RL Midproject Report

Jonathan Mendelson, ID 308564293

1. Introduction and Methodology

The project was divided into three phases:

1. **Implementation** of pre-processing, reward shaping and agents. Feature number, reward parameters and learning parameters were left adjustable to allow experimentation.
2. **Investigation** of the parameter space by testing each hyper-parameter individually.
3. **Training & Validation** of the best-preforming agent using the parameters from section 2.

2.1. Preprocessing

Normalization: each feature is normalized by the min-max range. Features are normalized by the range [0,512] (horizontal and vertical). After some experimentation, the vertical speed is normalized to [-20,20] because values do not exceed this range in practice.

Feature Extraction: The total number of features is reduced to 5 by computing the vertical distance of the bird to the vertical center of the next and next-next pipe. For example, for the next pipe the computation is: `next_y_dist = [bottom_y+top_y]/2-player_y`. These features can be between [-512,512] and are normalized to [0,1] so that 0 is a perfect alignment.

The resulting observation vector is: `[player_y_velocity, next_x, next_y_dist, next_next_x, next_next_y_dist]`. This feature space captures the important information while reducing the number of features. The reduction is observation space will yield a smaller Q-table as well.

Feature Elimination: Optionally, by setting the global flag `disregard_next_next = True`, we can disregard the next-next pipe. In this case only the 3 first values in the vector are taken, which further simplifies the feature space.

2.2. Reward Shaping

Goals: The built-in reward mechanism rewards the agent by +1 for every pipe passed. We can identify additional goals that can be rewarded:

- The bird should survive as long as possible – survival is correlated with passing pipes.
- Passing pipes is a sparse reward – we would need to add a more continuous component.

Reward shaping: We shape the reward to better align the agent with our goals:

- A strong reward (+10) for passing a pipe and a strong (-100) negative reward for crashing.
- A constant reward for each step survived – can be tuned by the variable `frame_bonus`.
- Reward for centering the bird with the next gap – computed as `centering_weight * distance`, where `centering_weight` is a global variable that can be tuned.

2.3. Agents

Bin Discretization: The environment produces continuous observations in the range [0,1]. The `discretize_observation` function discretizes each continuous feature using equal width bins and returns a unique state. The number of bins is passed when constructing the agent , for example `bins = [10] * 5` uses 10 bins for each feature (here the feature vector is `len = 5`).

It is worth noting that 10 bins per feature yields 10^5 states, which will yield a q-table of 20K. Optimization is performed on the number of bins in the next section.

SARSA was first implemented as it is the most straight-forward. It has a few methods:

RL Midproject Report
Jonathan Mendelson, ID 308564293

- `init`: initialized the super class and also stores the `bins` used for discretization and initializes a Q-table of size `(np.prod(bins), self.action_space.n)`. The size of the Q-table is the multiplication of all features sizes used for discretization.
- `select_action` updates ϵ using `_update_epsilon_decay`, and then selects the next action from the action space using an ϵ-greedy policy; if the `deterministic` flag is used, the policy is a greedy policy.
- `update_policy` updates the policy according to the SARSA update rule.
- `train` is the main training loop, which follows the scehme presented in class, with an additional step of discretization using `discretize_ observation` after each step is taken. The flag `full_log` is used to generate a .txt log of each step within an episode.
- The methods `save_policy` and `load_policy` are helper functions as defined in the template.
- `run_policy` runs an episode with a deterministic greedy policy, without policy update.

Q-Learning was implemented by inheriting the `SARSA_agent` class and overloading the `update_policy` function using the q-learning update rule (off-policy).

Epsilon-Decay was implemented in all agents, where the epsilon value decays in a linear fashion. The start and end values, as well as the decay period, can be configured.

Validation or testing of a policy is carried out by the `validate_agent` function, which uses the agent's `run_policy` method for a specified number of episodes and returns the mean, variance, max and min of both score and reward; they are plotted with a smoothing window.

3. Investigation Strategy

We will investigate the agent's **discretization** (Q-table size), **learning parameters** (α, γ, ϵ), **reward shaping**, and **agent type** one at a time. Assuming independence, we will identify the best settings for each and combine them to build and test the optimal agent. Table 1 details an initial baseline testing without any optimization; see figure 1 for convergence results.

