3MD3220: Reinforcement Learning Individual Assignment

Marius DRAGIC

CentraleSupélec

Abstract. This project explores the effectiveness of 2 leading RL algorithms applied to the simplified game Text Flappy Bird (TBF) [1] [3]. In this report, we study the performance of 2 agents trained using the Monte-Carlo and SARSA algorithms. The 2 agents are trained on the same environment in order to make relevant performance comparisons.

GitHub Repository: github.com/MariusDragic/RL-FlappyBird

Keywords: RL · TBF · Monte-Carlo · SARSA

1 Introduction

The aim of this project is to use Reinforcement Learning techniques to build a high-performance agent capable of playing the Text Flappy Bird game. We use a simplified Text Flappy Bird game in order to solve this problem more easily, since the importance of the project lies above all in the comparison of the SARSA and Monte-Carlo methods we are implementing.

In this report, we will analyze and compare the performance of these 2 methods, as well as the influence of the hyperparameters used. The ultimate aim is to present the relative strengths/weaknesses of each algorithm in the context of their application to a simplified problem.

1.1 Methodology

1.2 Environment

Text Flappy Bird (TFB) is a simplified grid-based version of Flappy Bird, where an agent controls a bird that must navigate through pipe gaps. The state at time t is represented as $s_t = (d_x, d_y)$, where d_x is the horizontal x distance to the next pipe, while d_y is the vertical y distance to the center of the next pipe. The agent chooses between two actions: flap ($a = 1$) to jump or do nothing ($a = 0$) to let gravity apply. The reward function grants +1 each timestep as long as it stays alive.

1.3 Agents

1.3.1 Monte Carlo Agent The Monte Carlo (MC) agent estimates the action-value function $Q(s, a)$ by averaging returns from complete episodes. Using an ϵ-greedy policy, it explores and exploits while updating $Q(s, a)$ as:

$$Q(s,a) \leftarrow \frac{1}{N(s,a)} \sum G_t$$

where G_t is the total return from state-action pair (s, a) and $N(s, a)$ is the visit count. This method ensures convergence as episodes accumulate.

1.3.2 SARSA Agent The SARSA(λ) agent updates $Q(s, a)$ incrementally at each step using:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left(R_{t+1} + \gamma Q(s',a') - Q(s,a)\right)$$

where (s', a') is the next state-action pair. Unlike MC, SARSA(λ) integrates real-time feedback, leading to faster adaptation.

1.4 Hyperparameters

After testing the influence of each hyperparameter (2.4), we choose the following hyperparameters for each agent, summarized in the table:

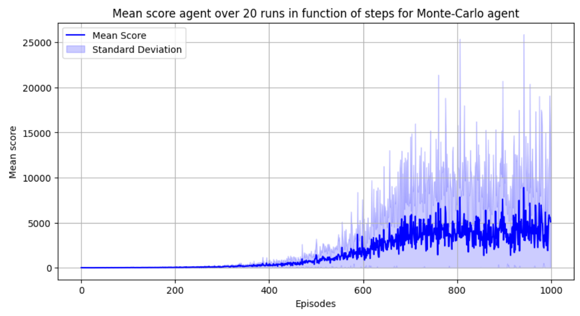
Hyperparameter	**Monte Carlo**	**SARSA(λ)**
Gamma (γ)	0.95	0.99
Epsilon (ϵ)	1.0	1.0
Epsilon Decay	0.990	0.995
Epsilon Min	0.001	0.001
Alpha (α)	-	0.1
Lambda (λ)	-	0.9
Number of Validation Games	50	50

Table 1: Comparison of hyperparameters for Monte Carlo and SARSA(λ) agents.

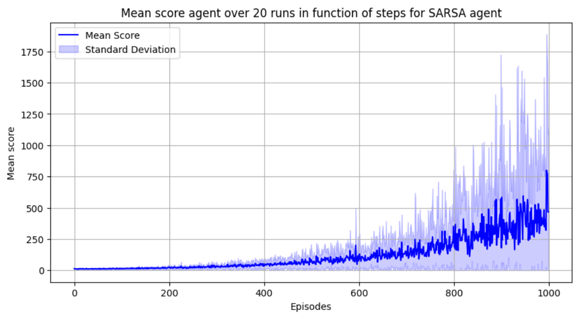
2 Results

2.1 Mean score per agent

The agents' performances are highly dependent on the conditions of the games, and therefore on the chance that generates them. In order to evaluate and compare their performance more accurately, we define a function *plot_avg_score* which performs 20 different learnings over 1000 episodes and records the average at each episode of all the learnings as well as their variance.



