# Text Flappy Bird with Basic Reinforcement Learning

Course: Basis Reinforcement Learning - EEE703116-1-1-24(N01)

Instructor: Vu Hoang Dieu

Student: Nguyen Huong Giang

November 6, 2024

# Contents

# Chapter 1

# Introduction

## 1.1 Overview

Flappy Bird is a game where the player controls a bird to avoid obstacles by tapping to make it jump up. Each time the bird passes through a pair of pipes, the player earns a point. The game ends when the bird hits a pipe or falls to the ground. This is an ideal reinforcement learning problem because it requires the agent to learn how to make decisions based on a continuously changing environment.

## 1.2 Problem Description

The main challenge here is to help the agent learn how to play Flappy Bird without human intervention. To achieve this, the agent needs to learn how to make real-time decisions based on the current state of the game, such as the bird's position, falling speed, and the distance to the upcoming pipes. The goal is to maximize the total score the agent achieves.

## 1.3 Objective

This report focuses on comparing different reinforcement learning algorithms to train an agent to play Flappy Bird:

- Q-learning: An off-policy method.
- Monte Carlo: Based on sample-based estimation methods.
- SARSA: An on-policy method.

## 1.4 Structure of Report

The report will be organized into the following main sections:

- Theoretical Background: Provides foundational knowledge on reinforcement learning and the algorithms used.

- Methodology: Describes problem formulation, algorithm selection, and solution implementation.
- Experimental Setup: Presents the environment setup, parameter configurations, and the experiments conducted.

# Chapter 2

# Theoretical Background

## 2.1 Core Concepts of Reinforcement Learning

Reinforcement Learning involves learning from experience to optimize an agent's actions in order to achieve the highest possible reward. The key concepts include:

- State: Information about the agent's current situation (e.g., The bird's position, the distance to the next pipe, the height of the pipe, and the bird's speed).
- Action: The actions are to either jump or not jump (maintain a free-fall state).
- Reward: The score the agent receives for taking a specific action in a particular state.
- Policy: The agent's strategy for choosing actions based on the state.

## 2.2 Overview of Algorithms

### 2.2.1 Q-Learning

Q-learning is an off-policy reinforcement learning algorithm that helps an agent learn how to estimate the Q-value for each state-action pair without needing to follow the current policy precisely.

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[ r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right]$$

where:

- $Q(s,\alpha)$ is the current Q-value for state $s$ and action $\alpha$.
- $\alpha$ is the learning rate, which determines how much new information overrides old information.
- $r$ is the reward received after taking action $a$ from state $s$,
- $\gamma$ is the discount factor, which balances the importance of immediate and future rewards.

- $s'$ is the next state resulting from taking action $\alpha$ in state $s$.
- $\alpha'$ is the next action chosen.

Q-learning aims to adjust the Q-values so that the agent can accurately predict the expected reward when taking an action from any given state. The agent always selects the action with the highest Q-value to optimize accumulated rewards.

### 2.2.2 SARSA

SARSA (State-Action-Reward-State-Action) is an on-policy algorithm, meaning the agent updates the Q-value based on the actions it actually takes according to the current policy.

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[ r + \gamma Q(s',a') - Q(s,a) \right]$$

where:

- $Q(s,\alpha)$ is the current Q-value for state $s$ and action $\alpha$.
- $\alpha$ is the learning rate, which determines the extent to which the current Q-value is adjusted.
- $r$ is the reward received after performing action $\alpha$ in state $s$.
- $\gamma$ is the discount factor, similar to that in Q-learning.
- $Q(s',\alpha)$ is The Q-value of the next state $s'$ when performing the next action $\alpha'$ that the agent chooses according to the current policy.

SARSA updates the Q-value based on the actual actions that the agent performs, which means that the Q-value is updated according to the agent's current policy, rather than an assumed optimal action as in Q-learning.

### 2.2.3 Monte Carlo Methods

Monte Carlo methods use complete playthroughs of the game to estimate the Q-value by averaging the accumulated rewards. The update rule for Monte Carlo is:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[ G_t - Q(s,a) \right]$$

where $G_t$ is the return (total reward) from time step $t$ to the end of the episode. The return is computed as:

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{T-t-1} \gamma^k r_{t+k+1}$$

## 2.3 Equations and Updates

### 2.3.1 Q-Learning Update Rule

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[ r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right]$$

### 2.3.2 SARSA Update Rule

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[ r + \gamma Q(s',a') - Q(s,a) \right]$$

### 2.3.3 Monte Carlo Update

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[ G_t - Q(s,a) \right]$$

where $G_t$ is the return (total reward) from time step $t$ to the end of the episode.

# Chapter 3

# Methodology

## 3.1 Problem Definition

The problem to solve is to train an agent to play the game Flappy Bird without human intervention. The agent will be set up to learn how to play the game by making decisions when encountering different game states.

Key elements in the problem definition:

- **State**: The states will include parameters such as the bird's x and y position, its current velocity, and the horizontal and vertical distances to the next obstacle pipe. These states provide the agent with enough information to make decisions.
- **Action**: The agent has only two choices of actions: jump (make the bird fly up) or do nothing (let the bird fall naturally).
- **Reward**:If the bird survives each frame, it will receive +1 reward point. If the bird hits a pipe or falls to the ground, the score will no longer increase, and the game session will end.
- **Objective of the Agent**: To find the optimal strategy (policy) for deciding actions in each state so that it can play the game for as long as possible and achieve the highest score.

## 3.2 Algorithm Selection

Since this is a Reinforcement Learning problem, the following algorithms are selected for comparison:

- **Q-learning**: A popular off-policy algorithm, suitable for simple problems and allows the agent to learn to optimize actions without needing to know the environment model.
- **SARSA**: An on-policy algorithm similar to Q-learning, suitable when the agent's behavior needs to accurately reflect what it is learning.

- **Monte Carlo Methods**:Suitable for environments where the model is incomplete or cannot be accurately predicted, as it relies on sampling from complete game sessions.

Criteria for Selecting Algorithms:

- **Ease of Implementation**: Some algorithms are easier to implement and suitable for simple problems, like Q-learning.
- **Learning Speed and Performance**: The selected algorithms differ in their approach and learning speed, so comparing them will help assess the overall effectiveness of each method.
- **Generalization Capability**:The agent needs the ability to make optimal decisions even in situations it has not encountered before.

## 3.3 Implementation

### 3.3.1 Step-by-Step Process

**Game Environment:**

- Create or use a simulated version of the Flappy Bird game. This environment must provide the necessary states for the agent and allow the agent to take actions (jump or do nothing).
- Define rules and rewards: The agent will receive a positive reward when passing through a pipe and a negative reward when colliding.

**Applying the Algorithms:**

- **Q-learning and SARSA**: Use a Q-table to store Q values for states and actions. As the game progresses, the agent updates these values based on the rewards received and the predicted value for the next state.
- **Monte Carlo**: Collect data from multiple complete game sessions and update the Q values based on the average reward from those sessions.

**Training the Agent:**

- Run the algorithms in the Flappy Bird environment multiple times so the agent learns how to play.
- Adjust parameters (learning rate, discount factor) to optimize learning speed and the agent's game performance..

**Evaluation and Performance Comparison:**

- After training, the agent will play the game without updating the Q-table or policy. The scores achieved will be recorded and compared to evaluate the performance of each algorithm.

