Reinforcement Learning Strategies with LunarLandingV3

Pantelis Kasioulis | 201801753 | p.kasioulis@liverpool.ac.uk
Ewa Fojcik | 201529790 | e.k.fojick@liverpool.ac.uk
Gabriel Thomas Newton | 201855293 | g.newton2@liverpool.ac.uk
Zoi Kallinaki | 201832458 | z.kallinaki@liverpool.ac.uk

1. Introduction

This report presents the implementation and analysis of a Deep Q-Network (DQN) agent for solving the Lunar Lander environment from OpenAI's Gymnasium. The Lunar Lander task requires an agent to learn to safely land a spacecraft on a landing pad without crashing. This environment presents a challenging control problem that requires precise actions based on the current state of the spacecraft. Our approach implements a DQN with different epsilon decay rates to examine how the exploration-exploitation trade-off affects learning performance and efficiency.

2. Problem Description

2.1. Lunar Lander Environment

The Lunar Lander environment simulates a spacecraft that must land safely on a landing pad. The state space consists of 8 continuous variables representing:

- Position coordinates (x,y)
- Linear velocities (x,y)
- Angle and angular velocity
- Two boolean flags indicating if each leg is in contact with the ground

The agent can take 4 discrete actions:

- Do nothing
- Fire left engine
- Fire main engine
- Fire right engine

The reward structure includes positive rewards for moving toward the landing pad and landing safely, negative rewards for moving away from the landing pad or crashing, and additional rewards/penalties for leg contact and fuel usage. The episode terminates when the lander crashes, lands successfully, flies out of bounds, or after a time limit.

2.2 Implementation Approach

A DQN agent was implemented to solve this environment. DQN combines Q-learning with neural networks to approximate the action-value function, allowing the agent to handle high-dimensional state spaces. Our implementation includes:

- A neural network to approximate the Q-function
- Epsilon-greedy exploration strategy with three different decay rates
- Immediate learning from experiences without a replay buffer
- MSE loss function to update the network weights

3. Methodology

3.1 Deep Q-Network Architecture

Our Q-network architecture consists of:

- Input layer with 8 nodes (matching the state space dimension)
- Two hidden layers with 64 neurons each using ReLU activation
- Output layer with 4 nodes (matching the action space)

3.2 Training Algorithm

A standard DQN algorithm was used with the following hyperparameters:

- Discount factor (gamma): 0.99
- Initial exploration rate (epsilon): 1.0
- Minimum exploration rate: 0.01
- Learning rate: 0.001
- Optimizer: Adam
- Loss function: Mean Squared Error (MSE)

A key aspect of our approach was experimenting with different epsilon decay rates to find an optimal exploration-exploitation balance:

- Slow decay: 0.003 per episode (decay rate = 0.997)
- Medium decay: 0.005 per episode (decay rate = 0.995)
- Fast decay: 0.01 per episode (decay rate = 0.99)

The training process involved 1000 episodes for each decay rate configuration. For each episode, the total reward and average loss were both recorded. Additionally, videos of agent performance at episodes 250, 500, 750 and 1000 were rendered to visualize learning progression. For these episodes, greedy was forced in order to properly understand the progression of the model, rather than rendering videos of potential explorative behaviour.

4. Results and Analysis

4.1 Training Performance

The training logs show the progression of rewards and losses across 1000 episodes for each decay rate.

