# 3MD4120: Reinforcement Learning Individual Assignment

Peter KESZTHELYI
peter.keszthelyi@student-cs.fr

GitHub: link

**Abstract.** This project implements two reinforcement learning agents to play the text-based Flappy Bird game, using Monte Carlo and Expected Sarsa algorithms.

## 1 Environment

### 1.1 Overview

The Text Flappy Bird game is represented by a 2D grid of arbitrary size, where the bird (the agent), and the pipes are represented by their corresponding coordinates in the grid.

### 1.2 Components

**Action Space.** The action space is a single scalar that encodes the two possible actions: doing nothing (being idle) or flapping the wings.

**State Space.** The state space captures crucial information about the position of the agent and the obstacles. Here the pros and cons of each are described.

*Screen Render Version.* This represents states as a screen render. Although it contains more information, the algorithms we intend to implement might not be able to efficiently learn from this representation due to the high number of possible states.

*Closest Pipe Distance Version.* This version returns the distance $(x, y)$ from the closest upcoming pipe. This representation allows for faster learning with tabular methods, as the state-space is smaller. The limitation is that the current and the next-next pipe distance is unknown. Despite this drawback, this version is more favorable for the project thanks to the lower dimensionality of the state space.

**Reward.** The cumulative reward is increased at each step until a terminal state is reached (i.e. the bird collides with an obstacle).

**Transition Dynamics.** At each step the environment gives a state, generated following the logic of the Flappy Bird game. Then, the agent takes an action, and the environment returns a reward, as well as the following state, until a terminal state is reached.

# 2 Agents

## 2.1 Algorithm Parameters

The $\boldsymbol{\epsilon}$ parameter controls $\epsilon$-greedy exploration, while $\boldsymbol{\epsilon}$**- decay** reduces the $\epsilon$ to a threshold over episodes. A step size $\boldsymbol{\alpha}$ and a discount factor $\boldsymbol{\gamma}$ is also used.

## 2.2 Monte Carlo Agent

A constant-$\alpha$ Monte Carlo agent was implemented. First, an episode is generated $\{S_t, A_t, R_t, S_{t+1}, A_{t+1} \dots, R_T, S_T\}$ per an $\epsilon$-greedy policy. Next, we iterate over the episode in reverse order ($t = T-1, T-2, \dots, 0$) updating estimates of state-action pairs:

1. Update the return: $G \leftarrow \gamma G + R_{t+1}$ (initialized $G \leftarrow 0$)
2. Update state-action value: $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[G - Q(S_t, A_t)]$

## 2.3 Expected Sarsa Agent

We implemented an on-policy expected SARSA algorithm, which is defined by the following update rule, performed at each step within each episode of learning.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[ R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t) \right]$$

# 3 Results and Comparison

## 3.1 Parameter Selection and Sensitivity

Parameter tuning over 1,000 episodes to select the best options. We set a maximum score of 1,500 to avoid extremely long episodes. To better understand the sensitivity to parameters, we visualize the rewards for $\epsilon$ and $\alpha$ (step size), shown in Figure 1 and 2 of the Appendix.

**Step Size ($\alpha$).** The result suggest that Monte Carlo may be more sensitive to step size, which directly affects the magnitude of updates. On the other hand, Expected SARSA TD-like updates might provide some stability compared to the full episode updates used in Monte Carlo.

**Exploration Parameter ($\epsilon$)**. Poorly chosen values of epsilon affect the balance between exploration and exploitation, leading to suboptimal policies. Expected SARSA seems to exhibit slightly less sensitivity to epsilon compared to Monte Carlo, as it updates Q-values based on TD errors, which can guide exploration more effectively.

### 3.2 Policies and State-Value Functions

**State Value Functions.** Figures 3 and 4 in the Appendix show a comparison of state-value functions between the Monte Carlo (MC) and the Expected SARSA (ES) agents.

For both agents, the state-value functions reveal similar findings: when the horizontal distance ($\Delta x$) is small, there are limited number of vertical distances ($\Delta y$) that result in a reward. When $\Delta x$ is larger, there exist more $\Delta y$ with higher values. However, when $\Delta x > 9$, the agent might still be traversing the previous pipe, necessitating caution to avoid collisions, therefore state-values for most $\Delta y$ tend to be lower in these regions.

The MC agent had a larger surface of high-value states compared to ES agent, especially when $\Delta x$ is either large or small. This suggests MC was better able to learn the right actions for these states. Overall, MC seems to be more conservative, due to the reliance on complete episode returns and the absence of expected values in updates.

