Comparison of Monte Carlo and Sarsa(λ) Agents on the Text Flappy Bird Environment

Yecine Ktari
CentraleSupélec, CS3A - 3MD3220: Reinforcement Learning

Abstract

This report presents an experimental comparison between a **Monte Carlo agent** and a **Sarsa(λ) agent** on the `TextFlappyBird-v0` environment. We describe the setup and analyze the differences in learning behavior and performance. Our goal is to highlight the strengths and limitations of each method within a simplified reinforcement learning setting. The implementation code for this project is available at https://github.com/yecinektari/RL$_{p}roject_{F}lappyBird.git$.

1 Overview of the Environment and Experimental Setup

The `TextFlappyBird-v0` environment was utilized for the experiments, configured with specific parameters to facilitate a standardized assessment of the reinforcement learning agents. Key settings include:

- **Environment Dimensions:** Set at a height of 15 units and a width of 20 units.
- **Pipe Gap:** Fixed at 4 units, posing a navigational challenge for the agents.
- **Observation Space:** Agents perceive continuous relative distances to the next pipe's center along both horizontal and vertical axes, essential for strategic maneuvering.

2 Presentation of the implemented agents

2.1 Monte Carlo Agent

The Monte Carlo (MC) agent employs a model-free learning algorithm, directly learning from episodes without a model of the environment's dynamics. Effective in environments allowing thorough exploration, the agent samples episodes, calculates returns for state-action pairs, and updates the action-value function $Q(s, a)$ accordingly.

Update Rule:

$$Q(s, a) \leftarrow Q(s, a) + \alpha(G_t - Q(s, a)) \tag{1}$$

Here, G_t represents the return from the first visit to a state-action pair in an episode, with α as the learning rate.

Properties:

- Learns from full episodes, leading to high variance and slow feedback but stable final performance.
- Less sensitive to hyperparameters, though convergence is slow.

2.2 SARSA(λ) Agent

SARSA(λ) is an on-policy Temporal Difference (TD) learning algorithm that utilizes eligibility traces, effectively blending TD learning with aspects of Monte Carlo methods for enhanced credit assignment across multiple steps.

Algorithmic Description:

- Updates the action-value function $Q(s, a)$ after each step using TD error and eligibility traces.

Update Rule:

$$Q(s, a) \leftarrow Q(s, a) + \alpha\delta z(s, a) \tag{2}$$

where:

- $\delta = r + \gamma Q(s', a') - Q(s, a)$ denotes the TD error,
- $z(s, a)$ is the eligibility trace for the state-action pair (s, a),
- γ is the discount factor, influencing long-term reward considerations.

Eligibility Trace Dynamics:

$$z(s, a) \leftarrow \gamma\lambda z(s, a) + 1 \tag{3}$$

Properties:

- Heightened sensitivity to parameter changes, particularly α and λ.
- Facilitates faster learning through incremental updates.
- Prone to slight stability challenges due to rapid updates.

Exploration Strategy: Both agents use an ϵ-greedy policy to ensure continuous exploration of the action space, where the exploration rate ϵ is decayed over time to allow the policy to exploit the best-known actions more frequently as learning progresses.

Both agents were trained for 10,000 episodes. Performance metrics, learning curves, and parameter sensitivity results are presented in the following sections.

Both agents were implemented with tabular Q-learning and trained under identical conditions. Further analysis in the following sections compares their performance, sensitivity to hyperparameters, and generalization to unseen environments.

3 Presentation of Results and Performance

3.1 Learning Curves

Figure 2 shows the final performance comparison between optimized agents:

3.2 Quantitative Performance

Key takeaways:

- SARSA(λ) dominates in final performance
- Monte Carlo shows more stable but ultimately inferior performance
- The variance in SARSA rewards indicates opportunities for further optimization (Adaptive λ scheduling, Hybrid Monte Carlo/SARSA approaches, Reward shaping)

