV. WORK DONE AND RESULTS ANALYSIS

A. Hyper parameters α and γ

In this section we discuss the effectiveness of our expert ludo agent by testing it with different parameters. In Q-Learning, we deal with two hyper parameters: learning rate (α) and discount factor(γ). Those two parameters have an effect on the agent's learning and for different values of α and γ, we will test the agent's win ratio.

- α: It is learning rate for the agent. It determines the extent to which agent updates its Q-Values.
- γ: It is the discount factor which determines the importance of future rewards compared to immediate rewards.

The amount by which the agent adjusts its Q-values depending on the discrepancy between the predicted reward and the present estimate depends on its learning rate. The agent is more receptive to previous experiences and may pick up new information faster with a greater learning rate, but it may also be more prone to overfitting. On the other side, the gamma parameter establishes the discount rate for potential benefits. A greater gamma indicates that the agent emphasises long-term incentives more, whereas a lower gamma indicates that the agent emphasises short-term rewards more.

We conducted a series of experiments by systematically varying the learning rate (α) and discount factor (γ) within a reasonable range. The results showed that setting the learning rate too high led to sub optimal performance of the agent. It exhibited volatile behavior, quickly adapting to new experiences but also forgetting previous knowledge rapidly. We observed that setting gamma too high caused the agent to exhibit short-sighted behavior. With a high discount factor, the agent prioritized immediate rewards less and assigned significant importance to future rewards.

To visualize the effect of different combinations of learning rate and gamma on the agent's performance, we have created a heatmap Figure 3 with the winrate values

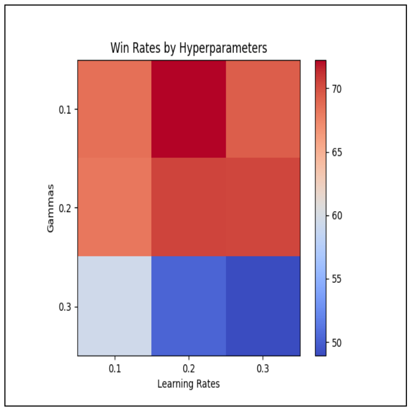

Fig. 3. Win Rate Heatmap for Different Values of γ and α

for each combination. Based on the heatmap, it seems like there is no clear pattern in terms of the optimal values for learning rate and gamma. However, some combinations appear to perform better than others.

For example, the highest winrate value of 72.257 is achieved for a learning rate of 0.2 and a gamma of 0.2, while the lowest winrate value of 48.916 is achieved for a learning rate of 0.9 and a gamma of 0.9. The other combinations of learning rate and gamma fall somewhere in between these extremes.

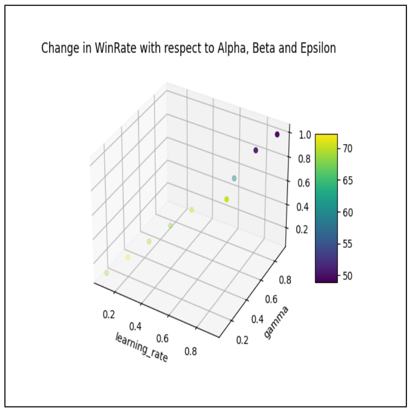

Fig. 4. Surface Plot of Win Rate with respect to γ and $\alpha and \epsilon$

Looking at the surface plot, we can see that the optimal values for the three hyperparameters are not concentrated in a specific region of the plot, but are scattered throughout. However, we can still observe some general trends.

First, we can see that the winrate tends to increase with increasing learning rate and gamma, up to a certain point. Beyond that point, increasing the learning rate and gamma may cause the performance to degrade due to overfitting.

Second, we can see that the optimal decay rate for epsilon depends on the specific combination of learning rate and gamma. For some combinations, a high decay rate may be optimal, while for others, a low decay rate may be optimal. This suggests that the agent's exploration strategy should be tuned based on the specific task and the values of the other hyperparameters.

Overall, the 4D surface plot provides a more detailed picture of how the Q-learning agent's performance is affected by different combinations of hyperparameters. By analyzing the plot, we can gain insights into the complex relationships between the hyperparameters and the agent's performance, which can inform future experiments and research in this field.

B. Epsilon Decay

One important aspect of training an expert Ludo agent using reinforcement learning algorithms such as Q-learning is the exploration-exploitation trade-off. The exploration phase allows the agent to discover new strategies and actions, while the exploitation phase focuses on exploiting the learned knowledge to make optimal decisions. In addition to learning rate and gamma, we have also experimented with different values of epsilon, which determines the trade-off between exploration and exploitation in the agent's action selection strategy. In this section, we investigate the effect of ϵ decay on the learning process of our expert Ludo agent, aiming to strike a balance between exploration and exploitation.