### 3.3.2 Pseudo-code

The pseudo-code for the key algorithms is presented below:

**Q-Learning Pseudo-code**

```
Initialize Q(s, a) arbitrarily
for each episode:
    Initialize state s
    while not done:
        Choose action a using epsilon-greedy
        Take action a, observe reward r and next state s'
        Q(s, a) = Q(s, a) + alpha * (r + gamma * max(Q(s', a')) - Q(s, a))
        s = s'
```

**SARSA Pseudo-code**

```
Initialize Q(s, a) arbitrarily
for each episode:
    Initialize state s
    Choose action a using epsilon-greedy
    while not done:
        Take action a, observe reward r and next state s'
        Choose next action a' using epsilon-greedy
        Q(s, a) = Q(s, a) + alpha * (r + gamma * Q(s', a') - Q(s, a))
        s = s'
        a = a'
```

**Monte Carlo Pseudo-code**

```
Initialize Q(s, a) arbitrarily
for each episode:
    Generate episode using current policy
    for each state-action pair (s, a) in episode:
        G = total reward from that point to the end of episode
        Q(s, a) = Q(s, a) + alpha * (G - Q(s, a))
```

# Chapter 4

# Experimental Setup

## 4.1 Environment Setup

Set up a simulated Flappy Bird game environment to run the experiments:

- **Environment:** Designed to provide state inputs, receive action inputs from the agent, and return rewards. The environment also needs to be able to simulate game physics, including the movement of the bird and pipes.
- **Simulation Configuration:**
  - **Frame Rate:** Configured to adjust the difficulty level.
  - **Difficulty Options:** Adjust the gap between pipes and the speed of pipe movement to test the agent's adaptability.

## 4.2 Data

Data is generated during the agent's gameplay, including:

- **State-Action-Reward Triplets:** Each time the agent performs an action, the system records the current state, the action taken, and the reward received. This data is used to update the Q-table or policy values.
- **Experimental Results:** Record the agent's scores across multiple game sessions to analyze performance.

## 4.3 Hyperparameter Tuning

### 4.3.1 Q-Learning

- **Learning Rate ($\alpha$):** Determines the extent to which new updates affect the current values. A high value can lead to instability, while a low value will cause the agent to learn slowly.
- **Discount Factor ($\gamma$):** Determines the importance of future rewards. The closer the value is to 1, the more the agent will focus on long-term rewards.

- **Epsilon ($\epsilon$) in $\epsilon$-greedy Strategy:** Adjusts the agent's exploration level. Increasing epsilon encourages more exploration, while decreasing it allows the agent to exploit the learned policy.

### 4.3.2 SARSA

The same parameters as Q-learning, but the Q-value updates are based on the actual actions taken by the agent. This requires additional experimentation to optimize the on-policy training process.

### 4.3.3 Monte Carlo Methods

- **Number of Episodes:** To improve the accuracy of the Q-values, a large number of playthroughs is needed to collect sufficient samples. This may require the agent to play thousands of episodes to converge.

# Chapter 5

# Results

## 5.1 Q-Learning

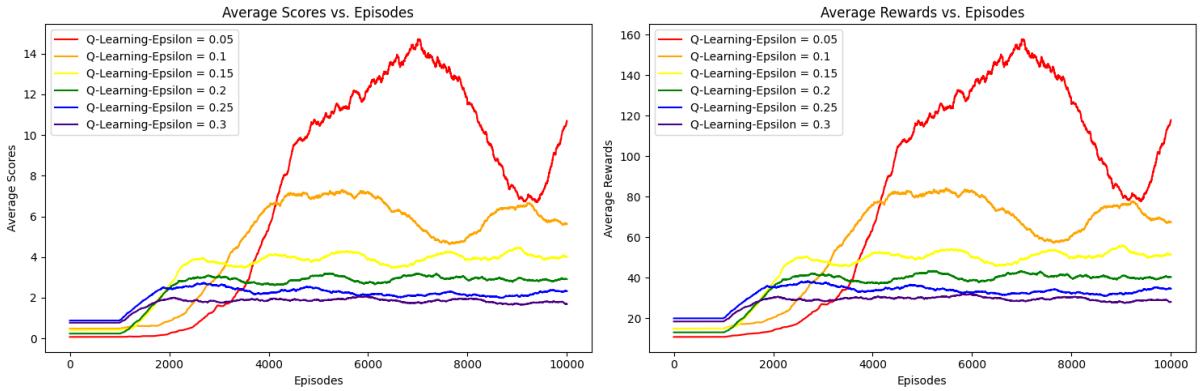


**Figure 5.1: Q-Learning with varying $\epsilon$**

Looking at the two plots, we can see the impact of different epsilon values on the average scores and average rewards in the Q-Learning model applied to the Text Flappy Bird Gym environment.

- **Left Plot (Average Scores):**
  - With smaller epsilon values (e.g., 0.05 and 0.1), the average scores increase more significantly and reach higher peaks compared to larger epsilon values. This indicates that the Q-Learning model with low epsilon tends to explore less and exploit more, leading to better performance when the Q-values have been optimized.
  - As epsilon increases (0.15, 0.2, 0.25, 0.3), the average scores are lower and tend to stabilize or increase slightly. Higher epsilon encourages the model to explore more, but in a complex environment like Flappy Bird, this can result in unstable behavior and prevent the model from achieving high scores as it does with lower epsilon values.
- **Right Plot (Average Rewards):**
  - The results are similar to the average scores plot. Lower epsilon values (0.05 and 0.1) lead to higher peak average rewards, while higher epsilon values have lower average rewards.

- Higher epsilon values encourage exploration and trying new actions, but this reduces performance because the model doesn't focus on the optimal actions it has learned.

- **Summary:**
  - Smaller epsilon values (0.05 and 0.1) tend to result in higher average scores and rewards because the model focuses on exploiting optimal actions.
  - Larger epsilon values lead to lower average scores and rewards due to more exploration, causing the model to exhibit suboptimal behavior in the Flappy Bird Gym environment.

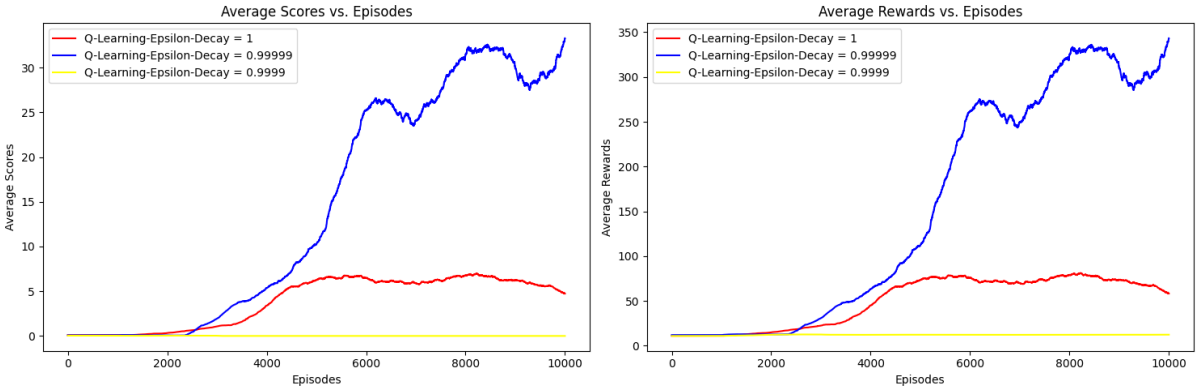


**Figure 5.2: Q-Learning with varying $\epsilon$ decay**

Looking at these two plots, we can analyze how different epsilon decay values affect the average scores and average rewards in the Q-Learning model applied to the Text Flappy Bird Gym environment.

