Reinforcment-Midterm Excercise

Stav Cohen & Avner Duchovni

January 2025

1 Introduction

In this project, we tackled the challenge of developing an autonomous agent to play the game "Flappy Bird" using reinforcement learning with tabular methods. The primary goal was to train the agent—represented by a bird—to navigate through gaps between pipes autonomously, without manual intervention and without colliding with the pipes. A collision with a pipe results in game over, thus, our success metric was defined as the bird successfully passing through at least 10 pipes with a gap size of 65.

We utilized the "Flappy Bird" game environment provided by the Python PLE (Pygame Learning Environment) library. This library sets up the game scenario, including the pipe configurations, the horizontal movement rate of the pipes, and the bird's default vertical movement, which is influenced by gravity. The critical task for our agent was to determine whether to flap its wings at each game step, based on the current game environment, to maximize the number of successful pipe navigations.

2 Methodology

We encountered several challenges when implementing actions for each step in the bird's environment:

2.1 Pre-processing

Initial Decision The action for each step must be determined, considering the large variability in step positions and actions. The pixel-based environment introduces a high number of potential states.

Environment Size The environment is defined by a grid of 512×288 pixels.

Parameters Eight parameters influence the decision-making process:

1. Bird's vertical position (y-coordinate)
2. Bird's velocity
3. Horizontal distance to the next pipe

4. Vertical position of the next pipe's top
5. Vertical position of the next pipe's bottom
6. Horizontal distance to the pipe after next
7. Vertical position of the pipe after next's top
8. Vertical position of the pipe after next's bottom

State Complexity Each parameter can take up to 500 discrete values, potentially leading to $512^2 \times 288^5 \times 20$ possible states. This complexity is computationally infeasible for polynomial-time training.

Reduction Techniques To manage state complexity, we applied three reduction techniques:

1. Binary State Transformation: We simplified the state by considering the vertical distance between the bird and the middle of the upcoming gap, encoding this distance as a binary state based on whether it exceeds six steps.
2. Feature Reduction to Five: We reduced the state representation to five binary features, leading to $2^5 = 32$ possible states:
   (a) Whether the horizontal distance to the next gap is under 45 steps.
   (b) Whether the vertical distance from the middle of the next gap exceeds four steps.
   (c) Whether the bird's velocity is positive.
   (d) Whether the horizontal distance to the pipe after next is under 60 steps.
   (e) Whether the vertical distance from the middle of the next-next gap exceeds eight steps.
3. Trinary and Binary Feature Combination: We used a combination of three-state and two-state features, resulting in $3 \times 2 \times 2 = 12$ possible states:
   (a) The vertical distance from the middle of the next gap is categorized as: a) far below (over 23 steps), b) slightly below (under 23 steps), c) above.
   (b) Whether the bird's velocity is positive.
   (c) Whether the next-next gap is above the next gap.

2.2 Reward-Shaping

Strategy The Reward Shaping strategy awards a small reward (+1) for each survival step, and imposes a large penalty (-50) if the bird reaches a terminal state by crashing into a pipe.

2.3 Policy-Agent Implementation

Policy Setup We use two tabular learning methods: **Trivial**, **Q-Learning**, and **SARSA**. The policy structure is a dictionary mapping state IDs to action probabilities, where each state ID is a tuple representing processed parameter values, e.g., "$(0, 1, 1, 0, 0)$".

Action Selection Actions are determined based on the epsilon-greedy method, controlled by hyperparameter ϵ:

1. With probability ϵ, a random action is chosen (50% UP, 50% DOWN).
2. With probability $1 - \epsilon$, the action with the higher weight is selected (UP if its weight exceeds that of DOWN).

Policy Update The updating of the policy varies by the learning method:

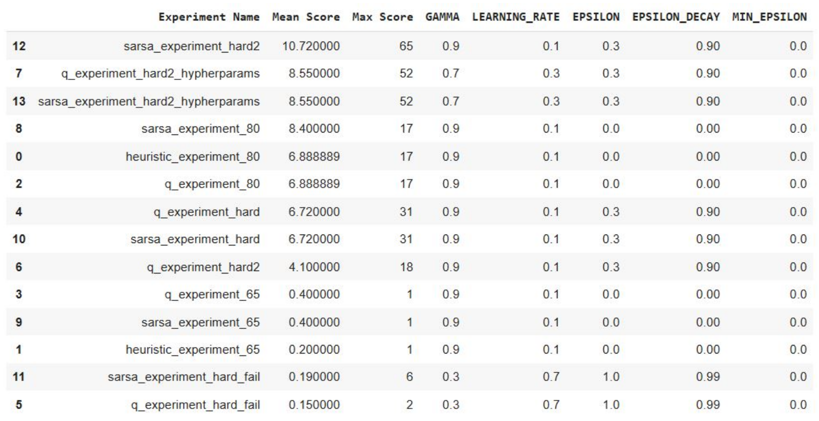
- For the Trivial Policy: $Q(s_t, a_t) = Q(s_t, a_t) + \alpha \cdot r_{t+1}$, where α is the learning rate and r_{t+1} is the reward of the next state.
- For Q-Learning: $Q(s_t, a_t) = Q(s_t, a_t) + \alpha(r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t))$, with γ representing the discount factor.
- For SARSA: $Q(s_t, a_t) = Q(s_t, a_t) + \alpha(r_{t+1} + Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$.

