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 policy from full-episode returns, while Sarsa(λ) applies incremental updates.

2 Methodology

2.1 Environment and State Representation

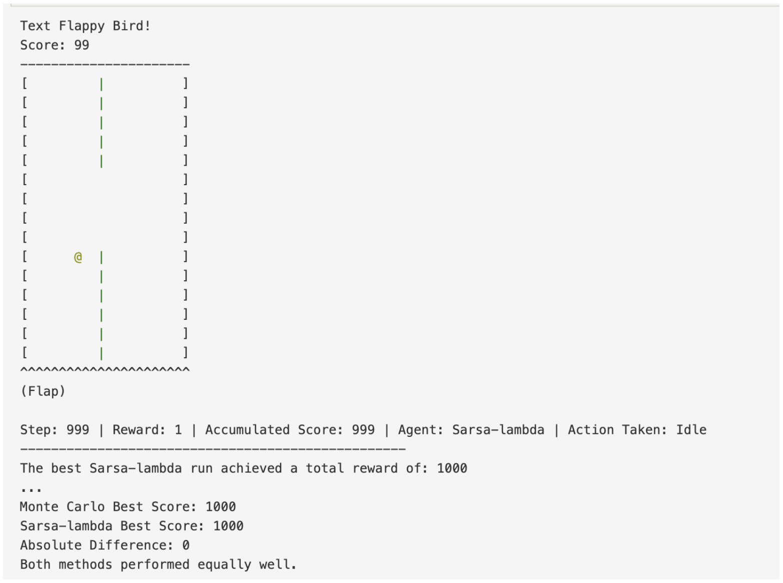
The environment is a 15×20 ASCII grid. The agent observes a numerical representation of the grid each timestep and decides to *flap* (move upward) or remain *idle* (fall downward). We extract a simplified state by locating the bird and the nearest pipe, binning relevant distances (horizontal gaps, vertical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



```
Text Flappy Bird!
Score: 99
----------------------
[         |          ]
[         |          ]
[         |          ]
[         |          ]
[         |          ]
[                    ]
[                    ]
[                    ]
[                    ]
[      @  |          ]
[         |          ]
[         |          ]
[         |          ]
[         |          ]
[         |          ]
^^^^^^^^^^^^^^^^^^^^^^
(Flap)

Step: 999 | Reward: 1 | Accumulated Score: 999 | Agent: Sarsa-lambda | Action Taken: Idle
--------------------------------------------------
The best Sarsa-lambda run achieved a total reward of: 1000
...
Monte Carlo Best Score: 1000
Sarsa-lambda Best Score: 1000
Absolute Difference: 0
Both methods performed equally well.
```

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, which corresponds to how long the agent survived before colliding with an obstacle.

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 per episode.

- Faster initial gains for Sarsa(λ): The step-by-step updates and eligibility traces allowed Sarsa(λ) to adjust its policy more quickly in the first few hundred episodes.
- Monte Carlo achieves stability later: MC's full-episode updates made its learning curve slightly slower, but its final policy reached comparable performance.

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 agents achieved the maximum possible score of 1000 points in their best runs, demonstrating that they successfully learned an optimal policy for navigating the game environment. The output log in Figure **??** shows an agent running for 999 steps, accumulating maximum rewards without failing.

3.3 Comparison and Insights

Although the final scores were identical, there are notable differences between the learning processes:

- Sarsa(λ) achieves early competency faster, likely due to its ability to adapt after every step.
- Monte Carlo requires more training episodes before reaching the same level of performance.
- Final policy behaviors appear nearly identical, suggesting that both methods successfully learn a near-deterministic survival strategy.

The fact that both agents reached the perfect score indicates that the state representation was effective, and the game structure was sufficiently learnable by tabular Q-learning approaches.

4 Conclusion

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

Future work could explore:

- Using deep RL models (DQN, PPO, or A3C) for more complex textual environments.

- Applying function approximation instead of a tabular Q-table.
- Modifying the game difficulty (e.g., reducing pipe gaps or increasing speed) to test more advanced policies.

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