- **Left Plot (Average Scores):**
  - With a high epsilon decay rate of 0.99999 (blue line), the average scores increase significantly, reaching higher peaks and remaining stable at higher values. This indicates that the model gradually reduces exploration, which helps it focus on exploiting optimal actions after sufficient exploration early on.
  - For the lower epsilon decay rate of 0.9999 (yellow line), the average score barely increases, indicating that the model continues exploring at a relatively high level and struggles to find a consistent optimal policy.
  - When epsilon decay is set to 1 (red line), there is no decay at all, so epsilon remains constant. The model doesn't reduce exploration over time, which results in a lower, more stable average score compared to when decay is applied.

- **Right Plot (Average Rewards):**
  - The pattern is similar to the average scores plot. With an epsilon decay of 0.99999, the average reward peaks at much higher values and maintains good performance over time, indicating effective learning.

- The decay rate of 0.9999 leads to minimal improvement in average rewards, suggesting that constant high exploration prevents the model from focusing on high-reward actions.
- For epsilon decay of 1 (no decay), average rewards remain low and stable, as the model continuously explores without exploiting learned optimal actions.

- **Summary:**
  - A high epsilon decay rate (0.99999) allows the model to balance exploration and exploitation effectively, resulting in much higher average scores and rewards.
  - A lower decay rate (0.9999) or no decay (1) leads to excessive exploration, preventing the model from exploiting learned strategies, resulting in lower average scores and rewards.

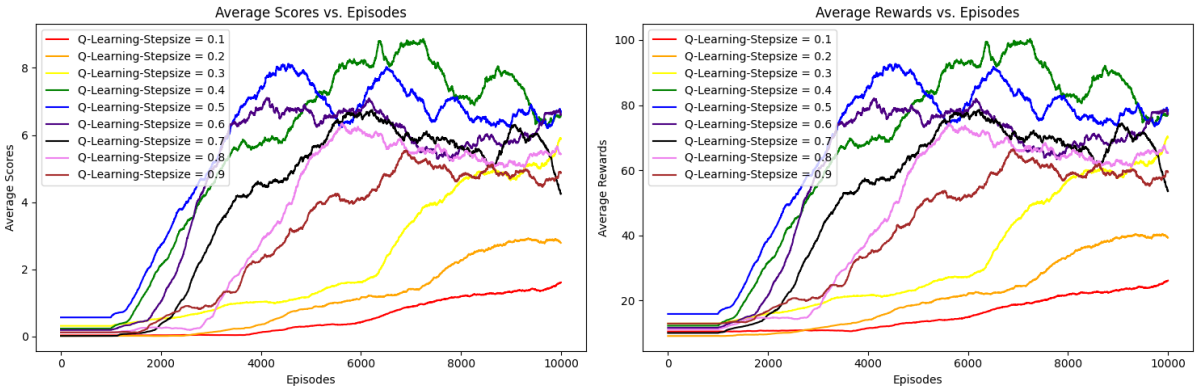


**Figure 5.3: Q-Learning with varying step size**

Looking at the two plots, we can see how different step sizes (learning rates) affect the average scores and average rewards in the Q-Learning model applied to the Text Flappy Bird Gym environment.

- **Left Plot (Average Scores):**
  - A moderate step size (0.3 to 0.5) tends to yield higher average scores, with the step size of 0.4 (green line) reaching the highest peaks and maintaining relatively stable performance. This suggests that with a well-balanced learning rate, the model effectively updates its Q-values without drastic fluctuations, leading to consistent learning and better performance.
  - When the step size is lower (0.1 and 0.2), the average scores improve but remain relatively low, indicating that the model is learning slowly. While it eventually reaches a stable policy, the slow learning rate prevents it from quickly achieving high scores.
  - For higher step sizes (0.6 to 0.9), especially at 0.9 (red line), the average scores tend to be lower. This is because a large step size causes the model to make large updates to the Q-values, which can lead to instability and prevent it from learning an optimal policy.

- **Right Plot (Average Rewards):**

- The patterns in average rewards are similar to those in average scores. Step sizes in the range of 0.3 to 0.5 achieve higher average rewards, with 0.4 showing the most consistent high performance.
- Lower step sizes (0.1 and 0.2) result in slow but steady improvements in average rewards, as the model requires more episodes to converge to a high-reward policy.
- High step sizes (0.6 to 0.9) show fluctuating or lower average rewards, with 0.9 again showing the least stability, indicating that excessive updates make it harder for the model to settle on effective strategies.

- **Summary:**
  - A moderate step size (0.3 to 0.5) provides the best balance, allowing the model to update its Q-values effectively, leading to higher and more stable average scores and rewards.
  - A low step size leads to slow learning, limiting performance improvements over time.
  - A high step size results in instability, making it difficult for the model to converge on an optimal policy, thus yielding lower scores and rewards.

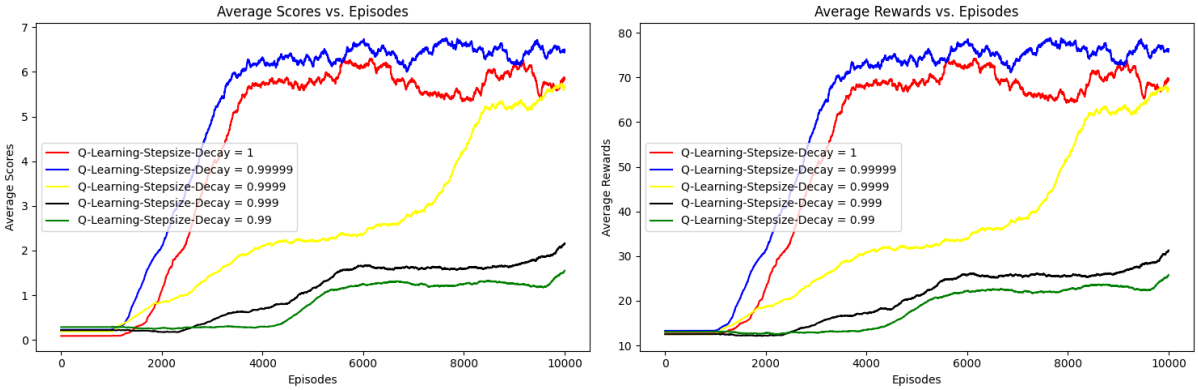


**Figure 5.4: Q-Learning with varying step size decay**

Based on the two plots showing the effect of step size decay values on the average scores and average rewards in the Q-Learning model applied to the Text Flappy Bird Gym environment, we can make the following observations:

- **Left Plot (Average Scores):**
  - Step size decay = 1 (red line): With the largest decay value, the model learns very quickly, reaching the highest average scores in a short amount of time. However, after many episodes, performance starts to fluctuate, indicating that updating the Q-values too quickly may lead to instability and make it harder for the model to converge.
  - Step size decay = 0.99999 (blue line): This value shows the best long-term performance. The model also learns quickly, achieving high scores rapidly and maintaining stability over most of the episodes, even surpassing the red line later on. This suggests a good balance between learning speed and stability.

- Step size decay = 0.9999 (yellow line): The model learns more slowly compared to the two larger decay values but still reaches relatively high average scores. Its performance is noticeably better than smaller decay values, although not as stable as the blue line.
- Step size decay = 0.999 (black line) and 0.99 (green line): These values lead to very slow learning. While the model improves over the episodes, it does not achieve high average scores compared to larger decay values. Especially with step size decay = 0.99, the learning rate is too slow, and the average scores improve very little over time.

