Reinforcement Learning Individual Assignment: MC Control and Sarsa(λ) learn to play Flappy Bird

Quentin Gopée

CentraleSupélec, Université Paris-Saclay
quentin.gopee@student-cs.fr

1 Introduction

Text Flappy Bird (TFB) is a minimalist adaptation inspired by the iconic NetHack game. Within the confines of a terminal, TFB unfolds using simple character elements, reminiscent of classic text-based adventures. It offers two distinct environments, each offering unique insights into gameplay dynamics. In *TextFlappyBird-screen-v0*, observations manifest as arrays representing the current state of the game screen, encoded as integers. Conversely, *TextFlappyBird-v0* presents observations as the horizontal and vertical distance of the player to the nearest upcoming pipe gap.

In this project, we implement and compare two different agents: Monte Carlo Control and Sarsa(λ).

The code can be found at: https://github.com/quentin-gopee/RL$_A ssignment$

2 Environment

There are 2 different environments provided on the Text Flappy Bird git repository:

TextFlappyBird-screen-v0:
Returns an array representing the current game screen state, encoded as integers, including player character, pipes, and other elements.

Limitations:

- High-dimensional observations: Large array sizes result in high-dimensional observation spaces, challenging RL algorithms, especially those struggling with high-dimensional inputs.
- Complexity: Processing the complete screen render may demand more computational resources and time, potentially slowing down learning.

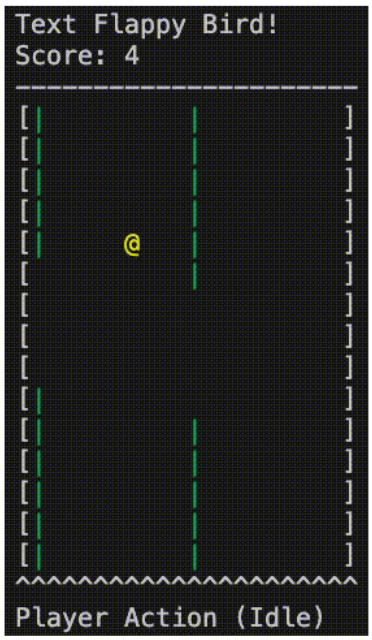

Fig. 1. Text Flappy Bird rendered in the terminal

TextFlappyBird-v0:

Provides horizontal and vertical distances of the player to the nearest upcoming pipe gap, focusing solely on gameplay-relevant information.

Limitations:

- Limited information: Observations are abstract, lacking details about the entire game screen, possibly hindering the agent's decision-making, especially in contextually critical situations.
- Reduced complexity: While simplifying observations aids faster learning, it may oversimplify the game environment, potentially overlooking vital details influencing gameplay strategies.

For its simplicity, we use the *TextFlappyBird-v0* environment in this project.

Note: The states of the original Flappy Bird game environment are continuous. Thus the agent presented in the next section woudn't work on this environment.

3 Method

3.1 Monte-Carlo Control

MC methods estimate the value function by averaging returns observed after visits to states (or state-action pairs). These returns are the total rewards obtained from the state onwards until the end of the episode.

The main equation for updating the value function in Monte Carlo (MC) methods is as follows:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left(G_t - Q(S_t, A_t)\right) \tag{1}$$

Where:

- $Q(S_t, A_t)$ is the action-value function
- G_t is the observed return
- α is the learning rate

3.2 Sarsa(λ)

Sarsa(λ) maintains an estimate of the action-value function and updates it using the observed rewards and transitions. It combines elements of TD learning with eligibility traces, which represent the accumulated future importance of states and actions.

The main equation for updating the action-value function in Sarsa(λ) is as follows:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \cdot \delta \cdot e(s, a) \tag{2}$$

where:

- $Q(s, a)$ is the estimated action value of state-action pair (s, a).
- α is the learning rate (step size).
- δ is the temporal difference error, which is calculated as $\delta = r + \gamma Q(s', a') - Q(s, a)$, where r is the reward received, γ is the discount factor, s' is the next state, and a' is the next action.
- $e(s, a)$ is the eligibility trace for the state-action pair (s, a), representing the importance of updating the action value of that pair. The eligibility trace is updated according to:

$$e(s, a) \leftarrow \gamma \lambda e(s, a) \tag{3}$$

where λ is the eligibility trace decay parameter

4 Results

4.1 Monte-Carlo Control

The main hyperparameters of the Monte-Carlo method to be tuned are the step size α and the ϵ-decay of the ϵ-greedy policy. The maximum training time was set to 5min to speed up the process and avoid infinite loops. We can see on Fig. 2 that the best performances are obtained for ϵ-decay $= 0.999$, while the step-size doesn't have much influence in the range explored. The best score obtained during the training is 1483, but the agent is able to easily go over 10000 (and maybe infinity) once trained when the ϵ-greedy policy is removed.