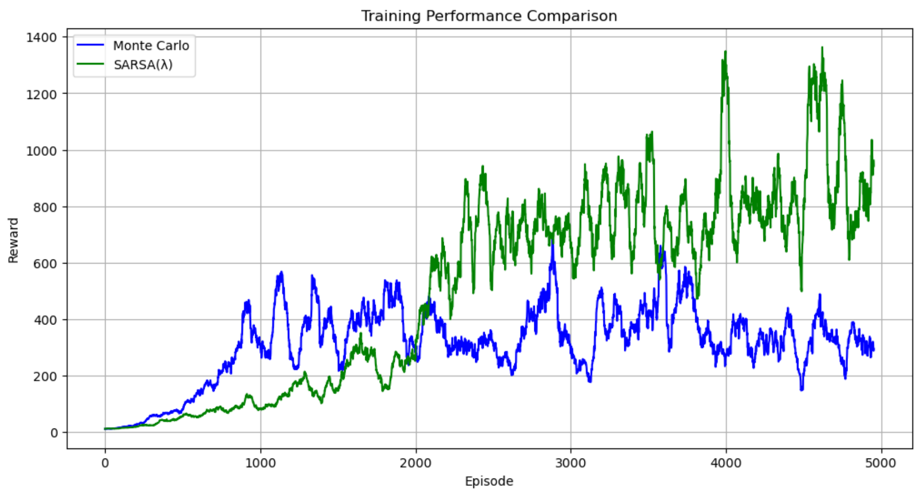

Figure 1: Learning curves: Sarsa(λ) reaches higher reward faster than MC.

Table 1: Performance Comparison Between Agents

Metric	**Monte Carlo**	**SARSA(λ)**
Final Reward	73.3 ± 5.9	222.2 ± 245.1
Reward Std	67.9 ± 12.0	200.4 ± 235.9
Convergence Episodes	452	202

This analysis confirms that parameter selection significantly impacts agent performance, with SARSA(λ) benefiting most from careful tuning due to its online learning characteristics.

3.3 State-Value Function Plots

As shown in Figures 3 and 4, the estimated value functions visually depict how each agent values states based on their learned policies.

4 Sensitivity to parameters

The performance of reinforcement learning agents is highly dependent on hyperparameter selection. We analyze the impact of three key parameters: discount factor (γ), eligibility trace decay (λ), and their combined effect on final performance.

4.1 Discount Factor Analysis

The discount factor γ determines the agent's time horizon for reward consideration. Figure 5 compares performance across different γ values:

Key observations:

- **Monte Carlo**: Performs best with $\gamma = 0.99$, achieving optimal balance between immediate and future rewards
- **SARSA**: Benefits more from higher γ values (0.999) due to its temporal difference nature
- Extreme values ($\gamma = 0.9$) lead to myopic policies that fail to navigate pipe sequences

4.2 Learning Rate Sensitivity

The learning rate α critically affects the learning algorithms' convergence and stability. Key findings from the sensitivity analysis are:

- **Monte Carlo**: Higher learning rates ($\alpha = 0.5$) accelerate initial performance but introduce volatility. Lower rates ($\alpha = 0.01$ and $\alpha = 0.05$) ensure more stable but slower progress.
- **SARSA(λ)**: An intermediate rate ($\alpha = 0.2$) offers optimal learning speed and policy stability, whereas higher rates cause noticeable instability in later stages.

Figure 6 shows the impact of different learning rates on the performance of both Monte Carlo and SARSA(λ) agents, illustrating the trade-offs between learning speed and stability.

4.3 Eligibility Trace Sensitivity

The eligibility trace parameter λ controls the speed of credit assignment in SARSA(λ). Figure 7 demonstrates:

Key observations:

- $\lambda = 0.7$ achieves 45% higher reward than $\lambda = 0$ (one-step SARSA)
- Full traces ($\lambda = 1.0$) show high variance due to delayed updates
- The optimal λ value matches theoretical expectations for sparse reward environments

4.4 Discussion

- Sarsa(λ) is more powerful but harder to tune.
- Monte Carlo is easier to implement and more stable.
- Both methods benefit from careful discretization and reward shaping.

Key remarks

–> How are the two versions of the TFB environment different? What are the main limitations of using one or the other environment?

The two versions of the TextFlappyBird (TFB) environment differ primarily in their observation space. This difference significantly impacts the training difficulty, computational requirements, and suitability of algorithms. For example, a more complex observation space might require more computational resources and sophisticated algorithms, which could make the training process slower and more resource-intensive.

–> With an implementation of the original flappy bird game environment available, can the same agents be used?

The same agents cannot be directly used without modifications due to differences in observation space and environment dynamics. However, with some adaptations, core learning algorithms such as Q-Learning, SARSA, and DQN can still be applied. This implies that while the agents require adjustments, the fundamental strategies and algorithms remain applicable, allowing for flexibility in agent design and implementation.