Observations using the next-next pipe weren't used, as score with 3 features (next pipe only) was satisfactory.

3.1. Discretization

We investigate the effect of the number of bins on the performance. A SARSA agent was trained for 20,000 episodes with $\alpha = 0.2,\ \epsilon = 0.1,\ \gamma = 0.9$. Only the closest pipe is considered so the observation vector is of size 3. Reward shaping is employed as explained in section 2, without a reward for survival (`frame_bonus = 0`) or for centering with the gap (`centering_weight = 0`).

The tested bin configurations are: Low [2, 5, 10], Mid [10, 20, 20] and High [20, 40, 80]; the resulting Q-table size is [100 X 2], [4000 X 2] and [64,000 X 2] respectively. A higher bin count was given to the 3rd feature as the Y range is larger than X - 512 px vs 288 px; see table 2.

A high bin count provides a better score by representing the state more accurately but takes more time to converge – see figure 2. For further experiments a mid-binning density of [10, 20, 20] bins per feature will be used, while a high bin density will be used in the final training.

3.2. Learning Parameters

The parameters to investigate include the ϵ decay rate, γ and α. We start with ϵ, by holding $\alpha = 0.2,\ \gamma = 0.9$ and varying the decay rate. The SARSA agent is used with medium binning and no additional reward for centering or survival. Full details are given in table 3.

RL Midproject Report

Jonathan Mendelson, ID 308564293

It was found that the epsilon decay has a significant effect on the convergence and performance of the agent. Holding ϵ at a constant low number prohibits the agent from learning and performance remains largely stochastic. A fast decay (after 5K episodes) has a positive effect, but a larger decay (10K\15K episodes) has a much more noticeable effect. See figure 3.

Early decay risks **overfitting**. For example, when ϵ decays after 5K episodes performance drops around 10K episodes. This highlights the importance of choosing the optimal training duration.

Next, we optimize α and γ by testing a few cases – see table 4. It was found that a high γ value yields better results, while a high α does not converge and performance can even fall after a while, such as in fig. 4 case 1. Therefore, we will keep $\gamma = 0.9$ and the learning rate $\alpha = 0.1 \sim 0.2$.

3.3. Reward Shaping

We first test `centering_weight` in the range [0, 0.1, 0.5, 1, 5]; a higher weight indicates a greater reward for keeping the bird centered with the next gap. The `frame_bonus` is 0 and all other parameters are held constant; full details are given in table 5. The best result is achieved with the `centering_weight = 5`. However, we choose to be conservative and keep the weight a bit lower (0.5 ~ 1) to prevent the centering reward from dominating the reward function entirely.

Next, we investigate `frame_bonus` in the range [0.01, 0.1, 1, 10]; this is the reward per survived frame. The `centering_weight` is set to 0; see full details in table 6. We were surprised that a reward of 0.01 yields the best result. We expected a higher survival bonus to correlate with pipes passed, but this may be redundant to the other rewards given; see figure 6.

3.4. Agent Type (Learning Algorithm)

Comparing the **Q-Learning** and **SARSA** algorithms in figure 7, we see that the q-learning agent converges faster and returns a significantly higher score. However, q-learning is very prone to overfitting with performance declining rapidly after convergence.
Therefore, for the final training Q-Learning will be used with a shorter training of 10K episodes. Results will be validated every 1K episodes of training to avoid overfit.

4. Final Training and Validation

Based on our experimentation we select the best parameters: **high** density discretization; **learning rate** is $\alpha = 0.15$ to stabilize the learning process; $\boldsymbol{\gamma} = 0.9$ and **ϵ-decay** after 5K episodes; `centering_weight = 0.1` and `frame_bonus = 0.01`. Full details are given in table 8.

The agent was trained for 10K episodes, with results saved and validated every 1K episodes. Validation included running 100 episodes, computing metrics, and generating a video for the 100th episode. A total of 10 video files were generated during training.

