Flappy Bird

Jatin Singh

Ms Data Science and Business Analytics – ESSEC x CentraleSupelec

Github: https://github.com/JatinSingh007/Flappy-Bird-RL/blob/main/RL_FlappyBird.ipynb

Abstract: This project aims to develop and compare two reinforcement learning agents that learn to play Flappy Bird: SARSA (λ) and Monte Carlo. Our goals are to track their rates of learning, pinpoint their shortcomings, and ascertain which approach more thoroughly understands the game strategy.

Keywords: Flappy Bird Game, Reinforcement Learning, Monte Carlo, Sarsa(λ)

1. Introduction and Motivation

1.1. Introduction

The task for this Reinforcement Learning assignment is to apply two different agents, Monte Carlo and SARSA (λ), to play Flappy Bird by using Reinforcement Learning techniques. The aim of this study is to conduct a comparative analysis of these agents by analysing their policies, state-value functions, and performance in the Flappy Bird game framework. Relevant metrics will be utilised for evaluation. The 'TextFlappyBird-v0' environment, a text-based emulation of the real game that offers distance information from approaching pipes, will be used for testing. The analysis's conclusions ought to be expanded upon to speculate on how well they would function in the more intricate 'TextFlappyBird-screen-v0' environment, which provides a full-screen visualisation. Comprehensive details on the issue set, such as the nature of the environment, the state and reward systems, and the actions of the agents.

2. Execution

2.1. Agent Selection

The Monte Carlo and SARSA (λ) agents were developed to utilize their respective reinforcement learning strategies, capitalizing on learning from each episode's interaction rather than only at the conclusion. This continuous learning is vital in games like Flappy Bird, where episodes may run long, and delaying learning until the end, as Sutton & Barto note, could be inefficient [1]. Both agents learn in an off-policy manner, adapting from ongoing experiences within the game. Although a perfect Flappy Bird policy may be difficult to predict in advance, these agents are designed to actively learn their tactics from their surroundings, even though there is a chance that they won't establish complete policies for states that haven't been encountered. Although SARSA (λ) updates value estimations based on both immediate and future projected rewards, Monte Carlo concentrates on learning from whole episodes. This differentiation may result in intriguing variations in their training-induced performance and learning.

2.2. Implementation

Arbitrary hyperparameters were first established for the SARSA (λ) and Monte Carlo agents for the Flappy Bird game by taking typical values from lab settings. The intention was to begin with a fundamental comprehension of the actions of the agents within the gaming environment. Later in training, I found that a fixed epsilon led to a less-than-ideal exploration-exploitation balance. In order to tackle this issue, I incorporated an epsilon-decay mechanism, which enabled the agents to transition from exploration to exploitation as they increasingly approached the ideal course of action. Optimising the hyperparameters of each agent was essential for achieving the best possible performance and convergence rate. The best values for α (alpha) = 0.2, ε (epsilon) = 0.05, γ (gamma) = 1, and λ (lambda) = 0.3 were discovered for the SARSA (λ) agent. The optimal outcomes for the Monte Carlo agent were found when γ (gamma) = 0.9 and ε (epsilon) = 0.1 were used. After a great deal of testing to determine the best combinations for every method, these parameters were developed. Owing to the high processing costs, especially when the agents started to converge, I established a 1500 point maximum score threshold at which the agent would terminate in the first training cycle. More practical training sessions were made possible by this. In order to preserve a controlled setting for comparing the two agents, the total number of episodes was limited to 10,000. After determining the ideal hyperparameters, I trained the agents again, this time with a higher maximum score limit of 7,000 points and a longer episode count of 15,000, in order to thoroughly train them and assess their capacity for long-term learning. To guarantee that the agents' taught procedures will withstand the unpredictability of the Flappy Bird game, a vigorous training programme was required.

3. Model Evaluation and Comparison

Three primary elements were compared between the SARSA (λ) and Monte Carlo agents in the Flappy Bird gaming environment: generalisation ability, learning progression as shown by rewards and scores over training episodes, and state-value functions.