3 Results

Using the 12-dimensions state space preprocessing function we were able to get the best results both with SARSA and Q-learning. The top results were obrained by SARSA with Gamma of 0.9 and learning rate of 0.1, along with an intial epsilon value of 0.3 and an epsilon decay factor of 0.9. As shown in Figure7, with this algorithm and hyperparameters we reached a max score of 65 pipes, and an average of 10.72 pipes per run.

The runner up algorithm was Q-learning, with a Gamma value of 0.3 and learning rate of 0.3, along with an initial value of epsilon set to 0.3 and epsilon decay factor for epsilot set to 0.9. As shown in Figure5, under these settings we reached a maximum of 52 pipes, and an average of 8.55 pipes per run.

Below is a summary of our results:



	Experiment Name	Mean Score	Max Score	GAMMA	LEARNING_RATE	EPSILON	EPSILON_DECAY	MIN_EPSILON
12	sarsa_experiment_hard2	10.720000	65	0.9	0.1	0.3	0.90	0.0
7	q_experiment_hard2_hypherparams	8.550000	52	0.7	0.3	0.3	0.90	0.0
13	sarsa_experiment_hard2_hypherparams	8.550000	52	0.7	0.3	0.3	0.90	0.0
8	sarsa_experiment_80	8.400000	17	0.9	0.1	0.0	0.00	0.0
0	heuristic_experiment_80	6.888889	17	0.9	0.1	0.0	0.00	0.0
2	q_experiment_80	6.888889	17	0.9	0.1	0.0	0.00	0.0
4	q_experiment_hard	6.720000	31	0.9	0.1	0.3	0.90	0.0
10	sarsa_experiment_hard	6.720000	31	0.9	0.1	0.3	0.90	0.0
6	q_experiment_hard2	4.100000	18	0.9	0.1	0.3	0.90	0.0
3	q_experiment_65	0.400000	1	0.9	0.1	0.0	0.00	0.0
9	sarsa_experiment_65	0.400000	1	0.9	0.1	0.0	0.00	0.0
1	heuristic_experiment_65	0.200000	1	0.9	0.1	0.0	0.00	0.0
11	sarsa_experiment_hard_fail	0.190000	6	0.3	0.7	1.0	0.99	0.0
5	q_experiment_hard_fail	0.150000	2	0.3	0.7	1.0	0.99	0.0

4 Discussion

Our project's success was driven by iterative improvements in the agent's preprocessing and state space configurations. A critical enhancement in our third iteration was dividing the bird's vertical position into three distinct zones, expanding the state space to twelve discrete states. This adjustment significantly improved the agent's performance, allowing it to navigate through 65 pipes and demonstrating the importance of balanced state space design.

The main challenge was optimizing the state space to facilitate learning rather than focusing solely on the algorithms. Our initial strategy of using only two states enabled quick early learning but limited the agent's ability to master complex behaviors. This highlighted the need for a balance between simplicity and richness in the learning environment.

Our findings underscore that preprocessing decisions and hyperparameter settings—like epsilon values and learning rates—are critical in shaping learning efficiency and outcomes. The reward system, focused mainly on survival, provided clear and immediate feedback, crucial for effective reinforcement learning. This straightforward approach, supplemented by careful hyperparameter tuning, achieved notable performance without excessive complexity.

As shown in Figure12, it was very very difficult for us to shape our ovservation space as the bird will be able to pass the next gap when it's much upper than the current one, but when the vertical distance to the next gap is a bit lower, as shown in Figure11, we could easily pass it

In future iterations, building on this foundational knowledge, gradually increasing the state complexity could allow the agent to develop more sophisticated behaviors without overwhelming the learning process.

5 Conclusion

The implementation of Q-Learning and SARSA algorithms within our defined environment demonstrated their robustness and practical applicability to problems with well-defined but challenging state spaces. While these algorithms are well-established, our project reaffirmed that the real challenge lies in appropriately preparing the learning environment through intelligent preprocessing and reward shaping.

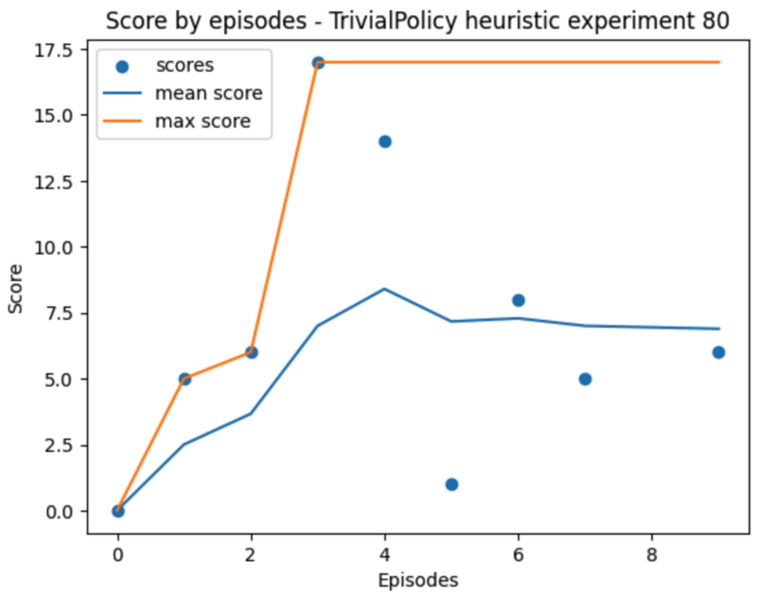
Future work could explore the effects of incrementally increasing the complexity of the state space or introducing additional dimensions to the reward system. Such explorations could enhance the agent's ability to learn more complex strategies and improve its adaptability to more varied scenarios beyond the relatively controlled environment of a game like FlappyBird.

