Play Flappy Bird using Reinforcement Learning Basic Algorithms

1 Introduction

1.1 Overview

Reinforcement Learning (RL) is a machine learning paradigm where an agent learns to make decisions by interacting with an environment to maximize cumulative rewards over time. Unlike supervised learning, which relies on labeled datasets for learning, RL operates on a trial-and-error basis. The agent explores the environment, observes the results of its actions, and adjusts its behavior based on the feedback it receives in the form of rewards or penalties. The agent's objective is to develop a policy—a strategy mapping states to actions—that maximizes the long-term reward. A hallmark of RL is the exploration-exploitation tradeoff, where the agent must balance between exploring new actions to improve knowledge about the environment and exploiting known actions to gain immediate rewards. Key RL algorithms include Q-learning, which estimates the value of actions in different states, and more advanced models and policy gradient methods, which leverage deep learning for more complex tasks.

Reinforcement learning has broad applications across various domains. In gaming, RL has produced agents capable of beating world champions in games like Chess and Go, with AlphaGo being a notable success [1]. It has also been applied in robotics for tasks such as motion planning, manipulation, and autonomous navigation [2]. In real-world scenarios, self-driving cars use RL to learn how to navigate complex traffic environments [3]. The financial industry employs RL to optimize trading strategies, while personalized recommendation systems use it to adapt to user behavior in real-time. As of now, RL is still in a state of rapid development, with research focused on improving sample efficiency, stability, and scalability of algorithms. While deep reinforcement learning has made significant strides in complex, high-dimensional environments, challenges remain, including the ability to transfer learning between tasks and ensuring safe exploration in real-world applications where mistakes can be costly. Despite these challenges, RL continues to be a key area in AI research, with increasing use in industries like healthcare, autonomous systems, and resource management.

In this project, we will apply reinforcement learning (RL) to solve the Flappy Bird game, utilizing RL's strengths in sequential decision-making. The game involves navigating a bird between gaps in pipes by deciding whether to "flap" or not, and the goal is to survive as long as possible to achieve a higher score. RL will allow the agent

to learn from its interactions with the environment, improving its performance over time based on the rewards (successfully passing pipes) and penalties (hitting obstacles or the ground). By using an RL algorithm, the agent will learn an optimal strategy to maximize its score. Training the agent on this game highlights RL's potential for solving environments with delayed rewards and dynamic challenges. As part of our approach, we will implement RL to train our agent to master Flappy Bird, improving its policy to achieve better gameplay performance over time.

1.2 Problem Description

The problem we aim to address in this project is developing an intelligent agent that can effectively play the Flappy Bird game using reinforcement learning (RL) [4]. Flappy Bird is a deceptively simple yet challenging game where the player controls a bird and must navigate through a series of pipes without crashing. The agent faces two actions at each time step: to "flap" or to do nothing, with the ultimate goal of staying alive as long as possible. The challenge lies in determining the correct timing for flapping, as poor timing leads to collisions with the pipes or the ground, resulting in game over.

Given the dynamic nature of the game, traditional rule-based approaches struggle to perform well, especially in environments with delayed rewards and non-linear dynamics. This makes Flappy Bird an ideal candidate for reinforcement learning, where the agent can learn to improve its strategy over time by interacting with the game environment. The problem is to train an RL agent that can maximize its survival time and thus its score, balancing exploration of new strategies and exploitation of known good moves. By applying reinforcement learning algorithms, we aim to teach the agent how to handle the complex decision-making process required to navigate the bird through the obstacles effectively.

1.3 Objectives

The primary objectives of this project are to apply reinforcement learning (RL) to the Flappy Bird game and to develop an intelligent agent that can effectively play the game using RL algorithms [5]. The Flappy Bird game is a challenging game that requires the player to navigate a bird through a series of pipes without crashing. The game is ideal for RL because it has a clear objective, a well-defined state and action space, and a simple reward function.