(a) Monte Carlo mean score agent



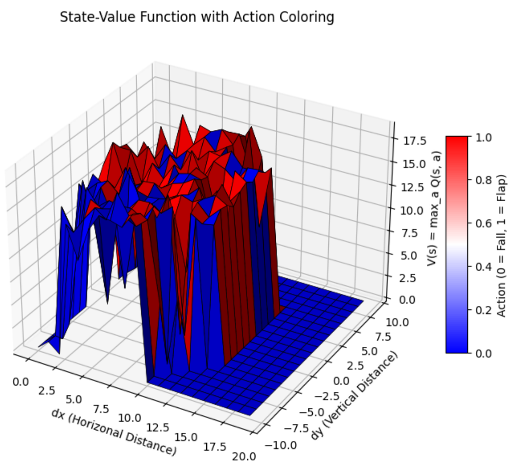
(b) SARSA mean score agent

Fig. 1: Comparison of Monte Carlo and SARSA mean scores

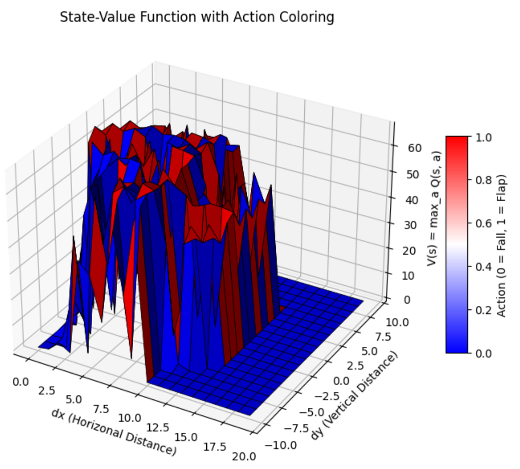
Over 1000 episodes, the Monte Carlo agent performs much better than the SARSA agent, with an average score of over 5000 compared with less than 1000 for the SARSA agent. The same applies to the variance of the Monte Carlo agent, which reaches 25000 compared with 1750 for SARSA. It's important to note here just how high the variance of the results is, with some training over 1000 episodes producing very good agents, as well as mediocre ones in some cases, hence underlying the importance of carrying out several trainings to obtain a more revealing average score.

Note that we're talking here about training agents with an epsilon greedy exploration strategy. If we then evaluate their performance with $\epsilon = 0$, their performance is much better (2.3).

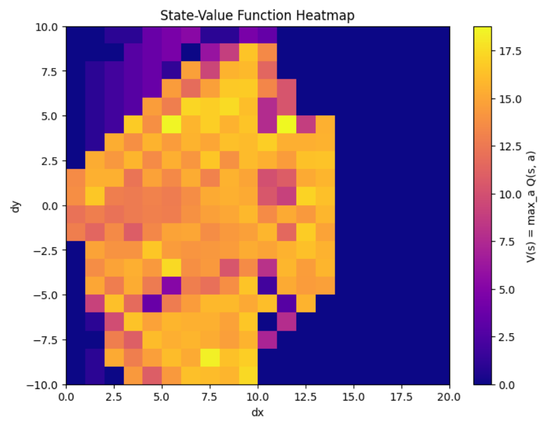
2.2 State-Value Function Plots



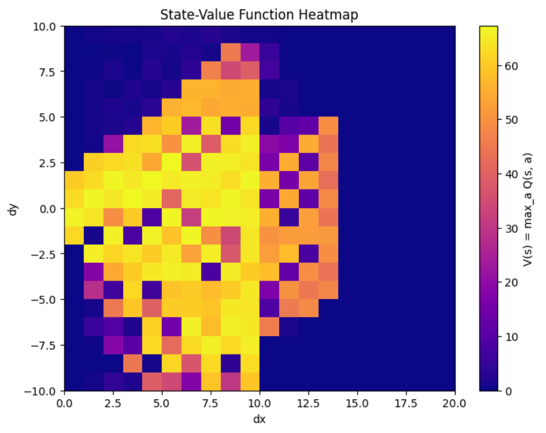
(a) Monte Carlo - State-Value Function



(b) SARSA - State-Value Function



(c) Monte Carlo - Heatmap



(d) SARSA - Heatmap

Fig. 2: Comparison of Monte Carlo and SARSA state-value function visualizations. The first row shows the state-value function as a 3D surface, while the second row presents heatmaps for better insight.