By structuring your report into these separate sections, you effectively guide the reader through your experimental journey, emphasizing the key findings, the analysis of those findings, and the broader implications of your work.

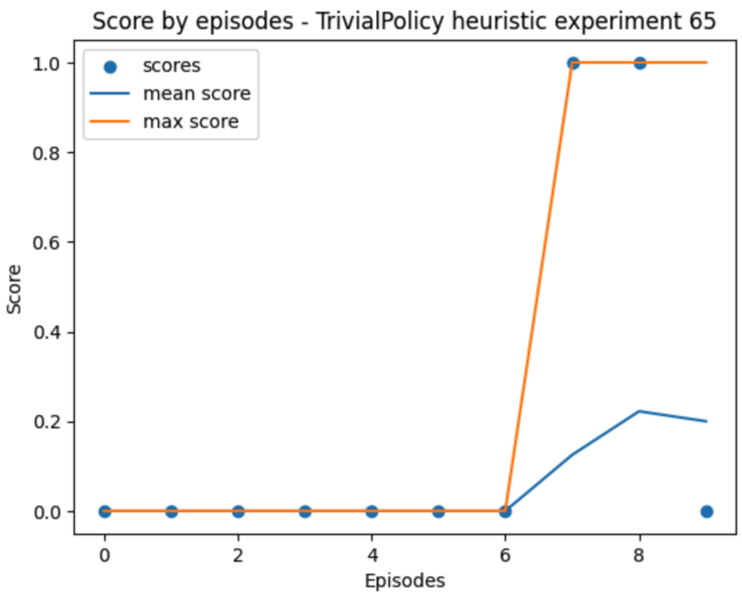
6 Appendix-Graphs

6.1 Training

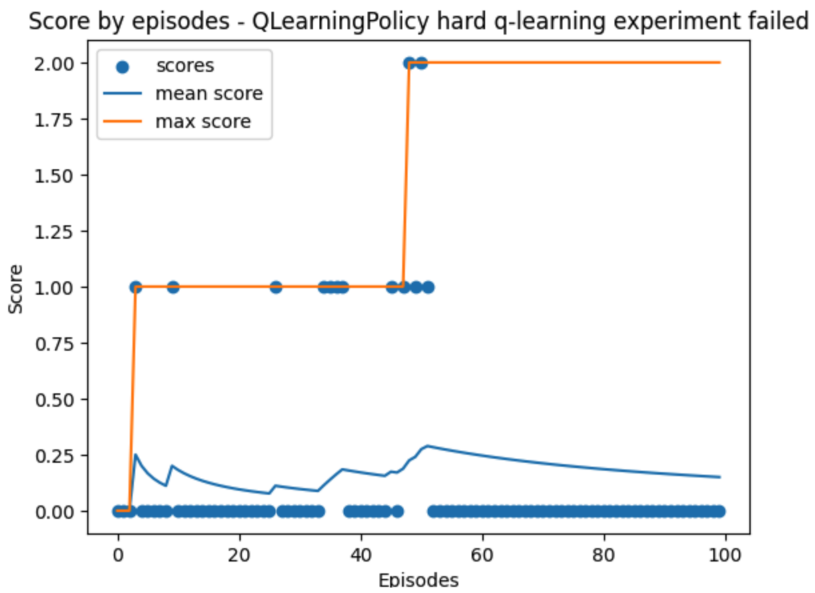
6.1.1 Figure 1 - Heuristic Results Gap of 80



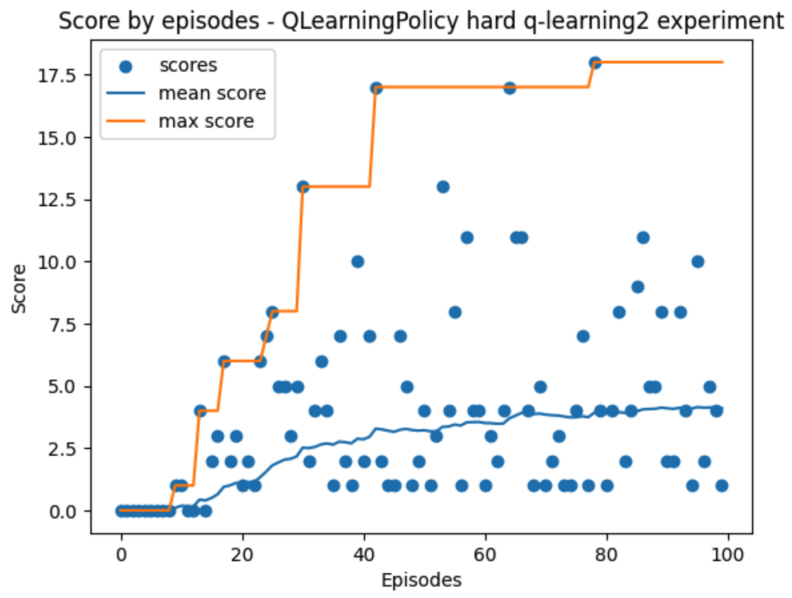
6.1.2 Figure 2 - Heuristic Results Gap of 65

