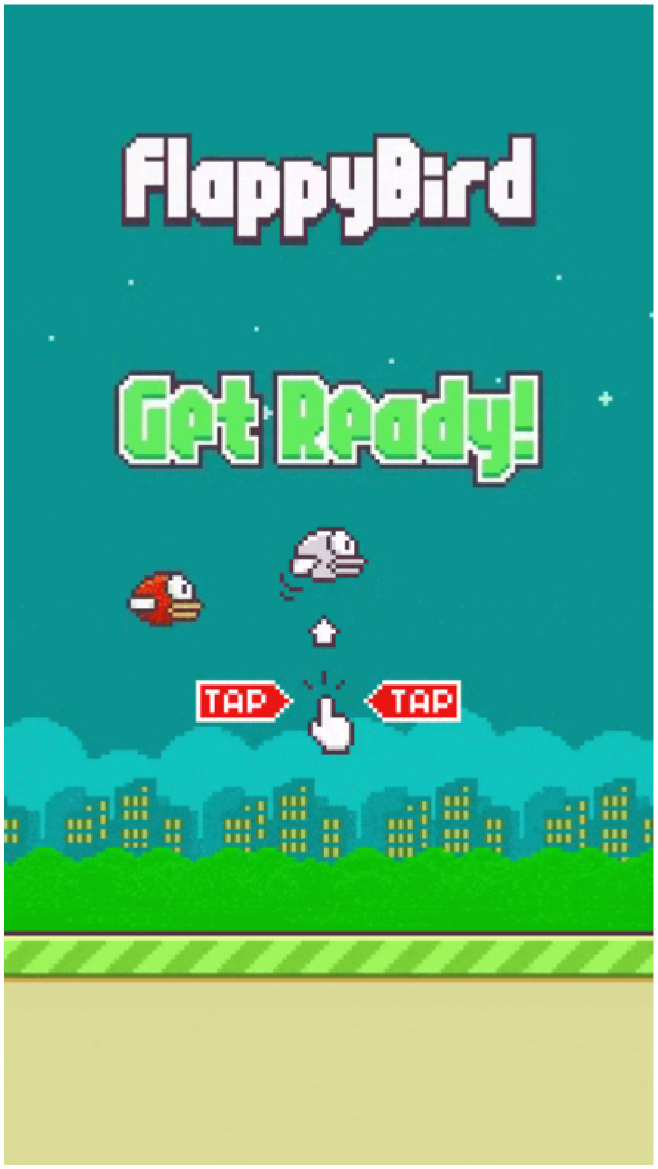
Reinforcement Learning Individual Assignment

Galdwin Leduc Msc AI

March 31, 2025

GitHub Individual Assignment Repository



Introduction

The intent of this study is to compare the performance of two reinforcement learning agents, Monte Carlo and Sarsa, on a Flappy Bird environment. In my case I even implemented 3 agents, the third being a more complex version of the Monte Carlo: Monte Carlo Reinforced. The implementation of these agents follows standard reinforcement learning principles while incorporating specific tuning techniques to improve the performance. This study examines their sensitivity to hyperparameters, convergence behavior, and overall performance.

1 Choice of environment

For this project we had a choice between two versions of flappy bird: TextFlappyBirdEnvSimple and TextFlappyBirdEnvScreen. The two versions of the Text-Based Flappy Bird (TFB) environment differ primarily in their state representations.

Flappy Bird Simple (Used in the Study)

Flappy Bird Simple provides a simplified, numerical observation space consisting of only two features:

- The horizontal distance between the bird and the next pipe.
- The vertical distance from the bird to the center of the pipe gap.

This environment utilizes a **discrete observation space**, making it more structured and easier for classical tabular reinforcement learning algorithms, such as Monte Carlo and Sarsa, to operate effectively. The simplicity of this representation also allows for faster training due to the reduced complexity of the state space.

In this environment, the agent receives reward signals based solely on survival, with no additional shaping rewards. This focus on survival makes it suitable for the study's objective, which is to apply and evaluate basic reinforcement learning methods.

Flappy Bird Screen (Not Used in the Study)

In contrast, *Flappy Bird Screen* provides a full-screen representation of the game state as a matrix, where each element contains pixel-like numerical values that correspond to the bird, pipes, and background. The observation space in this environment is much larger and more complex, making it more suitable for modern deep reinforcement learning approaches, such as deep Q-networks (DQN), which can process continuous inputs.

The environment utilizes a **continuous observation space**, which is significantly more complex than the discrete space used in the simple version. This allows for a more realistic representation of the game but would be better for more advanced function approximation methods, such as neural networks, for the agent to learn effectively. As a result, training in this environment is slower and demands more computational resources.

