Reinforcement Learning Strategies for Text Flappy Bird: A Comparative Study of Monte Carlo and SARSA(λ) Methods

Axel Nguyen–Kerbel[1,2]

[1] CentraleSupélec, Gif-sur-Yvette, 91190, France
[2] ENS Paris-Saclay, Gif-sur-Yvette, 91190, France

Abstract. In this paper, we explore and compare two reinforcement learning methods, Monte Carlo and SARSA(λ), in the context of mastering the Text Flappy Bird game. We describe the experimental setup, including the selection of environments and implementation details for each method. The Monte Carlo method employs an epsilon-greedy agent updating Q-values at the end of each episode, while SARSA(λ) updates Q-values after each step. Through cross-validation experiments, we determine optimal hyperparameters for each method. Additionally, we analyze the performance of both methods on different geometry configurations and discuss their adaptability to the original Flappy Bird game. Despite differences in computational requirements and convergence speeds, both methods achieve comparable performance. Analysis of Q-values matrices reveals insights into decision-making processes and potential gameplay strategies. Our findings suggest that while both methods offer effective learning approaches, further research may explore optimization techniques to adapt to new game environments.

Keywords: Reinforcement Learning · Monte Carlo · SARSA(λ)

1 Introduction

The code is available at : https://github.com/AxelCole/Text-Flappy-Bird

1.1 Monte Carlo Method Description

Here, we have chosen to use an epsilon-greedy agent because this way the agent balances exploration and exploitation by selecting actions based on a combination of prior knowledge (Q-values) and random exploration. The actions available to the agent in the Text Flappy Bird game are limited to two: flapping or remaining idle.

During each episode, the agent learns from its experiences and updates its knowledge represented by Q-values. More precisely, after each episode, the agent updates its Q-values using the following equation:

$$Q(state, action) = (1 - \alpha) \times Q(state, action) + \alpha \times G_t \tag{1}$$

Where:

- $Q(state, action)$: The Q-value corresponding to the current state-action pair.
- α: The learning rate determining the weightage given to new information during Q-value updates.
- G_t: The discounted future reward obtained from the trajectory, representing the agent's cumulative reward from the current time step t until the end of the episode.

We can see that several parameters and hyperparameters play crucial roles in shaping the MC method's behavior for this epsilon-greedy agent:

- Max_steps: Sets the maximum number of steps an agent can take during an episode to prevent infinite loops. ***In the following, it is set to 1,000 which ensures ample learning opportunities.***
- ϵ, ϵ-decay, ϵ-min: ϵ controls the balance between exploration and exploitation. Initially high, it gradually decreases to prioritize exploitation. ϵ-decay governs this reduction, while ϵ-min sets a lower bound to maintain exploration. ***ϵ-decay and ϵ-min are respectively set to 0.999 and 0.001 in the following.***
- α, α-decay, α-min: α is the learning rate determining Q-value updates. Initially high, it decreases over time for stability. α-decay manages this decrease, while α-min prevents the rate from dropping too low. ***α-decay, α-min are respectively set to 0.999 and 0.001 in the following.***

We will see how we find the best set of hyperparameters with cross-validation when examining the results.

1.2 Description of the SARSA(λ) Method

The implemented SARSA(λ) method also relies on an epsilon-greedy agent but it updates all Q-values after each step. We use an eligibility trace (e) that tells which action in the past in a given state was responsible for the current positive or negative reward.

The SARSA update rule at time t is defined as:

$$\delta_{SARSA}(t) = R_t + \gamma \cdot Q(next\_state, next\_action) - Q(state, action) \quad (2)$$

For all states and actions in Q, the Q-value update is governed by the following formula:

$$Q(s,a) = Q(s,a) + \alpha \cdot \delta_{SARSA} \cdot \left(1_{current\_state, current\_action}(s,a) + e(s,a)\right) \quad (3)$$

Where:

- $\delta_{SARSA}(t)$: The SARSA error at time t, calculated as the temporal difference between the reward at time t (R_t) and the discounted value of the next state-action pair.
- γ: The discount factor determining the importance of future rewards. ***It is set to 0.99 in the following.***
- λ: The eligibility trace decay parameter, determining the memory span of the eligibility trace.

2 Considerations about the environment

2.1 Environment Differences

The TextFlappyBird-screen-v0 environment is not limited to the distance of the agent to the center of the pipe but encompasses a matrix of the entire screen size. This matrix includes information about the agent's position as well as the positions of pipes. While this richer representation could give more information to the agent, it comes at the cost of more computations (convolutions if we use CNNs) to build a word representation from the screen. Therefore, due to our limited computational ressources we have chosen the simplest environment : TextFlappyBird-v0. However, it should be kept in mind that allowing the agent to build its own representation from the whole screen could be pivotal to make it generalize to new geometries and environments even if the initial learning process will be harder as the agent has to learn what is important in the screen on its own.