In our agent's learning process, we utilize an ϵ-greedy policy, where ϵ represents the probability of exploration. Initially, a higher ϵ value encourages the agent to explore more, while a lower value emphasizes exploitation. We apply an epsilon decay mechanism to gradually reduce the exploration probability over time, allowing the agent to shift its focus towards exploitation as it acquires more knowledge about the Ludo game.

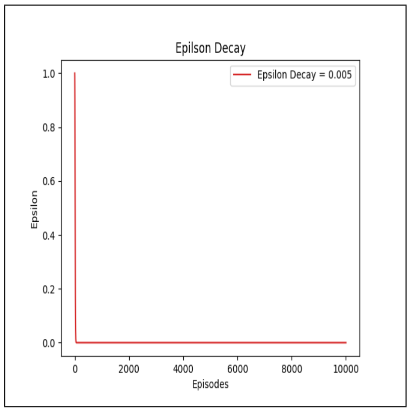

Fig. 5. Epsilon variation over episodes

Following the practice, we can see in Figure 4 that ϵ is given a value of 1.0 in start where it completely explores the environment, and randomly select the action. As the agent learns over increase episodes, the ϵ is decayed, and becomes asymptotic to the x-axis where not the agent has sufficient learning to make greedy actions based on the maximum Q-Value from the Q-table.

C. Episodic Learning

Episodic learning plays a crucial role in reinforcement learning algorithms by capturing and leveraging the temporal structure of experiences. It enables RL agents to learn from sequences of states, actions, and rewards, which are collectively referred to as episodes. In this section, we discuss the significance of episodic learning and its impact on the acquisition of optimal policies.

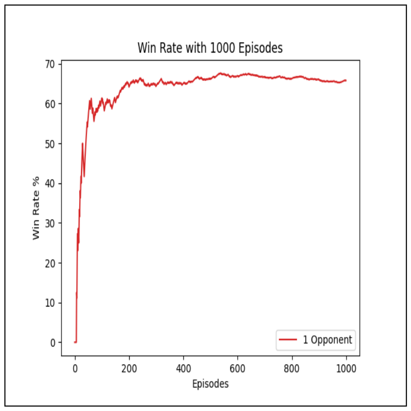

Fig. 6. Winning rate against 1000 episodes

Q-learning agent was trained for 1000 episodes, the average winrate achieved was 63.72%. This suggests that the agent was able to learn some effective strategies for the game of Ludo, but was not able to fully converge to an optimal solution in this limited training time. It's possible that with more training episodes, the agent would be able to refine its strategies and achieve a higher winrate.

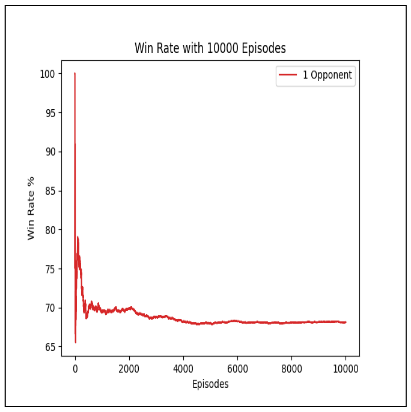

Fig. 7. Winning rate against 10000 episodes

Q-learning agent was trained for 10000 episodes, the average winrate achieved was 68.72%. This represents a significant improvement over the performance achieved with only 1000 training episodes. With more training time, the agent was able to explore more of the game space and refine its strategies, leading to more effective decision-making and ultimately a higher winrate. This result suggests that training for more episodes can be a worthwhile investment in order to achieve better performance in reinforcement learning tasks.

Overall, both scenarios provide valuable insights into the performance of the Q-learning agent and the effect of training time on its performance. However, the results from the scenario with 10000 episodes suggest that longer training times can lead to significant improvements in performance, indicating that further training could potentially yield even better results.

D. Q-Values for State-Action Pairs in a Reinforcement Learning Environment

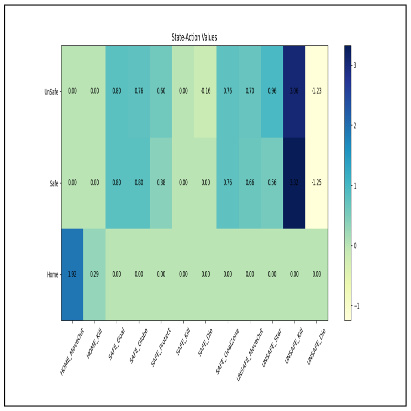

Fig. 8. Q values of State Action pair

n this study, we used a reinforcement learning approach to train an agent in a simulated environment. The environment consisted of three possible states: "Home", "Safe", and "UnSafe". The agent could take one of eleven possible actions in each state, resulting in a total of 33 possible state-action pairs.