3.1. Training and Testing

During training, the rewards and scores captured for the Monte Carlo agent (Figs. 3) show a positive trend in learning, with variability in rewards indicating ongoing exploration. This is further evidenced by the training statistics (Fig. 3), which show an average training score slightly higher than SARSA (λ), a higher median score, and a lesser maximum score, pointing towards a more consistent but less explorative approach.

3.2. State-Value Function

The state-value function for the Monte Carlo agent (Fig. 1) in the Appendix displays a distinct dichotomy of state values, with certain state regions consistently appraised as high-value and others as low. This suggests a definitive policy developed by the Monte Carlo method, which focuses on the cumulative reward structure and capitalizes on successful episode completions to inform its value estimations. In contrast, the SARSA (λ) agent's state-value function (Fig. 1) in the Appendix shows a more complex landscape with numerous local maxima and minima, indicative of a finer-grained policy that takes into account both immediate rewards and future potential rewards.

3.3. Convergence Time and Learning Progression

Regarding convergence time, the Monte Carlo agent's performance, seen in Fig. 2, presents an overall upward trend in average rewards as training episodes increase, yet exhibits occasional declines in performance. This reflects the agent's episodic approach, where the outcomes of entire episodes inform the learning rather than individual steps within episodes. For the SARSA (λ) agent (Fig. 2), a sharper increase in average rewards is followed by significant volatility, which result from the agent's continuous approximation and policy updates at each step, causing more pronounced fluctuations in performance. Performance sweep graphs (Figs. 2 for Monte Carlo and for SARSA (λ)) demonstrate that the Monte Carlo agent reaches a smoother performance increase, potentially indicating a more stable learning over time compared to the more erratic improvements of the SARSA (λ) agent.

3.4. Generalization Capacity

The generalization graphs (Fig. 2 for Monte Carlo and for SARSA (λ)) illustrate how each agent adapts to changes in the game's environment, specifically the varying gap sizes between pipes. The Monte Carlo agent shows a steady improvement in scores as the gap size increases, reflecting its capability to adapt to and exploit wider gaps more effectively. This pattern contrasts with the SARSA (λ) agent, which also improves but demonstrates a sharper decline when faced with certain gap sizes, due to its real-time learning algorithm being more sensitive to changes in state dynamics.

3.5. Comparing Results

Upon comparison, the SARSA (λ) agent seems to learn more quickly but with greater variance in rewards and scores, which indicate a more explorative yet less stable policy. The Monte Carlo agent, with its approach to learning from complete episodes, shows more gradual improvements and potentially more robust policy formulation, with a less volatile performance profile. The individual analyses of the graphs for each agent reveal distinct learning behaviors and generalization capabilities, providing insights into the trade-offs between episodic learning and step-by-step policy updates. The Monte Carlo agent's methodical learning progression contrasts with the SARSA (λ) agent's more dynamic adaptation, each with its strengths in the Flappy Bird environment.

4 Agents use in complex environments

4.1 Application in TextFlappyBird-screen-v0

An observation state space comprising of a tuple containing dx and dy from the pipe gap (in this case, pipe gap = 4, the second gap from the bottom) has been used to train the agent. The height and breadth of the screen are output as an array in this environment's state space. Although it should theoretically be able to train the agent directly on the space's array, doing so will probably be quite costly in terms of computation. The simplest workaround would be to retrieve the pipe's distance from the state space array. Creating an approximation of the space in a smaller state space is another potential method. This would necessitate changing the agents' classifications.