Figure 8 shows that while the agent's convergence is noisy, no overfitting occurred. Validation results in Figure 9 confirm that the max score improved, while the min score remained 0, and the success rate (games with scores > 10) reached a maximum of 87.

A few directions for improvement could be:

- **Additional training** to improve the score; however, further training may lead to overfitting.
- **Implementation of α decay** or **early stopping** to avoid overfit altogether.
- **Reward function reshaping**: to penalize losses more heavily, encouraging the agent to be more cautious and risk averted.

RL Midproject Report

Jonathan Mendelson, ID 308564293

Appendix 1: Tables and Figures

Table 1: Parameters and performance of the baseline agent

Parameter	**Baseline**			
Agent Type	SARSA			
Features	3 features [player_y_dot, next_x, next_y_dist]; normalized			
Binning	[5, 10, 10]			
Q-table dim	[500 X 2]			
Reward Structure	Frame bonus = 0, centering weight = 0			
Learning Parameters	$\gamma = 0.9, \epsilon = 0.05, \alpha = 0.2$			
Metric	**Mean**	**VAR**	**MAX**	**MIN**
Reward	-94.9	140.989	-40.0	-100.0
Score	0.51	1.40	6	0

Figure 1: baseline agent convergence

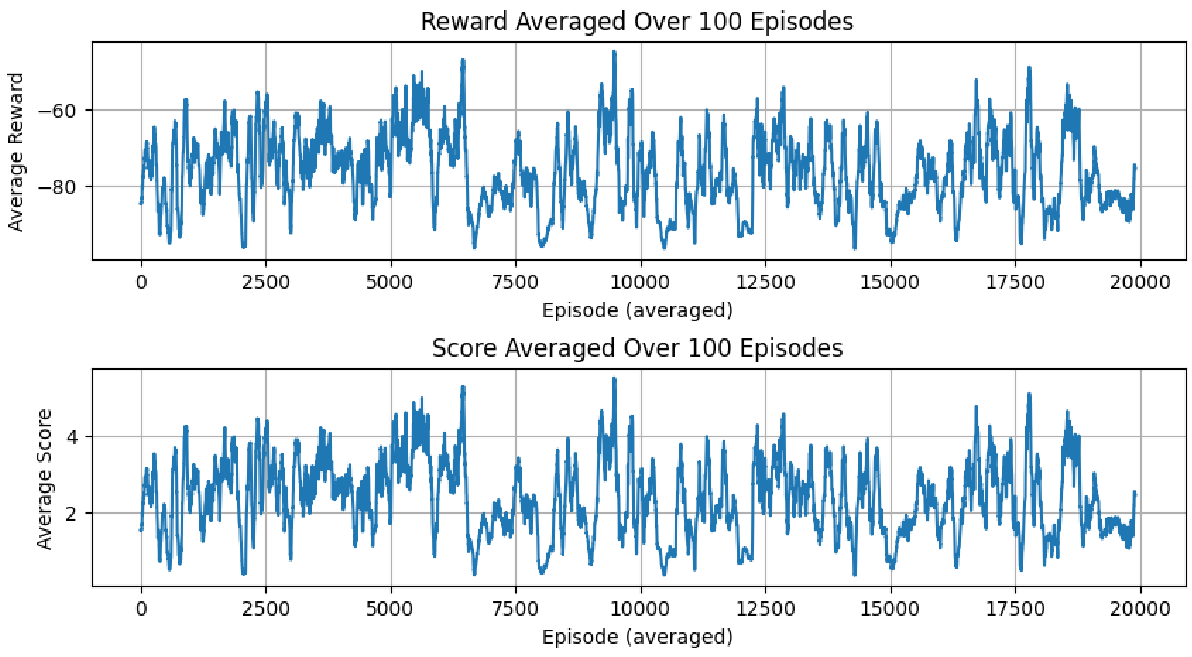


Table 2: Parameters for binning optimization

Parameter	**LOW**	**MID**	**HIGH**
Agent Type	SARSA		
Features	3 features [player_y_dot, next_x, next_y_dist]; normalized		
Binning	[2, 5, 10]	[10, 20, 20]	[20, 40, 80]
Q-table dim	[100 X 2]	[4000 X 2]	[64000 X 2]
Reward Structure	Frame bonus = 0, centering weight = 0		
Learning Parameters	$\gamma = 0.9, \epsilon = 0.1, \alpha = 0.2$		

