Comparative Analysis of Q-learning and SARSA in the Flappy Bird Game with Enhanced Reward Functions

Phan Duy Kha, Nguyen Dinh Huy

***Abstract*—This report analyzes the performance of Q-learning and SARSA in training an agent for the Flappy Bird game. An enhanced reward function is utilized to encourage prolonged flight, reward bonus actions, and penalize collisions. The report describes the game mechanics, detailed state representation, action dynamics, reward structure, learning algorithms, experimental evaluation, and discussion on results and future improvements.**

***Index Terms*—Reinforcement Learning, Q-learning, SARSA, Flappy Bird, Reward Functions.**

I. Introduction

This report analyzes the performance of Q-learning and SARSA in training an agent for the Flappy Bird game. An enhanced reward function is utilized to encourage prolonged flight, reward bonus actions, and penalize collisions. The structure of the report includes a description of the game mechanics, a detailed analysis of state representation, actions, and reward functions, followed by the methodology, experimental evaluation, and discussion on results and future improvements.

II. Game Description and Mechanics

The Flappy Bird game offers a challenging environment for Reinforcement Learning experiments. Implemented using `pygame`, it comprises several key components that simulate realistic game dynamics.

A. Game Elements and Dynamics

- **Visual Elements:** The game features a background, moving ground, pipes as obstacles, and a bird controlled by the agent. A bonus element, referred to as a "gift", is also included.
- **Dynamics:**
  - The bird is affected by gravity and can flap to gain altitude.
  - Pipes continuously move leftwards at a constant speed, with new pipes generated as the older ones exit the screen.
  - The gift moves along with the pipes; when collected, it provides an extra reward.

B. State Representation and Actions

The agent's perception of the environment is defined by its state representation and the available actions.

1) State Representation: The state of the game is represented as a tuple capturing essential information related to the bird's position and the environment:

- **Horizontal Distance to the Next Pipe:** This is calculated as the difference in x-coordinates between the bird and the next set of pipes, normalized by a fixed factor to maintain consistency.
- **Vertical Distance to the Next Pipe:** This component measures the difference in height between the bird and the lower edge of the upper pipe, with adjustments to ensure all values are non-negative.
- **Position of the Next Gift:** Both the x and y coordinates of the gift are incorporated; the x-coordinate is normalized similarly to that of the pipes, while the y-coordinate is computed relative to the bird's current vertical position.

This detailed representation ensures the agent captures all relevant spatial information to make informed decisions.

2) Actions: The agent has two possible actions:

- **Action 0 (Do Nothing):** The bird is allowed to descend naturally under gravity.
- **Action 1 (Flap):** The bird flaps its wings to gain altitude, counteracting gravity.

Action selection is implemented using a greedy or ϵ-greedy strategy, where the agent occasionally takes a random action to promote exploration while predominantly choosing the action with the highest expected reward.

C. Enhanced Reward Function

The reward function is designed to provide clear, multidimensional feedback to guide the agent's learning:

- **+200:** Awarded for successfully passing through a pipe.
- **+500:** Granted for collecting a gift.
- **-50:** Penalizes the agent for missing a gift.
- **-1000:** Imposed when the bird collides with a pipe or the ground, leading to the termination of the episode.

This reward structure balances positive reinforcement for desirable actions with penalties for mistakes, thereby shaping the agent's behavior effectively.

III. Methodology

A. Environment and Agent Architecture

The environment is simulated using `pygame`, which manages state updates, applies action effects, and assigns rewards based on the bird's interactions. The agent employs a Q-table,

stored as a dictionary where each state tuple is associated with an array of Q-values for the available actions. States are added dynamically as they are encountered, and actions are chosen based on either a greedy or ϵ-greedy strategy, with the exploration rate decaying over time to shift the focus from exploration to exploitation.

B. Learning Algorithms

Both Q-learning and SARSA are implemented within the same framework for state discretization and Q-table management. Their primary difference lies in the update mechanism:

- **Q-learning:** An off-policy method that updates Q-values using the maximum expected future reward from the next state. This approach typically converges faster, though it may promote riskier strategies.
- **SARSA:** An on-policy method that updates Q-values based on the actual action taken in the next state, leading to a more conservative and stable learning process.

Key parameters such as the learning rate (α), discount factor (γ), and exploration rate (ϵ) play critical roles in both algorithms.

IV. Experimental Evaluation

A. Setup

The training process is configured with the following parameters:

- Learning rate (α): 0.1
- Discount factor (γ): 0.9
- Exploration rate (ϵ): Initially high and gradually decaying
- Number of training episodes: 1000

B. Performance Metrics

The agent's performance is evaluated based on:

- **Average Score per Episode:** Indicative of the overall performance.
- **Convergence Speed:** Reflected by the stabilization of Q-values across episodes.
- **Policy Stability:** Measured by the consistency of the agent's decisions during testing.
- **Bonus Collection Frequency:** The frequency of collecting gifts, which reflects the effectiveness of the reward function.

C. Results and Discussion

Preliminary observations indicate that:

- Q-learning converges faster but may adopt riskier strategies.
- SARSA provides a more stable learning process with fewer collisions due to its on-policy update mechanism.
- The enhanced reward function is critical in guiding the agent toward effective behavior.

V. Discussion and Conclusion

A. Discussion

A detailed comparison between Q-learning and SARSA reveals:

- **Q-learning:** Its off-policy nature and optimistic estimation of future rewards lead to rapid convergence but may encourage riskier actions.
- **SARSA:** By basing updates on the actual actions taken, SARSA tends to foster a more conservative and stable policy.
- The multidimensional feedback provided by the enhanced reward function is essential for effective learning.

B. Limitations and Future Work

While the results are promising, several limitations remain:

- The discretization of continuous states might omit some dynamic details of the game.
- Further parameter tuning may improve performance.
- Future work could explore more complex reward structures and advanced state representation techniques, possibly incorporating deep learning methods.

C. Conclusion

This report demonstrates that both Q-learning and SARSA can effectively train an agent in the Flappy Bird game using an enhanced reward scheme. Q-learning offers faster convergence, while SARSA provides a safer and more stable policy. The analysis provides a strong foundation for further improvements and deeper research into reinforcement learning strategies.