Reinforcement Learning for Text Flappy Bird: Monte Carlo and SARSA(λ)

Théau d'Audiffret

CentraleSupélec

https://github.com/theaudaudiffret/RL

1. Environment

This report explores the application of reinforcement learning (RL) techniques to solve the Text Flappy Bird (TFB) game. Specifically, we compare two RL algorithms: Monte Carlo (MC) and SARSA(λ). These methods are evaluated in the *TFB-v0* environment which returns the distance of the player from the center of the next upcoming pipe. The environment *TBF-screen-V0* returns the compelete screen observation. The environment used gives less information about the game but is easier to explore and exploit for the agent. To use the second environment Deep Reinforcement Learning could be needed to extract features from the screen display or classical RL but with feature extraction first. For example the same agents could be used after computing the feature position of the bird, center of the pipe distance.

2.Experimental setup

2.1 Monte Carlo Method

The implemented agent is a **Monte Carlo Control Agent** using an *incremental update* approach. This agent learns an optimal policy through episodic interactions with the environment, updating its action-value function $Q(s,a)$ based on full episode returns. The update rule follows the Monte Carlo control method with a learning rate α:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left(G - Q(s,a)\right)$$

where G is the total discounted return from time step t:

$$G = \sum_{k=0}^{T-t} \gamma^k r_{t+k}$$

with γ being the discount factor. The optimal state-value function $V(s)$ is then estimated as:

$$V(s) = \max_a Q(s,a).$$

To balance exploration and exploitation, the agent follows an **ϵ-greedy policy**, where with probability ϵ it selects a random action and with probability $1-\epsilon$ it chooses the action that maximizes $Q(s,a)$. The value of ϵ decays over time to encourage more exploitation as learning progresses.

2.2 SARSA(λ)

The implemented agent is a **SARSA(λ) Agent**, which uses *temporal difference learning* with eligibility traces. This method is an extension of the SARSA algorithm, incorporating *trace-based updates* to enhance learning efficiency. The Q-value function $Q(s,a)$ is updated according to the following rule:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\delta E(s,a)$$

where α is the learning rate, and δ is the temporal difference (TD) error:

$$\delta = r + \gamma Q(s',a') - Q(s,a)$$

with r being the reward, γ the discount factor, and $Q(s',a')$ the estimated value of the next state-action pair. The eligibility trace $E(s,a)$ is updated as:

$$E(s,a) \leftarrow E(s,a) + 1$$

and decays at each step:

$$E(s,a) \leftarrow \gamma\lambda E(s,a)$$

where λ is the trace decay parameter, controlling the balance between one-step TD updates ($\lambda = 0$) and Monte Carlo updates ($\lambda = 1$). The agent follows an **ϵ-greedy policy**, ensuring a trade-off between exploration and exploitation, with ϵ decaying over time to favor exploitation as learning progresses.

3. Comparison of the 2 agents

The Monte Carlo agent struggles to learn during 40,000 steps approximately and then starts to learn very quickly. For the same basic parameters (ϵ=0.2, γ=0.8, α=0.1), the Sarsa λ model learns more gradually but slower (not in time but in steps required) than MC since at the end of 40,000 steps the medium cumulative reward is 25 versus 200 for MC. However, Sarsa does not need the end of each episode to adjust. The Sarsa λ learning, as explained in [Sutton and Barto, 2020], is a good trade off between a Sarsa agent with temporal difference and faster convergence and a Monte-Carlo agent. Unlike standard SARSA, which only updates the current state-action pair, SARSA(λ) propagates updates backward to multiple past state-action pairs using eligibility traces. This means that useful learning signals spread through the value function more quickly, improving sample efficiency. With $\lambda = 0$ we go back to Sarsa and with $\lambda = 1$ it is Monte Carlo.

Parameter analysis - see the notebook for the experiments:
- ϵ: A higher epsilon make the algorithm converge faster at first, but then it is necessary to introduce epsilon decay to induce a convergence. For MC, a high starting epsilon leads to faster convergence but less stability in the training. In the end, a lower epsilon over performs. On the other hand, for Sarsa(λ) a higher epsilon prevent from exploiting the best solutions.
- γ: For MC a higher gamma leads to much better results since the agent learns how to anticipate, in the graph we can see precisely when this phenomena happens since the training is not linear. For Sarsa(λ) it is the same and it is even more important to compund the future highly since the estimation of the present reward is not always enough to judge of the quality of a situation.
- λ: For the Sarsa Agent a high lambda leads to much better results, this is due to the fact that FB needs a strong temporal consitency and thus a high lambda favors the use of multiple steps for the update.