RL Midproject Report

Jonathan Mendelson, ID 308564293

Figure 2: Binning optimization results – convergence graph

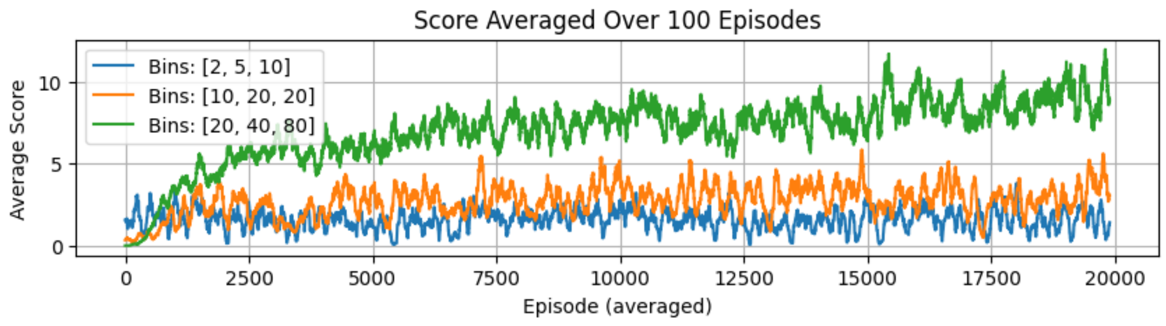


Table 3: Parameters for epsilon decay optimization

Parameter	Case 1	Case 2	Case 3	Case 4	Case 5
Agent Type	SARSA				
Features	3 features [player_y_dot, next_x, next_y_dist]; normalized				
Binning; Q-table dim	[10, 20, 20]; [4000 X 2]				
Learning Parameters	$\gamma = 0.9, \epsilon = 0.05, \alpha = 0.2$				
Decay strategy [start, end, episodes]*	Static $\epsilon = 0.05$	Static $\epsilon = 0.2$	Decay [1.0, 0.05, 5000]	Decay [1.0, 0.05, 1000]	Decay [1.0, 0.05, 15000]
Reward Structure	Frame bonus = 0, centering weight = 0				

* episodes denotes the number of episodes it takes to decay from start to end epsilon value.

Figure 3: Epsilon decay optimization results – convergence graph

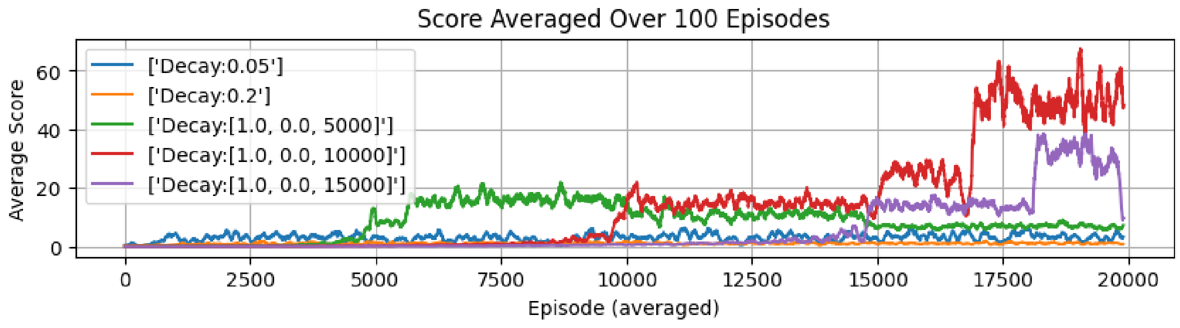


Table 4: Parameters for learning optimization

Parameter	Case 1	Case 2	Case 3	Case 4	Case 5
Agent Type	SARSA				
Features	3 features [player_y_dot, next_x, next_y_dist]; normalized				
Binning; Q-table dim	[10, 20, 20]; [4000 X 2]				
Decay Strategy	[Start, End, Episodes]: [1.0, 0.05, 5000]				
Learning Parameters	$\alpha = 0.5$, $\gamma = 0.9$	$\alpha = 0.8$, $\gamma = 0.9$	$\alpha = 0.2$, $\gamma = 0.5$	$\alpha = 0.2$, $\gamma = 0.99$	$\alpha = 0.2$, $\gamma = 0.9$
Reward Structure	Frame bonus = 0, centering weight = 0				

