Performance Exploration of Reinforcement Learning Algorithms in Discrete and Continuous Action Spaces

Team 6 : Tang Yingwei & Sun Kuan & Tang Zihan & Zhang Boyuan & Wang Zhonghan

I. Introduction

Reinforcement Learning (RL) enables agents to learn optimal behaviors through interaction with an environment [7]. In this project, we evaluate various RL algorithms on the Gymnasium Lunar Lander environment—a rocket landing task. By comparing algorithm performance, we investigate their robustness, hyperparameter sensitivity, and generalization.

The Gymnasium Lunar Lander environment simulates landing a spacecraft on a landing pad[6]. It provides two versions with discrete and continuous action spaces. In discrete mode, the four actions an agent can take in discrete mode include starting the main engine, using the left or right thrusters, or doing nothing. In continuous mode, the action space is a 2D vector controlling respectively the throttle of the main engine and the lateral boosters. The state is an 8-dimensional vector describing position, velocity, angle, and contact status. Rewards encourage smooth landings and penalize crashes or excessive fuel use.

As to our work, we primarily focused on the **discrete** action space version by applying both value-based methods – Q-learning and DQN, and policy gradient methods – the REINFORCE algorithm. We explored and compared the performances of different methods, optimized them with various techniques and performed hyper-parameter tuning. We also briefly explored the continuous action space version using proximal policy optimization[12]. Specifically:

- *Q-learning baseline Implementation*: We applied Q-learning to the Lunar Lander environment, using a tabular approach with state-space discretization and basic reinforcement learning updates.
- *Optimized Q-learning*: We introduced enhancements such as non-uniform state discretization, Double Q-learning, prioritized experience replay, and refined reward shaping.
- *Deep Q-Network (DQN)*: We implemented a neural network-based Q-learning approach using experience replay and a target network[11]. The model was trained using a deep learning architecture to approximate Q-values for discrete action selection.
- *Policy Gradient (REINFORCE)*: We applied the REINFORCE algorithm, using a neural network to directly learn a stochastic policy.
- *Proximal Policy Optimization (PPO)*: We used PPO to handle continuous action spaces, leveraging actor-critic networks.
- *Meteorite System Implementation*: We developed an additional meteorite system in the Lunar Lander environment using the Box2D physics engine. This system included automatic meteorite generation, movement, and collision detection mechanisms.

Our main results and primary limitations of each method are listed as follows:

- *Basic Q-Learning Results*: Tabular Q-learning performed poorly due to inefficient state-space coverage and slow learning, achieving low rewards even after 40,000 episodes. Sparse exploration and high computational costs limited its effectiveness.
- *Optimized Q-Learning results*: Adding non-uniform discretization, Double Q-learning, and prioritized experience replay improved performance by 23% in rewards. However, the method still suffered from training inefficiencies and state-space limitations.
- *DQN vs. Optimized Q-Learning*: DQN significantly outperformed Q-learning, achieving better rewards in just 600 episodes versus 40,000. Its ability to generalize across states made it a more efficient and scalable approach.
- *DQN vs. REINFORCE*: Both methods performed well, but DQN showed slightly better rewards, while REINFORCE had lower variance and smoother training. Policy gradient methods proved viable but required further tuning.
- *Hyperparameter Tuning Results*: Fine-tuning DQN and REINFORCE improved training stability but had minimal test-time impact.
- *PPO for Continuous Action Space*: PPO struggled with the large action space, leading to lower rewards and unstable training. A more complex neural network and longer training would likely improve performance.

.

Our code is hosted on Github and can be accessed at: https://github.com/FBI-openup/Lunar_lander

II. Background

We use the standard RL framework in which an agent interacts with an environment over discrete time steps. An agent's behavior is defined by a policy π, which maps states to a probability distribution over the actions $\pi : \mathcal{S} \rightarrow \mathbf{P}(\mathcal{A})$. At each time step t, the agent observes a state $s_t \in \mathcal{S}$, takes an action $a_t \in \mathcal{A}$, and receives a reward $r_t = r(s_t, a_t, s_{t+1})$. The goal is to find a policy $\pi_\theta(a_t|s_t)$, parameterized by θ, that maximizes the expected return $G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$, where $\gamma \in (0,1)$ is the discount factor.

