Using Q-learning to play Flappy Bird

José María Ibarra, a01706970@tec.mx

Abstract—*This report examines the use of Q-learning to train an agent to play Flappy Bird. By applying reinforcement learning techniques, the agent learns through interaction with the game environment. The study analyzes how different parameters, such as learning rate and discount factor, impact the agent's performance in both simple and complex settings. Results show that higher discount factors favor long-term rewards, aligning with the game's goal of survival. Additionally, adjusting the learning rate improves performance in more challenging and varying scenarios.*[1]

Index Terms—Q-learning, Flappy Bird, Reinforcement Learning

1 Introduction

Since computers have been able to process algorithms, different ways of solving problems have emerged. ML in particular has opened a broad path to computer learning, from *simple* supervised learning algorithms to general learning through reinforcement learning algorithms like Q-learning.

For the latter, different approaches may be taken to solve different problems and scenarios. It's relatively easy to model state-based environments where handy agents can act, get feedback, and calculate metrics in relation to their current states. As would be the case for many games, the popular game Flappy Bird can be modeled like so, and Q-learning can be applied to train an agent and obtain inhuman scores. This work focuses on this implementation, recovering previous code and analysis where it was determined a *perfect* bird was possible to train. This focuses on analyzing training performance over different bases, exploring harder environments for the bird.

2 Background

2.1 Reinforcement Learning

Reinforcement Learning (RL) consists of interacting with an environment to maximize cumulative rewards. Unlike supervised learning, RL doesn't rely on labeled data but rather learns from feedback received after taking action. Q-learning enables agents to learn optimal policies through exploration and exploitation of the environment, mapping situations to actions [1]. To learn, the agent first *observes* the environment and chooses what to do from a set of actions S that alter the environment. It then receives a *reward* related to that certain new state and action. The reward can be granted by an agent module or by another external observer. [2] The goal is to find the policy that maximizes rewards over time.

2.2 Q-learning

The Q-learning algorithm was first introduced by Chris Watkins in 1989, and it allows interactive agents to map Markovian environments as actions, i.e., a system where its next state depends only on the current state and action, not on how the environment got to that state. [3] In this implementation, given the current state s and the current action a, we compute $Q(s, a)$ values with:

$$Q(s,a) = (1-\alpha)Q(s,a) + \alpha\left(r(s,a) + \gamma \max_{a'} Q(\delta(s,a), a')\right) \tag{1}$$

1. Code is available at: https://github.com/jmibarrap/Flappy-bird-Q-learning

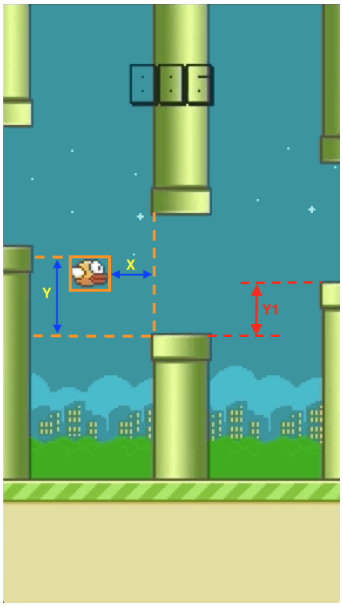


Fig. 1: *Recovered from* [4]. Diagram showing the spatial values for each state. The agent is able to determine the distance to the nearest (current) pipe, and the vertical difference between that and the next pipe. This forms a strategy that allows the agent to decide how to attack the current set of pipes in a way that allows it also to attack the next set successfully. This improves the performance in the long term.

where α is the step size, $\delta(s, a)$ gives the new state s', and γ is the discount factor, which determines the importance of immediate rewards over accumulated rewards.