**Policies.** Figures 5 and 6 in the Appendix show the policies learned by the agents. The ES agent seems to learn a more consistent behavior depending on $\Delta y$ – flapping when positioned below the pipe opening. However, neither agent learns to consistently position itself optimally above the pipe opening's extremities, possibly due to limitations in the learning process or the complexity of the environment. As seen on Figure 7, the MC agent performed better over 1,000 episodes of training, even though its policy reveals a less obvious pattern of actions.

### 3.3 Further Discussion

**Generalization to Different Environment Configuration.** The agents are generalized to different width, height, and pipe-gap configurations by selecting the idle action in unexplored regions of the state-space.

Our experiments show that the agents are not able to generalize to new configurations (unless the pipe gap is increased leaving all else unchanged). This suggests that changes in the outlay of the game require changes in the agent's strategy. Furthermore, the missing state-action values in the new configuration could be crucial for survival.

**Using in the Original Flappy Bird Environment.** The original flappy bird environment has a continuous 2D state-space, which does not directly allow our agents to play, unless we discretize the X- and Y-distance observations returned. Additionally, we once again encounter the configuration issue mentioned above. Finally, the number of frames rendered by the original flappy bird game might be much larger, making the implementation of the agents to the new environment computationally challenging.

## References

1. Richard S. Sutton and Andrew G. Barto (2020). Reinforcement Learning: an introduction.
2. Szepesvári, Csaba. Algorithms for reinforcement learning (2010). Synthesis lectures on artificial intelligence and machine learning
3. GitHub: https://github.com/peter-b-k/ReinforcementLearning-TextFlappyBird