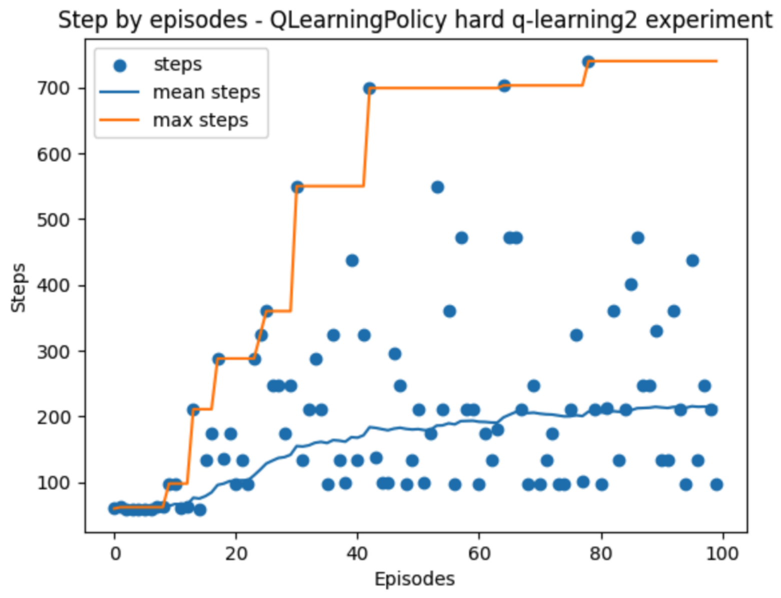


6.1.3 Figure 3 - QLearning Preprocess 2 Learning rate of 0.7 and epsilon greedy of 1 (Supposed to always fail)

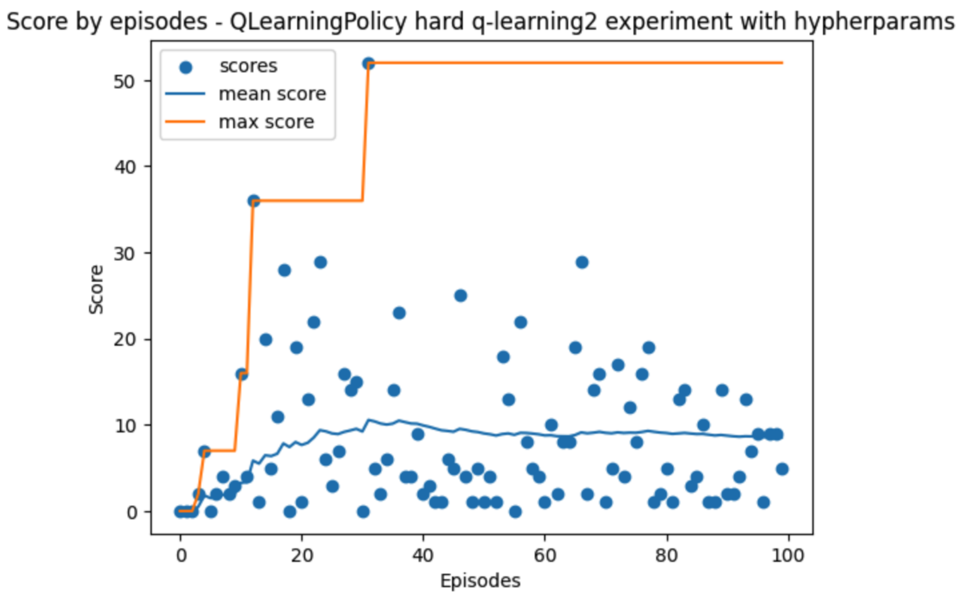


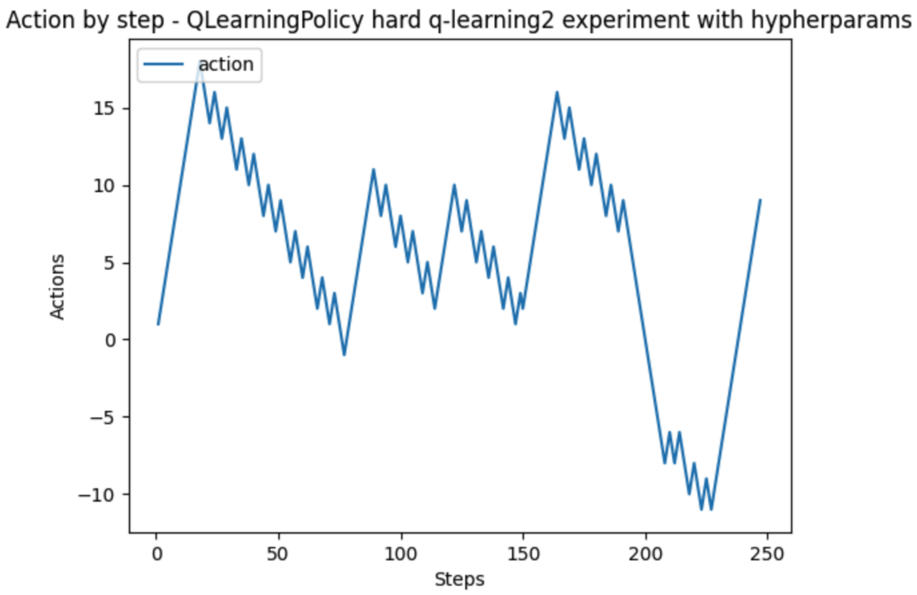
6.1.4 Figure 4 - QLearning Preprocess 3 Learning rate of 0.1 and epsilon greedy of 0.3