The specific objectives of this project are to implement three different RL algorithms (Monte Carlo[6], Q-Learning [7], and SARSA[8]) to solve the Flappy Bird game. Each of these algorithms has its own strengths and weaknesses, and by implementing them, we can evaluate their effectiveness in achieving the highest score in the game. Additionally, we aim to optimize the agent's policy using each RL algorithm to achieve the best possible score in the game.

Another important objective of this project is to analyze the tradeoff between exploration and exploitation for each RL algorithm. Exploration is the process of trying new actions to gather more information about the environment, while exploitation is the process of using what we already know to make the best decision and get the most

rewards. Striking the right balance between exploration and exploitation is crucial in RL, as it can significantly impact the agent's performance. By analyzing the tradeoff between exploration and exploitation, we can determine the optimal balance between the two for each RL algorithm.

By achieving these objectives, this project aims to demonstrate the effectiveness of RL algorithms in solving complex problems like the Flappy Bird game and to contribute to the development of more advanced RL techniques. The results of this project can be used to improve the performance of RL agents in a variety of applications, from gaming to robotics and beyond.

1.4 Structure of Report

2 Methodology

2.1 Reinforcement Learning Basics

Reinforcement Learning (RL) is based on several key components that work together to help an agent learn how to make decisions by interacting with its environment. These components include the agent, environment, state, action, reward, policy.

- **Agent:** The agent is the learner or decision-maker in Reinforcement Learning. It's the part of the system that interacts with the environment to learn how to achieve a goal. In a game, for example, the agent would be the player trying to win.
- **Environment:** The environment is everything that the agent interacts with. It includes the space, objects, and rules that define the task or game the agent is trying to perform. For instance, in a game like Flappy Bird, the environment includes the bird, pipes, and the sky.
- **State:** A state represents the current situation or condition of the environment. It's like a snapshot of everything important at a given moment. For example, in Flappy Bird, a state might include the bird's height and the position of the next pipe.
- **Action:** An action is what the agent decides to do in a given state. In each state, the agent has a set of actions it can choose from. In Flappy Bird, the actions are either "flap" (to move upward) or "not flap" (to let the bird fall).
- **Reward:** The reward is feedback from the environment to the agent. When the agent takes an action, it receives a reward based on how good or bad the action was. In Flappy Bird, a reward could be a positive score for passing through a pipe or a negative score (penalty) for hitting a pipe.
- **Policy:** The policy is the strategy the agent uses to decide which action to take in each state. It's like a guide that helps the agent choose its actions to get the most rewards. The policy can change over time as the agent learns what actions work best.

A key challenge in RL is balancing **exploration** and **exploitation** [9]. **Exploration** is when the agent tries new or less familiar actions to gather more information about the environment. **Exploitation**, on the other hand, is when the agent uses what it already knows to make the best decision and get the most rewards. Striking the right balance between exploration and exploitation is crucial: if the agent explores

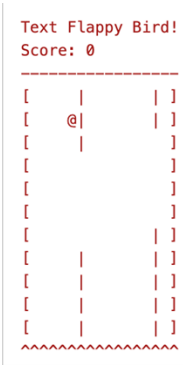

Fig. 1 Environment Overview

too much, it might waste time on bad actions, but if it exploits too soon, it might miss out on better strategies it hasn't discovered yet.

2.2 Experimental Setup

The experimental setup for using the TextFlappyBird-v0 environment in reinforcement learning focuses on establishing a structured framework that allows for effective testing and algorithm development. The environment is characterized by several parameters: height, width, pipe gap, and a custom seed. The height determines the vertical size of the pipes, while the width specifies the horizontal distance between two pipe columns, influencing the spacing of the obstacles. The pipe gap is the crucial width that allows the bird to navigate through the pipes, and the custom seed enables the generation of either random or fixed pipe columns, ensuring reproducibility in experiments.

