Implementation of Flappy Bird Agents

Yasmina Hobeika

yasmina.hobeika@student-cs.fr

Abstract. The objective of the assignment is to apply Reinforcement Learning techniques to train two agents and compare their performance based on state-value functions, policy, and raw performance metrics. The agents will be evaluated in the TextFlappyBird-v0 environment, which outputs the distance between the agent and the next pipe. The analysis should also include projections of the agents' performance in the TextFlappyBird-screen-v0 environment, which provides a complete observation of the game screen. Overall, this assignment gives an opportunity to explore and evaluate Reinforcement Learning methods in the context of a game environment, and to gain insights into the performance and limitations of different agents. Code can be found here [1]

Keywords: QL Learning, Expected Sarsa, Agents. Hyperparameters

1 How did you choose agents for the task?

In the TextFlappyBird-v0 environment, the state of the agent is a tuple that contains the x and y distance from the next pipe. To store each unique state, it is more interesting to use a dictionary with the tuple as the key and the index of the unique state as the value. The environment offers two actions: 1 for flapping and 0 for remaining idle, and the agent receives a reward of 1 at each time step with the goal of maximizing the reward. The game ends if the agent collides with the pipe, flies too high, or falls to the ground. Additional information provided by the environment includes the agent's score, position on the screen, and distance from the pipe. By using the agent's distance from the pipe to train an agent through reinforcement learning, it is possible to optimize its performance and keep it alive for as long as possible.

The Q-Learning and Expected SARSA algorithms were chosen to model the two agents because of their Temporal Difference property, which allows learning to occur during the running of episodes instead of waiting until the end. This property is especially crucial for games like Flappy Bird, which can have long episode runtimes. By implementing Temporal Difference methods, the agents learn actively at each step within the episode. These algorithms can also learn off-policy, allowing them to learn actively from the environment. While defining an optimal policy for Flappy Bird could be complex and tedious, it is more interesting to have the agent learn a policy actively from the environment. The main difference between Q-Learning and Expected SARSA is that the former chooses the maximum state-action value pair, while the latter considers the likelihood of each action occurring under the current policy. This adjustment in an

otherwise similar algorithm could have interesting implications for training, and it is worth observing the differences between the two.

2 How are the two implemented agents different?

2.1 Hyperparameter Tuning

To avoid infeasible training durations caused by the possibility of the agent running indefinitely, a maximum score was set to automatically kill the agent once achieved. At the start of the training, hyperparameters were chosen arbitrarily to get an initial understanding of the problem. Initially, both agents were implemented with a fixed epsilon, but this led to the agents exploring too much and not maximizing performance after training. Therefore, the algorithms were later modified to follow an epsilon decaying algorithm, which allowed the agents to exploit the greedy action from their learned policy once they approached an episode with a close to optimal policy. Hyperparameter tuning was then performed to find the optimal parameters for each algorithm that could maximize convergence time and performance. Due to the significant computational time required for training, the maximum score was limited to 1000, and the number of episodes was restricted to 12,000 per run. When tuning hyperparameters, two specific parameters were focused on: epsilon and step size. To find the optimal value for epsilon, an initial estimate was used to strike a balance between allowing the agent to explore for a few thousand episodes while still converging within a reasonable training time. This resulted in setting the starting epsilon to 1, the epsilon decay to 0.999, and the epsilon minimum to a very small value such as 1e-08. As for step size, seven different values ranging from 0.01 to 1.0 were tested for both Q-Learning and Expected SARSA to determine the optimal step size. In **Fig. 1**, two graphs are plotted, one for Q Learning and one for Expected Sarsa, where each curve is the moving average of the scores for each step size. For step size equal to 0.01, 0.1 and 1.0, we can see that the two agents don't behave well compared to step size in the range of 0.2 till 0.5. After performing tuning for 12,000 episodes, we can see that the best step size for Q Learning was found to be 0.4, while that for Expected SARSA was found to be 0.2.

