RL Assignment - Text Flappy Bird

Demy Ugo
ugo.demy@student-cs.fr

CentraleSupélec, MDS - SDI

Abstract. This report presents the implementation and comparison of two reinforcement learning agents for playing Text Flappy Bird: Expected SARSA and Q-learning. The objective of the agents is to maximize the cumulative reward, which is defined as the number of times the agent survives the game. Both agents are trained using the same environment and parameters, and their performances are evaluated in terms of cumulative reward and convergence time. In addition, an exploration part was realised using a Deep Q-Network agent with convolutions to learn directly from the screen render.

https://github.com/Nibleash/RL_TextFlappyBird

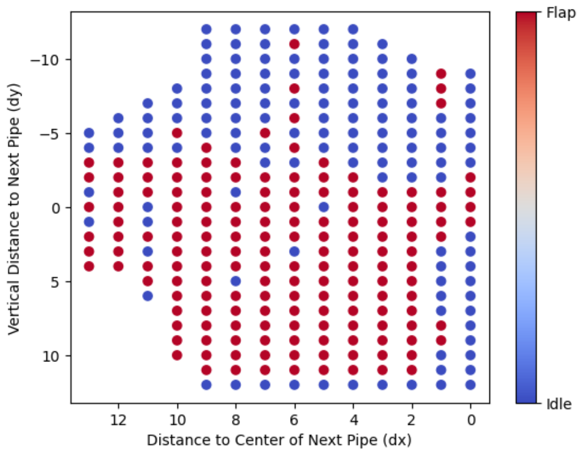
1 Expected SARSA Agent

The first agent was built using the *TextFlappyBird-v0* environment. The Expected SARSA agent is a model-free, on-policy reinforcement learning algorithm that estimates the Q-value function using a linear function approximator. I used a *collections.defaultdict* to store the Q-values corresponding to each distance. Each one of them is then updated using the expected value of the next state-action pairs, rather than the maximum Q-value as in Q-learning. The agent was trained following an epsilon-greedy policy for 25000 episodes and took 82.54 seconds. The following parameters were found to be the most appropriate ones:

- Epsilon: 0.07
- Learning rate: 0.4
- Discount factor: 0.9
- Epsilon decay: 1.0
- Learning rate decay: 1.0
- **Mean cumulative reward: 105.0244**

The Expected SARSA algorithm is known to perform well in environments with discrete state and action spaces and as expected, it is computationally efficient and converges to an optimal policy in a reasonable amount of time. Here is the action heatmap showing the decision taken by the algorithm depending on the bird's position:

Fig. 1. State-Value plot - Expected SARSA



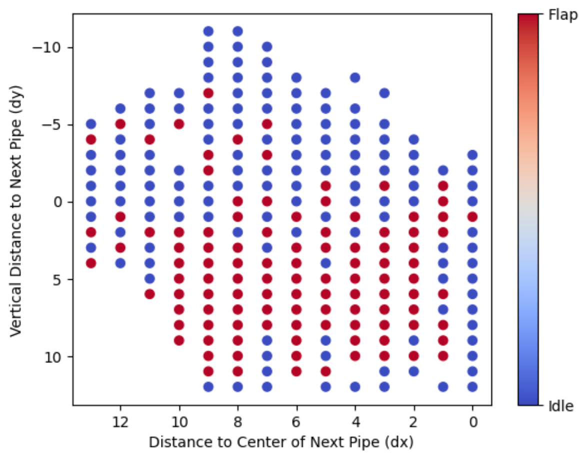
2 Q-learning Agent

Q-learning is another popular model-free, off-policy reinforcement learning algorithm. The Q-values are updated using the maximum Q-value of the next state-action pairs. The agent was also trained following an epsilon-greedy policy for 25000 episodes and took 659.09 seconds. Compared to the Expected SARSA agent, the hyper-parameters that changed were the *learning rate* from 0.7 to 0.75, the *epsilon* from 0.04 to 0.05 and the *decays* from 1.0 to 0.99999. The Q-learning agent reached a mean cumulative reward during its training of **674.56352**.
Q-learning is known to perform well in the same environments as Expected SARSA but is usually less stable than this one since it updates the Q-values with a maximum instead of an expected value. However, fine-tuning the algorithm also made me realise its sensitivity to hyper-parameters such as the learning rate, discount factor, and exploration rate. Here is the action heatmap showing the decision taken by the algorithm depending on the bird's position:

3 Exploration: DQN Agent

This third agent has been implemented more as an exploration part rather than to study its raw performances. Indeed, when the two previous ones could have some difficulty to play in the *TextFlappyBird-screen-v0* environment, the DQN agent can take advantage of convolutionnal neural networks to solve such problems. The Deep Q-Network (DQN) agent is a model-free, off-policy reinforcement learning algorithm that uses a neural network as a function approximator for estimating the Q-value function. The agent updates its neural network using a combination of stochastic gradient descent and experience replay. DQN is a powerful algorithm for learning complex policies in high-dimensional state

Fig. 2. State-Value plot - Q-Learning



and action spaces. DQN has been successfully applied to various reinforcement learning problems, including playing Atari games, and has shown to achieve state-of-the-art performance. Indeed, this agent was really interesting to train since it achieves really good scores on TFB (over 15 in general and more than 30 on the good runs) and is quite satisfying to look at while playing as you can see on the notebook. However, one of the main drawbacks of such algorithms is its training time and the number of episodes required for such complex environments. Mine took $\tilde{3}0$ minutes to complete 5000 episodes and was starting to get pretty good results. However, I had to stop its training due to lack of time and resources.

4 Results & Conclusion

Both agents were trained using the Text-Flappy-Bird environment and even if they do have similar action heatmap, the Q-learning agent (more than 600 cumulative reward on average) tends to perform better than the Expected SARSA one (100 cumulative reward on average) on the long run. However, while both of them would struggle to adapt to an original Flappy Bird game environment and would require prior computations to get the bird-pipe distance, the Deep Q-Learning agent can directly play on it and achieve satisfying results, even better if well-trained.

In conclusion, we mainly implemented and compared two reinforcement learning agents for playing Text Flappy Bird: Expected SARSA and Q-learning. Both agents achieved convincing cumulative rewards and converged to optimal policies in a reasonable amount of time. The Expected SARSA agent agent showed faster convergence time than the Q-learning agent, but this last one showed a more stable learning curve and higher robustness to noise in the environment.

References

1. Kevin Chen: Deep Reinforcement Learning for Flappy Bird. Stanford paper.
2. V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattle, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. Nature, 518(7540):529-533, 2015.
3. Tai V., Tran L.: FlapAI Bird: Training an Agent to Play Flappy Bird Using Reinforcement Learning Techniques. ResearchGate (2020)