For the slow decay rate (0.003), the agent started with a reward of –378.24 and loss of 108.56. By episode 250, the reward had decreased to –61.13 with a loss of 1.7174, showing initial struggles. At episode 500, the reward remained negative at –86.05 with a loss of 1.6626. By episode 750, performance improved significantly to 168.44 with a loss of 12.2229. Surprisingly,

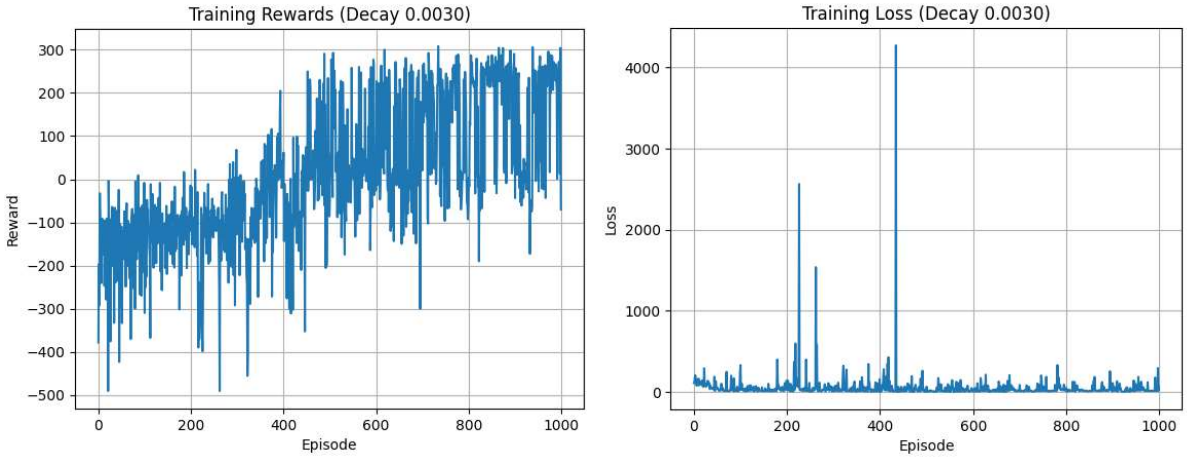
by episode 1000, the reward dropped to –70.14 with a high loss of 65.4380, indicating continued instability despite extensive training.

With the medium decay rate (0.005), the agent began with a reward of –342.36 and a high loss of 532.9626. It showed dramatic improvement by episode 250, reaching a reward of 235.08 with a loss of 21.1217. Performance continued to improve through episodes 500 and 750, with rewards of 276.62 and 280.76 respectively, and decreasing losses of 7.2383 and 7.5576. By episode 1000, the agent maintained good performance with a reward of 260.18 and a loss of 5.6427.

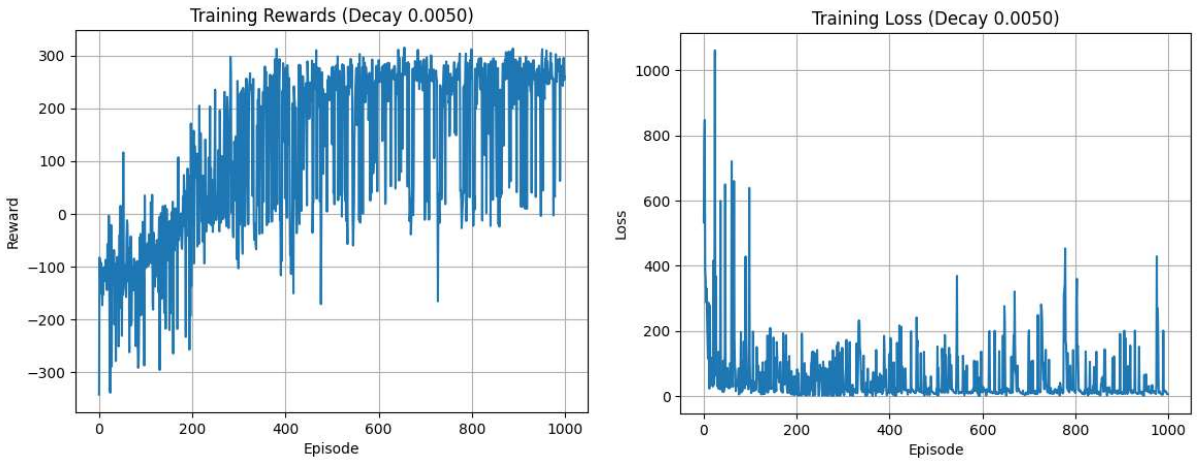
The fast decay rate (0.01) demonstrated the most effective learning progression. Starting from a reward of –138.63 and a very high initial loss of 1519.7164, the agent quickly achieved a substantial positive reward of 240.41 with a much lower loss of 17.2097 by episode 250. The agent maintained consistent performance through episodes 500, 750 and 1000, with rewards of 274.27, 295.28 and 259.04 respectively, while the loss steadily decreased to 9.8179, 8.5564 and finally 8.8076.