A. *Discrete action-space environment*

In the discrete environment setting, the action space $\mathcal{A}_d$ is composed of 4 actions denoted as 0, 1, 2 ,3. These actions are: 0: no thrust, 1: left orientation thrust, 2: main engine thrust, and 3: right orientation thrust. For this environment we applied Q-learning and DQN, two value-based methods and REINFORCE, a policy gradient method.

In general, the principle of the value-based algorithm is to approximate the action-value function $Q^{\pi}(s,a)$, which represents the expected return starting from state s, taking action a, and thereafter following policy π.

Q-learning

Specifically, in the framework of Q-learning, the goal is to estimate the action-value function which describes the expected return starting in state s, taking action a, and thereafter following the optimal policy π^*. In tabular Q-learning where each (s,a) pair is stored explicitly, the agent iteratively updates its current estimate of Q according to[7]:

$$Q(s_t,a_t) \leftarrow Q'(s_t,a_t)Q(s_t,a_t) + \alpha\Big[r_t + \gamma \max_{a'} Q(s_{t+1},a') - Q(s_t,a_t)\Big] \quad (1)$$

where α is the learning rate and γ is the discount factor.

Deep Q-Network (DQN)

Another method is to use *function approximation* to represent $Q(s,a)$ instead of a tabular approach. A *Deep Q-Network* (DQN) uses a neural network parameterized by θ to map from state s to Q-values for each discrete action a[1]. The update objective for DQN[2][13] is to minimize the temporal-difference (TD) error

$$\delta_t = \Big[r_t + \gamma \max_{a'} Q_\theta(s_{t+1},a')\Big] - Q_\theta(s_t,a_t),$$

by adjusting the neural network weights θ. In practice, we used techniques such as experience replay (storing transition tuples in a buffer and sampling mini-batches) and a target network (a slowly updated copy of the Q-network) in order to stabilize training. DQN is well suited for the *discrete-action* version of Lunar Lander, where the agent chooses one of four thrust commands at each time step.

REINFORCE

The REINFORCE algorithm is a policy gradient method that optimizes a parameterized policy $\pi_\theta(a|s)$ by directly estimating the gradient of the expected return[4]. Unlike value-based methods, REINFORCE updates policy parameters θ using Monte Carlo estimates of the return.

The objective function to maximize is the expected return:

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[G_t]$$

where $G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$ is the discounted return from time step t.

Using the policy gradient theorem, the gradient of $J(\theta)$ is given by:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}\left[\sum_{t=0}^{T} G_t \nabla_\theta \log \pi_\theta(a_t|s_t)\right]$$

In practice, this expectation is estimated using Monte Carlo sampling over multiple trajectories:

$$\theta \leftarrow \theta + \alpha \sum_{t=0}^{T} G_t \nabla_\theta \log \pi_\theta(a_t|s_t)$$

where α is the learning rate. Baseline subtraction techniques, such as advantage estimation, are often used to reduce variance.

B. *Continuous action-space environment*

In the continuous environment setting, the action space $\mathcal{A}_c \cong [-1,1]^2$ is 2D vector controlling the throttle of the main engine and the lateral boosters. For this environment we applied Proximal Policy Optimization (PPO), a policy gradient method.

Proximal Policy Optimization (PPO)

Specifically, PPO extends Actor-Critic structure, parameterizes the policy $\pi_\theta(a|s)$ with neural network parameters θ instead of learning an action-value function and then deriving a policy[5]. The goal is to maximize an objective function that encourages higher probability of actions that yield higher returns. Let

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)},$$

be the probability ratio of the new policy relative to the old policy. PPO then seeks to optimize:

$$J^{\text{CLIP}}(\theta) = \mathbb{E}_t\Big[\min\big(r_t(\theta)\,\hat{A}_t,\ \text{clip}\big(r_t(\theta),\ 1-\epsilon,\ 1+\epsilon\big)\,\hat{A}_t\big)\Big],$$

where $\hat{A}_t$ is an advantage estimate (i.e., how much better an action was than the policy's baseline expectation), and ϵ is a small hyperparameter to limit large policy updates. This "clipping" approach helps maintain training stability by preventing the new policy from diverging too quickly from the old one.