2.2 Challenges in Transferring to the Real Flappy Bird Environment

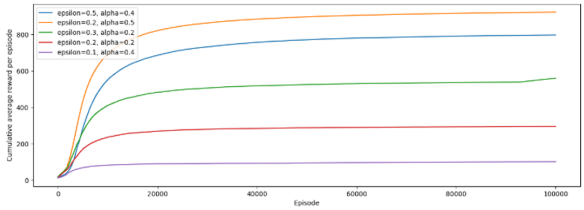
Text Flappy Bird simplifies the game mechanics by providing a minimalist textual representation of the environment. However, the real Flappy Bird game introduces a multitude of complexities absent in its textual counterpart. Notably, the real game features dynamic obstacles such as inclined pipes and moving obstacles in higher difficulty levels. Moreover, adversaries like bombs and carnivorous plants pose additional challenges that the agent must adapt to. Unlike Text Flappy Bird, the real game's observation space is more complex and includes visual and auditory signals. Additionally, the real game is non-stationary, requiring the agent to continuously adapt its policy to evolving gameplay dynamics—a feature absent in the simplified Text Flappy Bird version.

Addressing these challenges requires careful consideration of domain adaptation techniques and continuous learning strategies to ensure effective transfer of the agent's learned behaviors to the real Flappy Bird environment.

3 Methods and hyperparameter tuning

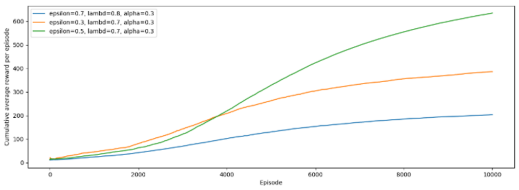
To determine the optimal hyperparameters ϵ, α, and λ for each method, we have done a cross-validation experiment. Due to our computational constraints, the experiment was limited to 5,000 episodes and a small number of hyperparameter combinations. It's important to note that performance over the initial 5,000 episodes may not accurately reflect long-term learning quality. That's why we further tested the best combinations identified during cross-validation over an extended training period to select the best final combination after 100,000 episodes.

Both the Monte Carlo and SARSA agents exhibit variability in results based on chosen hyperparameters. Despite potential variations, interesting scores are



ϵ	α	Average reward
0.5	0.4	797
0.2	**0.5**	**923**
0.3	0.2	560
0.2	0.2	295
0.1	0.4	102

Fig. 1. Monte Carlo method tested on 100,000 episodes with the 5 best hyperparameters combinations found during cross-validation.



ϵ	α	λ	Average reward
0.7	0.3	0.8	203
0.3	0.3	0.7	386
0.5	**0.3**	**0.7**	**635**

Fig. 2. SARSA(λ) method tested on 10,000 episodes with the 3 best hyperparameters combinations found during cross-validation.

consistently observed. In terms of learning rates, both methods require relatively low values for stability and convergence. Yet, in the Monte Carlo setup, the biggest alpha of 0.5 is identified as ideal at the end as shown in Figure 1, while the SARSA method performs well with a learning rate of 0.3 (Figure 2)

The SARSA method demonstrates a preference for greater exploration at the beginning of learning, as indicated by the favorable performance of higher epsilon values. Conversely, the Monte Carlo method benefits from a more conservative exploration/exploitation ratio, with a low initial epsilon value yielding optimal results. Furthermore, the SARSA method's effectiveness is enhanced by a higher lambda coefficient, indicating the importance of greater memory retention in the agent's learning process. This contrasts with the Monte Carlo method, where simpler memory updating mechanisms suffice due to the nature of the learning process. Overall, both methods offer viable approaches but they exhibit nuanced differences in their sensitivity to hyperparameters and learning dynamics. The SARSA method's adaptability to exploration and memory retention makes it well-suited for environments with non-stationary dynamics as we will se in the generalization section. Conversely, the Monte Carlo method's simplicity and conservative exploration approach may be advantageous for stability, robustness and time efficiency.

4 Results and discussion

4.1 Performances and convergence time

The main difference between the Monte Carlo and SARSA methods lies in their computation time. SARSA requires significantly more computation time due to updating Q-values after each step and updating all the Q-values of previously visited states. This contrasts with Monte Carlo, which updates Q-values after each episode and only for the states of the trajectory. Consequently, Monte Carlo reaches satisfactory solutions with around 10,000 episodes while SARSA may require more episodes for convergence. One takeaway is that SARSA exhibits slower performance compared to Monte Carlo, making the latter more efficient in achieving satisfactory results.

4.2 Comparative analysis of State-Value representations

In this section we analyze the final Q-value matrices, obtained after extensive learning with optimal hyperparameters.