The state-value function displayed on Fig. 3 shows something looking like a triangle converging to the center of the gap between the pipes, which makes sense. However this function is not really clear yet, maybe the agent needs to be trained for a longer time in order to converge.

The policy displayed on Fig. 4 seems also coherent: there are more "Idle" tiles at the top of the grid than "Flaps", and it's the opposite at the bottom.

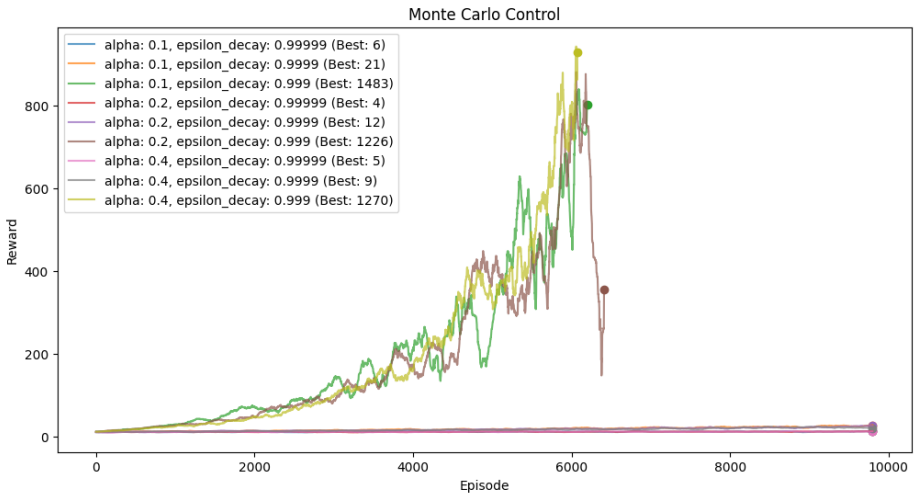

Fig. 2. Monte-Carlo hyperpameters tuning

4.2 Sarsa(λ)

The main hyperparamaters to tune for Sarsa(λ) are the learning rate α, the discount factor γ and the eligibility trace decay λ. In the experiments, the ϵ-decay is set to 0.99, and the maximum training time was set to 5min to speed up the