In this environment, the agent's state is represented as a tuple of two values: (dx, dy), where dx is the horizontal distance from the bird to the center of the gap, and dy is the vertical distance from the bird to the center of the gap. This simplified representation allows the agent to focus on critical distances, facilitating effective learning. The reward structure is designed to incentivize survival, with the agent receiving a reward of +1 for each time step it remains alive. Consequently, the cumulative reward increases as long as the bird successfully avoids collisions with pipes and the ground.

The agent's behavior is governed by its velocity, which accumulates based on its actions. The action space consists of two discrete actions: Action 0, which increases the bird's velocity by 2 (resulting in a downward movement), and Action 1, which sets the bird's velocity to -1 (allowing the bird to ascend by 1 unit). This dynamic is critical for developing strategies that enable the bird to navigate through the obstacles effectively. The implementation of a suitable reinforcement learning algorithm, such as Q-learning or DQN, will allow the agent to learn from its experiences and optimize its decision-making over time.

Training is conducted through repeated episodes, where the environment is reset at the beginning of each episode. During each episode, the agent interacts with the environment by selecting actions based on its current policy, transitioning between states, and receiving rewards accordingly. Performance metrics, such as average cumulative

reward and the number of successful episodes, are monitored throughout the training process to evaluate the effectiveness of the learning algorithm. After training, the agent's performance is assessed by allowing it to navigate the game using the learned policy, with metrics like survival time and average reward per episode being recorded.

Finally, the experimental setup encourages parameter tuning and analysis. By experimenting with different configurations of height, width, pipe gap, and seed values, the impact of these parameters on the agent's learning performance can be understood. Hyperparameters of the chosen reinforcement learning algorithm can also be adjusted to optimize learning outcomes. Documenting the experimental results, including training curves and notable agent behaviors, provides valuable insights into the effectiveness of the algorithm and the defined environment, paving the way for further research and improvement in reinforcement learning strategies.

2.3 Algorithms Overview

2.3.1 Monte Carlo

Monte Carlo methods are employed in reinforcement learning to enable an agent to learn from its experiences by completing entire tasks or games before reflecting on the outcomes. Rather than updating its knowledge after each individual move, the agent waits until the task concludes, allowing for a comprehensive evaluation of its actions.

The concept is straightforward: the agent plays from the beginning to the end of the game, meticulously tracking every action taken and the resulting outcomes. Once the game is finished, the agent reviews its actions to identify which decisions led to favorable results and which ones did not. Over time, this accumulated information helps the agent enhance its decision-making process for future tasks.

Input: a policy π to be evaluated
Initialize:
    $V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$
    $Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Loop forever (for each episode):
    Generate an episode following π: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
        $G \leftarrow \gamma G + R_{t+1}$
        Unless S_t appears in $S_0, S_1, \ldots, S_{t-1}$:
            Append G to $Returns(S_t)$
            $V(S_t) \leftarrow$ average($Returns(S_t)$)

Fig. 2 Monte Carlo Algorithm

Monte Carlo methods are particularly effective for tasks that can be neatly divided into complete episodes, such as games. The more the agent plays, the more it learns from its collective experiences, ultimately leading to improved decision-making. However, this approach can be slower than other methods, as it waits until the end of an episode to update its strategy.

In this framework, the exploration rate is governed by the parameter epsilon, which is typically set between 0.1 and 0.3. The discount factor, gamma, usually around 0.95,

helps balance the significance of immediate rewards against future gains. The environment is structured to allow two actions, denoted as $N\_ACTIONS = 2$, reflecting the movement options available to the bird. The Q-values, initially set to zero in the Q dictionary, are refined using the *update_Q* function, which applies a First-Visit Monte Carlo approach to average the returns for each state-action pair, leveraging the visit counts recorded in the N dictionary.

2.3.2 Q-Learning

Q-Learning is a straightforward yet powerful reinforcement learning algorithm that enables an agent to learn the optimal actions to take in various situations without needing prior knowledge of the environment's rules. This characteristic is what makes it a model-free algorithm, as the agent does not require a model or map of the environment to facilitate its learning process.