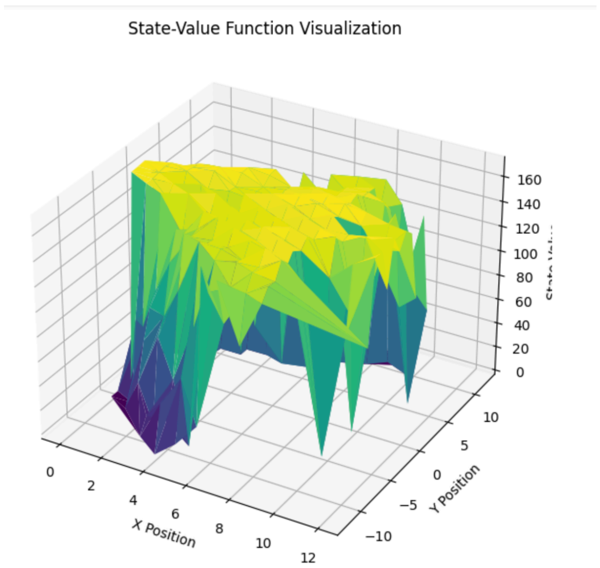
4.2 Utilisation in the Initial Flappy Bird Configuration

From a memory perspective, running the agents in the original Flappy Bird configuration can present challenges. Kernels might crash right away, and it would probably need quite powerful specs to function properly. Moreover, the problem lies in the continuous output of the observation (which is floats for the distance from the pipe) instead of a discrete state array location from the pipe. As a result, the agents would not be able to function in this environment as they currently do, and agents that can handle continuous observations in real-time or by approximation would need to be employed in their place.

5 References

1. R. S. Sutton, A. G. Barto: Reinforcement Learning: An Introduction, Cambridge. 2nd edn. The MIT Press, Cambridge MA (2018).

6 Appendix



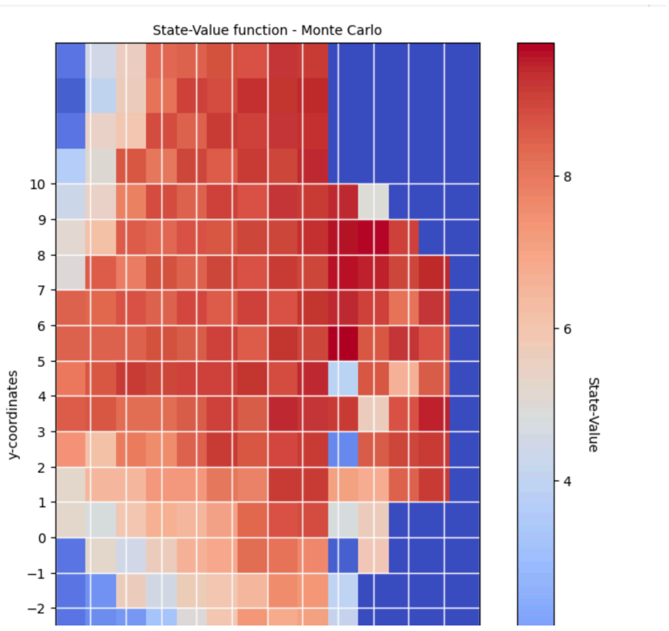
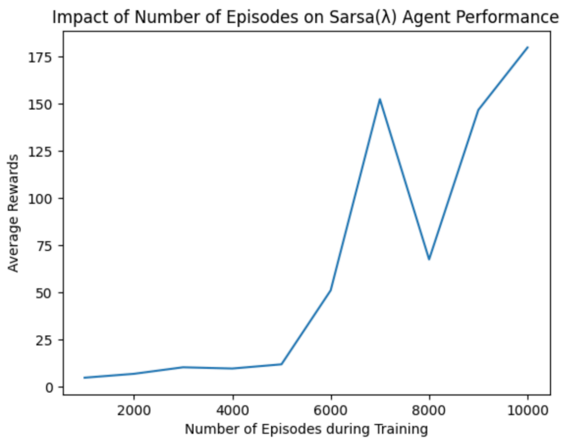
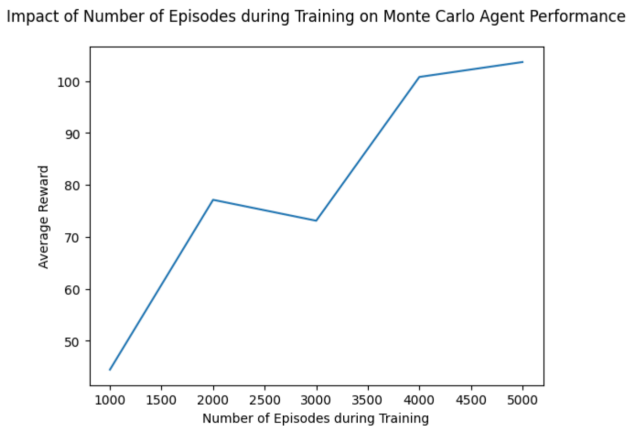