3 Agent Implementation

The Q-learning implementation to play Flappy Bird uses code from a Github Repository by Tony Xu [4]. If (x, y) determine the current position of the bird, the environment is determined by the set of states S of the agent (bird), which consist of values x_0 and y_0, referring to the bird's horizontal and vertical distances to the nearest pipe, respectively. v for the current flying velocity of the bird and y_1, the difference in height from the next lower pipe to the current lower pipe. This last value was implemented to improve the agent's performance over several training episodes, where the bird may face the scenario where consecutive pipes are highly separated in the y axis [4]. The other state basis and game code come from the work developed by Survy Vaish in [5].

As assessed in [4], the reward function is implemented as:

$$r(s,a) = \begin{cases} 0 & \text{if the bird is alive.} \\ -1000 & \text{if the bird is dead.} \end{cases} \tag{2}$$

This proved to be better for high-performing agents, in contrast to the original proposal where $r(s, a) = 1$ if the bird was alive. This improvement happens because we deny the bird of accumulating points for successful passes, thus denying the bird of minimizing the possible scenario where two sets of pipes are greatly separated. We care more about the bird not dying than actually accumulating points.

4 Training

4.1 Original setting

In the original code, the standard pipe gap size is 100px, more than enough for the 24px tall bird to pass through.

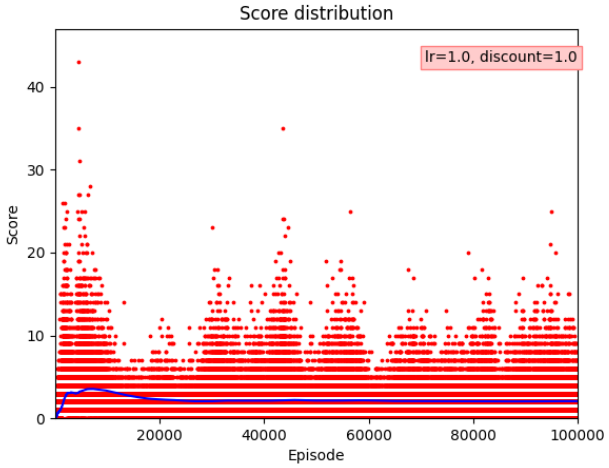


Fig. 2: Score distribution when both $\alpha, \gamma = 1$. No improvement is seen after 100,000 training episodes, with a max score of 43 and a 2.07 average score. The agent keeps seeing every state as new and does not learn.

We train agents with these parameters for 100,000 episodes, set a max score of 1,000, and a stop criterion where training will conclude if 10 consecutive episodes reach the max score, proving consistency to some level. We train varying the values for learning rate α and discount γ. Arbitrary values are chosen from $\alpha = [0.3, 0.5, 0.8]$, and $\gamma = [0.4, 1.0]$ We do not care about $\alpha = 1.0$, for that in our code that would disregard the current Q value and focus only on the new reward and state, essentially considering every state as non-explored (Figure 2).

For the chosen hyper-parameters, we see different training behaviors, where runs with low values for discount (0.4) yielded more chaotic and worse-performing agents. Figure 3 displays the results for the three training runs for $\gamma = 0.4$, and it's clearly seen that agents did not tend to improve after several episodes. Since this setting didn't give high importance to future rewards, the agent kept a steady performance after reaching a *ceiling* near the score of 40, with very low but consistent averages. On the other hand, the results show discount values close to 1 do produce better agents (Figure 4), where the change in learning rate made training convergence faster. All of these three runs reached *consistency* at around the 11,000 episode mark, presumably because of the low-complexity environment, where low learning rates are not necessary in order to explore all the possible states.

These results support the fact that in order to obtain high scores, the agent must prioritize accumulating rewards over time, or in this case, avoiding getting negative rewards in the long run, i.e., avoiding dying. And since the discount factor refers to this specific concept, having high values results in better-trained agents.

4.2 Making it harder to Flap

We now explore the behavior of our agent when changing the game setting. The new setting faces the bird with variable pipe gaps. Every time a new set of pipes is generated, the gap takes values from 80px to 95px, making it harder to flap in narrower spaces, and creating a more complex environment. Training runs are set to 200,000 episodes with $\alpha = [0.8, 0.5, 0.3]$, and $\gamma = 1.0$. Following what we previously learned, giving the bird a more complex game to face should reward runs with lower learning rates.