The 3D surface plots (a) and (b) show that both agents learn to assign higher values to states closer to the pipe gap, where action selection is most critical. However, Monte Carlo displays a smoother and more consistent value surface, suggesting more stable learning. This is confirmed by the heatmaps (c) and (d), where Monte Carlo's value function is more uniform and shows less variance. The state-value functions of the 2 agents remain very similar in terms of shape. The more we get close to the pipe gap, the more high value states are concentrated at the horizontal center. The agent learns to maximize its state value by remaining vertically centered while approaching the next obstacle.

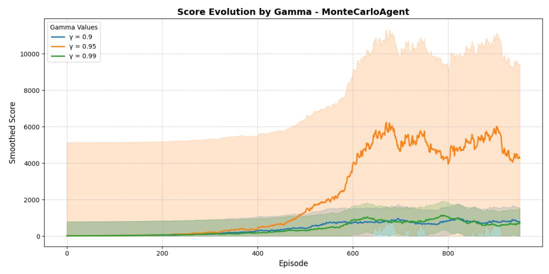
2.3 Best model saving strategy

The following graph 1 illustrates how scores vary from episode to episode as the Q-table is updated. So to deal with this variability, each episode, if the agent exceeds the previous highest score, it is subjected to a multi-game evaluation (n_game_validation=50) to ensure that this improvement is not due to luck. If the average score of this evaluation exceeds the best previously recorded average score, then the model is saved, ensuring that only truly successful policies are retained.

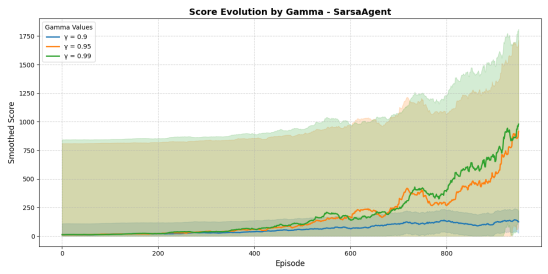
Thanks to this backup strategy, it's possible to save models that perform very well. For example, over just 1000 episodes, it is possible to obtain a Monte Carlo agent with an average score over 100 games of over 10000 (since we consider the game is finished after a score of 10000) with $\epsilon = 0$(check model in *./models/best_mc_agent.pth*) and a SARSA agent with an average score of over 5703 (*./models/best_sarsa_agent.pth*).

2.4 Influence of parameters gamma and epsilon decay

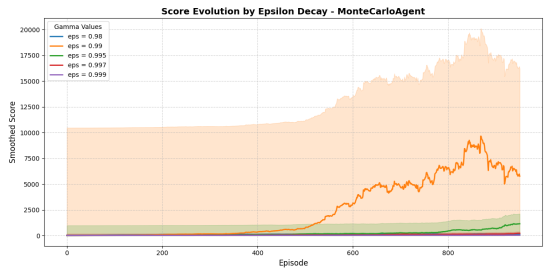
We choose to measure the impact of the gamma epsilon decay hyperparameters, since they are common to both agents. To measure their influence, we simply plot the agents' scores over 1000 episodes as a function of the value of each parameter.



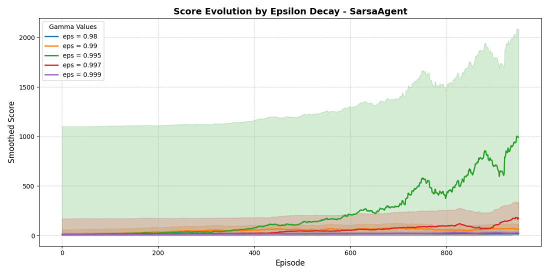
(a) Monte Carlo - Gamma influence



(b) SARSA - Gamma influence



(c) Monte Carlo - Epsilon Decay influence



(d) SARSA - Epsilon Decay influence

Fig. 3: Comparison of Monte Carlo and SARSA influence of core parameters Gamma and Epsilon Decay

The influence of the gamma and epsilon decay parameters is quite obvious. In the case of Monte Carlo, the best performances are obtained with $\gamma = 0.95$ and $\epsilon_{decay} = 0.99$, showing a preference for prolonged exploration. SARSA, on the other hand, achieves better scores with $\gamma = 0.99$ and $\epsilon_{decay} = 0.995$, relying on longer-term rewards.