RL Midproject Report

Jonathan Mendelson, ID 308564293

Figure 4: Learning optimization results – convergence graph

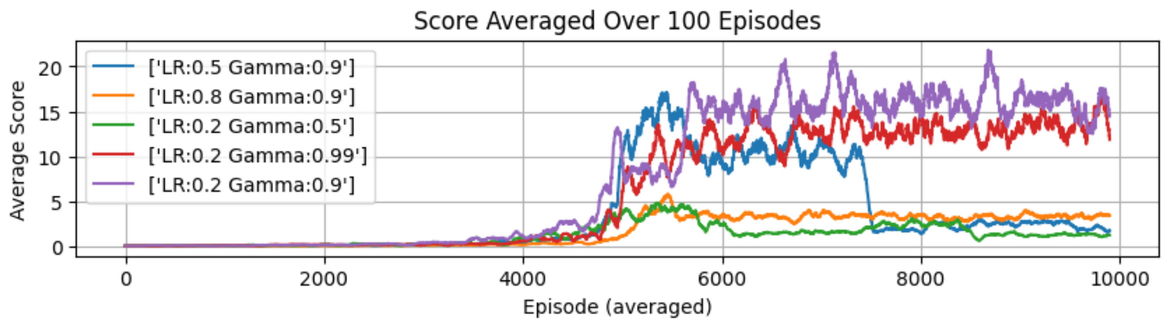


Table 5: Parameters for centering_weight optimization

Parameter	**Case 1**	**Case 2**	**Case 3**	**Case 4**	**Case 5**
Agent Type	SARSA				
Features	3 features [player_y_dot, next_x, next_y_dist]; normalized				
Binning; Q-table dim	[10, 20, 20]; [4000 X 2]				
frame_bonus	0	0	0	0	0
centering_weight	0	0.1	0.5	1	5
Learning Parameters	$\gamma = 0.9,, \alpha = 0.2$, epsilon-decay from 1.0 to 0 in 5K episodes				

Figure 5: centering_weight optimization results – convergence graph

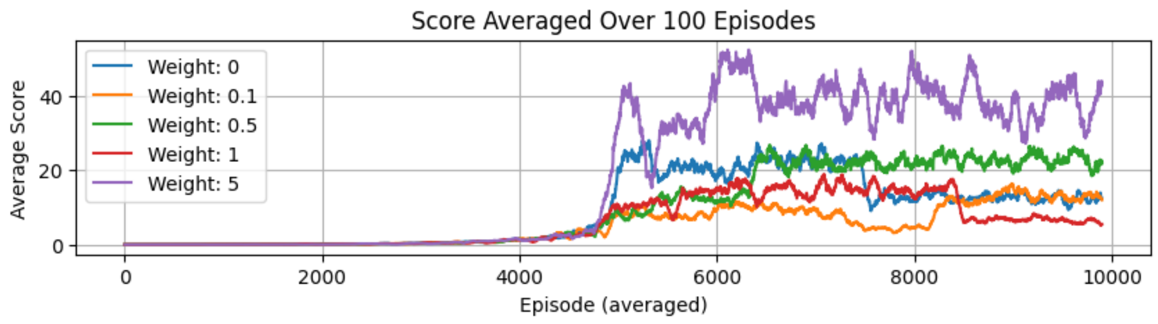


Table 6: Parameters for frame_bonus optimization

Parameter	**Case 1**	**Case 2**	**Case 3**	**Case 4**
Agent Type	SARSA			
Features	3 features [player_y_dot, next_x, next_y_dist]; normalized			
Binning; Q-table dim	[10, 20, 20]; [4000 X 2]			
frame_bonus	0.01	0.1	1	10
centering_weight	0	0	0	0
Learning Parameters	$\gamma = 0.9, \alpha = 0.2$, epsilon-decay from 1.0 to 0 in 5K episodes			