Fig 1. State Value Function Plots





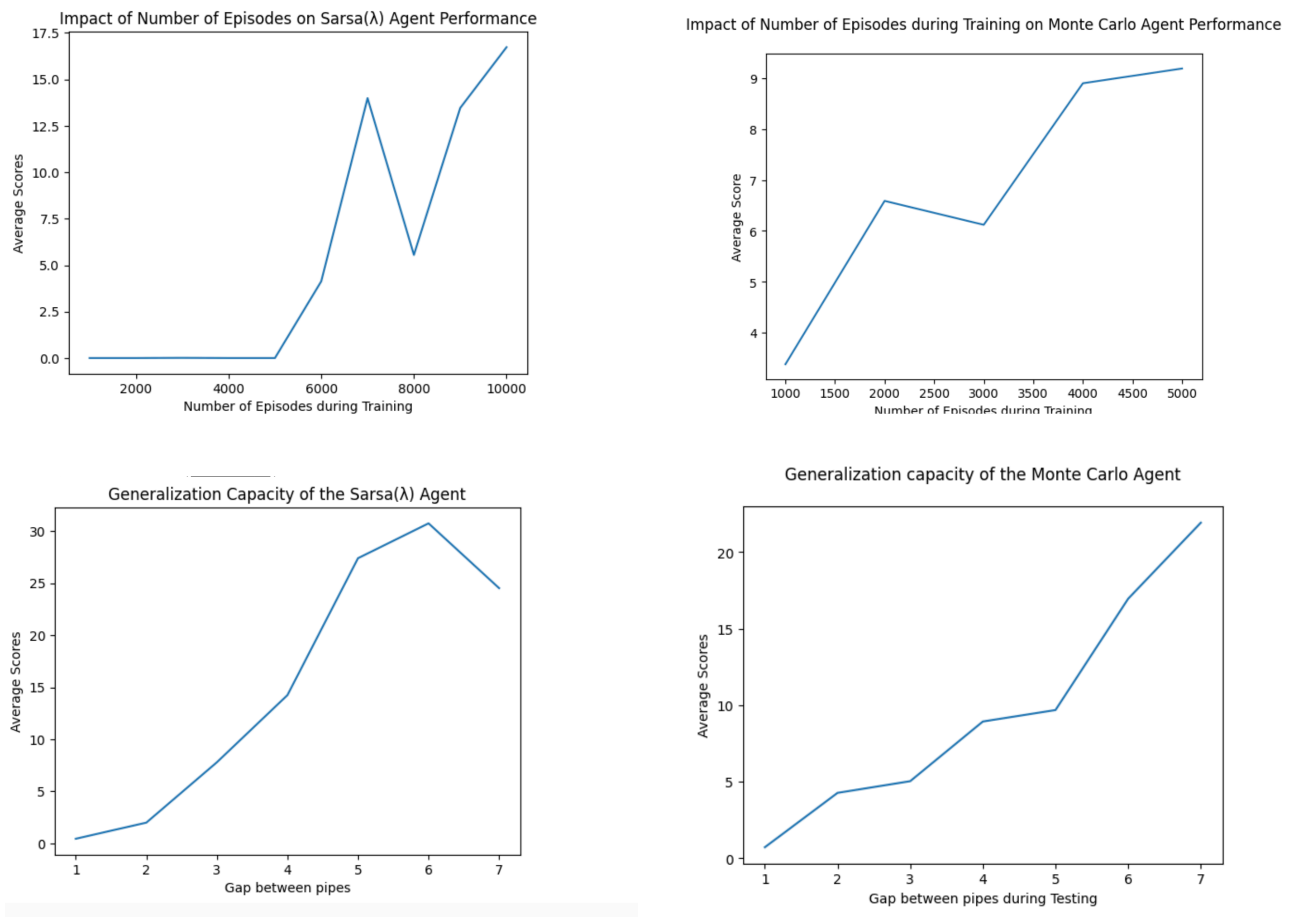
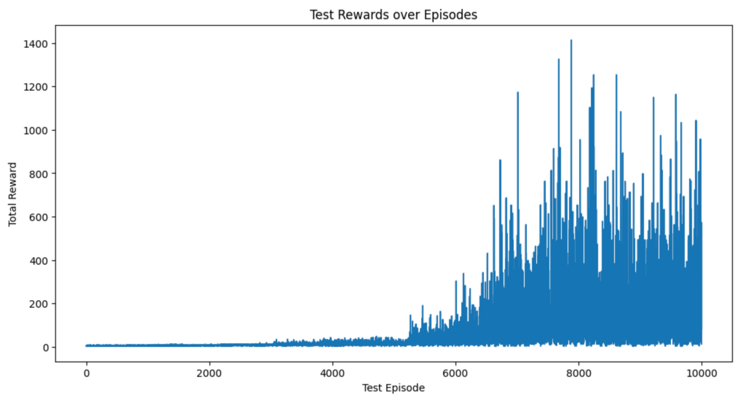
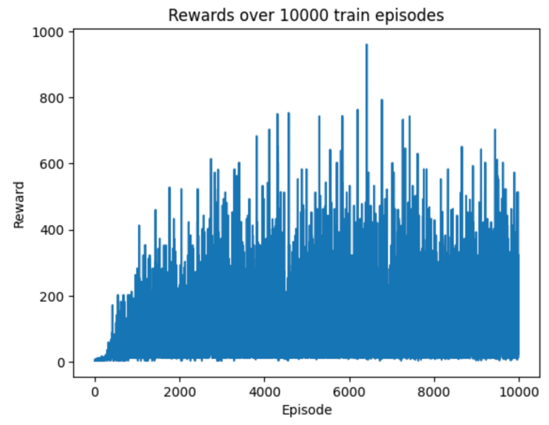