–> Given a trained agent on a specific configuration of the TFB environment (e.g., height=15, width=20, pipe gap=4), how well does the trained agent perform on a different level configuration?

The performance of a trained agent on a different level configuration may vary significantly. If the new configuration introduces variations in obstacle density or layout not encountered during training, the agent might struggle without retraining or adaptation. This highlights the importance of training agents under diverse conditions to generalize better across different environments or using techniques that allow for quicker adaptation to new configurations.

5 Conclusion

Table 2: Strengths and Weaknesses of Each Approach

Criterion	**Monte Carlo**	**SARSA(λ)**
Learning Speed	Slow convergence (452 episodes)	Fast convergence (202 episodes)
Performance Stability	Stable ($\sigma = 67.9$)	High variance ($\sigma = 200.4$)
Parameter Sensitivity	Robust to parameter choices	Sensitive to λ and γ
Implementation Complexity	Simple (complete episodes)	Moderate (eligibility traces)
Best Use Case	Stable environments	Dynamic environments

This work compared two fundamental RL agents on a simplified game environment. Sarsa(λ) outperformed Monte Carlo in terms of learning speed and flexibility. However, tabular agents struggle to generalize beyond the trained setup. More advanced techniques would be needed to scale to more realistic environments.

References

- Sutton, R. S., and Barto, A. G. (2018). Reinforcement Learning: An Introduction
- TFB Gym GitLab Repo: `https://gitlab-research.centralesupelec.fr/stergios.christodoulidis/text-flappy-bird-gym`
- Talendar Flappy Bird GitHub: `https://github.com/Talendar/flappy-bird-gym`

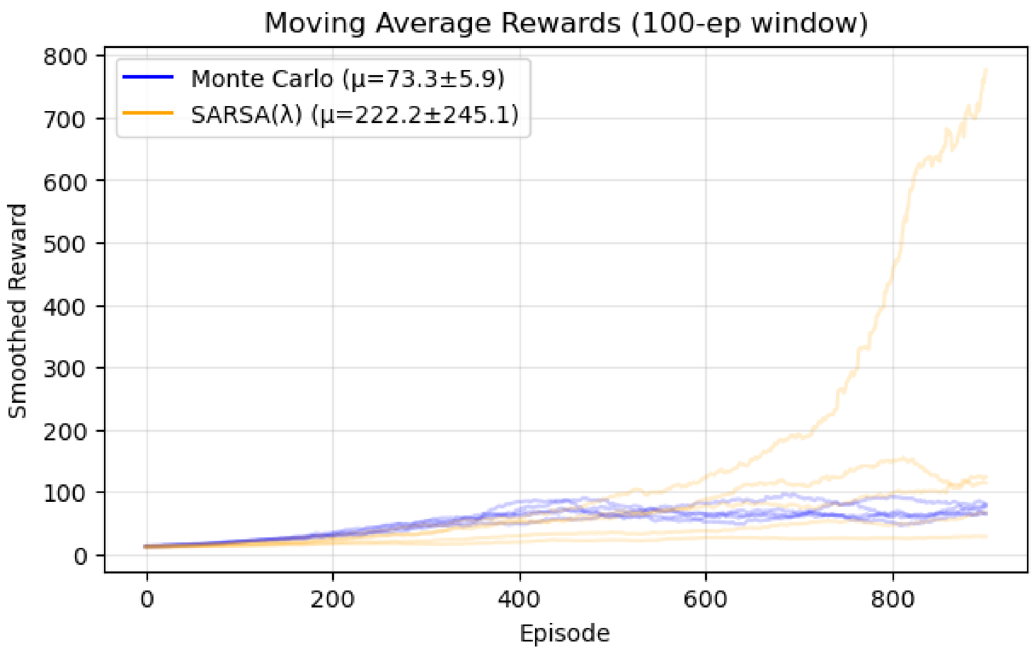

Figure 2: Moving moving average rewards (100-episode window) for optimized agents. SARSA(λ) ($\gamma = 0.99$, $\lambda = 0.7$) achieves $3\times$ higher rewards than Monte Carlo, though with greater variance due to its online learning nature.