Figure 6: frame_bonus optimization results – convergence graph

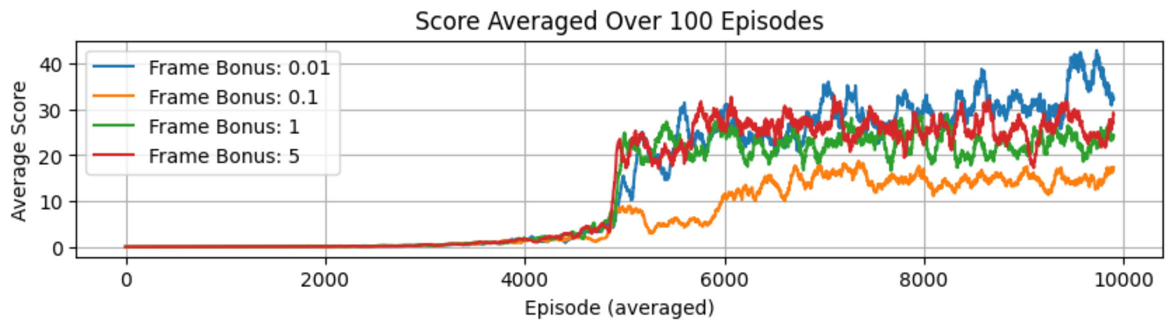


Table 7: Parameters for agent comparison

Agent Type	SARSA	Q-Learning
Features	3 features [player_y_dot, next_x, next_y_dist]; normalized	
Binning; Q-table dim	[10, 20, 20]; [4000 X 2]	
Reward Structure	Frame bonus = 0, centering weight = 0	
Learning Parameters	$\gamma = 0.9, \alpha = 0.2$, epsilon-decay from 1.0 to 0 in 5K episodes	

Figure 7: Agent comparison results - convergence graph

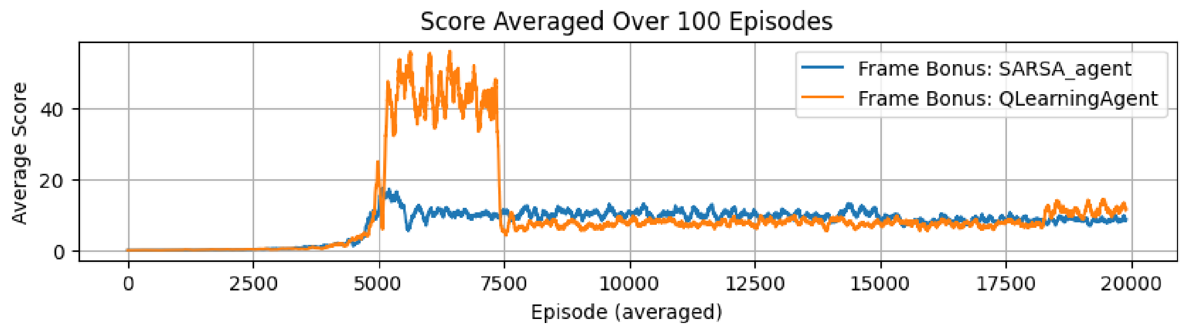


Table 8: Optimal agent parameters

Parameter	
Agent Type	Q-Learning
Features	3 features [player_y_dot, next_x, next_y_dist]; normalized
Binning; Q-table dim	[10, 40, 40]; [4000 X 2]
Reward Structure	Frame bonus = 0.01, centering weight = 0.1
Learning Parameters	$\gamma = 0.9, \alpha = 0.15$, epsilon-decay from 1.0 to 0 in 5K episodes

