Tools of Artificial Intelligence Final Project

Adam Wolkowycki

University of Southern Denmark, Campusvej 55, 5230 Odense M, Denmark
adwol21@student.sdu.dk

Abstract. The project aims to explore the application of artificial intelligence (AI), specifically Reinforcement Learning (RL), in the context of learning how to play popular board game Ludo. I will focus on using Temporal Difference (TD) Learning algorithms, including Expected SARSA and Double Q-learning, to develop intelligent agents capable of self-improving their gameplay.

Keywords: Reinforcement Learning, Q-learning, SARSA

1 Introduction

By studying the application of reinforcement learning algorithms to the Ludo game, this project aims to contribute to the understanding of how AI agents can learn and adapt to the simple environments. These techniques have shown great promise in enabling agents to learn optimal strategies through trial and error interactions with the highly stochastic environment, which is a typical feature of the Ludo game. The findings and insights gained from this research can pave the way for further advancements in game-playing AI and provide valuable knowledge for developing intelligent agents capable of tackling real-world challenges.

2 Materials and Methods

The illustrative taxonomy of different AI paradigms has been presented on the Venn diagram (fig. 1), where multiple overlapping circles represent these concepts starting from the broadest category and narrowing it down to specific subcategories. The second outermost circle represents **Reinforcement Learning (RL)**, a general approach to AI that involves an agent interacting with an environment, learning from experiences, and optimizing its decision-making policy to maximize cumulative rewards. Moving inward, the middle circle represents **value-based** methods, which form a significant subset of RL techniques. They aim to estimate the value function associated with state-action pairs, providing guidance for decision-making in RL scenarios. One prominent subcategory of value-based methods is **Temporal Difference (TD) Learning** that consists of two distinct approaches depicted by the innermost circles: **off-policy** and **on-policy** methods. The off-policy approach refers to methods where the agent

learns from data generated by a different behavior policy. Similarly, the on-policy approach represents methods where the agent learns from data generated by the current policy itself. It is noteworthy that discussed picture is focused on the methods applied for this project, which are a small slice of all AI methodologies.

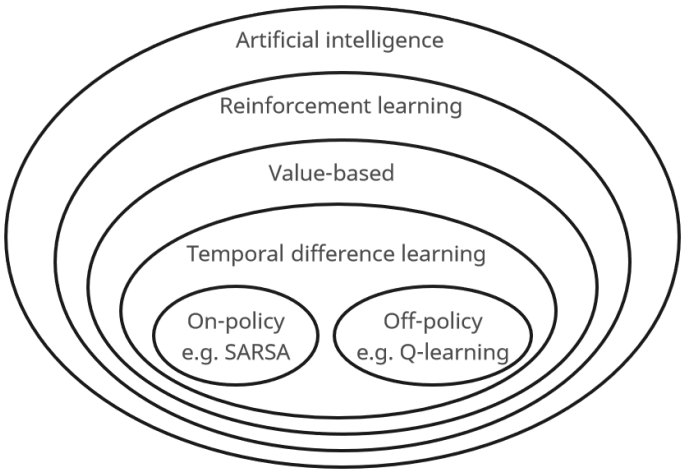


Fig. 1: Venn diagram showing how temporal difference learning is a kind of value-based methods, which is in turn a kind of reinforcement learning, which is used for many but not all approaches of AI.

2.1 Environment

The Ludo game is played on a game board consisting of a track divided into several sections. Each player has four tokens that aim to move from the starting area to the goal zone. The game environment presents quite limited, but various possibilities for the actions. There has been distinguished nine of them for each state and presented on the table (tab. 1) with their associated rewards. The rewards are dependent on both the action taken by a player and the current state of a game, specifically, the position of a token. Positive values indicate favorable outcomes, while negative values represent their opposites. The magnitude of the reward indicates the strength of its positive or negative impact. For instance, killing another player's token is associated with the lowest negative rewards, while successfully moving a token into the goal position from regular board field yields a substantial positive reward, as it indicates progress towards winning the game.

Table 1: Rewards table.