After training the agent, we computed the Q-values for each state-action pair. The Q-value represents the expected future reward that the agent will receive by taking a particular action in a particular state. The Q-values were computed using the Q-learning algorithm with a discount factor of 0.9.

We found that some state-action pairs had Q-values of 0, indicating that the agent had not learned any useful information for those pairs. As a result, we removed those state-action pairs from the analysis. This left us with a total of 20 state-action pairs for analysis.

The Q-values for the remaining state-action pairs ranged from -1.247 to 3.325. The highest Q-value was for the action "$UNSAFE_MoveOut$" in the "$UnSafe$" state, indicating that taking this action in this state is expected to result in the highest future reward. On the other hand, the lowest Q-value was for the action "$SAFE_Die$" in the "$Safe$" state, indicating that taking this action in this state is expected to result in the lowest future reward.

Overall, our results demonstrate that the Q-learning algorithm was able to successfully train the agent to make informed decisions based on the current state of the environment. The Q-values provide a measure of the expected future reward for each possible action in each state, and can be used to guide the agent's decision-making process in real-world applications.

In future work, we plan to further refine the training process to improve the agent's performance, as well as explore the use of other reinforcement learning algorithms to compare their effectiveness in this task.

V. CONCLUSIONS

In this paper, we developed a Reinforcement Learning model to train an agent to play the Ludo board game. We investigated the effectiveness of our model by testing it against random agents, and we explored the impact of different hyperparameters on its performance. Our results show that our Reinforcement Learning model can effectively learn to play Ludo and outperforms the random agents.

We have identified several important contributions in this work. First, we have demonstrated that Reinforcement Learning is a promising approach for training agents to play complex board games like Ludo. Second, we have shown that appropriate hyperparameters are crucial to achieving optimal performance, and we have identified the optimal settings for our model. Third, our work provides a foundation for further exploration of Reinforcement Learning algorithms for playing Ludo.

Our agent, once trained for 10000 episodes, showed convergence at a steady win rate of around 70% against one Random Agent. It can be concluded from this that the agent was able to learn a good policy overtime and make good decisions by adapting to its environment without actually having a true sense of what its environment was.

Our findings suggest several potential future directions for research. First, it would be interesting to investigate the effectiveness of other Reinforcement Learning algorithms, such as Deep Reinforcement Learning, for playing Ludo. Second, we could explore different reward functions to encourage the agent to learn different strategies. The training model currently used is to train the RL agent against the Random-based agent. We can even improve the policy of the agent by experimenting with newer ways to train the RL agent. Such as

1) Training against heuristic-based Agents
2) Training against another RL based agent
3) Making use of all 4 players to Train the agent.

Currently, we are only training it against one random agent. We can try to increase the number of random agents. Once the RL agent learns policy against random agents, make it play against heuristic-based agents using the same policy and allowing it to improve. Once this is done, we can then make two or more RL agents with similar policies to compete against each other and then finally test the learning against human players.

In conclusion, our work contributes to the advancement of Reinforcement Learning techniques for training agents to play board games. By demonstrating the effectiveness of our model, we hope to inspire further research in this field and provide valuable insights into its real-world applicability.

ACKNOWLEDGMENT

We would like to thank Dr. Syeda Saleha Raza, Associate Professor, Computer Science, Habib University, for her valuable comments and suggestions to improve the quality of our work. There is a lot of room for improvement in our research, and we'd like to continue working on this and align it with the feedback provided by Dr. Saleha Raza. Secondly, we are also grateful to Dr. Faisal Alvi, Assistant Professor, Computer Science, Habib University for helping us with some issues and providing feedback during the project.

REFERENCES

[1] Dr. Saleha Raza (2023). Computational Intelligence Notes. Habib University. [online]. Available: `https://hulms.instructure.com/courses/2501`

[2] Majed Alhajry, Faisal Alvi, and Moataz Ahmed. (2012). TD and Q-Learning Based Ludo Players. [online]. Available: `https://www.researchgate.net/publication/261279306_TDl_and_Q-learning_based_Ludo_players`

[3] Nicolai Nielsen. (2022). Ludo RL. [online]. Available: `https://github.com/niconielsen32/LudoRL`.

[4] Gabriel Vieira de Vasconcelos Vilaça Pinto. (2019). REINFORCEMENT LEARNING IN LUDO ARTIFICIAL INTELLIGENCE THROUGH SARSA AND GENETIC ALGORITHMS.

[5] Nathan Durocher. YARAL: Yet Another Reinforcement learning Approach to Ludo.

[6] Simon Soerensen (2023). LudoPy. [online]. Available: `https://github.com/SimonLBSoerensen/LUDOpy`