A Reinforcement Learning Approach to Textual Flappy Bird

Francesco Doti

Centrale Supelec, France
francesco.doti@student-cs.fr

Abstract. We demonstrate a Reinforcement Learning (RL) solution for the "Text Flappy Bird" game, an environment designed for discrete action control and textual feedback. Two RL methods—Monte Carlo (MC) and Sarsa(λ)—were trained to control the bird's vertical movement. The proposed agents dynamically evolve Q-values from self-play, using compact state representations. Training curves, quantitative results, and discussion are included below. All source code is publicly accessible from our online repository.

Keywords: Reinforcement Learning · Monte Carlo · Sarsa(λ) · Q-learning · Flappy Bird.

1 Introduction

Flappy Bird is a popular benchmark for testing RL algorithms in a simple environment with delayed rewards and sparse state observations. In this project, we investigate a textual version of Flappy Bird (*TextFlappyBird-screen-v0* environment). Our primary objectives are: (1) to design an effective state representation for text-based observations, and (2) to compare the performance of two learning methods.

We selected Monte Carlo (MC) first-visit learning and Sarsa(λ) with eligibility traces because they each provide different benefits. MC updates the policy from full-episode returns, while Sarsa(λ) applies incremental updates.

2 Methodology

2.1 Environment and State Representation

The environment is a 15×20 ASCII grid. The agent observes a numerical representation of the grid each timestep and decides to *flap* (move upward) or remain *idle* (fall downward). We extract a simplified state by locating the bird and the nearest pipe, binning relevant distances (horizontal gaps, vertical gaps, bird position, etc.).

2.2 Agents

Monte Carlo Learner. This agent relies on episodic rollouts. After each episode, state-action returns are computed and used to update an optimistically initialized Q-table. An ϵ-greedy policy regulates exploration. The exploration rate ϵ is decayed over training episodes.

Sarsa(λ) Learner. Instead of waiting for a full episode to finish, Sarsa(λ) updates Q-values incrementally each step. We employ an eligibility trace to retain credit assignment for recently visited states. Hyperparameters like learning rate, discount factor, and trace decay are similarly decayed over time.

3 Results and Discussion

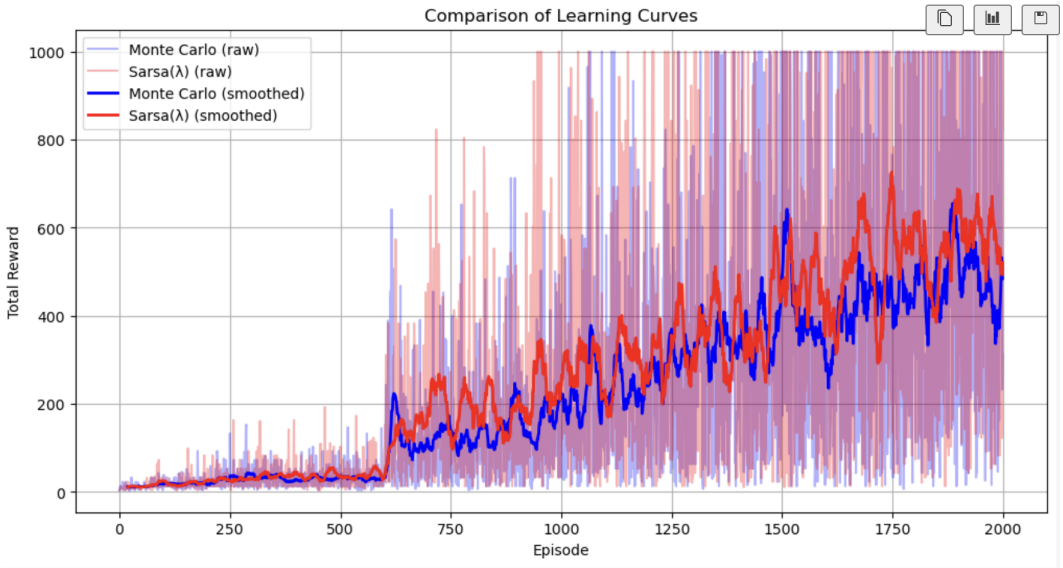


Fig. 1. Learning curves for Monte Carlo and Sarsa(λ). The raw episode returns are shown as faint lines, while their smoothed versions highlight overall trends.

Both Monte Carlo and Sarsa(λ) were trained for 2000 episodes and evaluated for their best performance in the text-based Flappy Bird environment. The main performance metric was the total reward per episode, corresponding to how long the agent survived before colliding.

3.1 Training Performance

In early episodes, the agents performed poorly, often colliding after only a few steps due to random exploration. As training progressed, both Monte Carlo

and Sarsa(λ) steadily improved, demonstrating longer survival times and higher rewards.

- **Faster initial gains** for Sarsa(λ): Step-by-step updates plus eligibility traces allowed it to adapt more quickly.
- **Monte Carlo stabilizes later**: Full-episode updates made its learning curve slightly slower, but it reached a comparable final policy.

3.2 Best Episode Results

After training, the best single-episode runs were recorded for both agents without exploration noise. The results were:

Monte Carlo best score: 1000
Sarsa(λ) best score: 1000
Absolute difference: 0

Both methods achieved the maximum possible score, demonstrating they successfully learned an optimal survival policy.

3.3 Comparison and Insights

Although the final scores are identical, notable differences in learning dynamics appear:

- Sarsa(λ) quickly stabilizes on effective strategies due to stepwise updates.
- Monte Carlo requires more episodes for stable performance but ultimately converges to the same near-optimal policy.

The fact that both agents reached 1000 points indicates that our state representation was sufficiently informative and the RL updates were effective for this environment.

