Text Flappy Bird

Maya Janvier

CentraleSupélec

1 Introduction

The goal of this project is to try different reinforcement learning agents on the Text Flappy Bird (TFB) game. The environment of TFB exists in two setups: one with the state being the distance to the center of the pipe, the other giving the complete screen render of the game. I decided to use the first one, as it is less costly in memory usage, in the configuration height=15, width=20, pipe gap=4.

2 Agents

I implemented two different agents: one on the Monte Carlo side, namely **MC Control**, and one on the temporal difference side, namely **SARSA(λ)**. Both are model-free reinforcement learning algorithm: they learn directly from interactions by observing state transitions and rewards. Their policy are both updated using the **epsilon-greedy algorithm**. I used a **decaying** ϵ as well to increase the performances. As an agent can become too good at the game, a **maximum reward** (1500) is set to avoid too large computational times. Finally, we will perform a **tuning** on two hyper-parameters they share: the **step-size** and the **starting** ϵ.

MC Control In MC control, updates are made based on the complete returns obtained at the end of each episode. This makes MC Control a heavy agent to train as it needs to store and process the complete returns for each episode before updating action values.

SARSA(λ) This algorithm learns online, updating its estimates at each time step of an episode. This makes SARSA(λ) a lighter algorithm to train. It uses **eligibility traces**: when an action is taken in a state, the eligibility trace for that state-action pair is increased. This trace decays over time, but not instantly. This process credits future rewards to preceding states and actions. This enables the algorithm to propagate credit backward through time, aiding in credit assignment.

3 Performances

3.1 Hyperparameters tuning

Figure 1 represents the evolution of the averaged reward (window size 100) along 10000 episodes for both agents. I performed a grid search with the following setup:

- step size: [0.1, 0.4, 0.8, 1]
- ϵ start: [0.05, 0.1, 0.99]
- ϵ decay: 0.999 (MC), 0.99 (SARSA)
- ϵ min: 1e-4 (MC), 0.01 (SARSA)
- γ (discount): 0.95 (MC), 0.99 (SARSA)
- λ (SARSA): 0.5

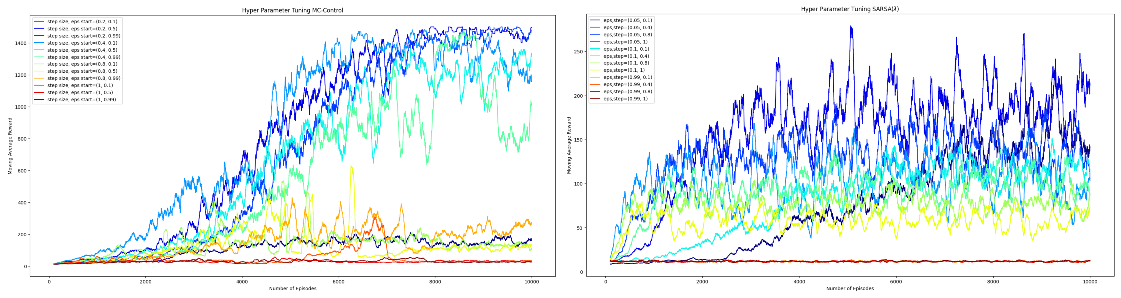

Fig. 1. Tuning: MC Control (left), SARSA(λ) (right)

MC Control yields better performances than SARSA when fine-tuned as it reaches a maximal reward of 1500 vs 250 for SARSA. This value is "fake" and could become much bigger as we increase the maximum reward authorized. The learning of SARSA is quicker in general as it reaches a plateau after 2000 episodes. This may indicate a sub-optimal policy due to the weak parametrization, whereas MC Control has a higher sensitivity to the parameters. Trying the same setup with $\lambda = 0.9$ did not yield better results however and a finer tuning may be needed on other parameters.

3.2 State-Value and Policy

Figure 2 and Figure 3 depict the training rewards, state value function and policy of the best agents of each method, the maximum training reward being set at 2000. We see that their policies are pretty similar. Both never explore the corners, probably because of the pipes. One main difference is that when far from the next pipe (x=12) and at the level of the pipe(y $\in$ [-2,2]), MC idles wheras SARSA flaps. The maximum state value of SARSA is higher (60) and more reached than the MC's one (20), however it covers less states than MC. SARSA state value is also smoother.