	State (position of a token)		
	Home	Target	Regular field
Moving a token out of home	1.6	0.4	0
Moving by the number of dice eyes	-0.79	0.01	-0.39
Moving into goal position	0	0.8	1.2
Moving on a star	0	0.8	0.4
Moving to a safe point (globe)	-0.4	0.4	0.8
Moving a token to with an intention to protect it from being killed	-0.6	0.2	0.6
Killing another player's token	0.7	1.5	1.9
Moving to a field with two or more opponent's pieces	-1.3	-0.5	-1.3
Moving into goal zone	-0.8	0.2	0.6

2.2 Parameters setup

All the experiments of training the reinforcement learning agents were conducted using specific parameter values. These parameters play a crucial role in shaping the learning process and determining the performance of the algorithms. The learning rate of 0.2 strikes a balance between adapting to new rewards and maintaining stability. The discount factor of 0.5 indicates a preference for short-term gains while still considering future rewards. An exploration rate of 0.9 encourages active exploration of actions to discover better strategies. The epsilon decay rate of 0.05 gradually decreases exploration over time to shift towards exploitation. The choice of 1000 episodes provides sufficient iterations for the agents to learn and refine their strategies.

2.3 SARSA

SARSA (State-Action-Reward-State-Action) belongs to the family of on-policy temporal difference methods, where the agent learns directly from its interactions with the environment. In SARSA, the agent maintains an action-value function, which estimates the expected return for taking a particular action in a given state.

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha[r_t + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t)]$$

Through iterative updates, it learns to improve its action selection by updating the action-value function based on the observed state-action-reward-state-action transitions. The updates are performed using the temporal difference error, which measures the difference between the estimated and actual values.

2.4 Expected SARSA

Expected SARSA is a variant of SARSA that is also on-policy TD control algorithm that learns the optimal action-value function through interaction. The key

difference between SARSA and Expected SARSA lies in how the next action is selected. In SARSA, the next action is chosen based on the epsilon-greedy policy, meaning there is a probability of selecting a random action for exploration purposes. However, in Expected SARSA, instead of randomly selecting the next action, the algorithm calculates the expected value of the action based on the current action-value estimates and the corresponding probabilities of selecting each action.

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha[r_{t+1} + \gamma \sum_a \pi(a|s_{t+1}) Q_t(s_{t+1}) Q_t(s_{t+1}, a) - Q_t(s_t, a_t)]$$

By taking the expected value of the action, Expected SARSA accounts for the full range of possible actions and their associated probabilities, rather than relying on a single randomly selected action. This approach enables a more accurate estimation of the action-values and can lead to improved learning efficiency and better performance.

2.5 Q-learning

Unlike SARSA, Q-learning belongs to the family of off-policy methods allowing the agent to learn from the actions of an exploratory policy while still aiming to maximize the value of the optimal policy. In Q-learning, the agent maintains an action-value function, known as the Q-function, which estimates the expected return for taking a specific action in a given state.

$$Q_{t+1}(s_t, a_t) = (1 - \alpha) Q_t(s_t, a_t) + \alpha(r_t + \gamma \max Q(s_{t+1}, a))$$

Through iterative updates, it learns to improve its action selection by updating the Q-values based on the observed state-action-reward transitions. The updates are performed using the maximum expected future reward, also known as the Bellman optimality equation. By selecting the action that maximizes the Q-value for the next state, Q-learning learns to approximate the optimal action-values. This approach enables the agent to progressively refine its policy, converging towards an optimal strategy that maximizes the expected cumulative rewards.

2.6 Double Q-learning

Double Q-learning is an extension of the traditional Q-learning algorithm that addresses the overestimation bias issue commonly encountered when estimating action-values. While Q-learning can sometimes overestimate the values of certain actions, Double Q-learning mitigates this problem by decoupling action selection and action evaluation. We have two separate sets of action-values, commonly referred to as Q-functions or Q-tables.

$$Q^A_{t+1}(s_t, a_t) = Q^A_t(s_t, a_t) + \alpha_t(s_t, a_t)(r_t + \gamma Q^B_t(s_{t+1}, \arg\max Q^A_t(s_{t+1}, a)) - Q^A_t(s_t, a_t)$$

$$Q^B_{t+1}(s_t, a_t) = Q^B_t(s_t, a_t) + \alpha_t(s_t, a_t)(r_t + \gamma Q^A_t(s_{t+1}, \arg\max Q^B_t(s_{t+1}, a)) - Q^B_t(s_t, a_t)$$

During the learning process, one set of action-values is used for action selection, while the other set is used for action evaluation. This decoupling helps to reduce the bias introduced by overestimation, leading to more accurate value estimations.