4.2 Reward and Loss Plots

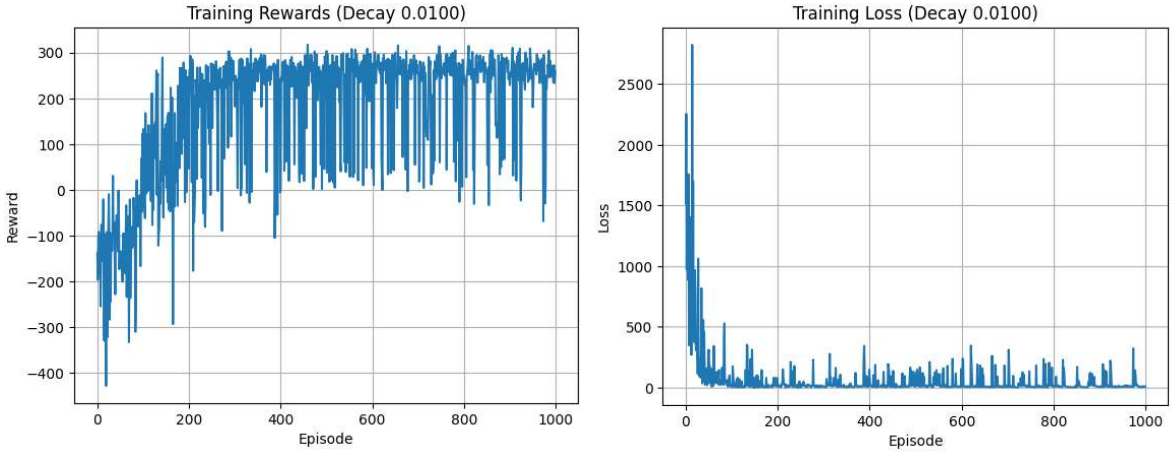
The **slow decay** (0.003) reward plot shows high volatility throughout all 1000 episodes, with frequent large negative rewards even in later stages. The loss plot reveals occasional extreme spikes up to 4000, suggesting unstable learning.



The **medium decay** (0.005) reward plot shows clearer learning progression, transitioning from predominantly negative rewards in the first 200 episodes to mostly positive rewards thereafter. The loss plot shows high initial values that quickly decreased and stabilized, with occasional moderate spikes.



The **fast decay** (0.01) reward plot demonstrates the most efficient learning curve, with rapid transition to consistently positive rewards around episode 150. The loss plot shows extremely high initial values but the fastest stabilization to consistently low values.



4.3 Comparative Analysis

The three decay rates revealed significant differences in learning efficiency and performance:

Learning Speed: The fast decay rate (0.01) learned most quickly, achieving good performance by episode 250 and maintaining it thereafter. The medium decay showed good but slower learning, while the slow decay struggled to develop a consistent policy even after 1000 episodes.

Stability: The fast decay showed the most stable rewards after initial learning, with fewer extreme drops. The medium decay exhibited good stability after episode 300 but with occasional significant drops. The slow decay displayed high volatility throughout training.

Loss Patterns: The fast decay demonstrated efficient stabilization of loss values, quickly dropping from the highest initial values to low consistent values. The medium decay followed a similar pattern with more fluctuations, while the slow decay showed erratic loss patterns with concerning increases in later episodes.