3.3 Play in a new environment

How well our agents perform in a new environment ? I set the height and width to 10, and changed the pip gap as it was the more discriminant feature for

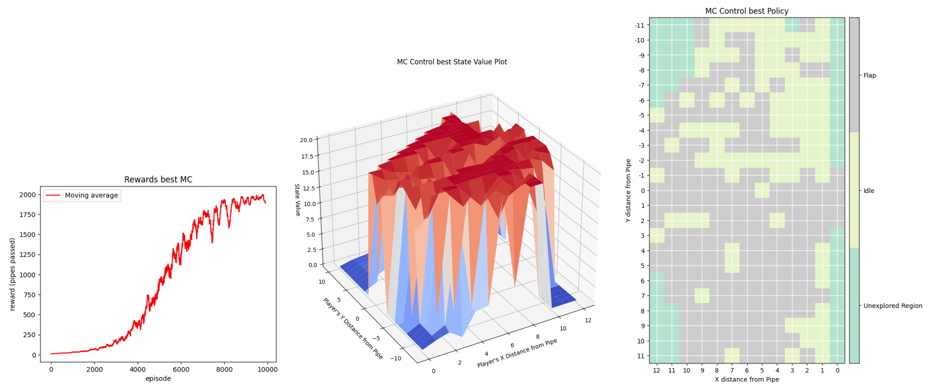

Fig. 2. Best MC Control agent

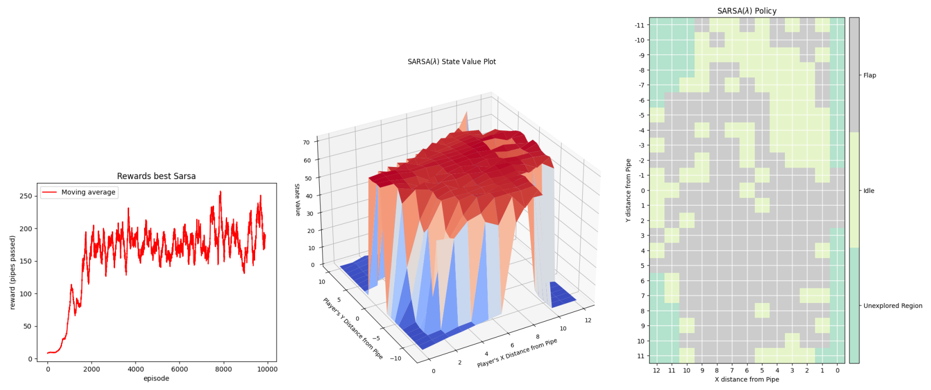

Fig. 3. Best SARSA(λ) agent

the performances. The mean, min and max of each agent of 1000 episodes are gathered Figure 4. The maximal playing reward is set at 10000 for MC Control, but it could probably play infinitely when min=max=10000.

We see that MC Control is more robust to the change of screen size, but both perform badly when the pipe gap is decreased, as their policy are probably not precise enough to manage smaller gaps and also because the game is more difficult.

4 Discussions

The second version of TFB is more precise, however certainly more computationally very expensive, as we will need to store almost empty matrices instead of a float, especially for MC Control as it needs to store all of the states before updating its estimates. SARSA may be more adapted.

Fig. 4. Playing in new environment

Agent	**Height/Width**	**Pipe gap**	**Mean**	**[Min,Max]**
MC Control	15/20	4	10000	[10000,10000]
SARSA(λ)	15/20	4	257	[4,2510]
MC Control	10/10	4	10000	[10000,10000]
SARSA(λ)	10/10	4	112.6	[3,986]
MC Control	10/10	3	12.64	[6,76]
SARSA(λ)	10/10	3	9.8	[5,61]
MC Control	10/10	6	10000	[10000,10000]
SARSA(λ)	10/10	6	194.7	[3,1160]

Using the same agents in the original Flappy bird game environment could also be tricky in terms of memory. They would also have to deal with continuous observations, and we will need tiles.

5 Conclusion

I implemented two agents that were trained on TFB and played the game in different configurations. MC Control performs better in the end probably because of a better parametrization, whereas SARSA needs more tuning. MC Control is however less flexible when it comes to more complex environments such as the screen rendering version due to heavy memory costs.