Whilst results show that these trainings are considerably more chaotic and less uniform (Figure 5), decreasing α does appear to support the hypothesis: lower learning rates obtain better results in more complex environments. Even if these trained agents can't be considered *good* players, these can be due not only to the training parameters but also to some level of insurmountable difficulty in the proposed setting, to which further investigation would be needed. Perhaps it's simply impossible to get more than 3 points in a row with three consecutive 80px sets of pipes, which is a possible state in this setting.

5 Conclusion

While no ground-breaking results were obtained, this implementation reflects how the setting relates to the chosen specifications (parameters) of the agent, concretely for Q-learning in simple contexts. Discount factors close to 1 work better in general because they support long-term rewards, which is consistent with the goal of the game: keeping the bird alive for as long as possible. For the original setting, high learning rates provided better results due to the simplicity of the game. In contrast, lowering the learning rate produced better results when facing the bird to a bigger challenge: a more complex and variable environment.

Further work could reveal if the *complex* setting proposed is actually appropriate for the nature of the game, and if it is possible to train a better agent. If so, a direct improvement to the code would be to tell the agent in each state the gap of the next set of pipes, since this is not considered as the original game keeps this value constant.

References

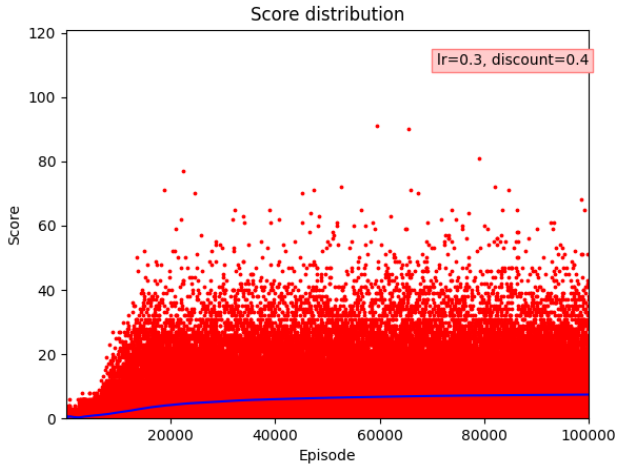
[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA and London, England: MIT Press, 2nd ed., 2015.

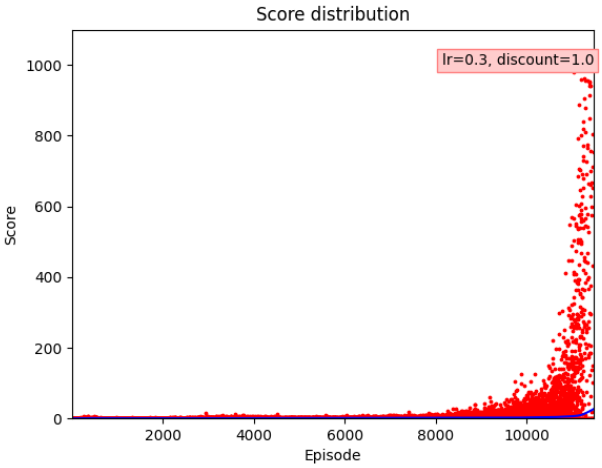
[2] M. G. Mendoza, "Reinforcement learning," 2016. Tecnológico de Monterrey, Available: mgonza@tec.mx.