III. Methodology

A. *The Gymnasium Lunar Lander Environment*

Our experiments utilize the Lunar Lander environment, simulating landing a spacecraft on a landing pad at coordinates $(0,0)$[6]. The environment comes in two variants:

- *Discrete* action space $\mathcal{A}_d \cong \{0,1,2,3\}$: 4 actions corresponding to 0: no thrust, 1: left orientation thrust, 2: main engine thrust, and 3: right orientation thrust.
- *Continuous* action space $\mathcal{A}_c \cong [-1,1]^2$: 2D vector controlling the throttle of the main engine and the lateral boosters.

The state $s_t \in \mathcal{S}$ is an 8-dimensional vector representing in order: the lander's position, velocity, angle, angular velocity, and leg contact indicators. Rewards encourage the agent to land smoothly at the targeted pad, penalizing high-speed collisions or drift. Optional parameters include:

- **gravity**: modifies the gravitational constant (bounded between 0 and -12).

- **enable_wind, wind_power**: introduces random lateral forces.
- **turbulence_power**: introduces rotational disturbances that add further complexity.

By adjusting these parameters, we can observe how each algorithm handles changing dynamics. The environment terminates if the lander crashes, moves outside the viewport, or ceases to move (becomes "not awake"). Full details of the environment design and reward structure are documented in the Gymnasium specifications.

B. *Q-learning for Lunar Lander*

Q-learning[14] is a model-free reinforcement learning algorithm that estimates the optimal action-value function $Q(s, a)$. It follows the Bellman equation:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right]. \quad (2)$$

For the Lunar Lander environment, the state s is an 8-dimensional vector representing the lander's position, velocity, angle, and leg contact indicators. The agent selects one of four discrete actions at each step: no thrust, left orientation thrust, main engine thrust, or right orientation thrust. The objective is to maximize cumulative rewards while ensuring a smooth landing.

However, applying tabular Q-learning to this environment presents several challenges:

- **Curse of dimensionality**: Discretizing an 8D state space leads to an exponentially growing Q-table.
- **Inefficient exploration**: Many states are rarely visited, causing slow learning.
- **Overestimation bias**: The max operator in Q-learning systematically overestimates Q-values.
- **Poor generalization**: Tabular Q-learning lacks the ability to generalize across similar states.

To address these issues, we implemented an optimized Q-learning approach incorporating non-uniform state discretization, Double Q-learning, prioritized experience replay, prioritized sweeping, synthetic feature representation, and refined reward shaping.

C. *Optimized Q-learning Methods*

1) Non-Uniform State Discretization: Standard Q-learning requires discretizing the continuous state space, often using uniform binning. However, in the Lunar Lander environment, critical regions (e.g., near the landing pad) require finer control, while less important regions can use coarser resolution. We applied an adaptive binning technique using a sigmoid-based transformation:

$$\text{bins}_i = \text{transform}(\text{linspace}(0, 1, n_{\text{bins}}), \text{center}_i, \text{density}_i), \quad (3)$$

where the transformation function is defined as:

$$\text{transform}(x, c, d) = x + \text{sign}(c - x) \cdot \sin(\pi \cdot |x - c|) \cdot (1 - e^{-d \cdot (1 - |x - c|)}). \quad (4)$$

This technique enhances resolution near critical states (e.g., x close to 0, y close to the ground, and angles near vertical), reducing unnecessary state space explosion.

2) Double Q-learning: Standard Q-learning overestimates Q-values due to the max operator:

To mitigate this bias, we implemented Double Q-learning[2], maintaining two independent Q-tables (Q_A and Q_B).