- **Right Plot (Average Rewards):**
  - Step size decay = 1 (red line) and 0.99999 (blue line): These high decay values show the model achieving high average rewards quickly, with step size decay = 0.99999 being the most stable and best performer over the long term.
  - Step size decay = 0.9999 (yellow line): Similar to the average scores plot, this value reaches high average rewards, though it takes more episodes to become stable.
  - Step size decay = 0.999 (black line) and 0.99 (green line): These smaller decay values result in much lower average rewards and slow improvement. The slow learning rate prevents the model from quickly finding an optimal policy.

- **Summary:**
  - High step size decay values like 1 or 0.99999 allow the model to learn quickly and achieve high performance, but may become unstable over long periods.
  - Moderate step size decay like 0.9999 provides a good balance between learning speed and stability, leading to more consistent long-term performance.
  - Low step size decay values like 0.999 or 0.99 result in slow learning, limiting the model's ability to find an optimal policy in a shorter time frame.

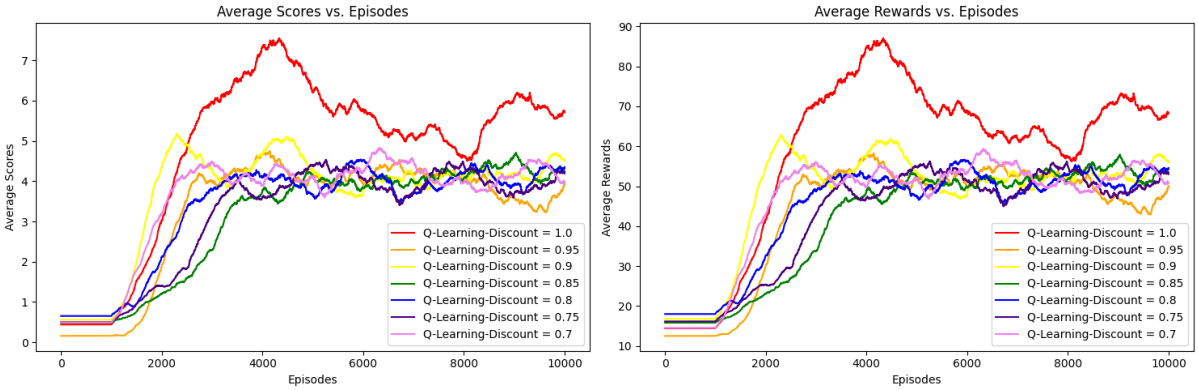


**Figure 5.5: Q-Learning with varying discount**

Based on the two plots showing the effect of discount values (gamma) on average scores and average rewards in the Q-Learning model applied to the Text Flappy Bird Gym environment, we can make the following observations:

- **Left Plot (Average Scores):**
  - Discount = 1.0 (red line): The highest discount value results in very strong initial performance. The model quickly achieves the highest average scores, peaking at over 7 points. However, after around 4,000 episodes, the average scores start to decline and fluctuate significantly, indicating instability in the long run.
  - Discount = 0.95 (orange line) and 0.9 (yellow line): These values also show good performance. Although the learning rate is slower than with discount = 1.0, both achieve relatively high average scores within a short time. In particular, discount = 0.9 seems to maintain more stability later on compared to discount = 1.0, though it does not reach the same peak.
  - Discount = 0.85 (green line) and 0.8 (blue line): These medium discount values lead to slower learning. The model gradually improves its scores, but neither achieves the high average scores seen with larger discount values. Discount = 0.85 performs slightly better than discount = 0.8, but the difference is not very large.
  - Discount = 0.75 (purple line) and 0.7 (pink line): These lower discount values result in slower learning and lower average scores. The model improves very slowly over episodes and fails to reach the high performance seen with larger discount values.
- **Right Plot (Average Rewards):**
  - Discount = 1.0 (red line): This value enables the model to achieve the highest average rewards after around 2,000 episodes, but there is significant fluctuation afterwards. Although it peaks at a high level, the rewards start to decrease, showing instability.
  - Discount = 0.95 (orange line) and 0.9 (yellow line): These values also yield high average rewards, with discount = 0.9 maintaining more consistent rewards without the sharp decline seen with discount = 1.0, indicating a better balance between learning speed and stability.
  - Discount = 0.85 (green line) and 0.8 (blue line): These values result in slower reward growth, but still achieve relatively good performance after many episodes. Discount = 0.85 performs slightly better than discount = 0.8 in terms of overall rewards.
  - Discount = 0.75 (purple line) and 0.7 (pink line): The lowest discount values show slow reward improvement and fail to reach the high levels of rewards seen with higher discount values. This suggests that lower discount values limit the model's learning ability.
- **Summary:**
  - High discount values (1.0 and 0.95) help the model learn quickly and achieve high scores and rewards, but may lead to instability in the long term.
  - Medium discount values (0.9 and 0.85) provide a better balance between learning performance and stability, with discount = 0.9 being the best choice for consistent scores and rewards.

– Low discount values (0.8 and below) result in slow learning and limit the model's ability to achieve high scores and rewards.

## 5.2 SARSA

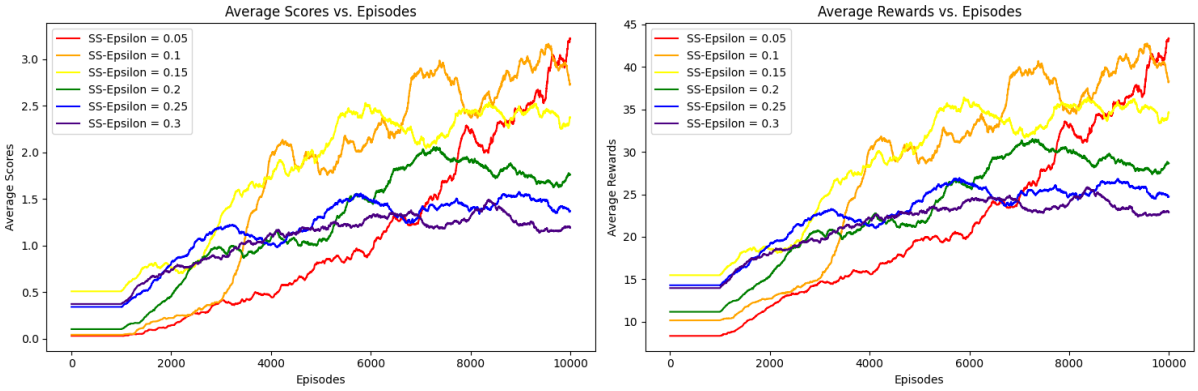


**Figure 5.6: SARSA with varying $\epsilon$**

Looking at these two plots, we can analyze how different epsilon (exploration rate) values affect the average scores and average rewards in the SARSA model applied to the Text Flappy Bird Gym environment.

- **Left Plot (Average Scores):**
  - With a low epsilon value of 0.05 (red line), the average scores are relatively high, especially in later episodes. This indicates that the model focuses heavily on exploitation, as it reduces exploration significantly, leading to a more stable and effective policy that maximizes scores over time.
  - Epsilon values around 0.1 (yellow line) and 0.15 (orange line) also yield high scores, though they show a bit more variability than epsilon = 0.05. These moderate exploration rates allow the model to explore occasionally while primarily focusing on exploiting learned strategies.
  - As epsilon increases (e.g., epsilon = 0.2 in green, epsilon = 0.25 in blue, and epsilon = 0.3 in purple), the average scores decrease. Higher epsilon values encourage more exploration, which can prevent the model from stabilizing on an optimal policy and achieving higher scores.
  - At the highest epsilon value of 0.3 (purple line), the model maintains a low average score throughout, suggesting that excessive exploration hinders the model's ability to focus on high-scoring strategies.
- **Right Plot (Average Rewards):**
  - The pattern in average rewards follows a similar trend to the average scores. With epsilon = 0.05 (red line), the average rewards peak at the highest values, indicating effective learning focused on maximizing rewards over time.
  - Epsilon values around 0.1 (yellow) and 0.15 (orange) also yield relatively high rewards, though with slightly more fluctuation. These values strike a balance between exploration and exploitation, allowing for good performance without excessive instability.