2.5 Influence of environment configuration

In order to compare the influence of the environment configuration, we set a seed so that the generated TFB game is the same for the 5 different configurations we test, otherwise comparisons would be irrelevant, since a difficult part could be generated with an easy configuration, distorting the interpretation.

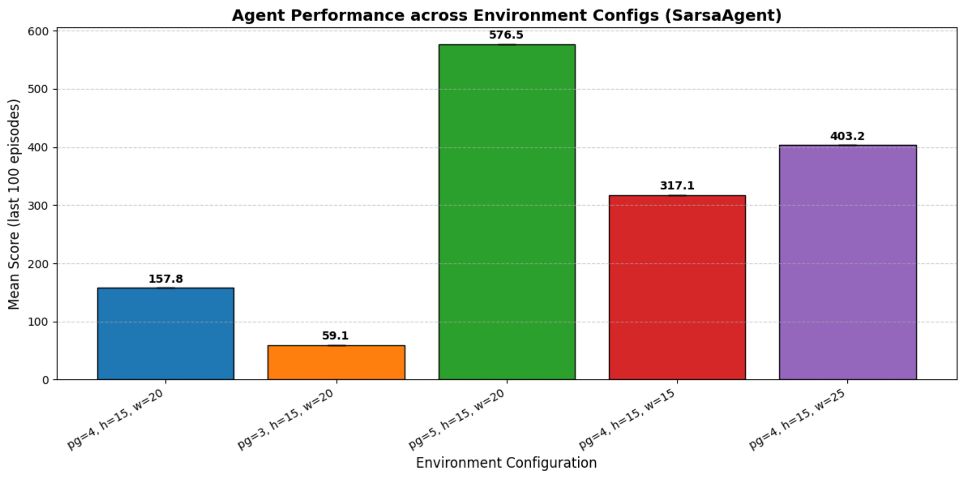


Fig. 4: Perfomances of SARSA agent in function of the environment

Unsurprisingly, compared with the base configuration *pipe gap=4, h=15, w=20*, the agent performs better when the pipe gap is large, with a mean score of 576 versus only 157 for the base configuration. What's more, the agent also performs better with a wider gap (mean score of 403), as the environment is wider and the agent can anticipate better.

2.6 Limitations of TFB and generalization to the original game

The two versions of the Text Flappy Bird (TFB) environment mainly differ in the way they represent the state space. The `TextFlappyBirdEnvSimple` environment provides a compact, low-dimensional observation consisting of the horizontal and vertical distances to the center of the next pipe gap. This makes it highly suitable for tabular reinforcement learning methods. On the other hand, the `TextFlappyBirdEnvScreen` environment returns a full 2D grid-like observation of the game screen, which includes all visible elements of the environment.

The main limitation of the simple version is its lack of rich environmental context, which may hinder generalization to more complex or visual environments. Conversely, while the screen-based version is more expressive and closer to real-game settings, it results in a high-dimensional observation space that cannot be handled effectively with basic algorithms like Monte Carlo or SARSA without incorporating function approximation methods such as Deep Q Learning for instance.

3 Conclusion

This project studied and compared the performance of 2 leading algorithms in LR: SARSA and Monte Carlo. The Monte Carlo algorithm proved to be much more efficient than SARSA once the right hyperparameters had been found. Nevertheless, despite differences in performance, the 2.2 section shows that training the 2 agents converges them towards a very similar state-value function. Finally, the 2.4 section shows the extent to which hyperparameters can influence the agents' overall performance, confirming the need for an in-depth search for the optimal hyperparameters presented here 1.

References

1. Stergios Christodoulidis. Text flappy bird gym. `https://gitlab-research.centralesupelec.fr/stergios.christodoulidis/text-flappy-bird-gym`, 2024. Accessed: 2025-03-23.
2. Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, 2nd edition, 2018.
3. Talendar. Flappy bird gym. `https://github.com/Talendar/flappy-bird-gym`, 2023. Accessed: 2025-03-23.