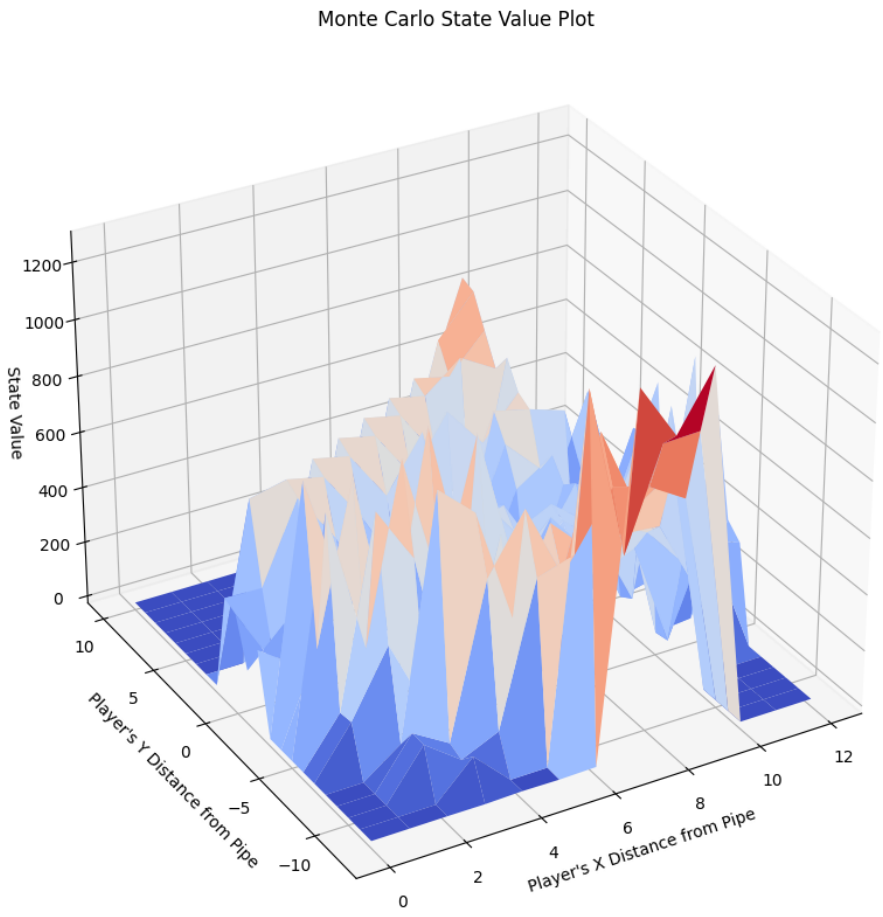

Fig. 3. Monte-Carlo State Value Function

process and avoid infinite loops. We can see on Fig. 5 that some configurations perform really well, espacially the ones with $\lambda = 0.9$ that have the reward exploding after a few hundreds of episodes.

We can see a really clear triangle on the state-value function in Fig 6 converging to the center of the gap between the pipes, wich makes perfect sens.

The policy displayed in Fig. 7 looks similar to the one obtained with Monte-Carlo and seems coherent as well.

4.3 Performance on different environment configurations

The two agents are trained using the same configuration (height=15, width=20, pipe gap=4). Both of them give similar results when tested on other environment configuration in term of pipes gap: they fail to play when the gap is smaller (2 or 3) and do really great when it's equal or higher than 4 (more than 10000, the experiment was stop to avoid infinite loop).

References

1. Christodoulidis Stergios, Text Flappy Bird for OpenAI Gym, https://gitlab-research.centralesupelec.fr/stergios.christodoulidis/text-flappy-bird-gym
2. Flappy Bird for OpenAI Gym, https://github.com/Talendar/flappy-bird-gym