- Higher epsilon values (0.2, 0.25, and 0.3) yield lower average rewards. The model spends more time exploring rather than exploiting learned strategies, which limits its ability to focus on high-reward actions.
- The highest epsilon (0.3) results in the lowest average rewards, as the constant high exploration prevents the model from focusing on an effective strategy.

- **Summary:**
  - A low epsilon value (0.05) enables the model to prioritize exploitation, leading to higher and more stable average scores and rewards over time.
  - Moderate epsilon values (0.1 - 0.15) still allow the model to achieve relatively high scores and rewards, providing a good balance between exploration and exploitation.
  - High epsilon values (0.2 - 0.3) lead to excessive exploration, which prevents the model from exploiting learned strategies, resulting in lower average scores and rewards.

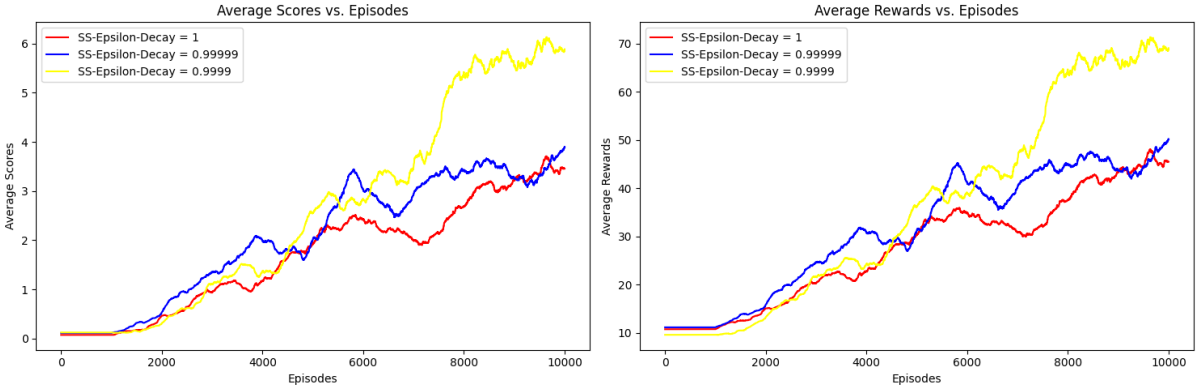


**Figure 5.7: SARSA with varying $\epsilon$ decay**

Looking at these two plots, we can analyze how different epsilon decay values affect the average scores and average rewards in the SARSA model applied to the Text Flappy Bird Gym environment.

- **Left Plot (Average Scores):**
  - With an epsilon decay rate of 0.9999 (yellow line), the average scores reach the highest values over time. This indicates that the model gradually reduces exploration, allowing it to focus more on exploiting optimal actions after sufficient exploration early on, leading to higher scores.
  - For an epsilon decay rate of 0.99999 (blue line), the model also performs well, though with slightly lower scores compared to the 0.9999 decay rate. This suggests that with a slower reduction in exploration, the model continues exploring for a longer period, which might prevent it from fully focusing on exploitation.
  - When epsilon decay is set to 1 (red line), there is no decay at all, meaning epsilon remains constant. The model does not reduce exploration over time, resulting in lower and more stable average scores, as it continues to explore rather than fully exploiting learned strategies.

- **Right Plot (Average Rewards):**
  - The pattern in average rewards is similar to that of average scores. With an epsilon decay of 0.9999 (yellow line), the model achieves the highest average rewards, indicating effective learning as it balances exploration and exploitation over time.
  - An epsilon decay of 0.99999 (blue line) results in moderate rewards, with performance slightly lower than the 0.9999 decay rate. The slower decay keeps the model in exploration mode longer, which impacts its ability to achieve consistently high rewards.
  - Without decay (epsilon decay = 1, red line), the average rewards are lower and remain stable. The constant exploration prevents the model from focusing on high-reward actions, leading to overall lower rewards.
- **Summary:**
  - A high epsilon decay rate (0.9999) allows the model to balance exploration and exploitation effectively, leading to higher average scores and rewards as it converges on optimal strategies.
  - A slower decay rate (0.99999) keeps the model exploring longer, which can slightly limit performance as it takes longer to focus on exploitation.
  - No decay (epsilon decay = 1) results in constant exploration, leading to lower scores and rewards, as the model fails to concentrate on the best strategies learned.

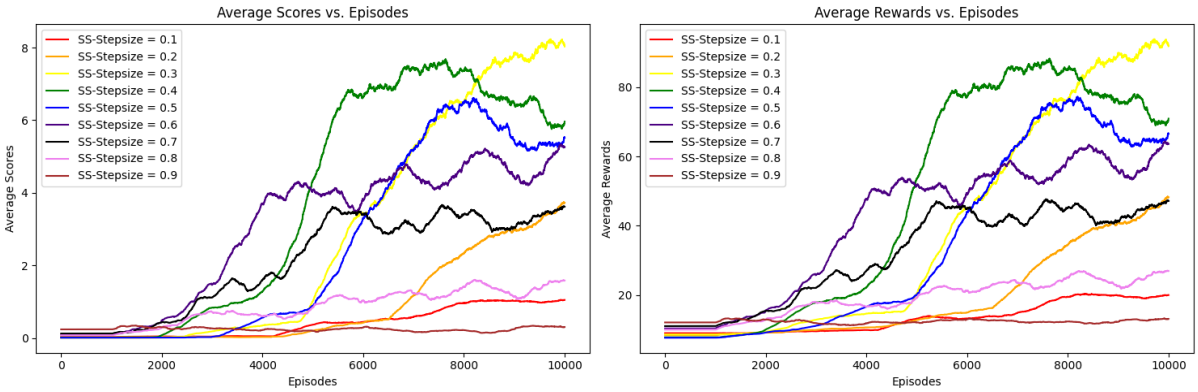


**Figure 5.8: SARSA with varying step size**

Looking at these two plots, we can analyze how different step size (learning rate) values affect the average scores and average rewards in the SARSA model applied to the Text Flappy Bird Gym environment.

- **Left Plot (Average Scores):**
  - Moderate step sizes (0.3 to 0.5) yield the highest average scores. The step size of 0.4 (green line) performs particularly well, reaching high scores and maintaining stable performance over time. This indicates that a balanced learning rate allows the model to update Q-values effectively without too much oscillation, leading to consistent improvement.

- Lower step sizes (0.1 and 0.2) lead to slower improvements in average scores. While the scores do increase over time, the model learns more gradually and fails to reach the high performance achieved by moderate step sizes, suggesting slower learning and convergence to an optimal policy.
- Higher step sizes (0.6 to 0.9) show more varied results. Step sizes of 0.6 (black line) and 0.7 (purple line) reach moderate scores, but the performance is unstable and fluctuates. Higher values, like 0.8 (pink line) and 0.9 (brown line), result in the lowest scores and show erratic performance, indicating that large updates lead to instability and prevent the model from converging on a reliable policy.

