Individual assignment report: Flappy Bird with Reinforcement Learning

Clement Wang[1,2]

[1] CentraleSupélec, University of Paris-Saclay, France
[2] Master MVA, ENS Paris-Saclay, France

Abstract. This report presents the application of reinforcement learning (RL) methods to the Text Flappy Bird (TFB) game environment. The goal of this assignment is to implement and compare two RL agents, namely Monte Carlo Control and Sarsa(λ), to tackle the TFB environment. I only chose to solve the simplified version with the distance to the pipe for simplicity. The report discusses the implementation details, experimental setup, results, and insights gained from the comparative analysis of these two RL algorithms in the context of TFB. The code is available here: https://github.com/clementw168/Flappy-bird-RL.

Keywords: Reinforcement learning · Monte Carlo Control · Sarsa(λ)

1 Experimental setup

1.1 Monte Carlo Control and Sarsa(λ)

Monte Carlo Control (Fig. 1a) is a reinforcement learning technique that learns optimal policies by directly experiencing the environment. It collects episodes of interaction, consisting of sequences of states, actions, and rewards, and then updates its policy based on the observed returns. It iteratively improves its policy to maximize long-term rewards by averaging the returns obtained from multiple episodes.

Sarsa(λ) (Fig. 1b) is another reinforcement learning algorithm that combines the temporal difference learning approach with eligibility traces. It learns action values for state-action pairs, updating these values based on the transitions experienced during interactions with the environment.

Decaying Epsilon Greedy exploration For both algorithms, I implemented a decaying Epsilon Greedy exploration strategy to balance exploration and exploitation during training. Epsilon Greedy exploration involves selecting a random action with probability ϵ and selecting the action with the highest estimated value otherwise. To encourage exploration early in training and exploitation later on, I decayed the value of ϵ over time.

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
  $Q(s,a) \leftarrow$ arbitrary
  $\pi(s) \leftarrow$ arbitrary
  $Returns(s,a) \leftarrow$ empty list

Repeat forever:
  (a) Generate an episode using exploring starts and π
  (b) For each pair s,a appearing in the episode:
    $R \leftarrow$ return following the first occurrence of s,a
    Append R to $Returns(s,a)$
    $Q(s,a) \leftarrow$ average$(Returns(s,a))$
  (c) For each s in the episode:
    $\pi(s) \leftarrow \arg\max_a Q(s,a)$

(a) Monte Carlo pseudo code

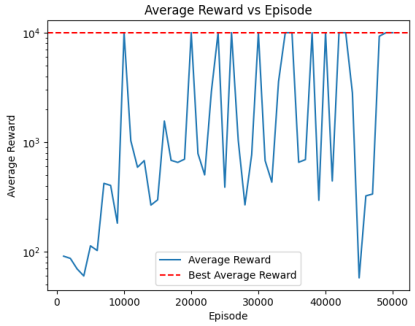
Initialize $Q(s,a)$ arbitrarily and $e(s,a) = 0$, for all s,a
Repeat (for each episode):
  Initialize s, a
  Repeat (for each step of episode):
    Take action a, observe r, s'
    Choose a' from s' using policy derived from Q (e.g., ε-greedy)
    $\delta \leftarrow r + \gamma Q(s',a') - Q(s,a)$
    $e(s,a) \leftarrow e(s,a) + 1$
    For all s,a:
      $Q(s,a) \leftarrow Q(s,a) + \alpha\delta e(s,a)$
      $e(s,a) \leftarrow \gamma\lambda e(s,a)$
    $s \leftarrow s'$; $a \leftarrow a'$
  until s is terminal

(b) Sarsa(λ) pseudo code

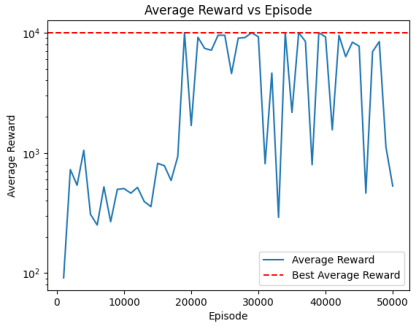
Fig. 1: Algorithm pseudo codes

1.2 Results

To assess the performance of our agent, I deploy it to play $k = 100$ episodes of the Flappy Bird game. To prevent potential infinite loops, I halt each game session after reaching step 10,000. The metric used for evaluation is the average reward across these k episodes. A higher average reward signifies a more effective algorithm. To evaluate MCC and Sarsa(λ), I used the policy determined solely by the highest estimated Q-values, avoiding any influence from epsilon-greedy selection to prevent potential biases in the estimation process.