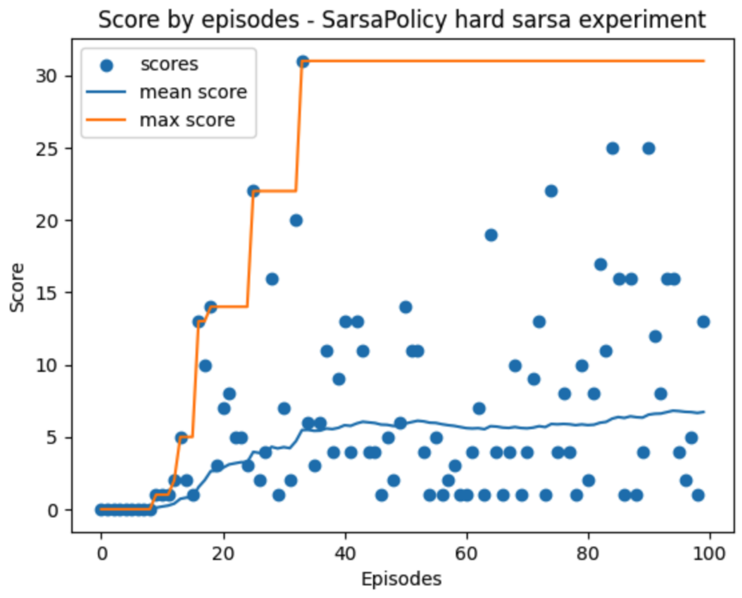


6.1.5 Figure 5 - QLearning Preprocess 3 Learning rate of 0.3 and epsilon greedy of 0.3

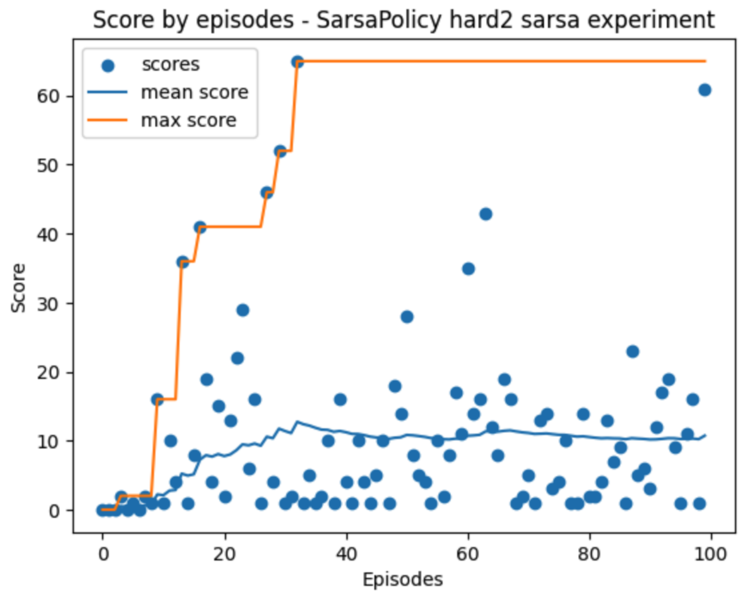




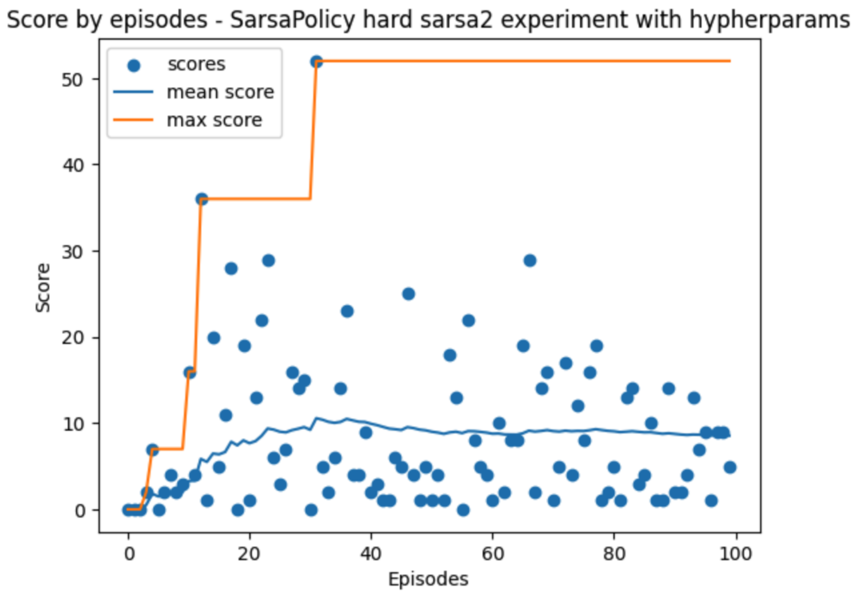
6.2 Figure 6 - Sarsa Preprocess 2 Learning rate of 0.1 and epsilon greedy of 0.3