As a result SARSA(λ) may be more sensitive to hyperparameters but also converges more continuously than MC that converges by steps and stagnation.

4. Results

In this section we present the results of the training of our best models.

The state-value function $V(s) = \max_a Q(s,a)$ was computed for each agent. Figure 1 shows the agent's state-value function.

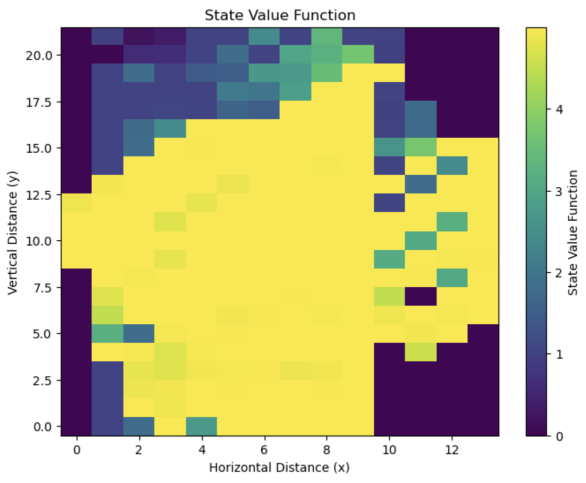
Figure 2 shows the shape of the training curve for both agents.

The state-value function is similar for both agents and it is interesting to see that for both agents the strategy is to be above the center of the pipe before passing through it which is coherent to a real game made by a human.

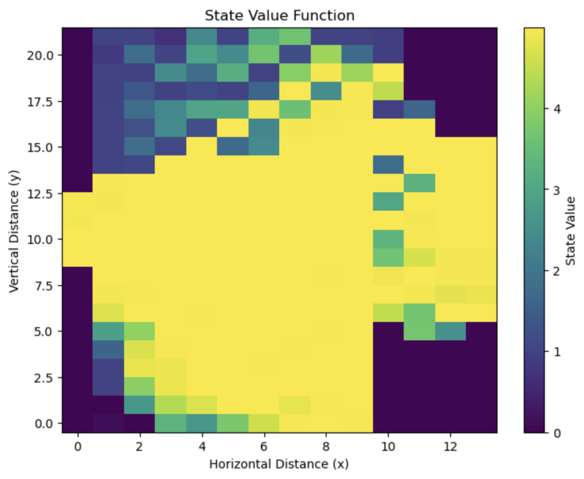
With $height = 15$, $width = 20$, $pipegap = 4$, and for a relatively similar amount of time. We created a MC agent and a Sarsa(λ) with good hyperparameters and the results are presented in Figure 3. The 2 agents are pretty close since we used a high λ for Sarsa but overall the MC approach seams better and more stable in the training.

References

Sutton and Barto, 2020. Sutton, R. S. and Barto, A. G. (2020). *Reinforcement Learning: An Introduction.* MIT Press, 2nd edition.

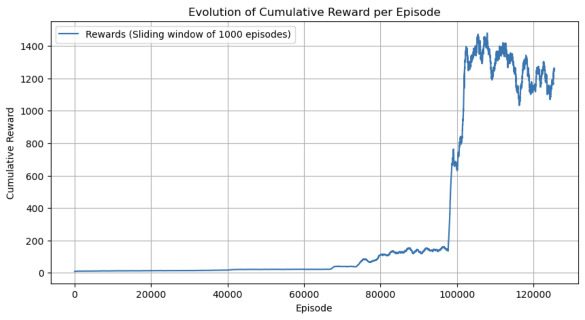


(a) State-value function for Monte Carlo agent.

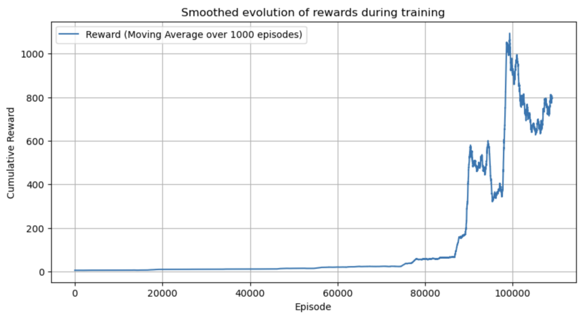


(b) State-value function for SARSA(λ) agent.

Fig. 1: Comparison of state-value functions for Monte Carlo and SARSA(λ) agents.