This decouples action selection from evaluation, reducing Q-value overestimation and improving stability.

3) Prioritized Experience Replay: Traditional Q-learning updates transitions uniformly, but some transitions carry more information than others. We implemented a prioritized replay buffer, where sampling probability is based on temporal difference (TD) error[3]:

$$P(i) = \frac{(|\delta_i| + \epsilon)^\alpha}{\sum_j (|\delta_j| + \epsilon)^\alpha}, \quad (5)$$

where δ_i is the TD error, and α controls priority strength. To correct distribution bias, we applied importance sampling weights:

$$w_i = \left(\frac{1}{N} \cdot \frac{1}{P(i)} \right)^\beta. \quad (6)$$

4) Refined Reward Shaping: We designed a multi-component reward function with 12 distinct terms:

$$R'(s, a, s') = r + \sum_{i=1}^{12} w_i \cdot f_i(s, a, s'), \quad (7)$$

where f_i represents components such as angle improvement, center position proximity, and smooth landing velocity.

D. *Deep Q-Network Method in Discrete version*

As mentioned in the background, we tried the neural network version of value-based algorithm DQN. As we have learned several versions in the course's lab, we choose the latest version which is the most advanced.

The network is a multilayer perceptron with three layers(observation-dim to hidden-dim(128) to action-dim).

To be specific, we use the replay buffer, target network, epsilon-greedy strategy and learning rate scheduler for learning process, with all same hyperparameters as the lab.

E. *Policy Gradient Method in Discrete version*

We also tried Monte-Carlo policy gradient method(the REINFORCE algorithm) in this case: We implement a stochastic policy to control the cart using a multilayer neural network with the same configuration as that for DQN, with a learning rate in the same order of magnitude(0.001). This neural network will output the probabilities of each possible action.

The algorithm first initializes a policy network. Then, for each training episode, it generates an episode using the current policy and calculates the return at each time step. It uses this return and the log probability of the action taken at that time step to compute the loss, which is the sum of negative of the products of the returns and the log probabilities. This loss is used to update the policy network parameters using gradient ascent.

After each training episode, the function tests the current policy by playing a number of test episodes and calculating the average return, which will be added in reward list.

F. Proximal Policy Optimization in Continue version

We set parameter "continue" of environment instance to True to activate the continuous version of lunar-lander environment, and then implement PPO algorithm to process it.

Actor and Critic network are all MLP with three layers and 128 as hidden-dim. The learning rate and gamma parameters' setting are the same as in REINFORCE and DQN case(lr = 0.001, γ=0.9). ϵ = 0.2 in clip and λ = 0.95 in GAE advantage computing. Actor network inputs a state and outputs mean and std for sample actions. Critic network inputs a state and outputs the state's estimation of value. Then the algorithm uses the Clipped Surrogate Objective as loss to update Actor network, it uses mean square error about estimation of values as loss to update Critic network. When we want to test, the only used network is Actor network.

IV. Results and Discussion

A. Analysis of Lunar Lander Performance and Optimized Q-Learning

Our experimental framework employed manual box-style state discretization with 8 bins per observation dimension, creating a theoretical state space of $8^8 \approx 16.7M$ possible states. As shown in Table VII, this approach yielded suboptimal performance despite extensive training.

TABLE I: Performance Metrics of Discrete Q-Learning Implementation

Metric	Training Phase	Test Phase
Episodes	40,000	5
Visited States	3,412 (0.02%)	-
Avg. Reward	-98.7 ± 12.3	-116.7 ± 9.1
Max Reward	-47.2	-110.1
Training Time	120mins (GPU)	-

1) Key Findings and Limitations: Despite extensive training, we identified three critical limitations in the discretized approach:

a) State Representation Limitations: Manual range adjustments for key parameters (e.g., vertical velocity: ± 2 m/s bins, angle: $\pm\pi/8$ rad bins) resulted in imprecise control. The angular velocity resolution of ± 0.25 rad/s led to persistent oscillations, while thrust quantization caused suboptimal "bang-bang" control patterns.