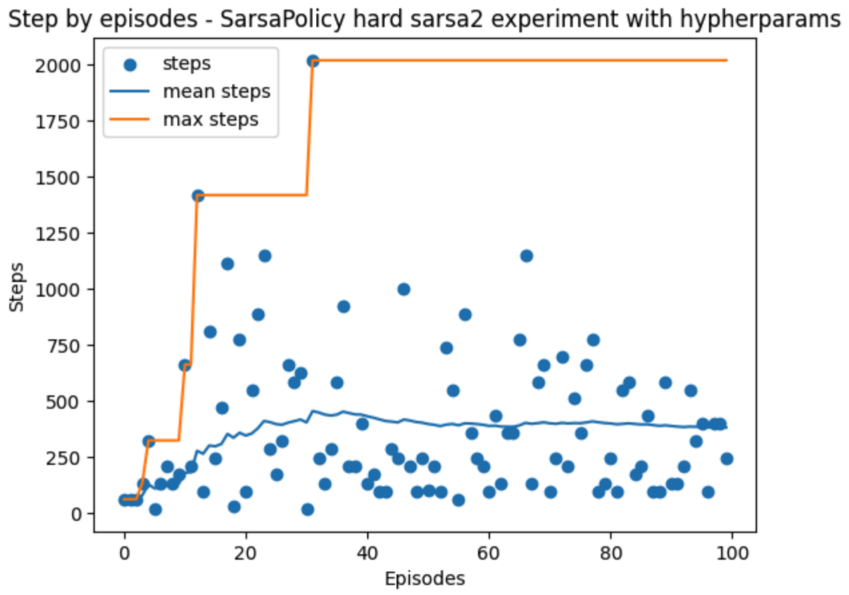


6.3 Figure 7 - Sarsa Preprocess 3 Learning rate of 0.1 and epsilon greedy of 0.3



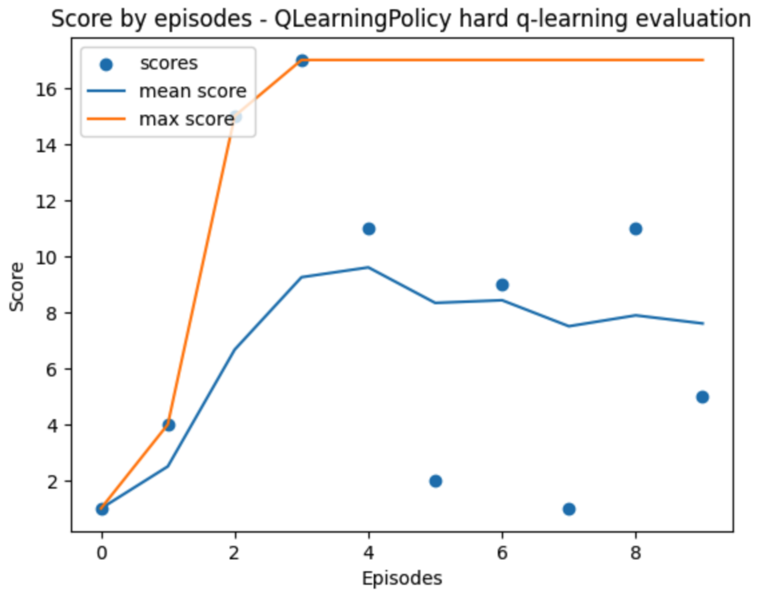
6.4 Figure 8 - Sarsa Preprocess 3 Learning rate of 0.3 and epsilon greedy of 0.3