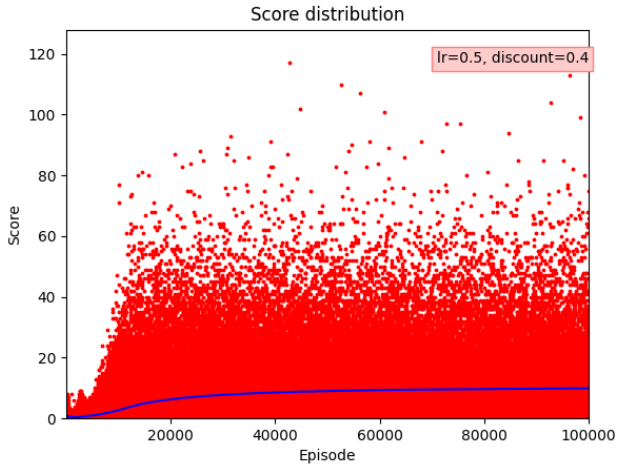
[3] C. J. C. H. Watkins and P. Dayan, "Q-learning," technical note, Centre for Cognitive Science, University of Edinburgh, Edinburgh, Scotland, 1992.

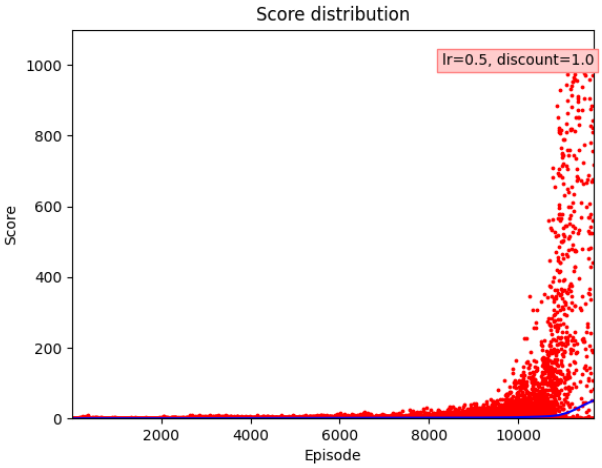
[4] T. Xu, "kyokin78's use reinforcement learning to train a flappy bird which never dies." https://github.com/kyokin78?tab=overviewfrom=2020-12-01to=2020-12-31, 2020. Accessed: 2024-05-23.

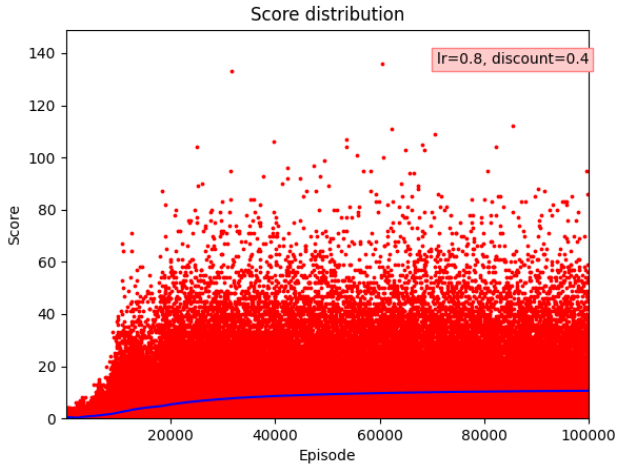
[5] Sarvagyavaish, "Flappy bird rl, flappy bird hack using reinforcement learning." https://sarvagyavaish.github.io/FlappyBirdRL/. Accessed: 2024-05-23.











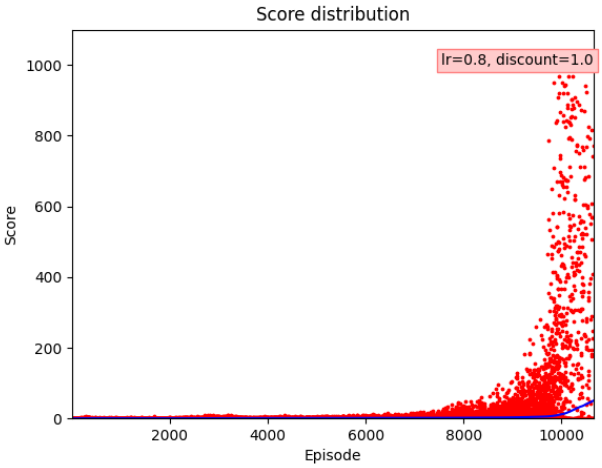


Fig. 3: Plots for training runs with $\gamma = 0.4$, varying learning rate: average scores and maximum scores (avg, max) were (7.42, 110), (9.93, 117), and (10.60, 136), respectively. Increasing the learning rate improves performance slightly, but overall the agent depends more on the discount factor.