Final Performance: The fast and medium decay rates achieved comparable final rewards (259.04 and 260.18) with low loss values (8.81 and 5.64). The slow decay ended with negative

reward (-70.14) and much higher loss (65.44), indicating failure to converge to an effective policy.

4.4 Visual Agent Performance

The recorded videos revealed clear differences in agent behaviour:

The fast decay (0.01) agent demonstrated quick development of effective landing behaviour, consistently landing between the flags by episode 500. The landings were smooth, with precise control and optimal fuel usage.

The medium decay (0.005) agent showed more gradual improvement, with reasonable but less precise landings. By episode 1000, it demonstrated good control but did not consistently land perfectly between the flags.

The slow decay (0.003) agent exhibited the slowest skill development with continued erratic behaviour. Even by episode 1000, landings remained inconsistent and often occurred outside the flags, aligning with its volatile reward pattern.

5. Discussion

5.1 Exploration-Exploitation Trade-off

Our results clearly demonstrate how the exploration-exploitation balance critically affects learning in the Lunar Lander environment:

The slow decay (0.003) maintained high exploration ($\varepsilon = 0.050$ at episode 1000), resulting in continued instability and poor final performance. This extended exploration prevented the agent from consolidating effective landing strategies.

The medium decay rate (0.005) provided better balance, with sufficient early exploration to discover good policies while transitioning to exploitation ($\varepsilon = 0.010$ by episode 1000) at an appropriate pace.

The fast decay rate (0.01) transitioned quickly to exploitation ($\varepsilon = 0.010$ by episode 500), allowing the agent to refine effective strategies earlier. Importantly, this did not lead to premature convergence to suboptimal policies, suggesting that for environments with clear feedback signals like Lunar Lander, extended exploration may be unnecessary and even counterproductive.

5.2 Neural Network Learning Dynamics

The loss plots provide valuable insights into how the network learned under different exploration schedules:

All configurations showed initially high loss values that generally decreased with training, but with notable differences in stabilization patterns. The fast decay configuration's rapid loss decreases and consistent low values in later episodes indicate that focused exploitation enabled more efficient function approximation.

The slow decay's inconsistent loss pattern, with periodic spikes and an increasing trend in later episodes, suggests that continued high exploration introduced conflicting updates that prevented stable convergence.

5.3 Limitations and Future Improvements

While our implementation successfully learned the Lunar Lander task with faster decay rates, several enhancements could further improve performance:

1.Experience Replay:

Implementing a replay buffer would improve learning stability by breaking correlations between consecutive samples.

2. Advanced DQN Variants:

Double DQN could reduce overestimation bias, while duelling architecture could improve state-value and advantage function separation.

3. Adaptive Decay Schedules:

Automatically adjusting the exploration rate based on performance metrics could optimize the exploration- exploitation balance.

6. Conclusion

Our DQN implementation for the Lunar Lander environment successfully demonstrated the critical impact of exploration-exploitation balance on learning performance. The fast decay rate (0.01) provided the best overall results, achieving quick learning, stable rewards, low consistent loss values, and successful landings between flags.

This finding challenges the conventional assumption that slower exploration decay is generally preferable. For environments with clear feedback signals like Lunar Lander, transitioning more quickly to exploitation can accelerate learning without sacrificing final performance.

The dramatic performance differences between decay rates highlight the importance of hyperparameter tuning in deep reinforcement learning. Even with identical network architecture and learning algorithms, the exploration strategy significantly influenced whether the agent successfully learned the task.

Problem 2 – Exploration and Exploitation in Deep Reinforcement Learning

One of the main challenges to come up when it comes to deep reinforcement learning is to find the right balance between the exploration and exploitation. When choosing its actions, in order to get a high reward, the agent must favour actions that have been tried before, and that have also been found to be effective in the reward production. The agent has to however first attempt different actions that have not been selected before, to identify such actions. In order to receive a reward, the agent must then take advantage of its past experiences (exploit), but it must also explore to choose better courses of action going forward (Sutton & Barto, p. 2).