- **Right Plot (Average Rewards):**
  - The patterns in average rewards are similar to those in average scores. Step sizes between 0.3 and 0.5 produce the highest rewards, with 0.4 (green line) again showing the most consistent high performance, suggesting effective Q-value updates.
  - Lower step sizes (0.1 and 0.2) show slow but steady improvements in rewards, as the model gradually converges to a high-reward policy. These values allow the model to eventually find an effective strategy, but the learning is much slower.
  - Higher step sizes (0.6 to 0.9) result in fluctuating or lower average rewards. Step sizes of 0.8 and 0.9 show especially poor performance, as large step sizes cause instability in the Q-values, preventing the model from settling on effective strategies.

- **Summary:**
  - Moderate step sizes (0.3 to 0.5) provide the best balance, allowing the model to update its Q-values effectively and leading to higher and more stable average scores and rewards.
  - Low step sizes (0.1 and 0.2) result in slower learning, as the model updates its Q-values cautiously, limiting performance improvements over time.
  - High step sizes (0.6 to 0.9) cause instability, making it difficult for the model to converge on an optimal policy, which results in lower scores and rewards.

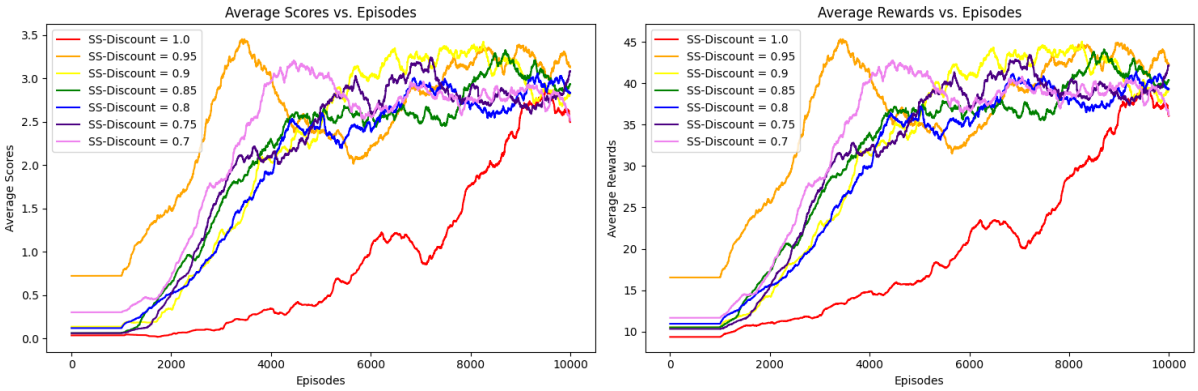


**Figure 5.9: SARSA with varying discount**

Based on the two plots showing the effect of different discount values on the average scores and average rewards in the SARSA model applied to the Text Flappy Bird Gym environment, the following observations can be made:

- **Left Plot (Average Scores):**
  - SS-Discount = 1.0 (red line): The highest discount value results in the lowest average scores. The model learns slowly and maintains a low, stable score throughout the episodes. This suggests that a discount of 1.0 makes it difficult for the model to focus on improving scores.
  - SS-Discount = 0.95 (orange line): The model learns faster and achieves a higher average score than discount = 1.0. However, the improvement is still not optimal.
  - SS-Discount = 0.9 (yellow line): This discount produces quite good results, with higher and more stable average scores. The model shows significant learning and score improvement.
  - SS-Discount = 0.85 (green line), 0.8 (blue line), and 0.75 (purple line): Discount values from 0.85 to 0.75 result in the highest average scores. This range of discount values seems to optimize learning and score improvement in the long term.
  - SS-Discount = 0.7 (pink line): The lowest discount value allows the model to learn fairly quickly but does not maintain high scores like the values between 0.75 and 0.85. This suggests that a low discount might not be sufficient for optimal long-term learning.
- **Right Plot (Average Rewards):**
  - SS-Discount = 1.0 (red line): The highest discount value similarly results in the lowest average rewards and slow learning. The model struggles to accumulate high rewards.
  - SS-Discount = 0.95 (orange line): This discount shows significant improvement compared to discount = 1.0, but it still performs lower than values from 0.9 and below.
  - SS-Discount = 0.9 (yellow line): This is one of the best-performing discounts, with high and stable average rewards. The model quickly accumulates rewards and maintains a high, stable level.
  - SS-Discount = 0.85 (green line), 0.8 (blue line), and 0.75 (purple line): Values from 0.85 to 0.75 achieve the highest and most stable average rewards, indicating this range is optimal for improving both scores and rewards in the long term.
  - SS-Discount = 0.7 (pink line): The lowest discount value yields high initial rewards but does not maintain high levels like the values between 0.75 and 0.85, suggesting that a low discount might not achieve maximum learning efficiency.
- **Summary:**

- Discount values from 0.75 to 0.85 provide the best performance, enabling the model to learn effectively and achieve the highest average scores and rewards.
- Higher discounts (0.9 and 0.95) also perform quite well but may not be as stable as values between 0.75 and 0.85.
- A discount of 1.0 is not effective, leading to slow learning and the lowest average rewards.
- Lower discount values (0.7) can speed up initial learning but may not optimize long-term performance.

## 5.3 Monte-Carlo

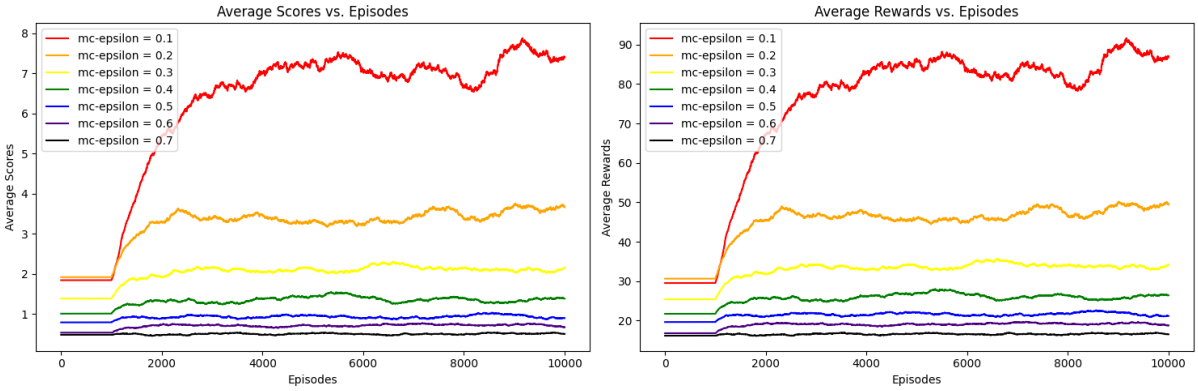


**Figure 5.10: Monte-Carlo with varying $\epsilon$**

Based on the two plots showing the effect of different epsilon values on the average scores and average rewards in the Monte Carlo (MC) model applied to the Text Flappy Bird Gym environment, the following observations can be made:

- **Left Plot (Average Scores):**
  - Epsilon = 0.1 (red line): The lowest epsilon value results in the highest average scores. The model learns very quickly, achieving a peak score of nearly 8, and then stabilizes with minimal fluctuations over the remaining episodes. This suggests that with low exploration (epsilon = 0.1), the model is able to quickly converge to a high-performing policy.
  - Epsilon = 0.2 (orange line): The model also achieves decent scores, though significantly lower than epsilon = 0.1. The average score stabilizes around 4 after the first 2,000 episodes, showing slower learning but still relatively good performance.
  - Epsilon = 0.3 (yellow line): With this epsilon value, the model shows even slower improvement, stabilizing around an average score of 3. The performance is noticeably lower than both epsilon = 0.1 and 0.2, indicating that increased exploration prevents the model from quickly learning the best policies.
  - Epsilon = 0.4 (green line), 0.5 (blue line), 0.6 (purple line), and 0.7 (black line): As the epsilon values increase, the average scores become progressively lower, showing minimal improvement across episodes. With epsilon = 0.7,

the model struggles the most, achieving an average score of less than 1. This indicates that with high exploration rates, the model spends too much time exploring and fails to focus on exploiting learned policies.

