Reinforcement Learning FlappyBird

Ofek Kirshenboim

1 Introduction

In this study, we implemented a Flappy Bird environment tailored to explore reinforcement learning approaches by focusing on two key algorithms: SARSA and Q-Learning. Our approach involved customizing the state representation and reward structure to improve learning efficiency and interpretability. For state preprocessing, we created a custom observation map that extracts and focuses on four critical features: the player's vertical position, velocity, distance to the next pipe, and the next pipe's top height. To further limit complexity, we discretized each parameter into 32 possible states by the parameters by 16, preventing the policy table from growing excessively large. Additionally, we shaped the reward function to encourage the wanted behavior before the appearance of the pipe, this was done by making a loss based on the height distance from the next pipe. Using these adjustments, we implemented and tested SARSA and Q-Learning, two reinforcement learning algorithms, to compare their effectiveness in training an agent to master the game.

2 Methodology

In this section, we describe our approach to preprocessing the state observations, shaping the reward function, and implementing the SARSA and Q-Learning agents.

2.1 Preprocessing

To reduce the complexity of the state representation and focus on the most relevant aspects of the game, we identified four key features from the environment to define the observation space: the player's vertical position, the player's velocity, the horizontal distance to the next pipe, and the height of the top edge of the next pipe. These features were selected to provide a concise yet informative representation of the game's state, allowing the agent to make decisions effectively while minimizing the computational overhead. To further reduce the complexity of the state space and maintain computational feasibility, we discretized the continuous positional values. By dividing the positions by 16, we limited the possible states to 32 per feature, thereby ensuring that the policy table remained manageable.

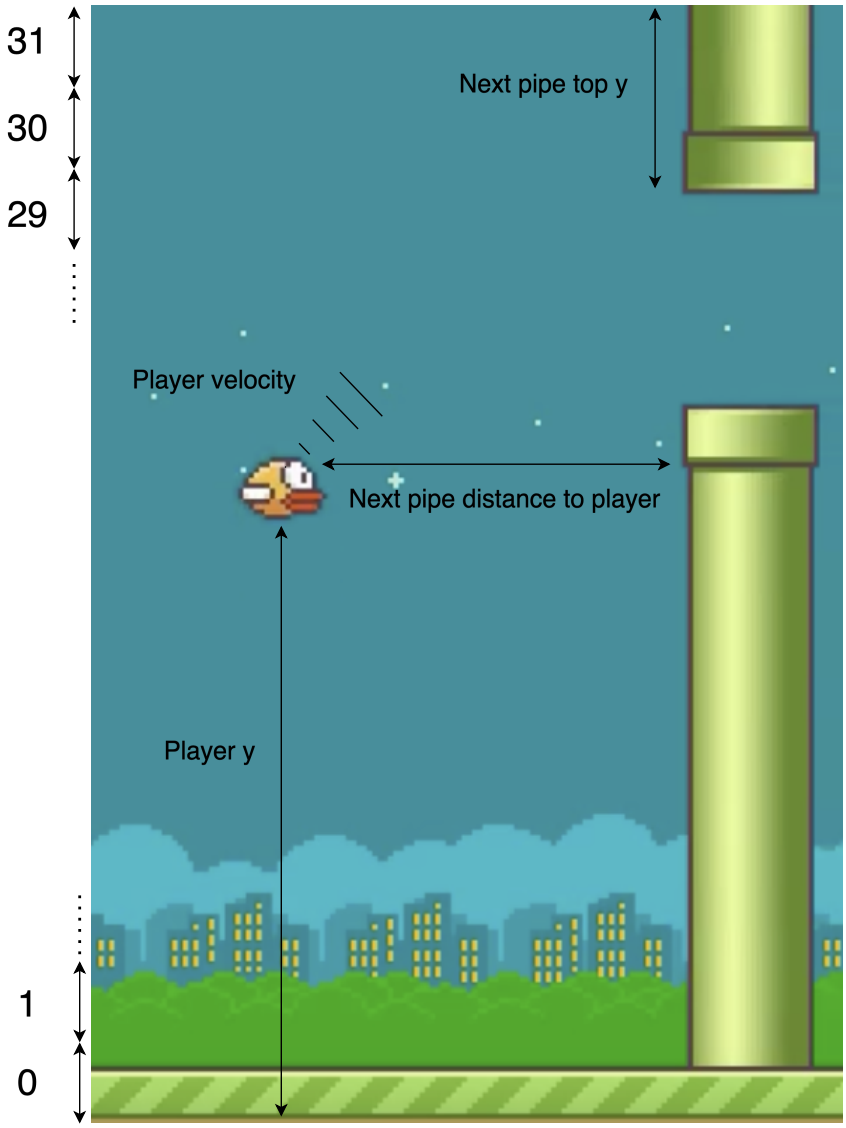

Figure 1: The observation space is constructed from four features: the player's vertical position (Player y), the player's velocity (Player velocity), the horizontal distance to the next pipe (Next pipe distance to player), and the height of the top edge of the next pipe (Next pipe top y). Each positional feature is discretized into 32 possible states based on its location on the screen.