Figure 8: Optimal agent convergence graph

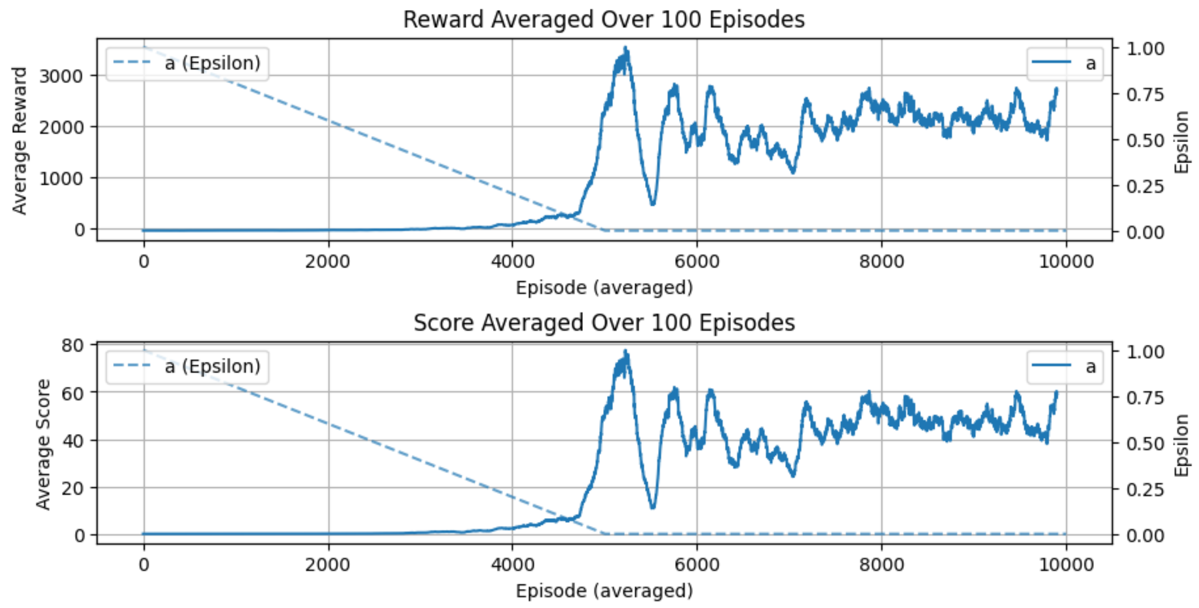
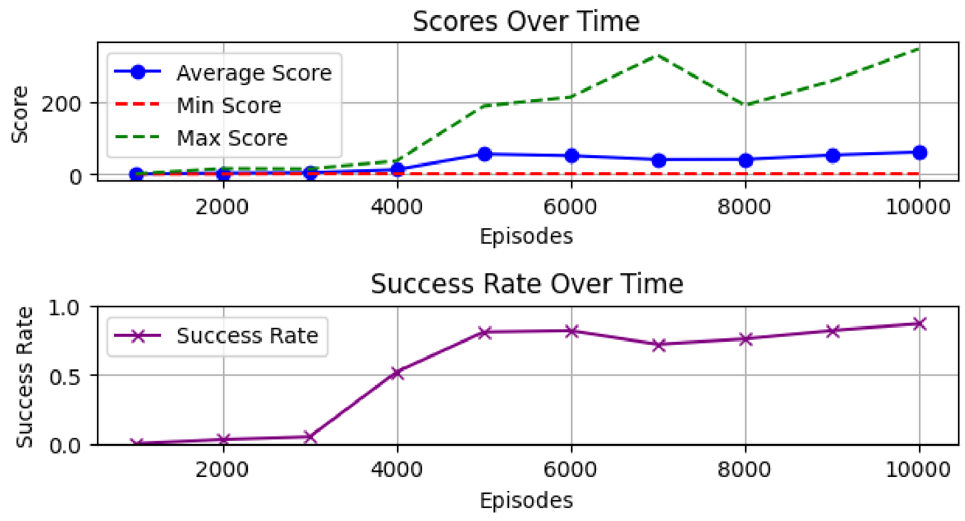


Table 9: Optimal agent validation results



RL Midproject Report

Jonathan Mendelson, ID 308564293

Appendix 2: External Links

Training is handled by the “Training_…” notebook.

Inference and validation are in the “Load_…” notebook.

Repository

Optimal Agent Training Videos

Colab Folder