Fig. 4: Plots for training runs with $\gamma = 1.0$, varying learning rate: average scores were 26.69, 50.98, and 50.90. Runs converged to the accepted max score at 11469 episodes, 11700 episodes, and 10663 episodes, respectively. $\alpha >= 0.5$ values have similar average scores, but $\alpha \approx 1$ values converge faster.

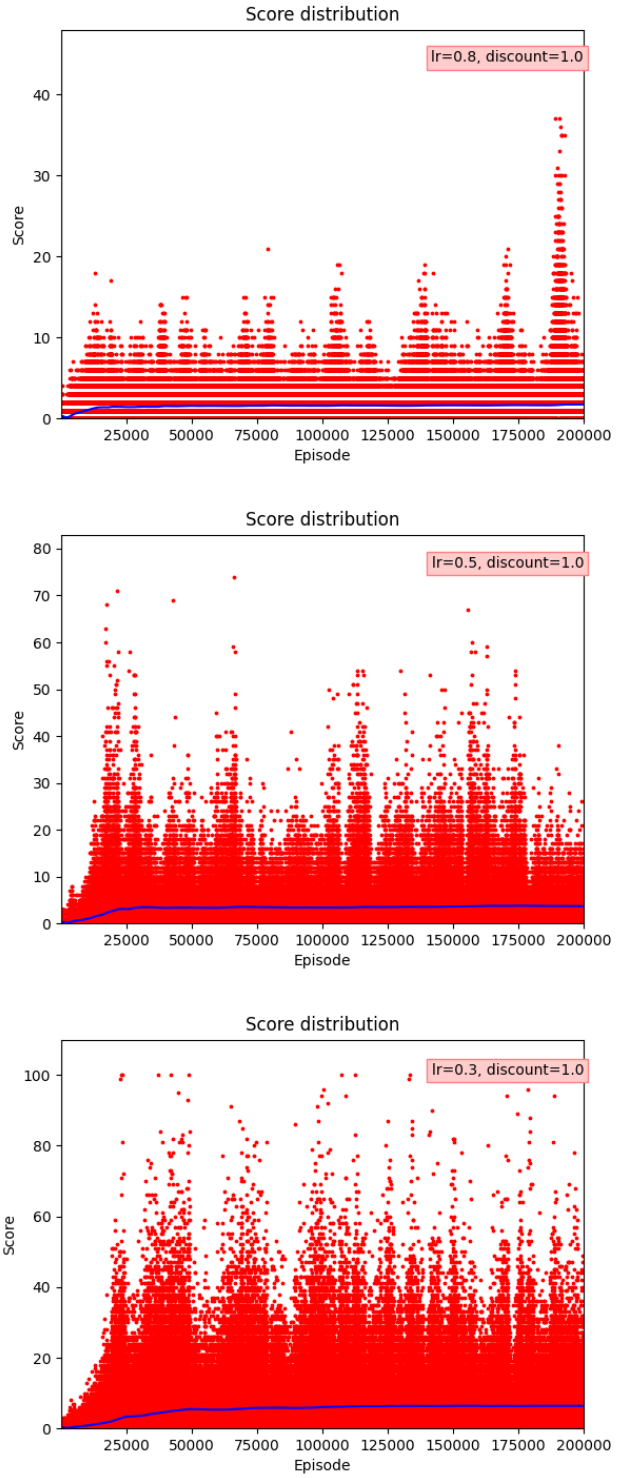


Fig. 5: Plots for training runs with a more complex environment reveal that lower learning rate values yield better-trained birds over the same number of training episodes. Average scores and maximum scores were (1.73, 44),(3.69, 76), and (6.39, 100), respectively.