This is further complicated by delayed rewards, where the consequences of an action might only be seen many steps later. Thus, the agent must explore strategically – not only to gather information, but to learn how that information translates to long-term success. In deep RL specifically, this becomes even more challenging because neural networks are used to estimate value functions or policies, often in large or continuous spaces where full exploration isn't realistic. The difficulty arises though, as neither of those two approaches can be used solely on its own. Different actions must be tested first and gradually give preference to those that seem the best. However, since the task involves a degree of randomness, each of the actions ought to be tested multiple times to assess their effectiveness (Sutton & Barto, p. 3).

Reinforcement learning environments also often appear to be nonstationary, indicating that the optimal actions and reward systems may evolve over time. Even in the stationary environments however, the agent encounters a series of bandit-style decision-making tasks, and they all undergo gradual modifications as learning advances and the agent's policy evolves. The balance task is therefore further complicated as strategies must account not only for uncertainty but also for the evolving nature of the task (Sutton & Barto, p. 23).

There are methods, like optimistic initial values and sample-average estimates, that can briefly encourage exploration. However, they do not work well with nonstationary problems, given the loss of their impact over time and not being change-adaptive (Sutton & Barto, p. 27).

Such a problem is especially visible in techniques such as Monte Carlo control, where adequate exploration is necessary to learn accurate action values. Policies that don't ensure all actions are tried, such as purely deterministic ones, risk missing better alternatives entirely (Sutton & Barto, p. 79). One workaround is to assume that episodes begin from randomly selected state–action pairs, known as exploring starts. While this can help in simulated environments, it's rarely feasible when learning from real-world experience. In practice, on-policy methods maintain exploration throughout learning, while off-policy methods separate the behaviour policy (used for exploration) from the target policy being improved (Sutton & Barto, p. 94). On-policy methods, such as ε-soft Monte Carlo control, incorporate exploration directly into the learning process by requiring the policy to always assign a non-zero probability to all actions. This ensures continual sampling, even if some actions currently seem suboptimal. Off-policy methods, on the other hand, separate the behavior policy from the one being improved, allowing the agent to learn a deterministic policy from exploratory trajectories (Sutton & Barto, p. 85).

Reinforcement learning employs training information, and gives back feedback on the quality of the actions taken, which is the main feature distinguishing it from other learning approaches. That therefore pushes the agent to actively explore and seek out better actions (Sutton & Barto, p. 19). As Sutton & Barton emphasize in the context of multi-armed bandits, exploration can eventually result in a larger total

reward by identifying superior actions. The exploitation however focuses on maximizing value in the short term (Sutton & Barto, p. 20).

This balancing dilemma is not apparent in supervised or unsupervised learning settings and despite many years of mathematical research, this exploration–exploitation trade-off remains unsolved (Sutton & Barto, p. 2). This is due to the fact that RL agents must learn by trial and error rather than being instructed the correct course of action (Sutton & Barto, p. 3). Bellman (1956) and Gittins and Jones (1974) introduced the theoretical basis for striking that. Despite their influence however, these methods are based on idealized presumptions that are difficult to apply in complex reinforcement learning contexts (Sutton & Barto, p. 35). Apart from focusing on what the optimal balance might be in theory, researchers have also looked at how efficiently agents can actually learn through exploration. This is often described in terms of sample complexity – basically, how many interactions with the environment are needed before the agent learns a good enough policy. In more practical terms, it's not just about whether the agent explores, but whether the exploration helps it learn useful actions quickly enough, especially when dealing with large or complex environments (Sutton & Barto, p. 34). As most of the proposed mathematical solutions rely on assumptions, such as stationary environments and prior knowledge of action values, these often don't apply to more complex reinforcement learning problems, such as the environments that may be more dynamic and uncertain (Sutton & Barto, p. 20).