3 Experiment and Results

The experiments have been conducted in two ways: by evaluating the RL agents' performance in a game against different number of players, whose decisions were randomized, and by testing them in a gameplay against each other.

3.1 Against random players

According to the Ludo rules, game is dedicated to two, three, or four players. If we assume that each player randomize their decisions, the probability of winning is equivalent for each of them. So, then we have 50% win probability if the game has two players, 33% for three of them, and 25% if three individuals are playing. In our problem, these values indicate the lowest level of winning rate that should be outperformed by every single, correctly trained RL agent. Therefore, I will treat these numbers as a kind of baseline and we can refer to them while comparing the final results.

Table 2: Win rate comparison

	Number of opponents			
	1	2	3	mean
Random player	0.5	0.33	0.25	0.36
SARSA	0.762	0.586	0.469	0.607
Expected SARSA	0.824	0.716	0.599	0.713
Q-learning	0.77	0.617	0.472	0.62
Double Q-learning	0.786	0.654	0.523	0.654

The results (tab. 2), provide insights into the performance of different learning algorithms in Ludo gameplay against varying numbers of opponents. The win rates achieved by the agents reveal their effectiveness in adapting to different game scenarios and competing against opponents. The win rate of the random player, as mentioned above, serves as a baseline, which indicates that the random player always has a lower chance of winning compared to the RL agent.

Among the RL algorithms, SARSA achieved a mean win rate of 0.607. This indicates that the SARSA agent was able to learn effective strategies and perform consistently better than the random player. However, the win rates achieved by Q-learning were even higher, with a mean win rate of 0.62. Both of the methods have been significantly outperformed by their variants: Double Q-learning and

Expected SARSA, which achieved mean win rates of 0.654 and 0.713, respectively. It indicates that tiny improvements, such es estimating probabilities in Expected SARSA and using double Q-tables in Double Q-learning can introduce several advantages that can increase the win rate by slight changes in the agent's behaviour.

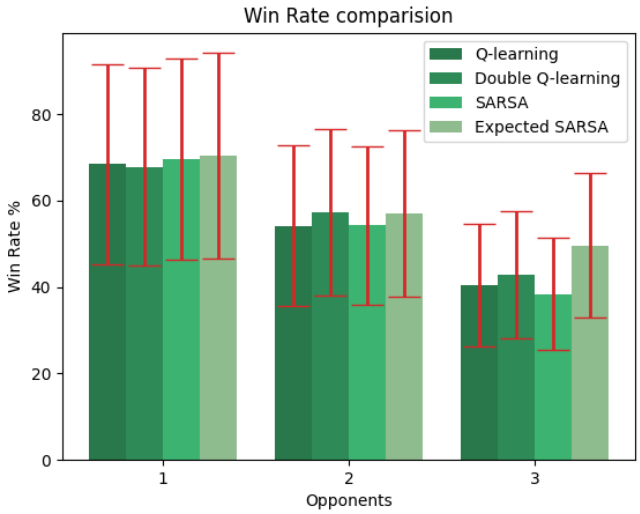


Fig. 2: Overall win rate comparison (means and standard deviations).

3.2 Against each other

The gameplay where agents rival against each other (fig. 3) actually brings us similar insights about their performance like playing with random opponents. When considering the performance of the RL algorithms in self-play scenarios, we observe that SARSA achieved a win rate of 20.7%, Q-learning achieved 22.5%, and Double Q-learning achieved a significantly higher win rate of 56.8%.