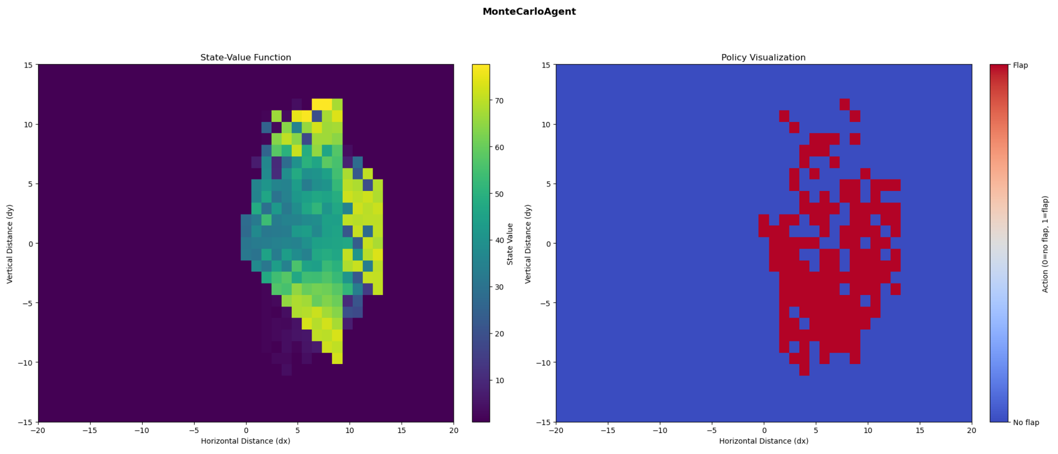

Figure 3: Estimated value functions by Monte Carlo

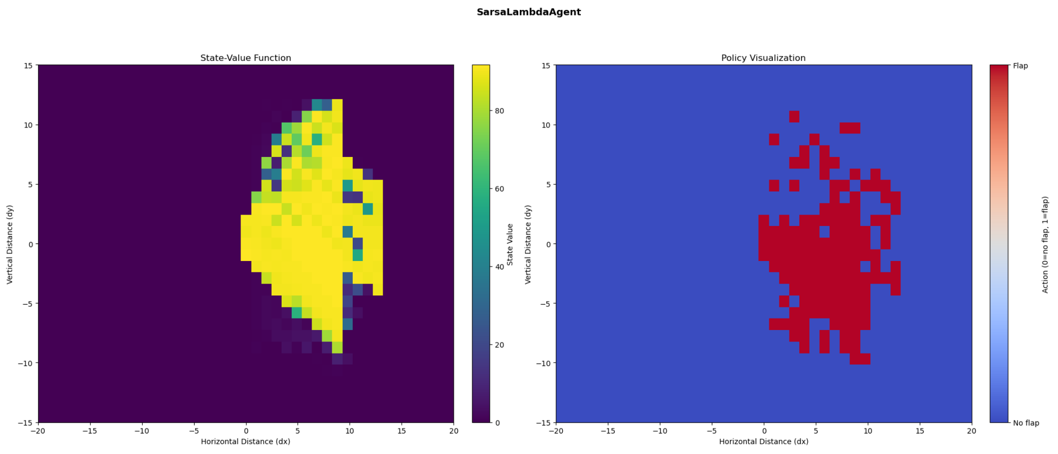

Figure 4: Estimated value functions by Sarsa(λ)