Fig 2. Performance and Parameter Sweep Graphs





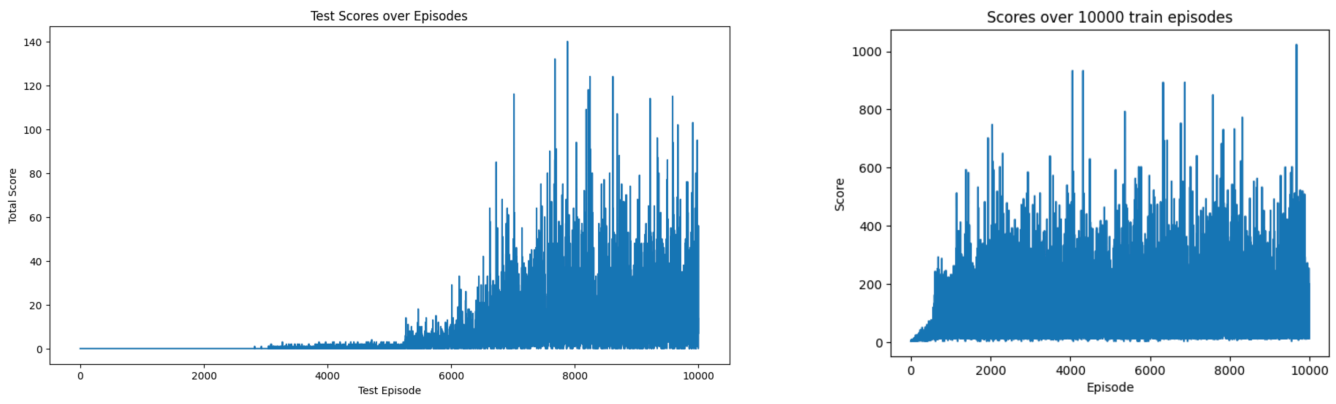


Fig 3. Graphs for the Optimal Models