The use of strategies like ε-greedy or entropy-based exploration is therefore more common, despite the fact that they are not formally optimal (Sutton & Barto, p. 22). Even though ε-greedy is one of the more common strategies because it's easy to use, it doesn't always make the best use of exploration. It picks random actions without really considering whether they're likely to be useful or not, which can slow down learning, especially in situations with a lot of possible actions or where rewards are sparse. Some other methods, like upper-confidence bounds or Boltzmann exploration, try to deal with this by choosing actions based not just on value, but also on how uncertain those estimates are (Auer et al., p. 236). In deep RL, these strategies are often built into algorithms like DQN or PPO, where the policy or value function is approximated by a neural network, and exploration noise or entropy regularisation is used to help the agent avoid premature convergence (Mnih et al., p. 530; Schulman et al., p. 3).

To sum up, finding the right balance between exploration and exploitation isn't just a basic choice between trying new things or sticking with what works. It's a much more complicated problem that depends on the environment, the algorithm, and how the agent learns over time. Because of that, there's no one perfect solution, and different strategies come with different trade-offs – some are easier to apply, but not always the most efficient. That's why managing exploration properly remains one of the biggest challenges in deep reinforcement learning.

References:

Auer, Peter, Nicolo Cesa-Bianchi, and Paul Fischer. "Finite-Time Analysis of the Multiarmed Bandit Problem." *Machine Learning*, vol. 47, no. 2–3, 2002, pp. 235–256.

Bellman, Richard E. "A Problem in the Sequential Design of Experiments." *Sankhya: The Indian Journal of Statistics*, vol. 16, 1956, pp. 221–229.

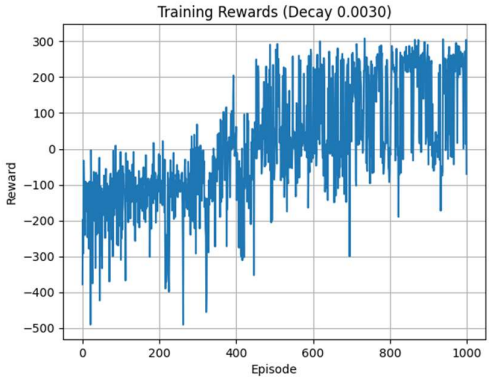
Gittins, John C., and D. M. Jones. "A Dynamic Allocation Index for the Sequential Design of Experiments." *Progress in Statistics*, edited by J. Gani, K. Sarkadi, and I. Vincze, North-Holland, 1974, pp. 241–266.

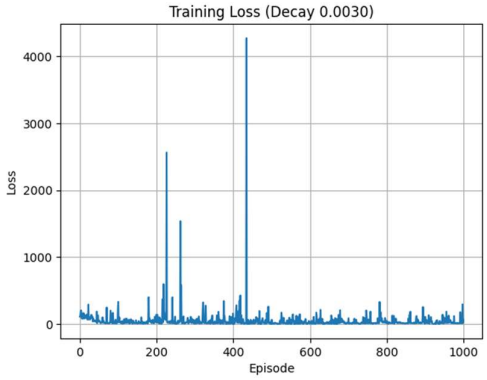
Mnih, Volodymyr, et al. "Human-Level Control through Deep Reinforcement Learning." *Nature*, vol. 518, 2015, pp. 529–533.

Sutton, H. "Peter Morgan Sutton." *BMJ*, vol. 348, no. mar31 11, 2014, p. g2466. *BMJ*, https://doi.org/10.1136/bmj.g2466.