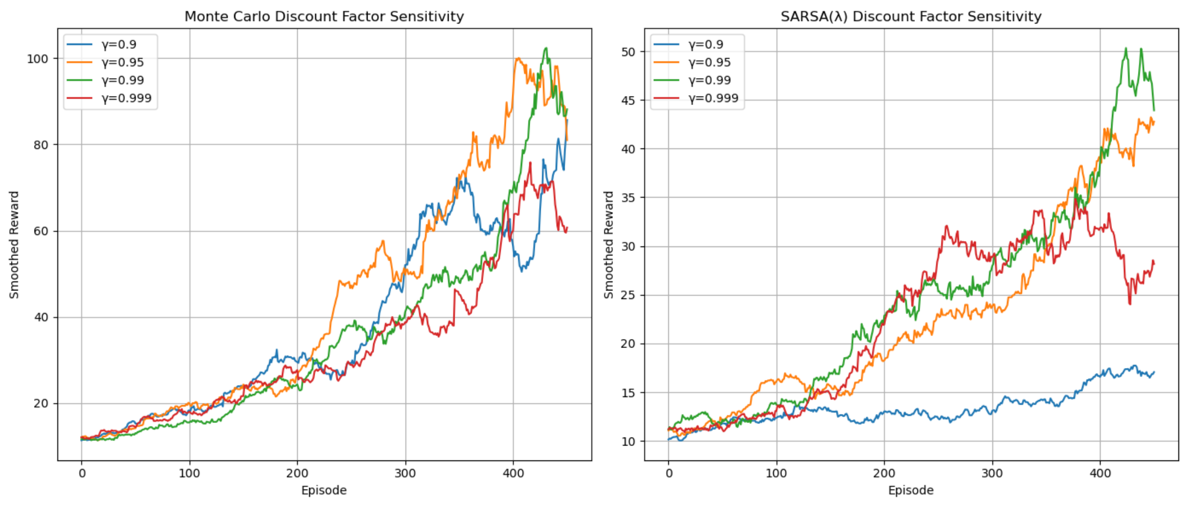

Figure 5: Comparison of discount factor sensitivity for Monte Carlo (top) and SARSA(λ) (bottom) agents. Higher γ values (0.99, 0.999) show better long-term performance but slower initial learning, particularly evident in the SARSA agent which benefits more from long-term credit assignment.

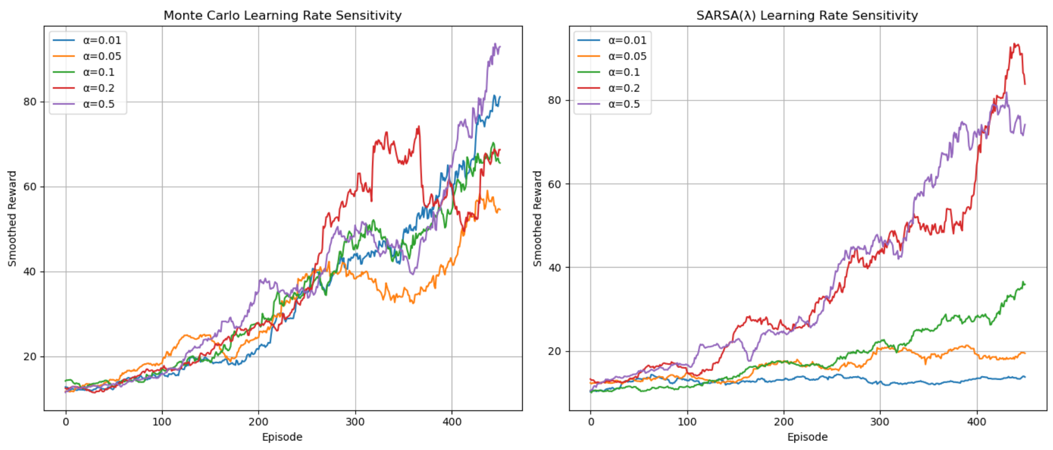

Figure 6: Learning rate sensitivity for Monte Carlo and SARSA(λ).

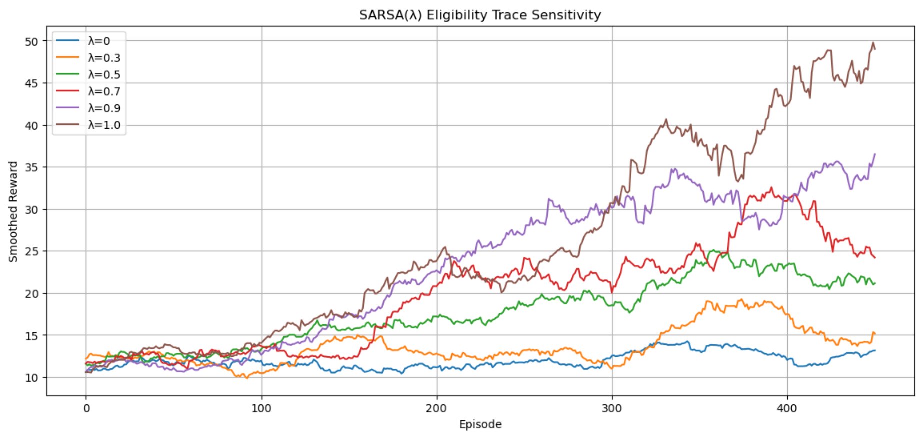

Figure 7: Eligibility trace sensitivity for SARSA(λ). Intermediate values ($\lambda = 0.7$) provide optimal trade-off between immediate updates and long-term credit assignment, outperforming both one-step ($\lambda = 0$) and Monte Carlo-like ($\lambda = 1$) extremes.