In Q-Learning, the agent maintains a Q-table, which tracks the value of each action in every possible state, known as Q-values. These Q-values help the agent estimate the long-term effectiveness of different actions, allowing it to learn which choices yield the best outcomes over time.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \; max_a \; Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

As an off-policy algorithm, Q-Learning allows the agent to learn the optimal strategy (or policy) independent of the actions it is currently taking. This means the agent can explore various actions, including random ones, while still acquiring knowledge about the most advantageous actions to take in the future. This flexibility is crucial for Q-Learning, as it enables the discovery of the optimal strategy without being confined to its current behavior.

Key parameters of the Q-Learning algorithm include the learning rate lr, which influences how much the Q-values are updated during training and can be set to values such as [0.4, 0.6, 0.8, 0.9]. The discount factor *discount* balances the significance of immediate rewards versus future rewards, with potential values including [1.0, 0.95, 0.9, 0.8, 0.7]. The exploration rate epsilon, starting at 0.5, affects the trade-off between exploration and exploitation, allowing the agent to choose between random actions and those with the highest Q-value. This exploration rate decreases over time with an *eps_decay* of 0.9995, encouraging the agent to rely more on its learned knowledge as it accumulates experience. Lastly, the number of actions $N\_ACTIONS$ is fixed at 2, representing the options for the bird to ascend or descend.

2.3.3 SARSA

SARSA (State-Action-Reward-State-Action) is a reinforcement learning algorithm closely related to Q-Learning, distinguished by its on-policy approach. Instead of learning from the optimal actions available, SARSA updates its knowledge based on the actual actions the agent takes. This sequence involves observing the current state, selecting an action, receiving a reward, transitioning to a new state, and then choosing

the subsequent action. The acronym SARSA encapsulates this process: State, Action, Reward, State, Action.

By learning from the agent's real actions—regardless of whether they are the best choices—SARSA tends to promote safer and more stable learning. If the agent opts for cautious actions, SARSA will continue to refine this conservative strategy. In contrast, Q-Learning focuses on learning the best actions available, which can sometimes encourage more aggressive behavior.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\Big].$$

The agent starts with an initial exploration rate (epsilon) of 0.5, which decays over time using a decay rate *eps_decay* of 0.9995, and a learning rate lr of 0.4 determines how quickly the agent learns from its experiences. The discount factor *discount* of 0.9 sets the importance of future rewards. Additionally, a random number generator *rand_generator* is used to select actions and update the Q-values, with a seed of None. The agent is trained for *n_episodes* episodes, with the average score over the last window episodes calculated, and then tested for *n_episodes* episodes, with the average score calculated. The agent can also be run for a single episode, returning the final score.

3 Results and Discussion

Our experimental results demonstrate the effectiveness of the reinforcement learning algorithms implemented to play the Flappy Bird game. We trained three agents using the Monte Carlo, Q-Learning, and SARSA algorithms, each with the goal of maximizing the cumulative reward and navigating the game for as long as possible.

The Monte Carlo method achieved a score exceeding 40 after 4,000 episodes, demonstrating a gradual improvement in the agent's performance over time. Notably, the agent attained a higher maximum cumulative reward after 8,000 episodes. During testing, the agent recorded its highest score, nearly reaching 100. The results of the training process are not as good as the testing process. That's because during the training process, exploration probability is still used to dynamically perform exploration and exploitation

Table 1 Select hyper-parameter (Mean scores)

Discount/Lr	0.4	0.6	0.8	0.9
1.0	22.0	2.2	5.9	5.0
0.95	2.7	4.4	1.6	0.3
0.9	4.6	4.0	1.5	0.1
0.8	2.5	1.0	2.4	2.2
0.7	2.3	2.0	1.9	2.2