Conclusion for environment choice

Since this study focuses on classical tabular RL methods, specifically Monte Carlo and Sarsa, the *Flappy Bird Simple* environment was a better option to have more time to test more things and do more analyses.

2 Agents

In this part, we will go through the presentation of the different agents that I implemented and their parametrization. An important aspect for the fine-tuning of these models was the average reward, which served as the key metric for evaluation.

2.1 Monte Carlo Basic

This Monte Carlo Agent is a basic implementation of the Monte Carlo method for reinforcement learning. In this use case, the agent work as follow:

- **Learning Process**: Q-values are updated after each episode based on the cumulative reward.
- **Exploration Strategy**: The agent follows an epsilon-greedy policy, with epsilon decreasing over time to shift from exploration to exploitation.
- **Reward Handling**: A discount factor is applied to prioritize immediate rewards while considering future outcomes.

In the agent I implemented, the hyperparameters were as follows.

- ***gamma*:** Discount factor for future rewards, balancing long-term and short-term goals.

- ***epsilon_start*:** Initial value of epsilon, determining the exploration rate at the beginning.

- ***epsilon_end*:** Final value of epsilon, ensuring a minimum exploration rate at the end of training.

- ***epsilon_decay*:** Rate at which epsilon decays over time, gradually shifting from exploration to exploitation.

- ***alpha_start*:** Initial learning rate, controlling the size of updates to Q-values.

- ***alpha_decay*:** Rate at which the learning rate decays over time, thus influencing the convergence speed of Q-value updates.

After fine-tuning the model using the average reward as the evaluation metric, the following hyperparameters were determined to produce the best results for the agent:

Hyperparameter	**Value**
gamma	0.9
epsilon_start	1.0
epsilon_end	0.05
epsilon_decay	0.995
alpha_start	0.1
alpha_decay	0.999

Table 1: Best Hyperparameters for the Monte Carlo Basic Agent

Correlation analysis (Figure 1) highlights *alpha_start*, *epsilon_start*, and *alpha_decay* as positively linked to performance, while a slight negative correlation is observed with *epsilon_end*. Most hyperparameters remain largely uncorrelated, supporting independent tuning.

2.2 Monte Carlo Reinforced Agent

This agent implements the Reinforced algorithm using a neural policy network trained via Monte Carlo returns. The learning process is as follows:

- **Policy Learning**: A feedforward neural network maps states to action probabilities, optimized using policy gradient updates.
- **Stochastic Action Selection**: Actions are sampled from a softmax distribution over outputs.
- **Return Computation**: Discounted returns are computed backward and standardized to stabilize training.

The tunable hyperparameters for this agent include:

- ***lr*:** Learning rate for the Adam optimizer.

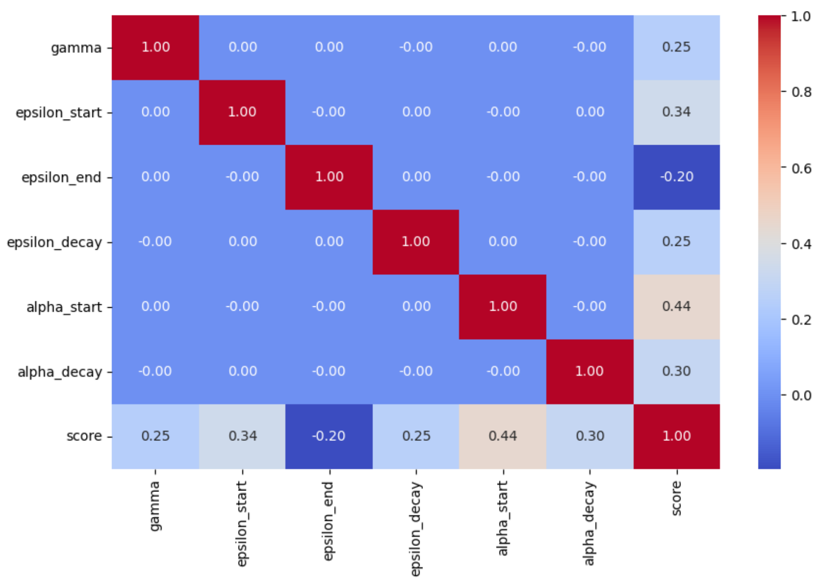


Figure 1: Correlation Matrix of Hyperparameters and Monte Carlo Basic Score

- ***gamma*:** Discount factor for return computation.
- ***smoothing_factor*:** Exponential smoothing for tracking performance trends.