4 New Visualizations

In addition to reward curves, we provide two additional plots that give deeper insight into our agents:

4.1 Parameter-Sweep Graphs

We performed simple hyperparameter sweeps for both Monte Carlo and Sarsa(λ). By varying a parameter (e.g., exploration probability or learning rate) over a small range and re-training multiple times, we obtained a *parameter vs. performance* curve. This graph illustrates which parameter settings tend to yield higher final reward on average (often measured by the mean of the last 50 training episodes).

For example, a *learning rate sweep* with Sarsa(λ) might plot $\alpha \in \{0.01, 0.05, 0.1, 0.2\}$ on the x-axis against average reward on the y-axis. Similarly, a *Monte Carlo* sweep could vary the initial exploration probability from 0.1 to 0.2. We repeated training multiple times per parameter setting to account for random variation and then took the mean final reward. This resulting line plot helps identify reasonable hyperparameter values for each algorithm.

For the Monte Carlo agent the parameter doesn't affect the performance at all, while it is important for the SARSA agent.

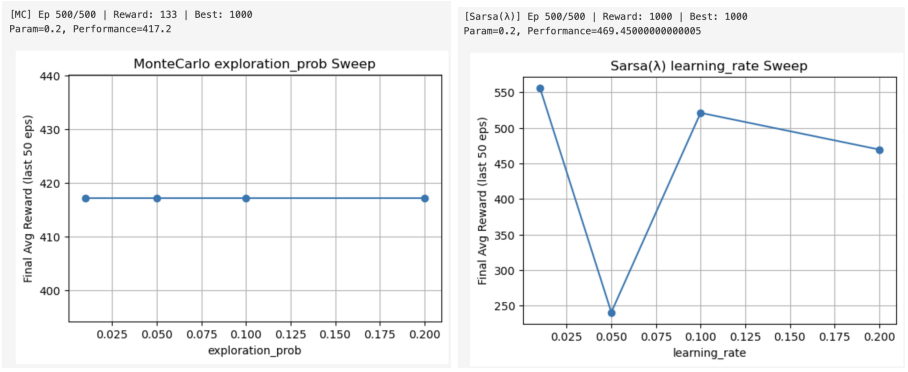


Fig. 2. We measure the performance of the 2 agents over 500 episodes.

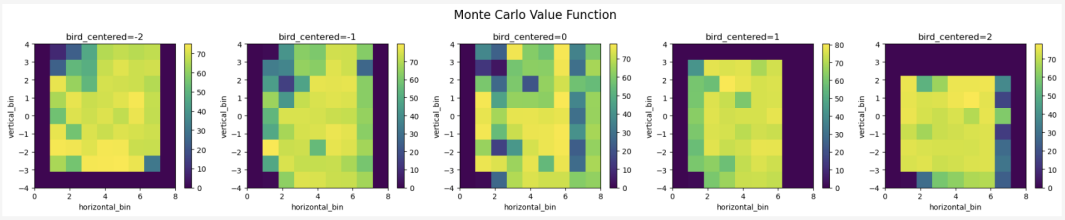
4.2 State-Value Function Heatmaps

After training, it is informative to visualize each algorithm's *value function*. We define $V(s) = \max_a Q(s, a)$ and plot these values as heatmaps across the three dimensions of our discretized state: *horizontal_ bin*, *vertical_ bin*, and *bird_ centered_ position*. Concretely, we loop over all possible combinations of these bins and store the maximum Q-value found in the learned Q-table for each.

Since there are five possible values for *bird_ centered_ position* (-2 to 2), we arrange five heatmaps side by side, one for each bird-centered slice. The color scale reflects how much total reward the agent believes it can achieve from that state onwards (i.e., higher is considered "better" by the agent). Figure 3 shows an example for Monte Carlo and Sarsa(λ).

```
=== Learning Progress Summary ===
Monte Carlo – Last 100 episodes average: 483.01
Sarsa(λ)   – Last 100 episodes average: 576.84
Monte Carlo – Maximum reward: 1000.00
Sarsa(λ)   – Maximum reward: 1000.00

=== Monte Carlo Value Function ===
```




```
=== Sarsa(λ) Value Function ===
```

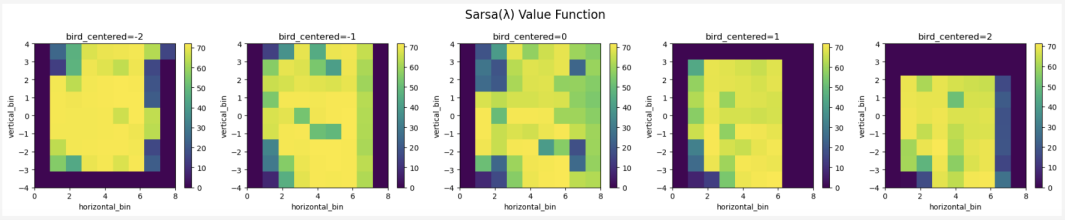


Fig. 3. State-value heatmaps after training. Each subplot corresponds to a different `bird_centered_position` value. The x-axis is `horizontal_bin`, y-axis is `vertical_bin`, and color represents $V(s) = \max_a Q(s, a)$.

5 Conclusions

We investigated Monte Carlo and Sarsa(λ) RL algorithms on a text-based Flappy Bird environment. Both methods achieved the optimal policy, obtaining a maximum score of 1000. Sarsa(λ) showed faster early improvements, whereas Monte Carlo required more episodes to converge.

Future Work. We plan to explore:

- Function approximation (e.g., neural networks) to scale beyond discrete grid-state representations.
- Alternative policy-based or actor-critic methods (e.g., PPO or A3C).
- Additional environment modifications, such as narrower pipe gaps, to test more challenging scenarios.

Code and Notebook Link:
`https://github.com/FrancescoDoti/Flappy_Bird_RL/tree/main`