2.2 Reward Shaping

To encourage progress and penalize inefficiencies, we designed a custom reward function. The reward is scaled based on the game's reward signal for passing pipes and adjusted to penalize the agent for its horizontal distance from the next pipe. The horizontal distance penalty reduces the agent's reward based on how far it is from the optimal trajectory:

$$R_{\text{shaped}} = R_{\text{game}} \cdot 60 - \frac{|d|}{255} \tag{1}$$

where d is the absolute difference between the player's y-position and the top position of the next pipe.

2.3 Agent Implementation

We implemented two reinforcement learning agents, SARSA and Q-Learning, to train the agent on the Flappy Bird environment. Both agents utilize a tabular method to store and update the state-action values (Q-values) for each state-action pair. The policy table is a five-dimensional structure with dimensions corresponding to the discretized state space and the action space. Specifically, the table has a size of $32 \times 32 \times 32 \times 32 \times 2$, where each of the four state dimensions has been discretized into 32 possible values, as described in the preprocessing step. This design ensures that the state space is computationally manageable while retaining sufficient granularity for effective learning. Initially, all values in the policy table are set to zero. The SARSA agent follows an on-policy approach, where the next action is selected based on the agent's current policy. The Q-value update for SARSA is given by:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma Q(s',a') - Q(s,a) \right] \tag{2}$$

where s and a are the current state and action, s' and a' are the next state and action, r is the reward, α is the learning rate, and γ is the discount factor.

The Q-Learning agent, in contrast, follows an off-policy approach by selecting the next action greedily, regardless of the current policy. The Q-value update for Q-Learning is:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right] \tag{3}$$

Both agents use an epsilon-greedy strategy for exploration, where the agent chooses a random action with probability ϵ and the action with the highest Q-value otherwise. This balance ensures sufficient exploration of the state space during training.

Algorithm 1: SARSA and Q-Learning

Data: State space S, Action space A, Learning rate α, Discount factor γ, Exploration rate ϵ, Number of episodes N

Result: Learned Q-value table $Q(s, a)$

Initialization:

Initialize $Q(s, a) = 0$ for all $s \in S$ and $a \in A$;

for *each episode n from 1 to N* **do**

Reset environment and observe initial state s;

Select action a using the epsilon-greedy policy:

$$a = \begin{cases} \text{random action from } A & \text{with probability } \epsilon, \\ \arg\max_{a'} Q(s, a') & \text{otherwise} \end{cases}$$

for *each step in the episode* **do**

Execute action a, observe reward r and next state s';

Select next action a' using the epsilon-greedy policy (SARSA only);

Update Q-value:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \begin{cases} Q(s', a') & \text{(SARSA)}, \\ \max_{a'} Q(s', a') & \text{(Q-Learning)} \end{cases} - Q(s, a) \right]$$

Update state and action:

$$s \leftarrow s', \quad a \leftarrow a' \quad \text{(SARSA only)}.$$

if *s' is terminal* **then**

Break the loop;

3 Results

The parameters chosen for both models are the same: $\alpha = 0.1$, $\gamma = 0.95$, $\epsilon = 0.1$. The results in Fig. 2 correspond to the scenario with a small gap between pipes (gap = 65).In the upper left plot of (Fig. 2), the normalized rewards show consistent improvement across episodes for both SARSA and Q-Learning. Q-Learning achieves slightly higher normalized rewards throughout training. In the upper right plot of (Fig. 2), both models demonstrate an increase in success rates as the number of episodes progresses. The graph is noisy, making it challenging to determine which model performs better in the training phase. In the bottom left plot of (Fig. 2), both models begin to pass three pipes after about 30,000 episodes, with their performance plateauing near a score of 3.5 pipes on average. In the bottom right plot of (Fig. 2), during the validation phase, the Q-Learning model passes an average of 8 pipes in 50 runs, while the SARSA model averages 6 pipes. Q-Learning also exhibits a longer tail

in its performance, with some runs passing up to 52 pipes and others reaching scores of 1, highlighting its variability. In Fig. 3, the plots show the results for the same parameters but with a larger gap between the pipes (gap = 85). Unlike Fig. 2, it can be observed that during the training phase, the SARSA model outperforms the Q-Learning model. By the end of training, the SARSA model achieves an average of 5.5 pipes passed and a success rate of 0.19, compared to Q-Learning's 4.5 pipes passed and a success rate of 0.125. However, in the validation phase, based on 50 runs, Q-Learning slightly outperforms SARSA, achieving an average of passing one more pipe. Despite this, the SARSA model exhibits a significantly greater variance in its performance.