4 Discussion

The project gave us insights on the strengths and characteristics of each algorithm. All the methods show superior performance compared to a random player, demonstrating the effectiveness of reinforcement learning algorithms in learning and improving gameplay strategies. Their win rates indicate the ability to adapt and make optimal decisions based on the game's dynamics. Additionally, the substantial improvement observed in the Double Q-learning approach suggests

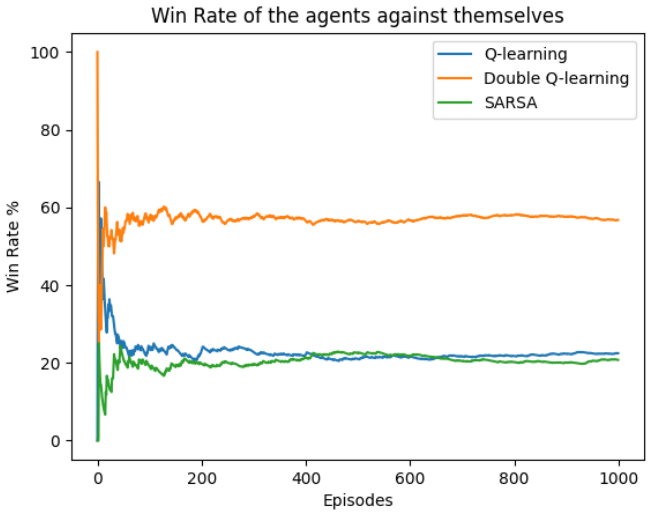
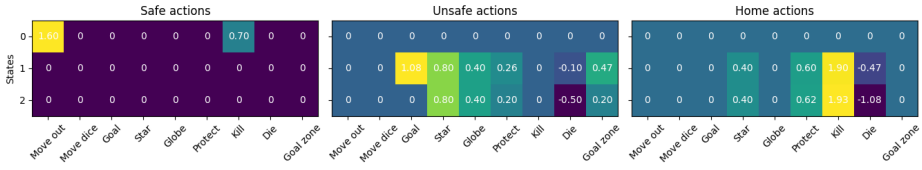


Fig. 3: Win rates (left) and their moving average (right) of the agents playing against themselves.

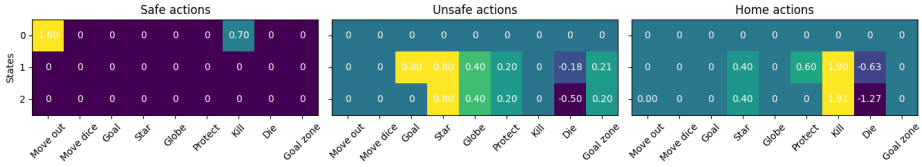
the advantage of using two value functions to mitigate the overestimation of action values, leading to more accurate estimations and better performance. It is worth noting that the difference between win rates of each method increases together with the growth of the number of players. The explanation of this phenomenom is quite simple, as with a higher number of players, the number of pawns also increases making the gameplay more strategic and less stochastic.

5 Conclusion

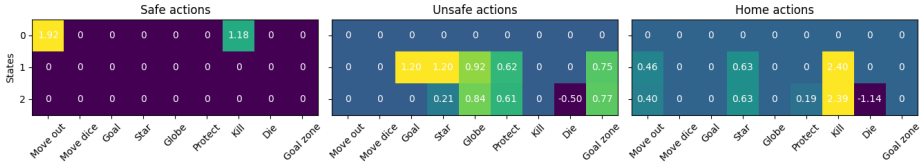
The project described various temporal difference learning methods in the problem of Ludo game. Future work would include the usage of other techniques, such as: Actor-Critic and genetic algorithms. By utilizing an actor network to approximate the policy and a critic network to estimate the value function, Actor-Critic algorithms have the potential to achieve more efficient and stable learning in complex environments like Ludo. Additionally, genetic algorithms provide an alternative approach to reinforcement learning. By applying principles of natural selection and evolution, genetic algorithms can evolve populations of agents with diverse strategies, allowing for the discovery of novel and effective gameplay tactics. Exploring the application of genetic algorithms in the context of Ludo could lead to the development of unique and adaptive playing strategies. They could be used, for instance, with combination with RL to fine-tune the rewards for the agent.