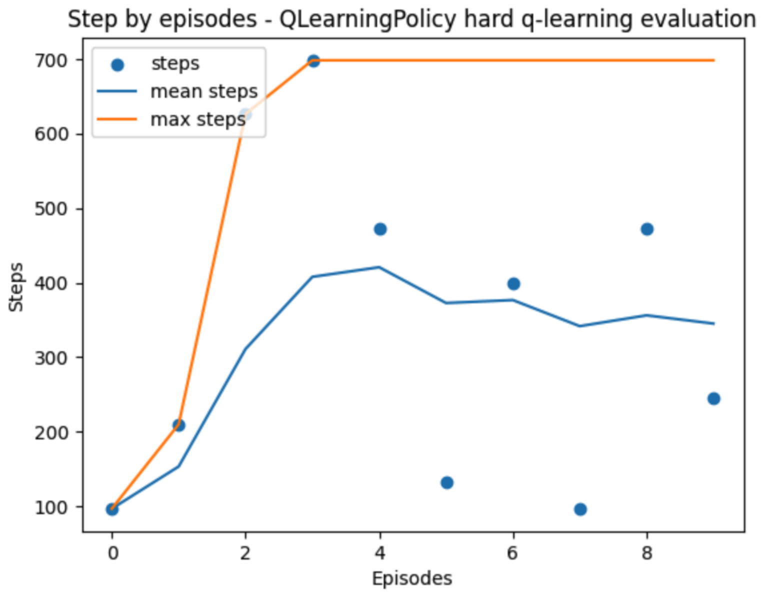


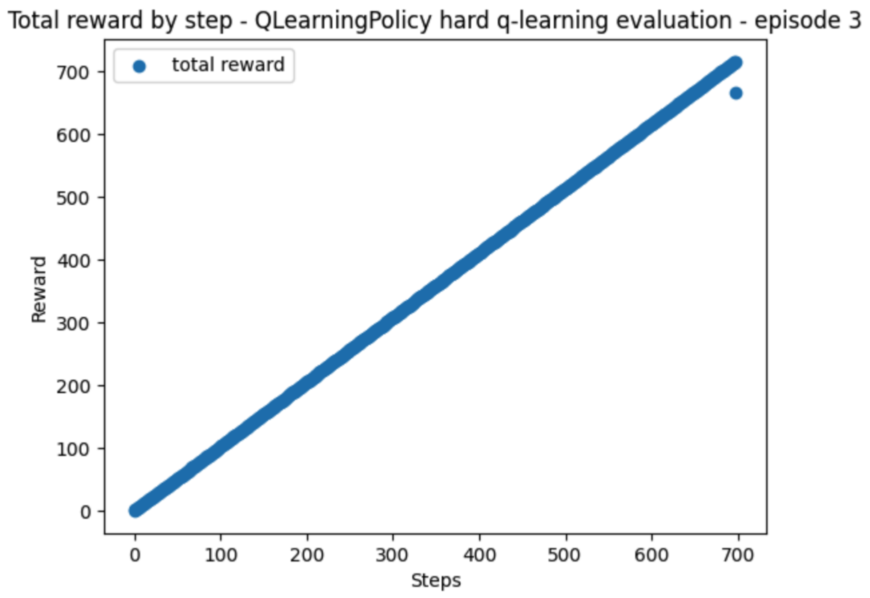


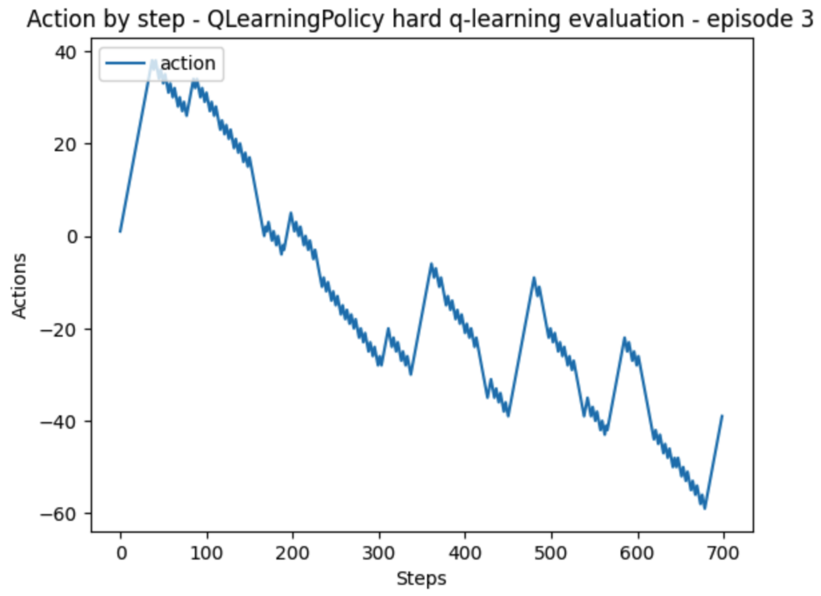
6.5 Evaluation

6.5.1 Figure 9 - QLearning Preprocess 3 Learning rate of 0.3 and epsilon greedy of 0.3

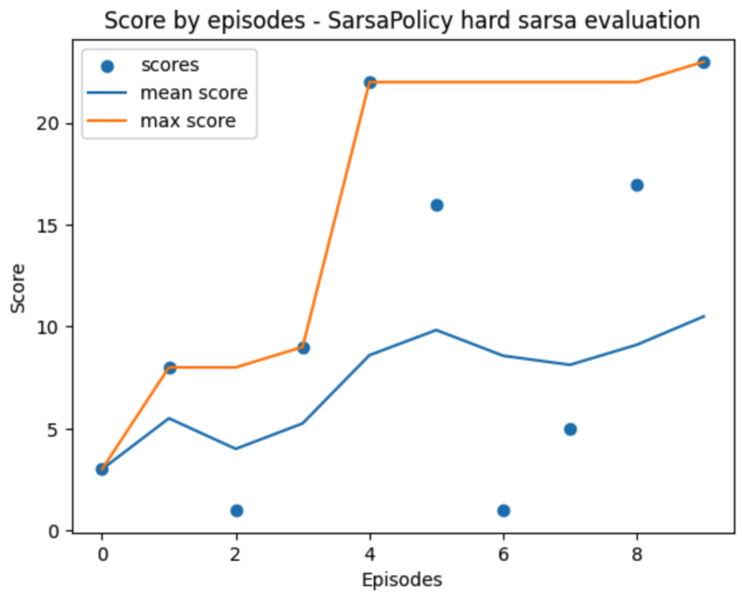


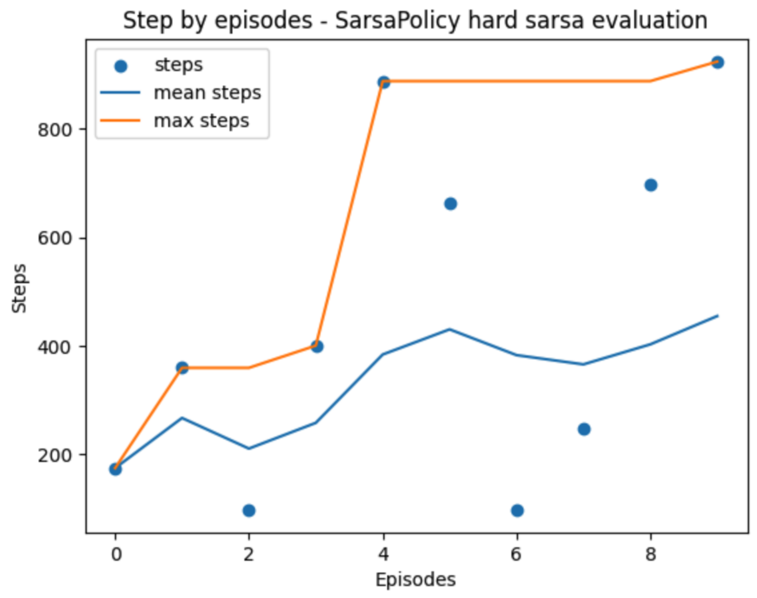


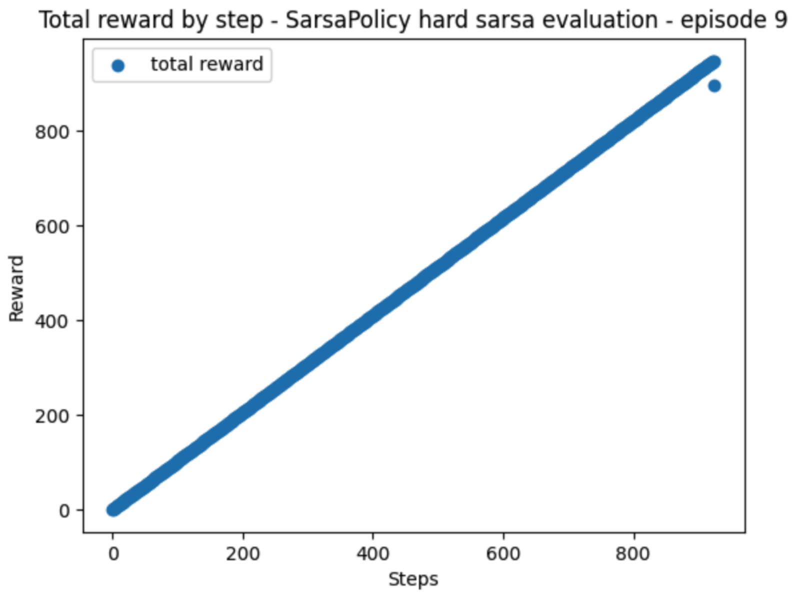

Total reward by step - QLearningPolicy hard q-learning evaluation - episode 3
total reward
Reward
Steps
0
100
200
300
400
500
600
700

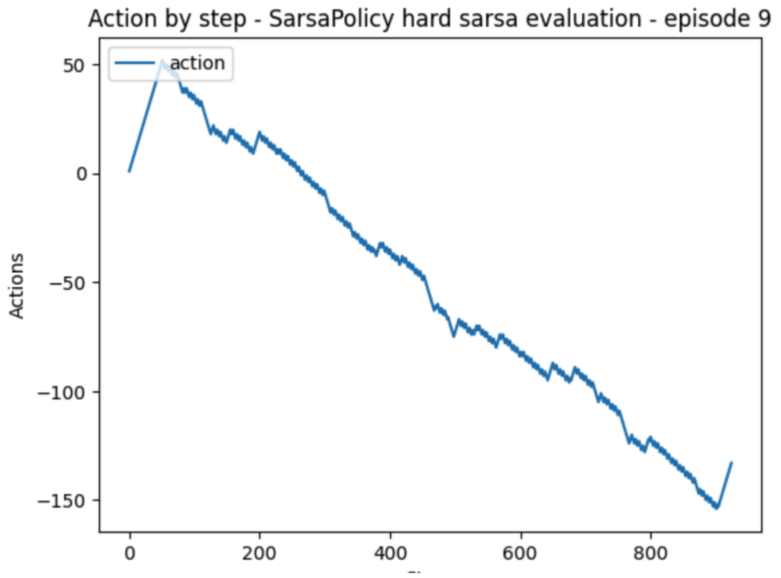

Action by step - QLearningPolicy hard q-learning evaluation - episode 3