- **Right Plot (Average Rewards):**
  - Epsilon = 0.1 (red line): Similar to the average scores, this value results in the highest average rewards, peaking at around 90 after 2,000 episodes. The model demonstrates both fast learning and stable performance, indicating that low exploration (epsilon = 0.1) leads to efficient learning and optimal policy exploitation.
  - Epsilon = 0.2 (orange line): The model also achieves fairly high rewards, though it stabilizes around 60, which is significantly lower than epsilon = 0.1. This shows that a bit more exploration (epsilon = 0.2) allows for good performance, but not as optimal as the lower epsilon value.
  - Epsilon = 0.3 (yellow line): The average rewards stabilize at around 50, indicating a slower learning process. The rewards are still decent, but the model does not perform as well as it does with lower epsilon values.
  - Epsilon = 0.4 (green line), 0.5 (blue line), 0.6 (purple line), and 0.7 (black line): Higher epsilon values result in significantly lower rewards. With epsilon = 0.7, the rewards remain around 20, showing that the model's excessive exploration prevents it from consistently exploiting good strategies and earning higher rewards.

- **Summary:**
  - Low epsilon values (0.1 and 0.2) allow the model to learn efficiently, leading to higher average scores and rewards. Epsilon = 0.1 provides the best performance, with quick convergence and stable results.
  - Moderate epsilon values (0.3 and 0.4) result in slower learning and moderate performance, as the model balances exploration and exploitation.
  - High epsilon values (0.5 and above) lead to too much exploration, limiting the model's ability to exploit learned strategies, and thus result in lower average scores and rewards.

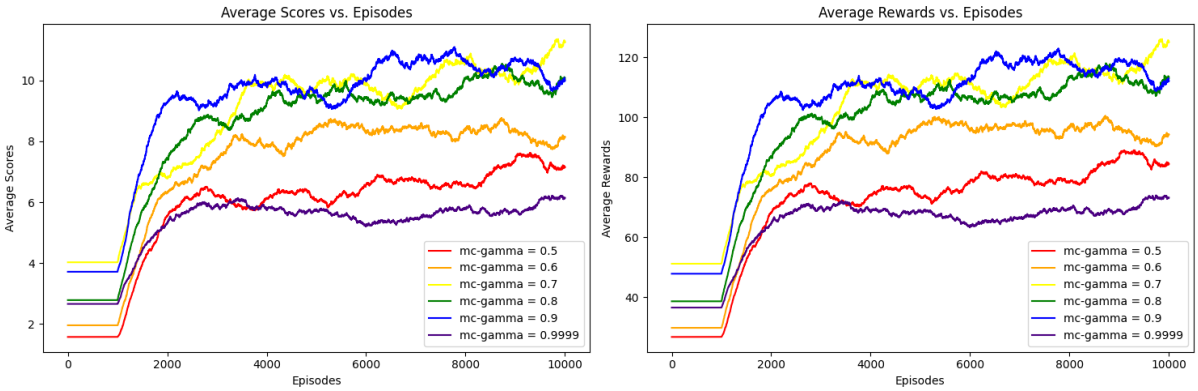


**Figure 5.11: Monte-Carlo with varying $\gamma$**

Looking at these two plots, we can analyze how different gamma (discount factor) values affect the average scores and average rewards in the Monte Carlo (MC) model applied to the Text Flappy Bird Gym environment.

- **Left Plot (Average Scores):**
  - With a high gamma value of 0.9999 (blue line), the average scores increase significantly and maintain a high peak, indicating that the model prioritizes long-term rewards and learns to achieve higher scores consistently. This helps the model focus on future rewards, leading to more strategic gameplay.
  - At gamma = 0.9 (green line), the average scores are slightly lower than for gamma = 0.9999, but the model still performs well. This value also encourages long-term rewards, though with less emphasis on very distant future rewards.
  - As gamma decreases (e.g., gamma = 0.8 in yellow, gamma = 0.7 in orange, and gamma = 0.6 in red), the average scores decrease. Lower gamma values focus more on immediate rewards, which may lead to a short-sighted strategy where the model fails to plan effectively for future rewards.
  - The lowest gamma value, 0.5 (purple line), results in the lowest average scores, indicating that the model places a high emphasis on immediate rewards, limiting its ability to learn strategies that yield higher scores over time.
- **Right Plot (Average Rewards):**
  - The pattern here is similar to the average scores plot. With a high gamma value of 0.9999 (blue line), the average rewards peak at the highest values and maintain stability, indicating effective learning that focuses on maximizing future rewards.
  - Gamma values around 0.9 (green) and 0.8 (yellow) also result in relatively high rewards, although they are slightly lower than the 0.9999 case. These gamma values still promote good performance, balancing between immediate and future rewards.
  - Lower gamma values (0.7, 0.6, and 0.5) yield lower average rewards, as these values cause the model to prioritize short-term gains, which limits its ability to learn strategies that could lead to higher rewards in the long run.
- **Summary:**
  - A high gamma value (0.9999) allows the model to focus effectively on future rewards, resulting in significantly higher average scores and rewards.
  - Medium gamma values (0.8 - 0.9) provide a reasonable balance, with moderately high scores and rewards but with slightly less focus on long-term rewards.
  - Low gamma values (0.5 - 0.7) cause the model to prioritize immediate rewards, leading to lower average scores and rewards as it struggles to adopt an optimal, future-oriented strategy.