4 Discussion

According to Fig. 2, we can see that the convergence rate of Q-Learning is greater. However, it is difficult to determine which model performs better during training because both curves are noisy. The success rate for both models is not satisfactory, which is consistent with the additional subplot showing that the average number of pipes passed stabilizes at around 3.5 after a significant number of episodes. This is likely due to the model's curiosity driven by the exploration parameter ϵ. If ϵ had been reduced, better results might have been achieved. In the validation phase, the performance improves compared to training, as the model acts according to the learned table without further exploration, making it less curious and more stable. In contrast, as shown in Fig. 3, both models successfully pass 10 pipes during training for the larger gap scenario. This highlights that the gap value is a significant factor influencing learning performance. Despite the lack of parameter optimization, both Q-Learning and SARSA achieved considerable success, suggesting that the increased gap simplifies the task for the agent, enabling better learning outcomes.

5 Conclusion

In conclusion, our findings demonstrate that Q-Learning generally outperforms SARSA in terms of convergence rate and overall performance in the small gap scenario. However, both models showed limitations, with an average performance in validation that remained below the target of passing 10 pipes. With additional computational resources, we could further optimize the hyperparameters, particularly the exploration rate (ϵ), learning rate (α), and discount factor (γ). Such optimizations could significantly improve the model's performance, potentially allowing it to consistently pass more than 10 pipes in the validation phase.

In contrast, for the large gap scenario, both algorithms exceeded the target of passing 10 pipes by a significant margin. This demonstrates that the gap value between pipes is a key factor affecting learning difficulty, with larger gaps simplifying the task and enabling better performance even without hyperparameter optimization.