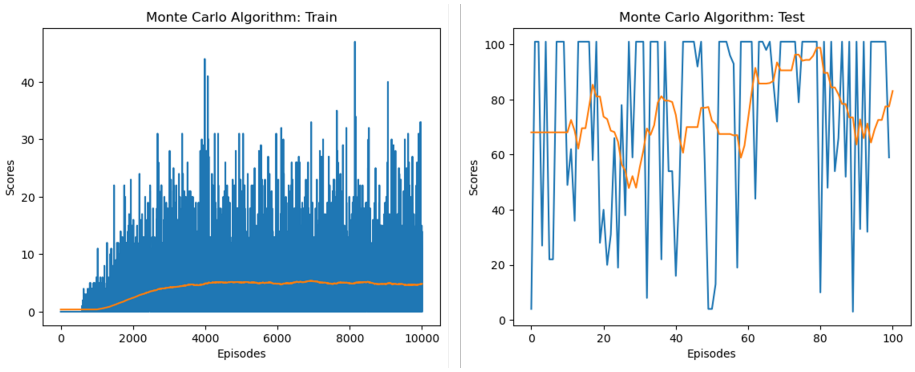

Fig. 3 History of Training and Testing Processes of Mento Carlo

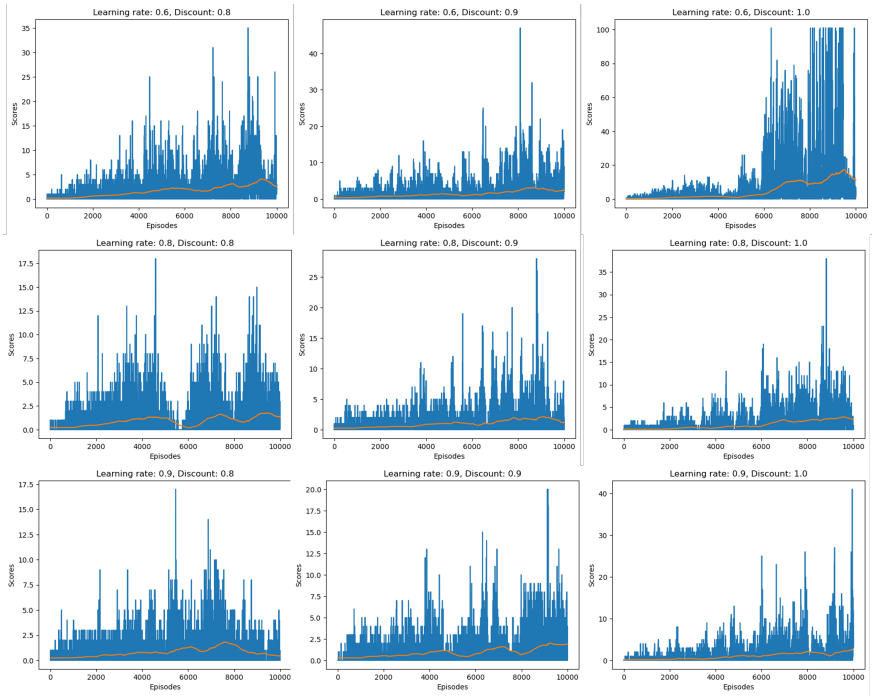

Fig. 4 History of Training and Testing Processes of Q-Learning

Both of the remaining two models (Q-learning and SARSA) are not performing well and achieving efficiency in Bird control implementation. The scores achieved are not kept at the best level, sometimes reaching the maximum score (100), sometimes reaching a very low level (less than 10). With the Q-Learning algorithm, even though the method of tweaking hyper-parameters was used to select the best hyper-parameters for the algorithm, it did not bring high results.