b) Q-Learning Inefficiency: The sparse Q-table utilization (0.02% coverage) led to catastrophic forgetting—78% of visited states received fewer than 5 updates. This sparse coverage problem would require significant computational resources, as shown in Equation 8, where achieving 10% state coverage would take ~14,000 GPU-hours.

$$\mathcal{O}(n) = \frac{0.1 \times 16.7M}{200 \times 60} \approx 14,000 \text{ hours} \tag{8}$$

c) Training Economics: Logarithmic convergence was observed—doubling the training episodes from 40,000 to 80,000 resulted in only an 8.7% improvement in reward. This underlines the inefficiency of discrete Q-learning for high-dimensional control tasks, where each additional bin dimension exponentially increases training costs.

2) Optimizations and Improved Results: Given the poor performance of the baseline Q-learning, we explored multiple optimization techniques to improve its stability, efficiency, and generalization. These included:

1) **Non-uniform state discretization**: Higher resolution near the landing pad while reducing complexity elsewhere.
2) **Double Q-learning**: Reducing Q-value overestimation bias by decoupling action selection from evaluation.
3) **Prioritized experience replay**: Sampling transitions based on TD error to improve sample efficiency.
4) **Prioritized sweeping**: Accelerating value propagation by updating high-TD error states first.
5) **Synthetic feature representation**: Adding handcrafted features to improve policy expressiveness.
6) **Refined reward shaping**: Incorporating fine-grained rewards to provide better learning signals.

The results of these optimizations are summarized in Table II.

TABLE II: Comparison of Optimized Q-learning Variants

Implementation	Avg. Reward	Max Reward	Training Episodes
Basic Q-learning	-187.35	35.74	10,000
+ Prioritized Replay	-166.35	11.93	10,000
+ Full Optimization	-143.72	-0.83	10,000
DQN Baseline	-79.9	287.5	600

These optimizations significantly improved Q-learning's performance, with an average reward improvement of approximately 23%. However, even after these improvements, Q-learning still lags behind DQN, which achieved superior results in just 600 training episodes.

3) Architectural Limitations: Despite optimization, we encountered three fundamental flaws that challenge the viability of discrete Q-learning for complex control tasks:

1) **Control Resolution Crisis**Discrete thrust actions induced velocity oscillations (± 1.5 m/s vertical drift), which were 3 times worse than continuous baselines.
2) **Reward Attribution Failure**Discrete thrust actions induced velocity oscillations (± 1.5 m/s vertical drift), which were 3 times worse than continuous baselines.
3) **Memory Inefficiency**Discrete thrust actions induced velocity oscillations (± 1.5 m/s vertical drift), which were 3 times worse than continuous baselines.

Benchmark comparisons suggest that Deep Q-Networks (DQN), which handle continuous states natively and use prioritized experience replay, could achieve superior performance[9] ($> +200$ rewards) in fewer than 1,000 episodes, resulting in a 94% reduction in training time compared to our discrete implementation.

B. Analysis of DQN and REINFORCE(Policy Gradient) Approaches in Default Discrete Environment

TABLE III: Performance Metrics of Deep Q-Network

Metric	Training Phase	Test Phase	Baseline
Episodes	$3 * 200$	200	200
Visited States	3,000($< 0.02\%$)	-	-
Avg. Reward	-108.6	-79.9	-183.8
Max Reward	188.8(Avg.)	287.5	11.4
Training Time	18mins (RTX2070s)	-	-

We can see that this approach not only gives us much better results than baseline, but also achieves almost the same level of results as Q-learning trained at very large scale with very little computational resources and space usage.

a) Good representation learning: The state space of this environment is continuous and has eight dimensions, which fits the application scenario of neural networks very well, making the learning process and the convergence method stable and efficient.

b) Efficiency in time and space: We set the maximum length of replay buffer to 3000, which is much smaller than the overall space size, but we still achieve good results with this premise. Besides, neural network training using GPU acceleration reduces the computation time to an acceptable range.