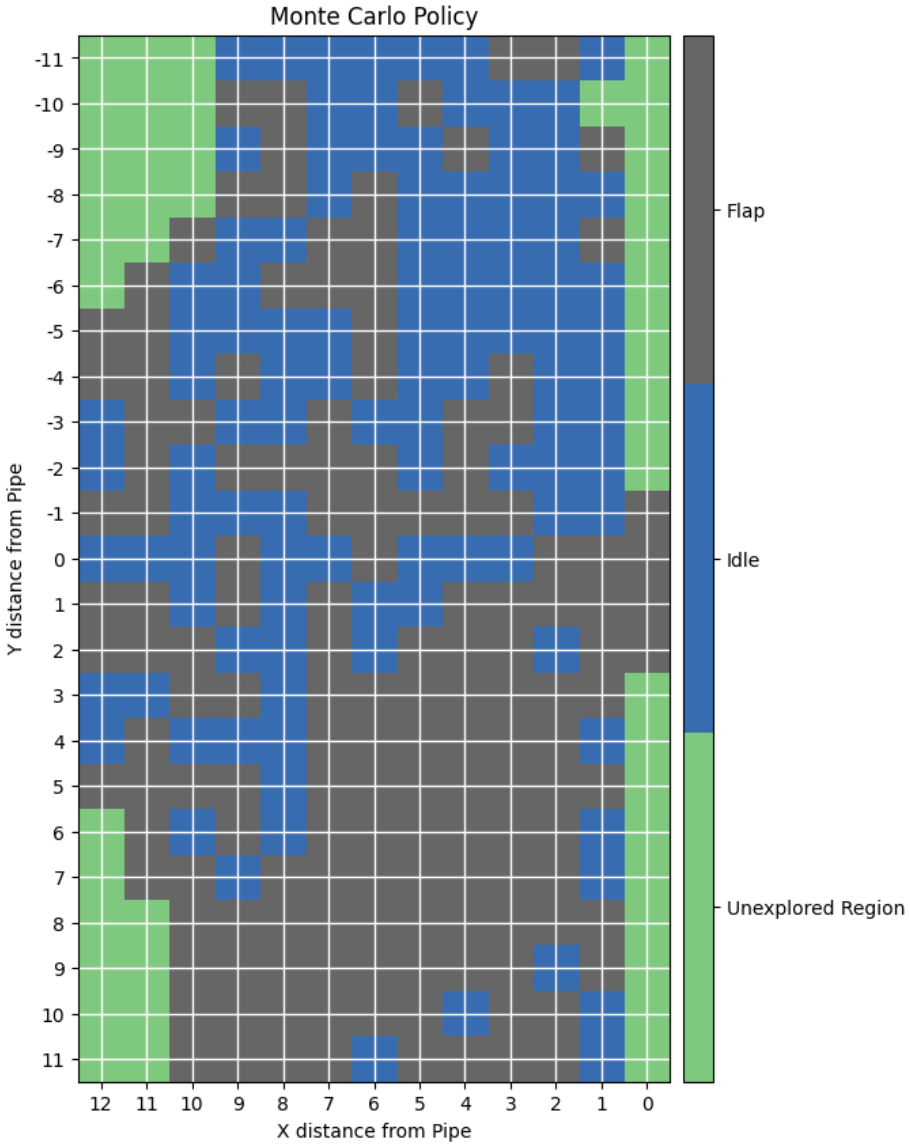

Fig. 4. Monte-Carlo Policy

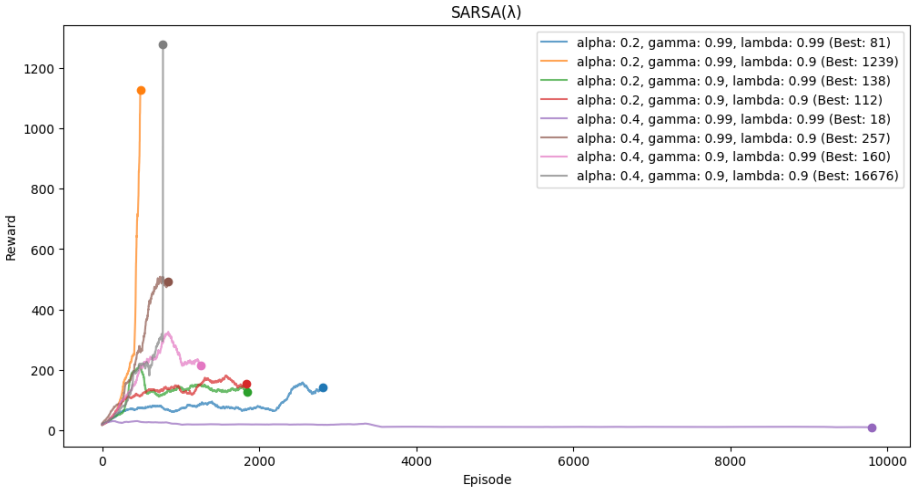

Fig. 5. Sarsa(λ) hyperpameters tuning