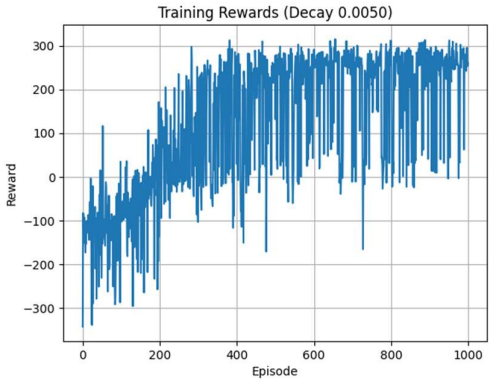
Code Output:

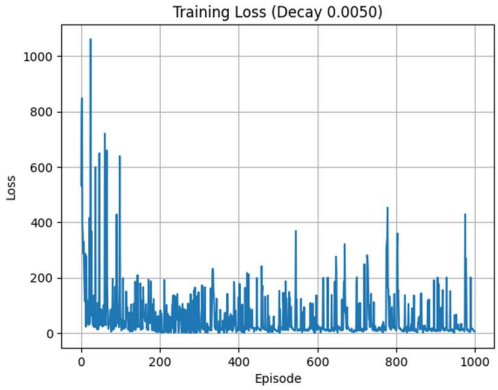
```
Starting training with decay rate: 0.9970
Decay 0.0030:   0%|          | 1/1000 [00:00<14:21,  1.16it/s]

Episode 1/1000, Reward: -378.24, Epsilon: 0.997, Loss: 108.5641
Decay 0.0030:  25%|██        | 250/1000 [01:18<27:58,  2.24s/it]

Episode 250/1000, Reward: -61.13, Epsilon: 0.472, Loss: 1.7174
Decay 0.0030:  50%|████      | 500/1000 [06:27<29:19,  3.52s/it]

Episode 500/1000, Reward: -86.05, Epsilon: 0.223, Loss: 1.6626
Decay 0.0030:  75%|██████    | 750/1000 [13:46<06:07,  1.47s/it]

Episode 750/1000, Reward: 168.44, Epsilon: 0.105, Loss: 12.2229
Decay 0.0030: 100%|████████| 1000/1000 [18:37<00:00,  1.12s/it]

Episode 1000/1000, Reward: -70.14, Epsilon: 0.050, Loss: 65.4380
```






```
Starting training with decay rate: 0.9950
Decay 0.0050:   0%|          | 2/1000 [00:00<02:48,  5.91it/s]

Episode 1/1000, Reward: -342.36, Epsilon: 0.995, Loss: 532.9626
Decay 0.0050:  25%|██        | 250/1000 [03:02<25:22,  2.03s/it]

Episode 250/1000, Reward: 235.08, Epsilon: 0.286, Loss: 21.1217
Decay 0.0050:  50%|████      | 500/1000 [08:38<10:38,  1.28s/it]

Episode 500/1000, Reward: 276.62, Epsilon: 0.082, Loss: 7.2383
Decay 0.0050:  75%|██████    | 750/1000 [11:55<05:02,  1.21s/it]

Episode 750/1000, Reward: 280.76, Epsilon: 0.023, Loss: 7.5576
Decay 0.0050: 100%|████████| 1000/1000 [15:03<00:00,  1.11it/s]

Episode 1000/1000, Reward: 260.18, Epsilon: 0.010, Loss: 5.6427
```






```
Starting training with decay rate: 0.9900
Decay 0.0100:   0%|          | 2/1000 [00:00<02:43,  6.10it/s]

Episode 1/1000, Reward: -138.63, Epsilon: 0.990, Loss: 1519.7164
Decay 0.0100:  25%|██        | 250/1000 [04:40<22:34,  1.81s/it]

Episode 250/1000, Reward: 240.41, Epsilon: 0.081, Loss: 17.2097
Decay 0.0100:  50%|████      | 500/1000 [08:18<07:20,  1.14it/s]

Episode 500/1000, Reward: 274.27, Epsilon: 0.010, Loss: 9.8179
Decay 0.0100:  75%|██████    | 750/1000 [11:11<04:00,  1.04it/s]

Episode 750/1000, Reward: 295.28, Epsilon: 0.010, Loss: 8.5564
Decay 0.0100: 100%|████████| 1000/1000 [14:27<00:00,  1.15it/s]

Episode 1000/1000, Reward: 259.04, Epsilon: 0.010, Loss: 8.8076
```