2.2 Convergence Time

The Q-Learning agent demonstrated a quicker learning curve in achieving higher scores in approximately 2000 episodes compared to the Expected SARSA agent. Despite this, both agents converged to the maximum set score in a relatively similar time interval. Notably, during the last few episodes, the Expected SARSA agent had a lower variance in scores compared to the Q-Learning agent, as shown in **Fig. 2.** Convergence time of Q Learning agent (Left) and Expected Sarsa agent (Right), indicating that while the left agent could quickly achieve higher scores, it was more likely to fail compared to the right agent.

2.3 State-Value Function

Both agents have successfully played and solved the same game, resulting in their state-value functions to have similar shapes as shown in **Fig. 3.** State-Value Function of Q Learning agent (Left) and Expected Sarsa agent (Right). When the dx approaches zero, there are only limited dy values that would result in a reward for the 2 agents, which means that the bird did not have enough time to adjust between the height of the 2 pipes and died immediately. When dx is between 1 and 9, we can see that the range of dy values that result in a reward with Q-learning agent is bigger than the ones with Expected Sarsa. The expected SARSA agent adopts a more cautious approach in this case. When the dx is greater than or equal to 10, the bird may still be passing through the previous pipe and must avoid a collision, that's why we also see limited dy values that would result in a reward. At this location, the Q-Learning agent displays a smoother value function compared to the expected SARSA agent. Most values are around 20 for Q Learning and around 15 for Expected SARSA. These are the times the birds could safely cross the pipes and move to the next pipe.

2.4 Policy Plots

The figure presented in **Fig. 4.** Policy plot of Q Learning agent (Left) and Expected Sarsa agent (Right)demonstrates that the Q-learning agent has come closer to the optimal policy function when the X distance is greater than or equal to 10, resulting in a smoother policy at that location. The unobserved states located in the 4 corners of the plots correspond to the position of the pipes, which explains their absence from the agents' learning. The unexpected number of inconsistencies in the middle, particularly between the agents, suggests that there is not a singular winning strategy for the game. The expected SARSA agent appears to have learned more irregularities than the Q-learning agent. It is intriguing to note that neither agent has been able to learn the ideal policy when the dy value is equal to 11. i.e., to flap, which may be due to the proximity of two consecutive pipes, one with a very low position of hole and the other with a high position of hole.

3 How are the two versions of the TFB environment different? What are the main limitations of using the same agent for the TextFlappyBird-screenv0?

The TextFlappyBird-v0 environment provides a state representation consisting of only two integers, whereas the TextFlappyBird-screen-v0 environment offers a state representation in the form of a 20x15 sparse 2D array (the height and width of the screen), where 1 represents the flappy bird and 2 represents the structure of the pipe. Although both environments work similarly, i.e., the score is stored in the same way, agent step function is the same… the difference of the state is the primary distinguishing factor between them, which can result in some agents being incompatible with both. For instance, the agents implemented in this assignment to tackle the TextFlappyBird-v0 environment cannot be employed to solve the TextFlappyBird-screen-v0 environment

because we stored all state-action values in an array, which is not feasible when the state is represented by a 15x20 array due to the enormous number of possible states. I tried even to change the 2D array into a tuple state to match the representation of the state in my agents but it didn't work. Therefore, to address the TextFlappyBird-screen-v0 environment, we must utilize techniques such as value function approximation, including linear or deep learning approximations.

4 With an implementation of the original flappy bird game environment available, can the same agents be used?

Using the agents in the original Flappy Bird environment might pose memory-related issues. It could result in immediate kernel crashes and would require highly powerful specifications to function correctly. Additionally, the problem arises because the distance observation from the pipe is continuously outputted as floats, rather than state array position. This continuous setting may require approximation, or else there could be an infinite number of state spaces theoretically. As a result, the agents, as they are, cannot be utilized.