## 5.4 Evaluation

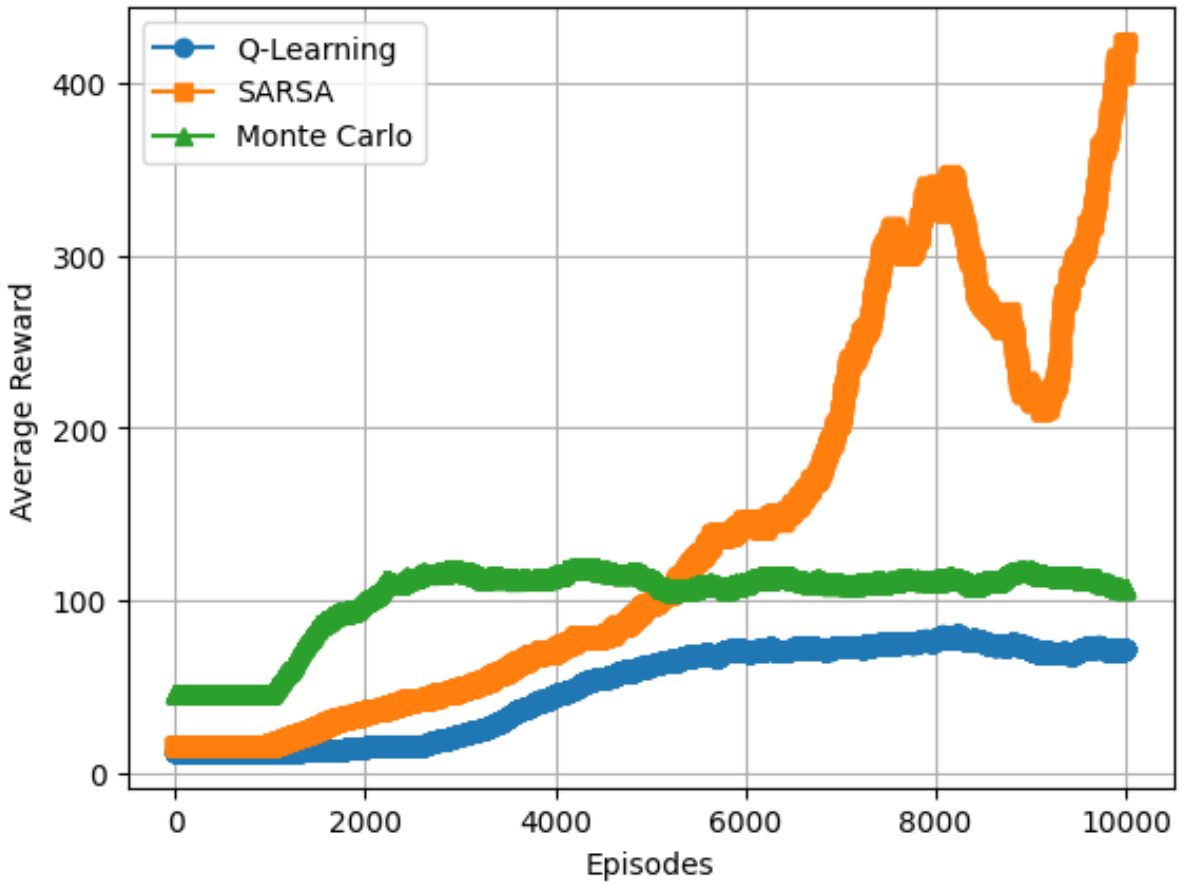


**Figure 5.12: Average Rewards with 3 models**

### 5.4.1 Q-Learning (Blue Line)

- **Convergence Speed:** Q-Learning exhibits a steady increase in average rewards during the initial phases, but its convergence speed is relatively slower compared to SARSA and Monte Carlo.
- **Average Reward:** The average reward reaches a relatively stable level after around 4,000 episodes and maintains that level for the remaining episodes. This indicates that Q-Learning converges and remains stable once it has learned the optimal policy.
- **Stability:** Q-Learning tends to maintain a consistent level of rewards, with no large fluctuations, meaning the algorithm behaves reliably after learning a good policy.

### 5.4.2 SARSA (Orange Line)

- **Convergence Speed:** SARSA converges faster than both Q-Learning and Monte Carlo. The average rewards increase sharply from the beginning and continue to rise gradually. This indicates that SARSA learns the policy faster than the other two algorithms.

- **Average Reward:** SARSA's average rewards keep increasing to a peak of over 400. This suggests that SARSA discovers a policy that yields higher rewards.
- **Stability:** While SARSA achieves the highest average rewards, it also has significant fluctuations towards the end of the graph (after around 8,000 episodes), which may indicate instability in the policy or the learning process.

### 5.4.3 Monte Carlo (Green Line)

- **Convergence Speed:** Monte Carlo converges faster than Q-Learning but not as quickly as SARSA. The average reward increases significantly early on and then gradually stabilizes.faster than the other two algorithms.
- **Average Reward:** Monte Carlo achieves a relatively high average reward ( 120) but lower than SARSA. This indicates that Monte Carlo learns a good policy but does not reach the highest performance level like SARSA.
- **Stability:** The average reward line for Monte Carlo is very stable and does not fluctuate much after convergence. This shows that the algorithm learns a stable policy and has good generalization ability.

### 5.4.4 Summary Recommendation

- **SARSA:** Has the highest average reward and fast learning speed, but lacks stability due to significant fluctuations after convergence. This may be because SARSA's on-policy learning approach leads to variations when encountering uncontrolled situations.
- **Q-Learning:** Slower convergence speed and lower average rewards compared to SARSA, but much more stable. It may be a good choice if a stable policy is needed after convergence.
- **Monte Carlo:** Achieves stable and relatively high average rewards, but not as high as SARSA. It is suitable when stability is required, and there is no need to quickly find an optimal policy.

Based on this analysis, the choice of algorithm will depend on specific requirements:

- If learning speed and high rewards are a priority, SARSA may be the best choice.
- If stability and consistent policy are needed, Q-Learning or Monte Carlo would be more suitable.

# Chapter 6

# Conclusion

## 6.1 Summary

In the experiments conducted with reinforcement learning (RL) algorithms, including Q-Learning, SARSA, and Monte Carlo on the Text Flappy Bird environment, each algorithm demonstrated distinct strengths and weaknesses. Q-Learning showed strong potential in finding the optimal policy, but it required longer convergence times due to its extensive exploration. SARSA, with its on-policy approach, achieved better performance in terms of convergence speed and stability, making it more suitable as the complexity of the environment increased. Monte Carlo, while providing accurate estimates based on complete episodes, struggled with computation time, especially in more complex settings, due to the need to run multiple full episodes for value estimation.

## 6.2 Lesson Learned

Several key lessons were drawn from the experiments:

- **Exploration and Exploitation:** Balancing exploration and exploitation is critical in optimizing RL algorithms. Adjusting the $\epsilon$ parameter effectively led to significant improvements in the convergence speed of both SARSA and Q-Learning.
- **On-Policy vs. Off-Policy Advantages:** SARSA was more stable because it updates values based on actual actions, whereas Q-Learning was more flexible but tended to explore more, which could lead to slower convergence. Understanding the nature of the environment and the desired behavior of the agent is crucial in choosing the appropriate algorithm.
- **Computation Time:** Monte Carlo required longer computation times as it relied on complete episodes for updates. This highlighted the importance of selecting an algorithm based on the system's time constraints and resource availability.

## 6.3 Future Work

To enhance the performance of RL algorithms in the Text Flappy Bird environment, several future research and development directions can be considered:

- **Improving Exploration Strategies:** Explore alternative strategies such as $\epsilon$-greedy decay, Boltzmann exploration, or methods that adaptively reduce $\epsilon$ over time to enhance the performance of Q-Learning and SARSA.
- **Combining with Deep Learning:** Integrate deep learning techniques with RL (e.g., Deep Q-Learning) to help the agent learn policies from more complex and larger state spaces, improving its ability to handle the complexities of Text Flappy Bird.
- **Experimenting with Advanced RL Algorithms:** Explore newer algorithms such as Actor-Critic, Double Q-Learning, or Proximal Policy Optimization (PPO) to find solutions that may converge faster and more efficiently across different environments.
- **Flexible Environment and Reward Structure:** Develop multiple versions of Text Flappy Bird with varying difficulty levels and configurations, which can help test the adaptability of RL algorithms when facing dynamic and ever-changing challenges.

# References


[1] Ge, V.: Solving planning problems with deep reinforcement learning and tree search. PhD thesis, University of Illinois at Urbana-Champaign (2018)

[2] Kroese, D.P., Rubinstein, R.Y.: Monte carlo methods. Wiley Interdisciplinary Reviews: Computational Statistics **4**(1), 48–58 (2012)

[3] Feng, D., Gomes, C.P., Selman, B.: Solving hard ai planning instances using curriculum-driven deep reinforcement learning. arXiv preprint arXiv:2006.02689 (2020)

[4] Pardo, F., Levdik, V., Kormushev, P.: Scaling all-goals updates in reinforcement learning using convolutional neural networks. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 34, pp. 5355–5362 (2020)

[5] Wiering, M.A., Van Otterlo, M.: Reinforcement learning. Adaptation, learning, and optimization **12**(3), 729 (2012)

[6] Watkins, C.J., Dayan, P.: Q-learning. Machine learning **8**, 279–292 (1992)