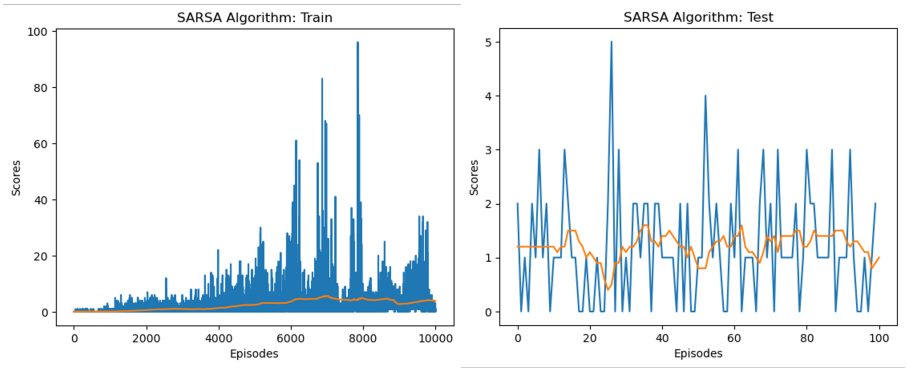

Fig. 5 History of Training and Testing Processes of SARSA

The Flappy Bird game environment has a lot of randomness, so it also greatly affects the results of the above algorithms, making these algorithms unstable or disturbing the updated values. In addition, a factor that affects the effectiveness of the models is the reward of the game, it is always 1 for every decision of the Agent, making it difficult to optimize actions to achieve the highest reward.

The results of this study contribute to the development of more advanced reinforcement learning techniques, which can be applied to a wide range of applications, from gaming to robotics and beyond. Future studies can build on this work by exploring the use of more advanced reinforcement learning algorithms, such as deep reinforcement learning, and by applying these techniques to more complex environments.

4 Conclusion

In conclusion, the experiment highlights the effectiveness of reinforcement learning algorithms in playing the Flappy Bird game, with the Monte Carlo algorithm emerging as the most successful. It consistently outperformed the other two algorithms, showcasing significant performance improvements over time. This underscores Monte Carlo's robustness as a solution for sequential decision-making problems, particularly when the exploration and learning rates are finely tuned.

In contrast, the Q-Learning and SARSA algorithms exhibited the least stable performance, often failing to complete tasks effectively. Its reliance on random exploration likely hindered its convergence speed, making it less suitable for complex problems.

Overall, the findings suggest that while Monte Carlo stands out as the most effective approach, Q-Learning and SARSA may struggle with stability, especially with this random environment and fixed reward. These insights can guide the development of advanced reinforcement learning algorithms for various applications.

References

[1] David Silver et al. "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529 (Jan. 2016), pp. 484–489. DOI: 10.1038/nature16961.

[2] Jens Kober, J Andrew Bagnell, and Jan Peters. "Reinforcement learning in robotics: A survey". In: *The International Journal of Robotics Research* 32.11 (2013), pp. 1238–1274.

[3] Ahmad EL Sallab et al. "Deep reinforcement learning framework for autonomous driving". In: *arXiv preprint arXiv:1704.02532* (2017).

[4] Aidar Shakerimov, Dmitriy Li, and Jurn-Gyu Park. "A Case Study: Characterization of Performance Inconsistency for Reinforcement Learning on Flappy Bird Game". In: *2021 International Conference on Information and Communication Technology Convergence (ICTC)*. 2021, pp. 1–6. DOI: 10.1109/ICTC52510.2021.9621017.

[5] Tai Vu and Leon Tran. *FlapAI Bird: Training an Agent to Play Flappy Bird Using Reinforcement Learning Techniques*. 2020. arXiv: 2003.09579 [cs.AI]. URL: https://arxiv.org/abs/2003.09579.

[6] Bruno Bouzy and Guillaume Chaslot. "Monte-Carlo Go Reinforcement Learning Experiments". In: *2006 IEEE Symposium on Computational Intelligence and Games*. 2006, pp. 187–194. DOI: 10.1109/CIG.2006.311699.

[7] Watkins. "Q-learning". In: (1989).

[8] Clark Kendrick Go et al. "A reinforcement learning approach to the shepherding task using SARSA". In: *2016 International Joint Conference on Neural Networks (IJCNN)*. 2016, pp. 3833–3836. DOI: 10.1109/IJCNN.2016.7727694.

[9] Melanie Coggan. "Exploration and exploitation in reinforcement learning". In: *Research supervised by Prof. Doina Precup, CRA-W DMP Project at McGill University* (2004).