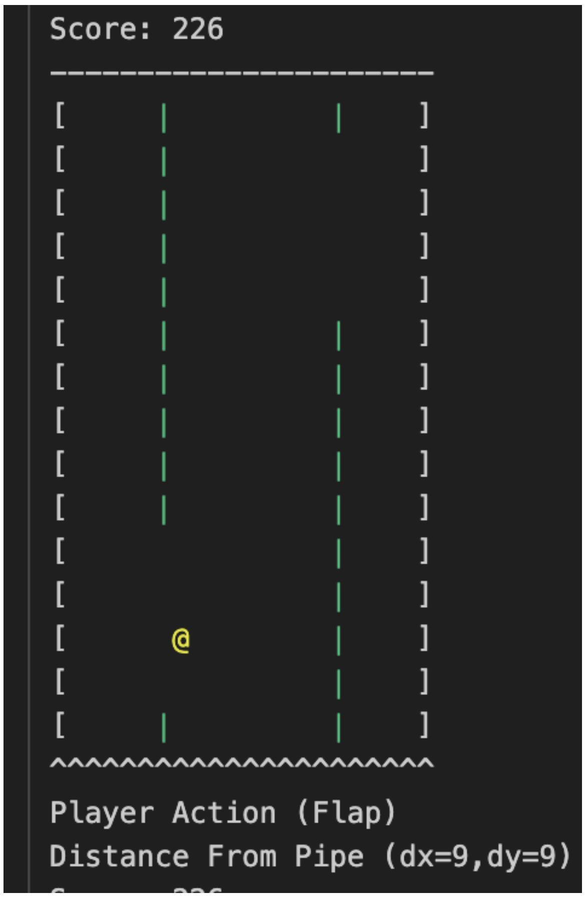


(a) Cumulative rewards during training for MC (smoothed).



(b) Cumulative rewards during training for Sarsa λ (smoothed).

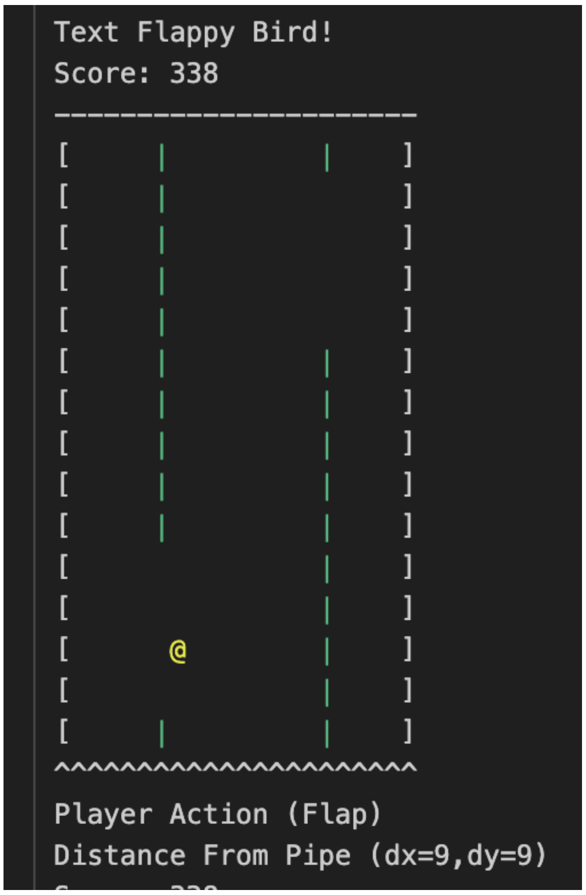
Fig. 2: Comparison of cumulative rewards and scores evolution.



```
Score: 226
----------------------
[     |         |    ]
[     |              ]
[     |              ]
[     |              ]
[     |              ]
[     |         |    ]
[     |         |    ]
[     |         |    ]
[     |         |    ]
[     |         |    ]
[               |    ]
[               |    ]
[      @        |    ]
[               |    ]
[     |         |    ]
^^^^^^^^^^^^^^^^^^^^^^

Player Action (Flap)
Distance From Pipe (dx=9,dy=9)
```

(a) Scores during training for MC



```
Text Flappy Bird!
Score: 338
----------------------
[     |         |    ]
[     |              ]
[     |              ]
[     |              ]
[     |              ]
[     |         |    ]
[     |         |    ]
[     |         |    ]
[     |         |    ]
[     |         |    ]
[               |    ]
[               |    ]
[      @        |    ]
[               |    ]
[     |         |    ]
^^^^^^^^^^^^^^^^^^^^^^

Player Action (Flap)
Distance From Pipe (dx=9,dy=9)
```

(b) Scores during training for Sarsa λ

Fig. 3: Comparison of cumulative rewards and scores evolution.