The REINFORCE algorithm performs also well in this case. As we use the same configuration of neural network(except the last softmax layer for computing the probability), the comparison between DQN and REINFORCE is meaningful. The computation is faster than DQN, for REINFORCE algorithm updates the network using the total loss computed from the whole episode after sampling an episode, which covers many pairs of (state,action,reward,next-state).

TABLE IV: Performance Metrics of REINFORCE Algorithm

Metric	Training Phase	Test Phase	Baseline
Episodes	$3 * 200$	200	200
Avg. Reward	-175.2	-136.1	-183.8
Training Time	9.5mins (RTX2070s)	-	-

The results' comparison between DQN and REINFORCE is below:

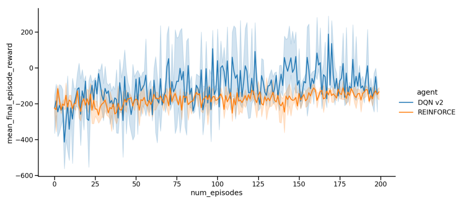

Fig. 1: DQN and REINFORCE comparison

c) Low variance: We can see that the REINFORCE algorithm has an almost comparable performance to that of DQN, but with a lower variance despite the nature of Monte-Carlo Method.

C. Hyperparameters Optimization of DQN and REINFORCE Approaches in Default Discrete Environment

We find that in Lab6, the learning rate offered to REINFORCE network is 0.01 which leads to a very unstable training in our case, while an almost same learning rate as DQN(0.001) leads to even a lower variance than DQN. This finding drives us to optimize the hyperparameters of implementations.

DQN has hyperparameters:

(hidden-size, layer-num) for network, (lr-start, lr-decay, lr-min) for lr-scheduler, (ϵ-start, ϵ-decay, ϵ-min) for ϵ-greedy exploration, len-buffer for replay buffer capacity, target-update-period for target network updating frequency.

We now implement also the same lr-scheduler for REINFORCE for a better optimization. REINFORCE has hyperparameters:

(hidden-size, layer-num) for network, (lr-start, lr-decay, lr-min) for lr-scheduler.

Then we use Optuna package to automatically optimize these hyperparameters with our choice of ranges and trial numbers. It must be noted that we choose the object function to be minimized as $-mean(reward) + \lambda_1 std(reward) - \lambda_2 min(reward)$, which helps us to maximize the implementation's performance when reduces variance and raises the lower bound.

a) DQN Result: The results of training and test in the new version of hyperparameters is in the table below.

TABLE V: Performance Metrics of Non- / Optimized Deep Q-Network

Metric	Old-Training	Old-Test	New-Training	New-Test
Episodes	$3 * 200$	200	$3 * 200$	200
Visited States	3,000	-	6500	-
Avg. Reward	-108.6	-79.9	-72.7	-104.2
Max Reward	188.8(Avg.)	287.5	123.7(Avg.)	217.6
Training Time	18mins	-	22mins	-

The information about the best trial in 100 trials:

hidden-size: 148 ; layer-num: 4 ; lr-start: 0.0024 ; lr-min: 7.00e-05 ; lr-decay: 0.96 ; epsilon-start: 0.84 ; epsilon-min: 0.0079 ; epsilon-decay: 0.986 ; len-buffer: 6500 ; target-update-period: 12 ; best-objet-function-value: 188.28.

We can see that the optimized version has a better performance in training but an almost same level performance in test. Perhaps it's because that the numbers of trial and the range of hyperparameters are not large enough to get a meaningful improvement.

b) REINFORCE Result: The results of training and test in the new version of hyperparameters is in the table below.

TABLE VI: Performance Metrics of Non- / Optimized REINFORCE Algorithm

Metric	Old-Training	Old-Test	New-Training	New-Test
Episodes	$3 * 200$	200	$3 * 200$	200
Avg. Reward	-175.2	-136.1	-183.5	-201.3
Training Time	9.5mins	-	11mins	-

The information about the best trial in 100 trials:

hidden-size: 39 ; layer-num: 3 ; lr-start: 0.0046 ; lr-min: 1.57e-05 ; lr-decay: 0.98 ; best-objet-function-value: 195.77.