Following grid search optimization based on average reward, the best-performing parameters were:

Hyperparameter	**Value**
lr	0.001
gamma	0.8
smoothing_factor	0.99

Table 2: Best Hyperparameters for the Monte Carlo Reinforced Agent

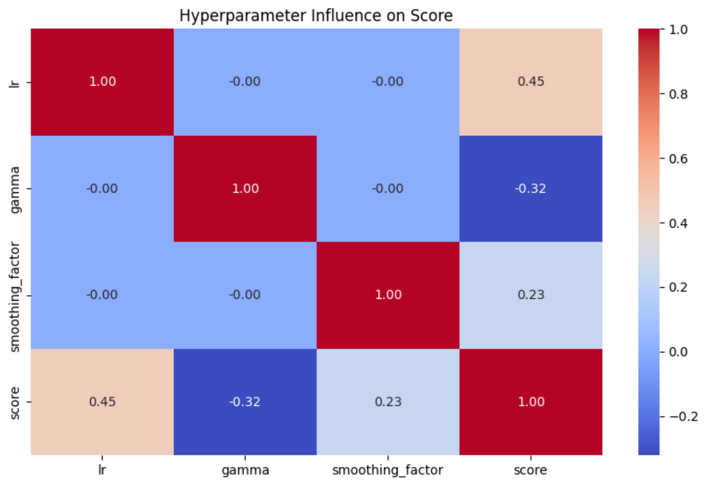


Figure 2: Correlation Matrix of Hyperparameters and Monte Carlo Reinforced Score

Correlation analysis (Figure 2) indicates that *lr* exhibits a moderate positive correlation with performance, while *gamma* has a weak negative correlation. The *smoothing_factor* shows a slight positive correlation with performance. Most hyperparameters appear to be weakly correlated with each other, consequently suggesting that they can be optimized independently to some extent.

2.3 SARSA Agent

This agent implements the on-policy SARSA (State-Action-Reward-State-Action) algorithm, using tabular Q-learning to iteratively update action-value estimates. Its behavior can be described as follows:

- **Policy**: Epsilon-greedy strategy balances exploration and exploitation during action selection.
- **Q-Value Updates**: Action values are updated incrementally using the SARSA update rule based on the agent's actual policy.
- **State Representation**: A dictionary-based Q-table is used, dynamically expanded as new states are encountered.

The agent was tuned over the following hyperparameters:

- *epsilon*: Exploration rate controlling the probability of random action selection.

- *alpha*: Learning rate determining the size of Q-value updates.

- *gamma*: Discount factor balancing immediate and future rewards.

The best-performing configuration after grid search (based on average reward) was:

Hyperparameter	**Value**
epsilon	0.1
alpha	0.5
gamma	0.99

Table 3: Best Hyperparameters for the SARSA Agent

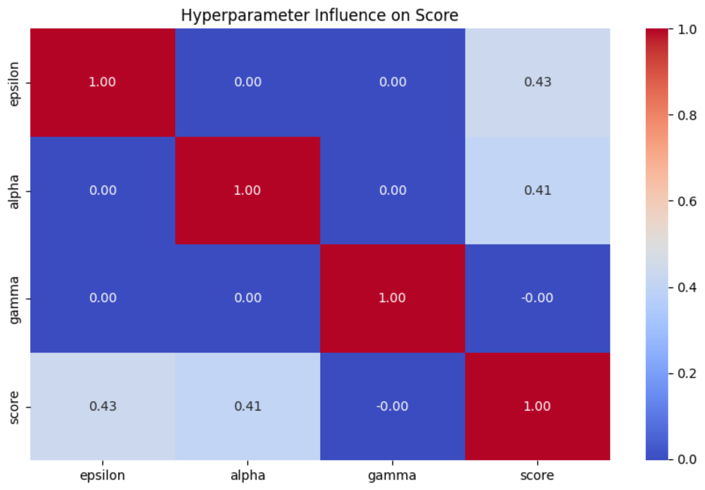


Figure 3: Correlation Matrix of Hyperparameters and Agent Score

As seen in Figure 3, *epsilon* and *alpha* show weak to moderate positive correlations with agent performance, indicating that exploration and learning rate have some influence on the final score. *Gamma*, however, exhibits no significant correlation, thus suggesting that discounting future rewards does not strongly impact performance in this setting.

3 Performance Analysis of Fine-tuned Agents

To evaluate the effectiveness of the fine-tuned agents, we analyze their training sensitivity and reward history using two key visualizations: a box plot capturing sensitivity and a curve plot illustrating training progress.