References

1. https://colab.research.google.com/drive/1pfhIL-VeKRyr5xwO9yOVfa9AJcIZdeeCt?usp=sharing

5 Appendix

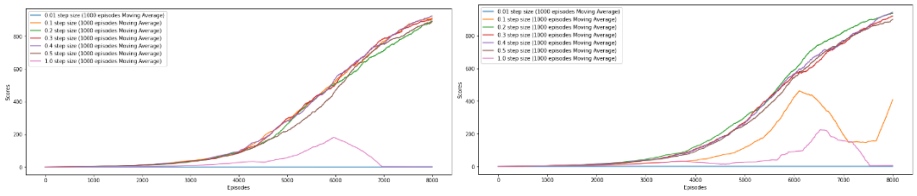

Fig. 1. Hyperparameter tuning of Q Learning step size (Left) and Expected Sarsa step size (Right)

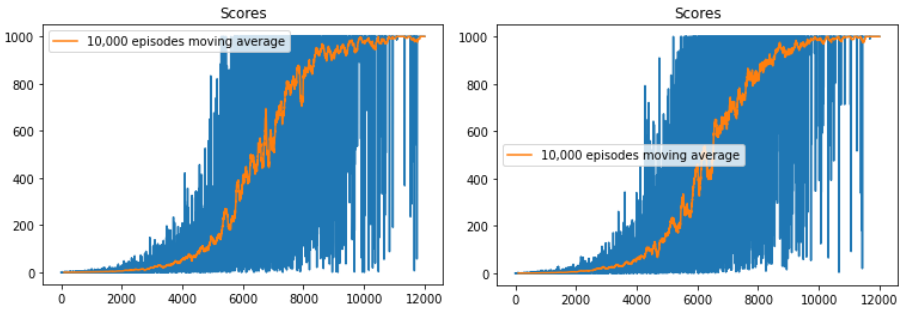

Fig. 2. Convergence time of Q Learning agent (Left) and Expected Sarsa agent (Right)

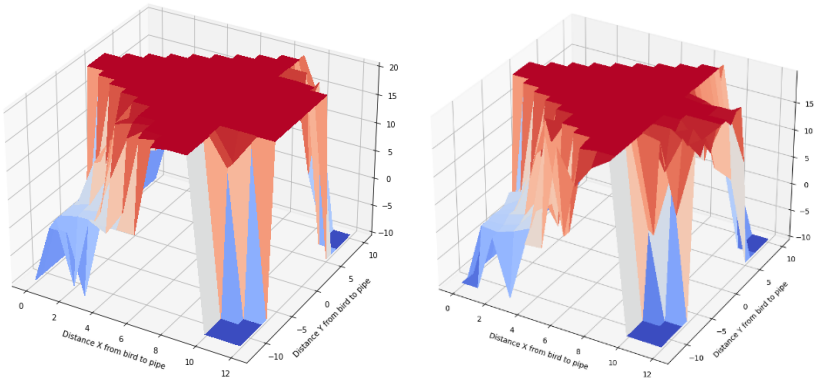

Fig. 3. State-Value Function of Q Learning agent (Left) and Expected Sarsa agent (Right)

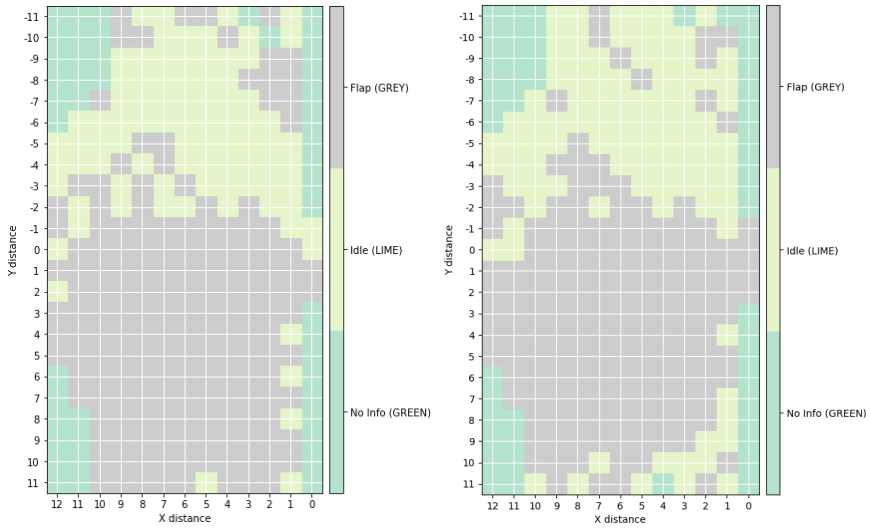

Fig. 4. Policy plot of Q Learning agent (Left) and Expected Sarsa agent (Right)