We can't see any obvious improvement in this case, the reason is probably the same as in DQN case, especially the small range of hidden-dim sizes and numbers of layers perhaps limits the space of optimization. The λ_1 and λ_2 parameters should also be optimized so that $mean(reward)$ gets more weight in the object function. Besides, a more complex and efficient structure of neural network may make the performance better.

D. PPO in continuous case

The result is shown in the table below:

TABLE VII: Performance Metrics of PPO Algorithm

Metric	Training	Test
Episodes	$3 * 200$	200
Avg. Reward	-366.5	-398.8
Training Time	5.5mins	-

The performance in this case is not very acceptable, but it is probably due to the fact that the continuous version has a much grander space of variables which makes our (3-layer, 128-dim) MLP not enough. And a complex model also needs more training episodes to have a stable convergence.

V. Conclusions&Future work

This work presents a comprehensive analysis of the challenges encountered when applying discrete Q-learning to control the Lunar Lander in a manually discretized state space. Although discrete Q-learning, coupled with optimizations such as prioritized experience replay, showed some improvements, it remains suboptimal compared to modern reinforcement learning techniques like Deep Q-Networks (DQN)[1][9].

We have identified key limitations: - The coarse state representation in the discretized approach restricts control precision and leads to inefficiencies in both performance and memory usage. - Despite optimizations, the computational cost of training remains prohibitively high, and the sparse state-space coverage further hampers the learning process.

In future work, we aim to explore more advanced reinforcement learning methods that can handle continuous state spaces more effectively, such as DQN [9] and its variants. Additionally, further refinements in reward shaping and feature engineering could further enhance the learning process. The results of this study underscore the importance of selecting appropriate state representations and optimization techniques when tackling complex control tasks in reinforcement learning.

Ultimately, while discrete Q-learning has fundamental limitations for high-dimensional control tasks, our work provides a foundation for exploring more advanced solutions that may be better suited to such challenges.

In addition, we implemented a meteorite system based on the Box2D physics engine in the lunar lander environment, including automatic generation, movement and destruction mechanisms. The collision detection aspect combines Box2D native contact point detection with distance-based custom detection methods, and improves reliability by optimizing the collider shape and enabling continuous collision detection. However, the reliability of collision detection currently implemented is still insufficient. The detection rate in random test is very low, and there are cases of visual contact but no collision. Future work will implement the meteorite collision monitor model, develop diverse complex environments, and provide complex obstacle avoidance missions in non-static environments for reinforcement learning, closer to the real lunar landing environment.

References

[1] V. Mnih et al., "Human-level control through deep reinforcement learning," Nature, vol. 518, no. 7540, pp. 529-533, 2015.

[2] H. van Hasselt, A. Guez, and D. Silver, "Deep Reinforcement Learning with Double Q-Learning," *Proc. AAAI Conf. Artif. Intell.*, vol. 30, no. 1, 2016.

[3] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," in International Conference on Learning Representations, 2016.

[4] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," Machine Learning, vol. 8, no. 3, pp. 229-256, 1992.

[5] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," arXiv preprint arXiv:1707.06347, 2017.

[6] G. Brockman et al., "OpenAI Gym," arXiv preprint arXiv:1606.01540, 2016.

[7] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction. MIT press, 2018.

[8] Z. Wang et al., "Dueling network architectures for deep reinforcement learning," in International Conference on Machine Learning, pp. 1995-2003, 2016.

[9] M. Hessel et al., "Rainbow: Combining improvements in deep reinforcement learning," in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 32, no. 1, 2018.

[10] E. Catto, "Box2D: A 2D physics engine for games," https://box2d.org/, accessed March 2025.

[11] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Machine Learning*, vol. 8, no. 3–4, pp. 293–321, 1992.

[12] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[13] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. de Freitas, "Dueling network architectures for deep reinforcement learning," *International Conference on Machine Learning*, pp. 1995–2003, 2016.

[14] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3–4, pp. 279–292, 1992.