# Appendix

**Fig. 1.** Expected Sarsa algorithm parameter sensitivity.

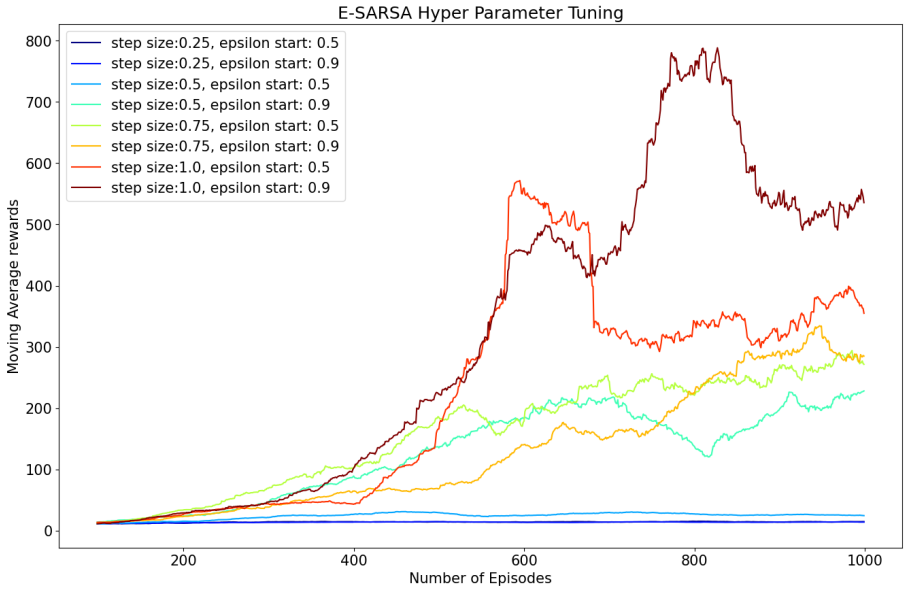


**Fig. 2.** Monte Carlo algorithm parameter sensitivities.

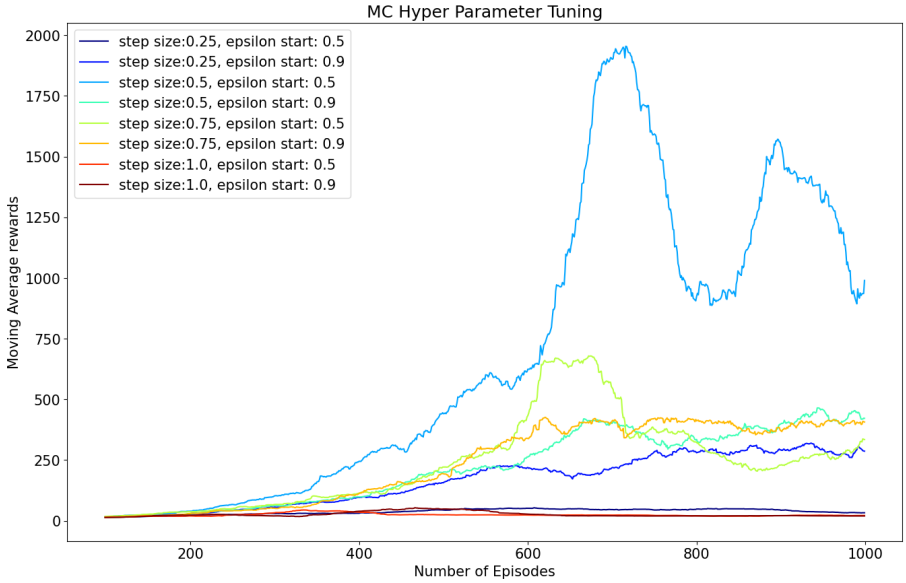

**Fig. 3.** MC State-Values

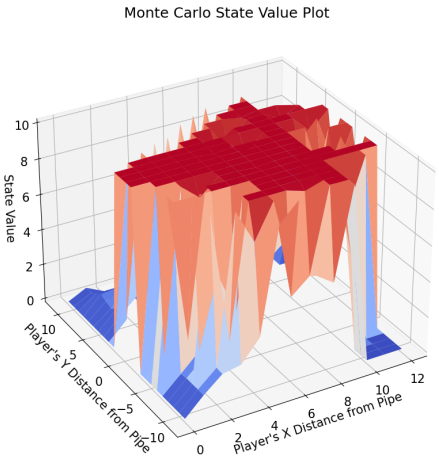


**Fig. 4.** E-SARSA State Values

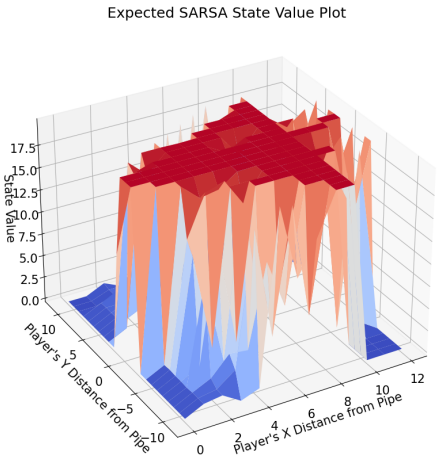


**Fig. 5.** Monte Carlo Policy

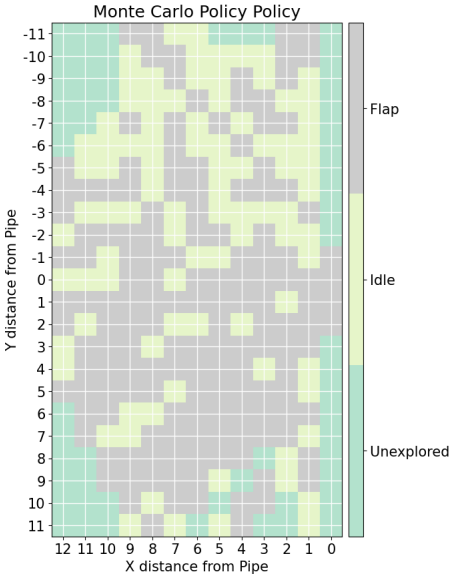


**Fig. 6.** Expected SARSA Policy

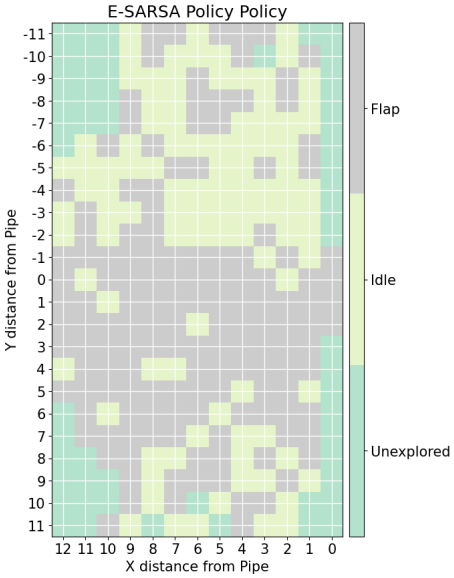

**Fig. 4.** Scores per Episode Comparison