(a) MCC reward curve



(b) Sarsa(λ) reward curve

Fig. 2: Comparison of reward curves

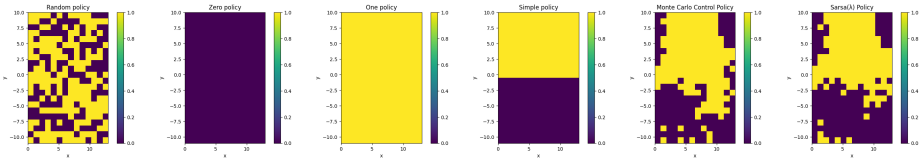
Both algorithms reach a nearly optimal policy within a reasonable timeframe (Fig 2). Table 1 presents a comparison with simple policies. The greedy policy consists of flapping when the vertical position is positive and remaining idle when it's negative. MCC demonstrates slightly quicker convergence, but repeated experiments reveal its instability compared to Sarsa(λ). Additionally,

MCC demands full episodes for training, while Sarsa(λ) can continuously learn at each step, rendering it more convenient for training purposes.

Policy	Random	Nothing	Always flap	Greedy	MCC learned	Sarsa(λ) learned
Average reward	12.0	4.0	13.0	244.782	**10001.0**	**10001.0**

Table 1: Comparison of average rewards for different policies

Fig. 3 displays the plotted policies obtained from the training process. In the case of Sarsa(λ) and MCC, it presents a direct comparison of Q-values for actions 0 and 1, as the policies were derived using the greedy policy approach based on Q-values. The learned policies align closely with our expectations: when the vertical position is negative, the optimal action is to remain idle, while a positive vertical position suggests flapping. In regions where the vertical position is greater than 2 and the horizontal position is less than 2, the algorithms do not learn to flap, as it's already too late to alter the reward outcome significantly. Moreover, for high values of x, the action's impact on the reward diminishes, potentially explaining why less intuitive actions are taken.

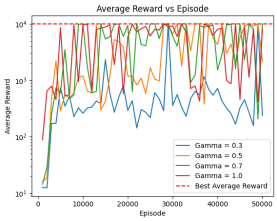


(a) Random (b) Zero (c) One (d) Greedy (e) MCC (f) Sarsa(λ)

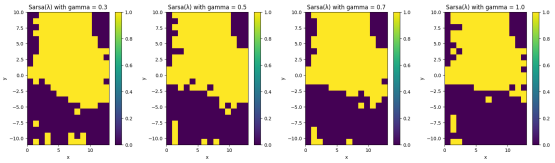
Fig. 3: Comparison of Policies

1.3 Parameters sweep

The parameter γ represents the discount factor, which determines the importance of future rewards in the learning process. It influences how much weight the agent places on immediate rewards versus future rewards when updating its action values. Smaller values of gamma tend to slow down the training process due to their impact on how the agent values future rewards. When gamma is small, the agent places less importance on future rewards compared to immediate rewards. As a result, the agent becomes more myopic, focusing predominantly on short-term gains and disregarding the potential benefits of long-term planning. This effect is directly observable in the learned policies, where the agent continues to choose to flap even when the vertical position is negative but the horizontal position is large.