3.1 Training Sensitivity Analysis

The box plot in Figure 4 presents the variation in average rewards across different runs for each agent. The spread of values reflects each agent's stability and robustness to hyperparameter tuning. The Monte Carlo Reinforced agent exhibits a wider range, indicating higher variance but also potential for superior performance. The SARSA agent shows more consistency but lower median rewards.

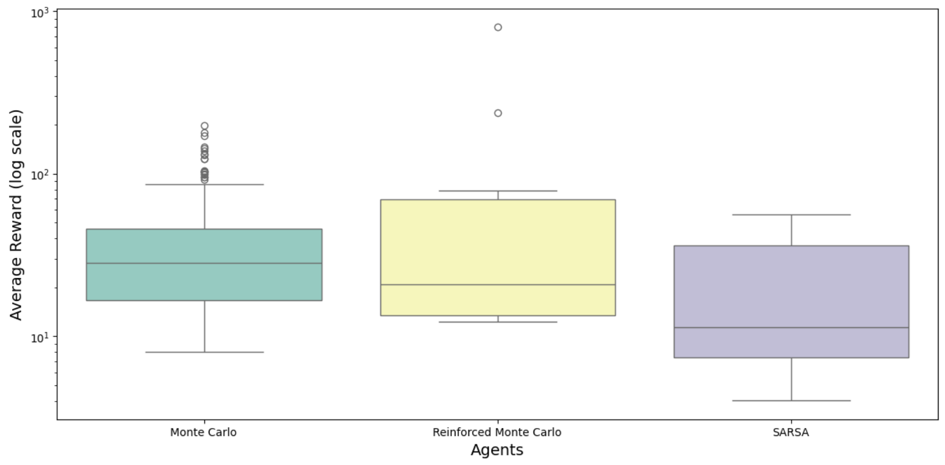


Figure 4: Sensitivity analysis of fine-tuned agents through hyperparameter variations.

3.2 Reward Progression During Training

Figure 5 illustrates the cumulative reward progression over training episodes for each agent. The Monte Carlo Reinforced agent demonstrates exponential growth in rewards, requiring early termination to prevent infinite training. The Monte Carlo Basic agent follows a steady improvement, whereas SARSA, despite its consistency, lags behind in long-term performance.

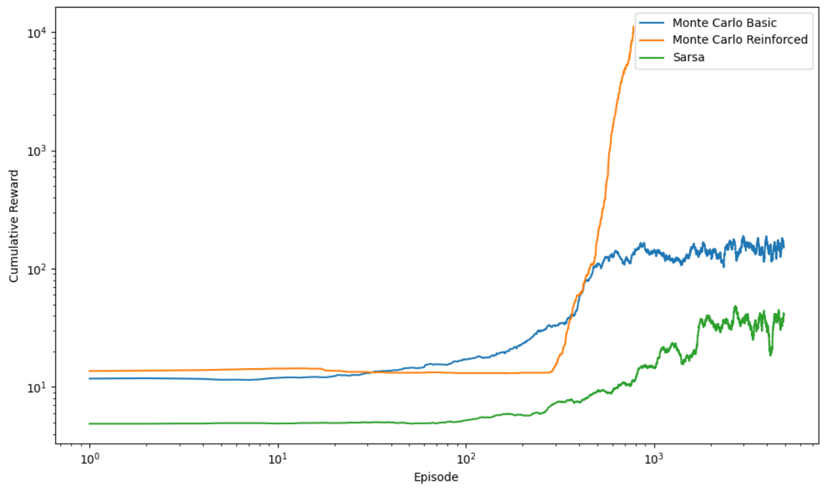


Figure 5: Reward progression during training for fine-tuned agents.

4 Conclusion

In this study we have compared the performance of Monte Carlo Basic, Monte Carlo Reinforced, and SARSA agents in a reinforcement learning environment. Grid search optimization helped fine-tune hyperparameters for optimal learning. Monte Carlo Reinforced emerged as the most effective agent, exhibiting rapid learning but requiring controlled training termination. Monte Carlo Basic provided steady improvements, whereas SARSA, though consistent, underperformed in the long run.

If we wanted to apply these agents to the original Flappy Bird environment, modifications in state representation and action space would be necessary to align with the game's mechanics. However, the fundamental learning principles remain applicable.

If we wanted to deploy a trained agent on a different configuration (e.g., height=15, width=20, pipe gap=4), performance might downgrade if the new setup alters state transitions and reward dynamics. To maintain effectiveness, transfer learning or domain adaptation techniques could be applied.