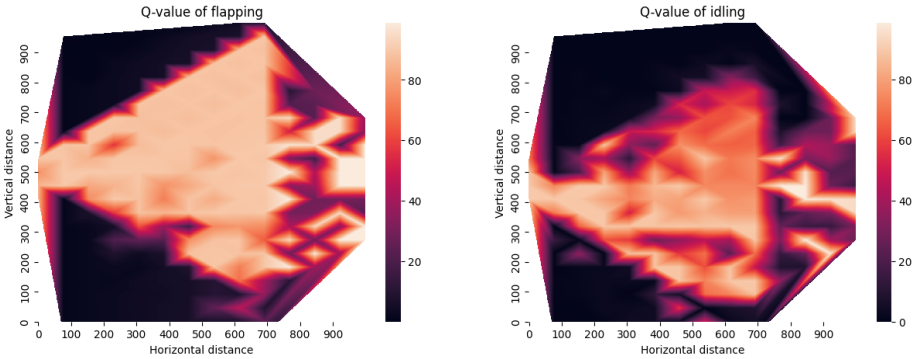


Fig. 3. Q-values after training for Monte Carlo

Those visualizations (Figures 3, 4) highlight several key insights:

- Sparse areas in the heatmap indicate regions where exploration was limited, suggesting not all states require visitation for effective learning.
- Black regions for both actions signify systematic failure areas, often near pipes and at extreme vertical positions.
- Light areas near pipes indicate high success probability, with precision required for optimal action selection.
- Triangular patterns in the heatmap illustrate the agent's learned behavior, favoring specific zones and actions.
- Despite different learning techniques, both Monte Carlo and SARSA methods yielded similar decision matrices, suggesting an optimal strategy for playing Text Flappy Bird.

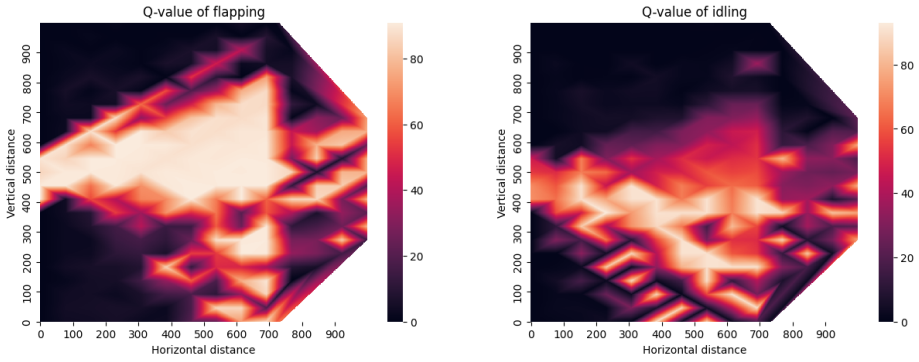


Fig. 4. Q-values after training for SARSA(λ)

4.3 Generalization to new geometries

Now that we have an optimal set of hyperparameters for both Monte-Carlo and SARSA(λ) agents, we can look at their abilities to adapt to new geometries. Are they able to generalize what they have learned to new settings ?

In assessing the performance of agents trained using different methods across varied geometries, the pipe gap remained fixed at 4, while adjustments were made to the height and width parameters around their base values of 15 and 20, respectively. For each geometry, each agent played 1,000 episodes and we are plotting the average reward per episode in Figure 5.

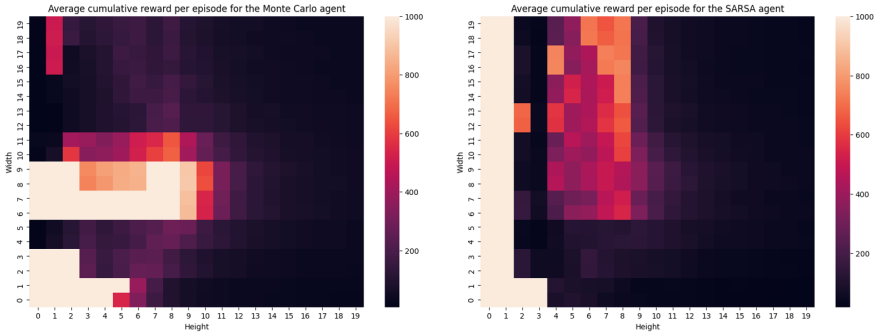


Fig. 5. Q-values after training for SARSA(λ)

Despite similar Q-values, the agents exhibit varying sensitivity to geometry changes. The Monte Carlo agent shows minimal sensitivity to screen height, while the SARSA agent is least affected by changes in screen width.

5 Conclusion

In this study, we compared two reinforcement learning methods, Monte Carlo and SARSA(λ), in the context of mastering the Text Flappy Bird game. We provided detailed descriptions of each method's implementation and experimental setup, including hyperparameter tuning through cross-validation.

Our results indicate that both methods offer viable learning approaches, albeit with nuanced differences. Monte Carlo demonstrates faster convergence and efficiency in achieving satisfactory solutions compared to SARSA(λ), which exhibits slower performance due to its more computationally intensive nature.

Analysis of Q-values matrices provided insights into decision-making processes, revealing similar decision patterns despite different learning techniques. However, both methods showed varying sensitivity to changes in game geometry, highlighting the need for further investigation into their adaptability to diverse environments.