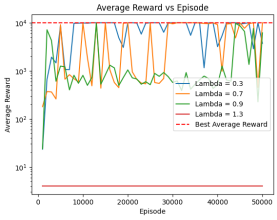
(a) Average reward



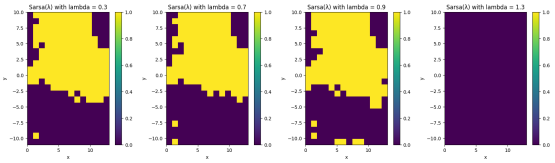
(b) Policies

Fig. 4: Gamma parameter sweep

In Sarsa(λ), the parameter λ controls the decay rate of the eligibility trace, which tracks the influence of past experiences on learning. A smaller λ leads to faster decay, prioritizing recent experiences, while a larger λ allows past experiences to have a more lasting impact. Essentially, λ balances between short-term gains and long-term planning in the agent's learning process, influencing its exploration-exploitation trade-off and memory retention. In the context of a relatively simple game, such as this one, a smaller lambda value tends to have faster convergence. Moreover, a lambda value greater than 1 can cause the Q-values to overflow, rendering the algorithm unusable.



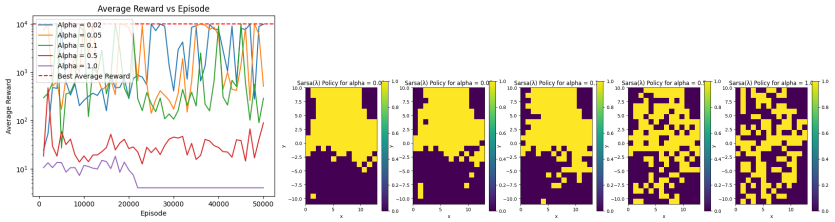
(a) Average reward



(b) Policies

Fig. 5: Lambda parameter sweep

The parameter α influences the rate at which the agent updates its Q-values based on new information. A smaller α value results in slower learning, as the agent gives less weight to new experiences and relies more on its existing knowledge. In my experiments, selecting a high value for alpha led to instabilities and caused the Q-values to overflow.



(a) Average reward (b) Policies

Fig. 6: Alpha Parameter Sweep

2 Discussion

2.1 Solving the other TBF and the Original Flappy Bird environment

The Text Flappy Bird (TFB) and the Original Flappy Bird games differ only in their observation representation. Both games could be effectively solved using the same algorithms if a feature extractor were employed to obtain the position of the next pipe. However, if one were to apply the algorithms directly without feature extraction, the observation space would become a 2D or 3D tensor. While tabular reinforcement learning methods might still be applicable, they would likely take significantly longer to converge due to the increased dimensionality of the observation space. In such cases, it would be more practical to utilize Deep Q-Learning, which can handle larger observation sizes and directly output learned Q-values or policies using algorithms such as DQN [1] or DDPG [2]. Moreover, we would benefit from using a Convolutional features extractor as the input is a 2D image.

2.2 Adaptation to other configurations of TBF

To evaluate the adaptability to different configurations, I trained an agent using Sarsa(λ) on TBF with specific dimensions (height = 20, width = 15, pipe gap = 4), then evaluated its performance on various TBF configurations not encountered during training. In instances where the agent encountered unseen observations during training, I filled the policy with the Idle action. The agent performed optimally when the pipe gaps were greater or equal and the heights were smaller or equal to the ones during the training phase. However, for other configurations, the average reward significantly decreased. This decrease can be attributed to the need for more precise actions in smaller pipe gaps, which might only be learned through training in such conditions. Additionally, increasing the height resulted in unseen observations where the agent's behavior wasn't corrected. This issue might be mitigated if the default action were to flap. As for changes in width, smaller widths had minimal impact, while larger widths didn't affect the observations significantly, as they always returned to seen training observations.

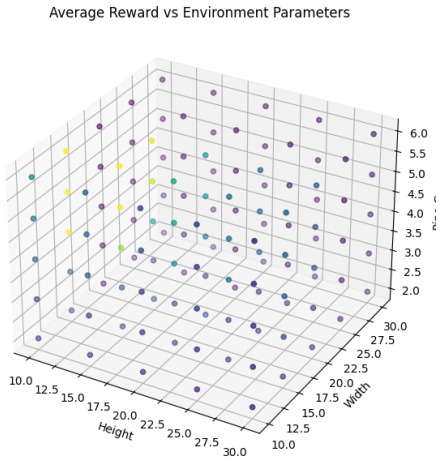


Fig. 7: Average reward on different configurations of TFB

References

1. Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013.
2. Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning, 2019.