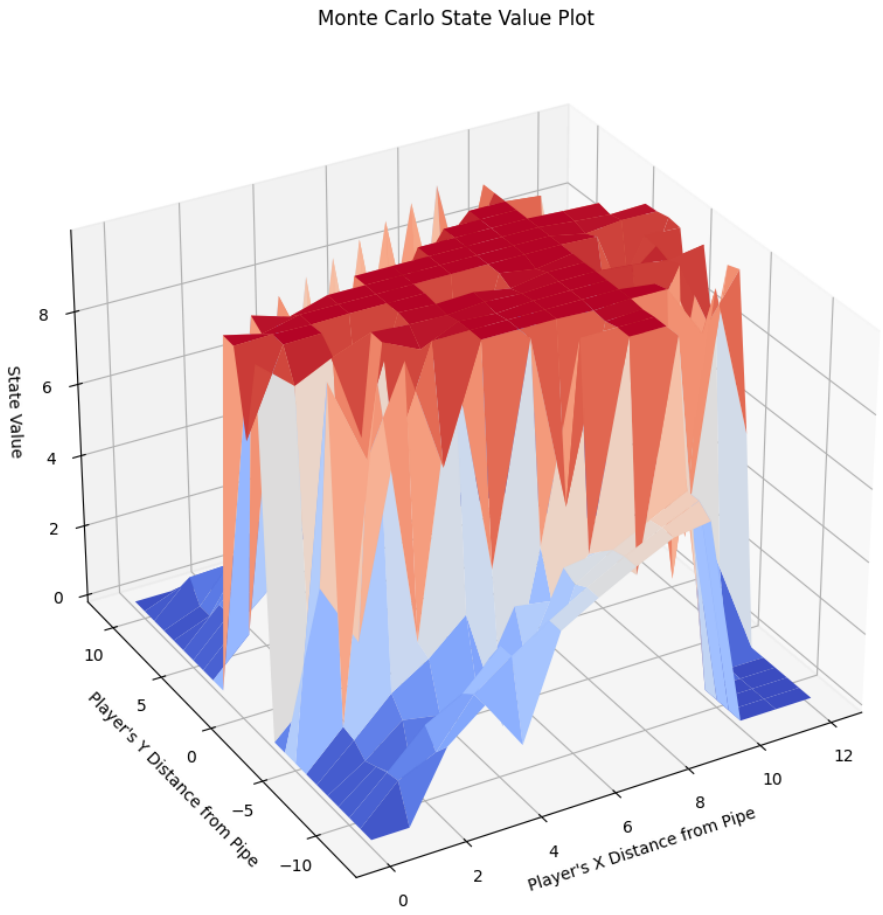

Fig. 6. Sarsa(λ) State Value Function

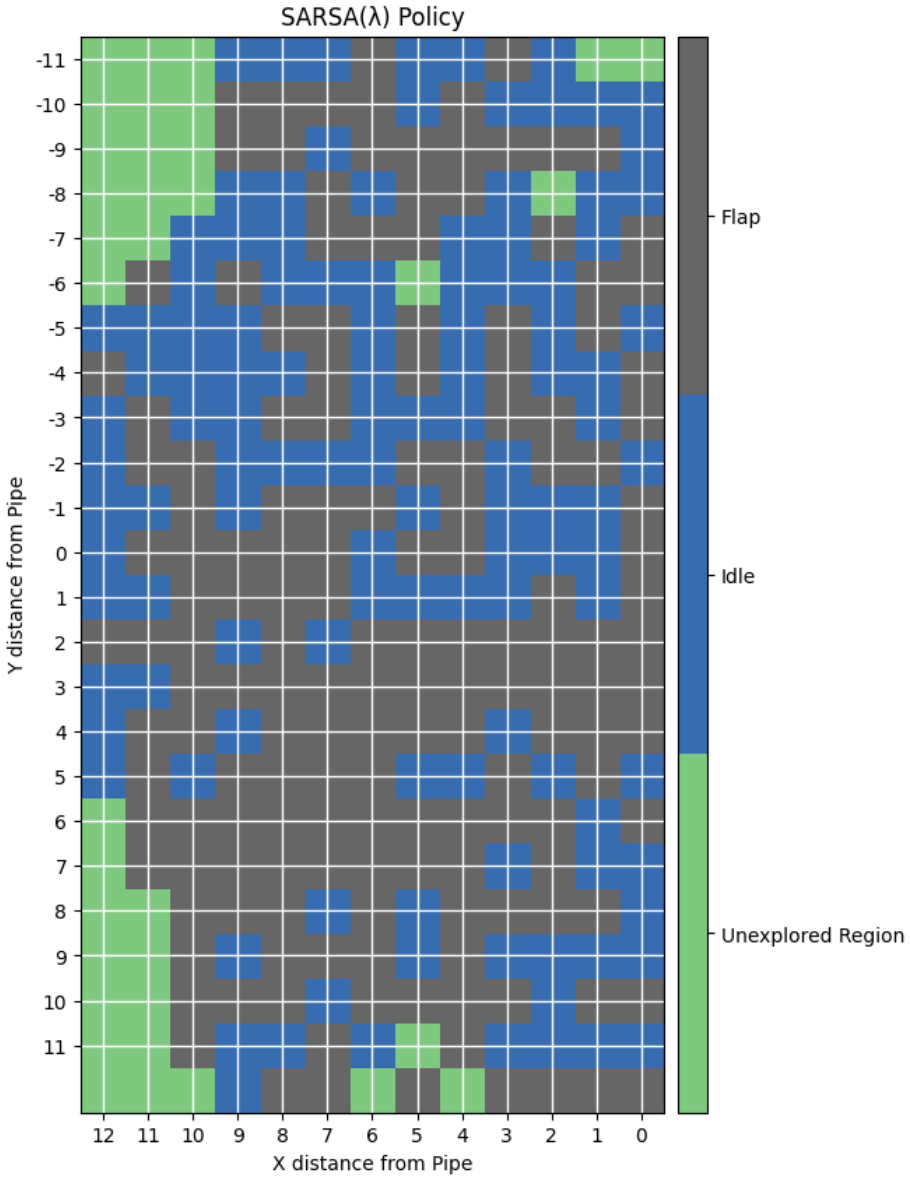

Fig. 7. Sarsa(λ) Policy