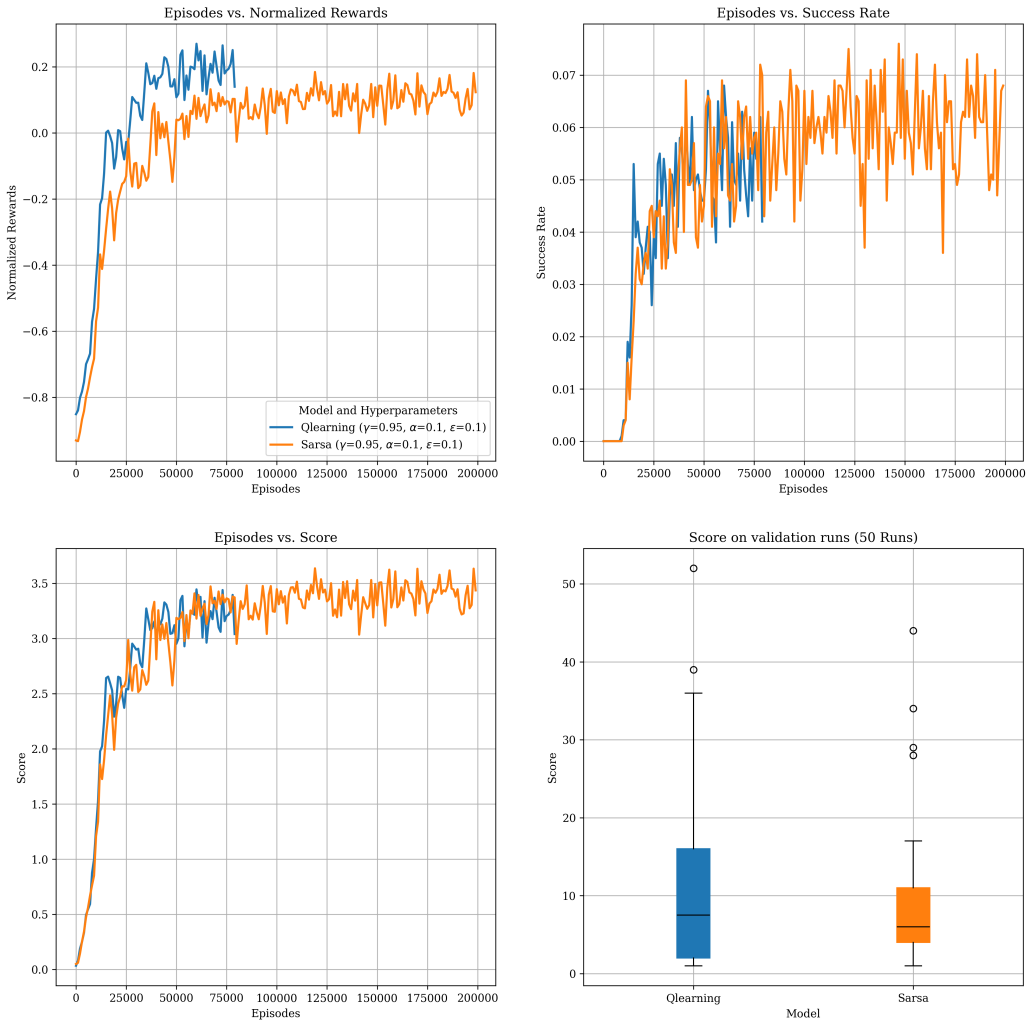

Figure 2: The plots correspond to the small gap scenario (gap = 65). The top left plot shows normalized rewards per episode for the Q-Learning and SARSA models during training. The top right plot shows the success rate (percentage of pipes passed) per episode during training. The bottom left plot shows the average score (number of pipes passed) per episode during training. The bottom right plot shows the validation scores for 50 runs, comparing the performance of Q-Learning and SARSA models.

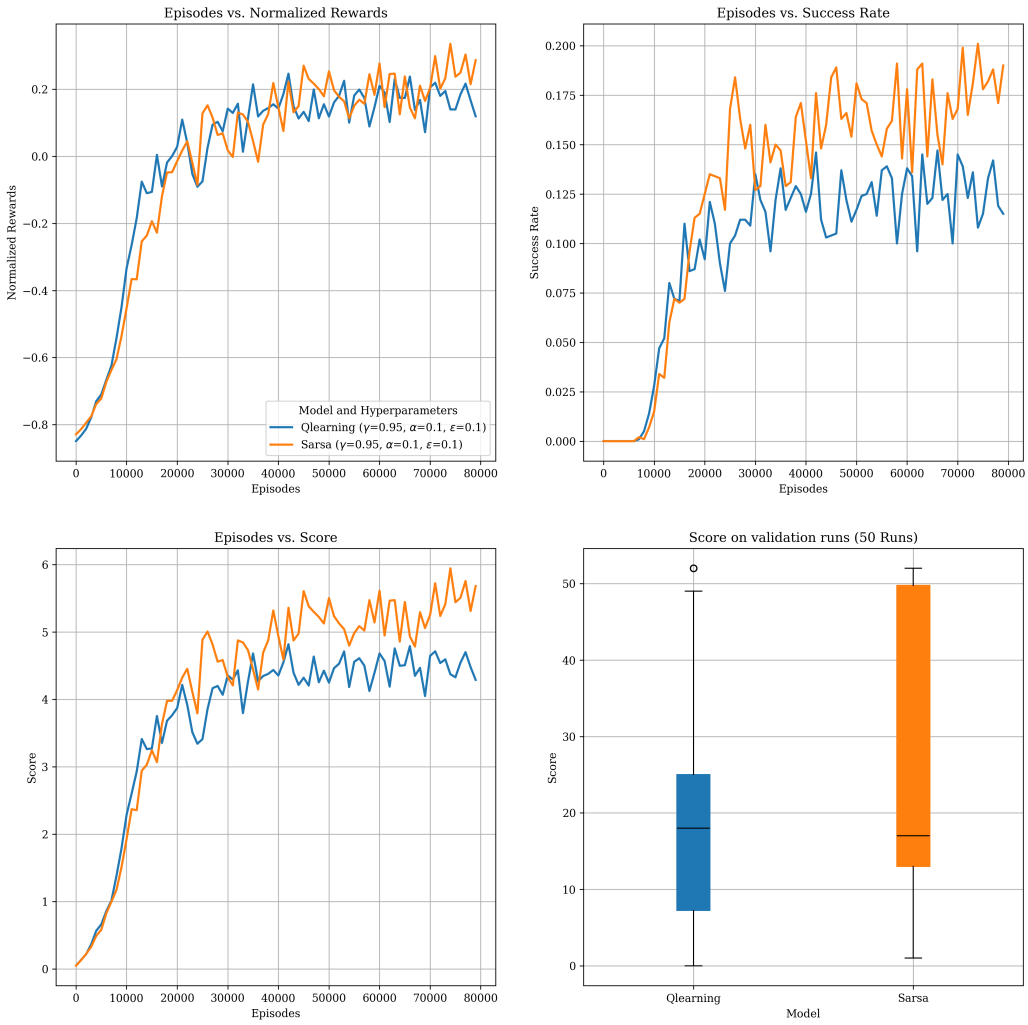

Figure 3: The plots correspond to the small gap scenario (gap = 80). The top left plot shows normalized rewards per episode for the Q-Learning and SARSA models during training. The top right plot shows the success rate (percentage of pipes passed) per episode during training. The bottom left plot shows the average score (number of pipes passed) per episode during training. The bottom right plot shows the validation scores for 50 runs, comparing the performance of Q-Learning and SARSA models.