action
Actions
Steps
40
20
0
−20
−40
−60
0
100
200
300
400
500
600
700

6.5.2 Figure 10 - Sarsa Preprocess 3 Learning rate of 0.1 and epsilon greedy of 0.3



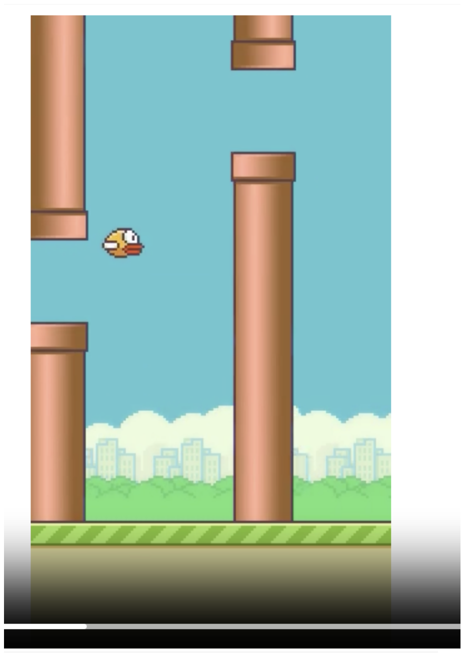


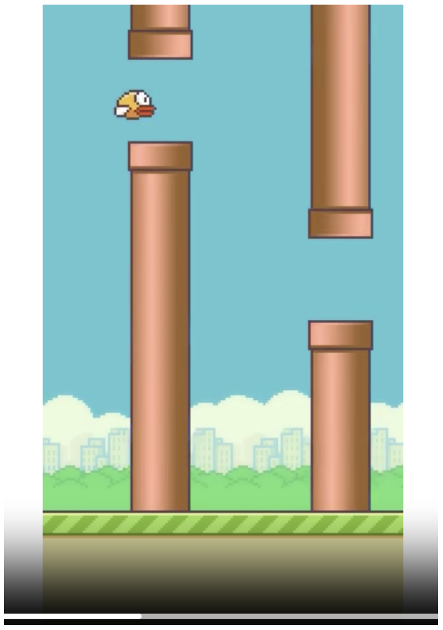

Total reward by step - SarsaPolicy hard sarsa evaluation - episode 9
total reward
Reward
Steps
0
200
400
600
800


Action by step - SarsaPolicy hard sarsa evaluation - episode 9
action
Actions
50
0
−50
−100
−150
0
200
400
600
800

6.6 Video Screenshots

6.6.1 Figure 11 - Success to pass high gap





6.6.2 Figure 12 - Failure to pass low gap