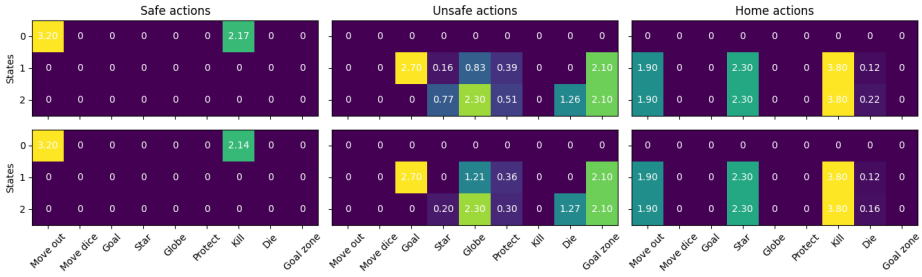
(a) SARSA



(b) Expected SARSA



(c) Q-learning



(d) Double Q-learning

Fig. 4: Q-tables

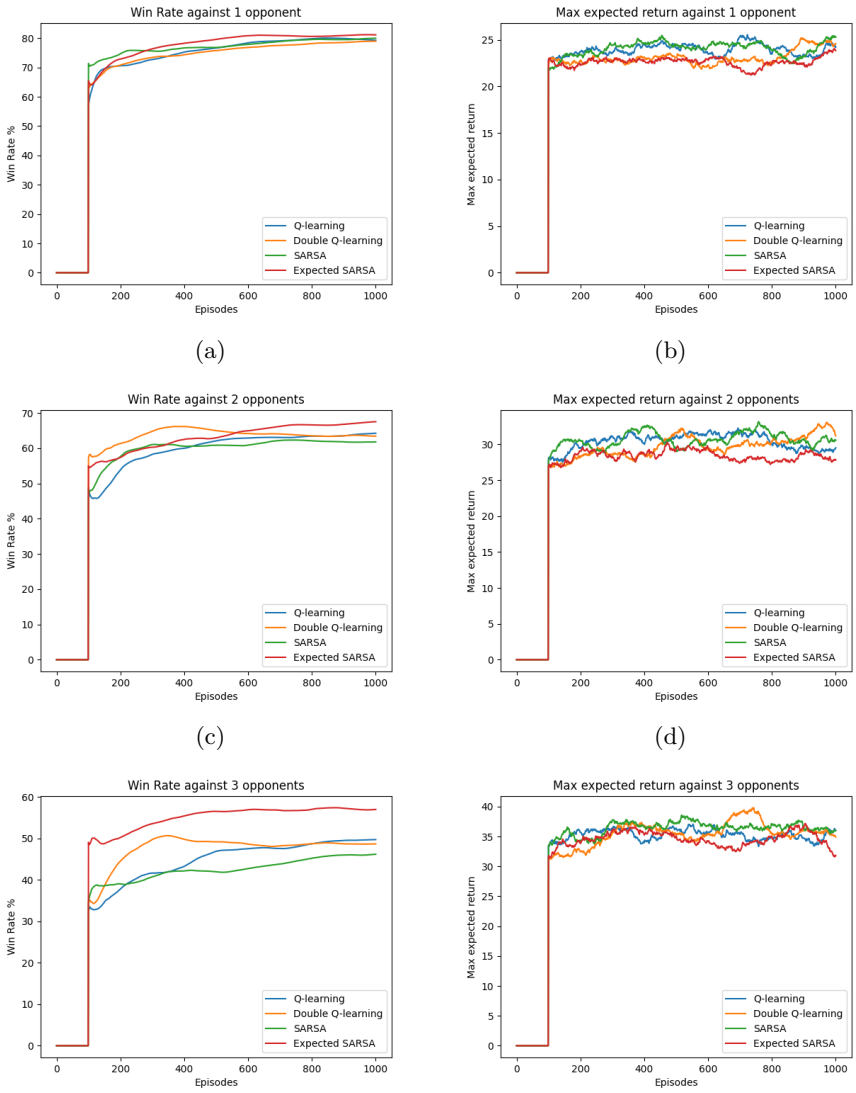


Fig. 5: Win rates against 2, 3, and 4 opponents (left) and maximal expected returns (right).