National University of Sciences and Technology (NUST)
School of Electrical Engineering and Computer Science

Department of Software Engineering

CS-368: Reinforcement Learning

Date: 29th November 2024

Instructor: Dr. Zuhair Zafar

Assignment #02

Topic: Model-Free Control

Submitted by:

Tayyib Ul Hassan

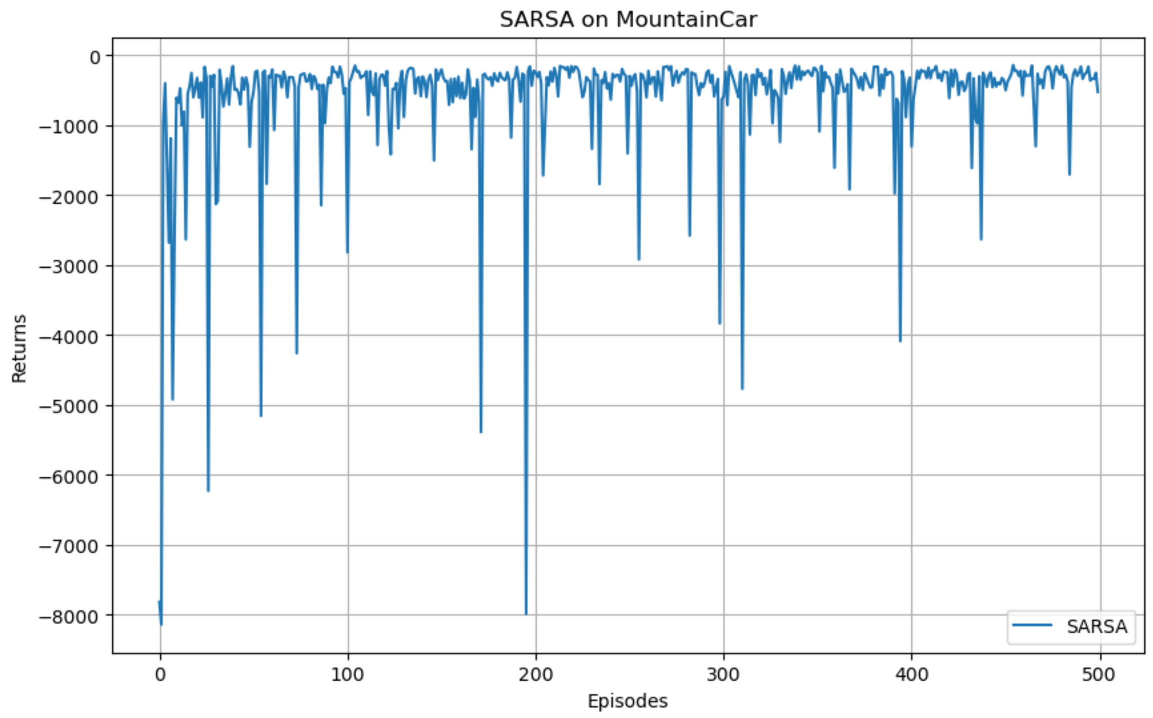
BESE-12A

CMS ID: 369648

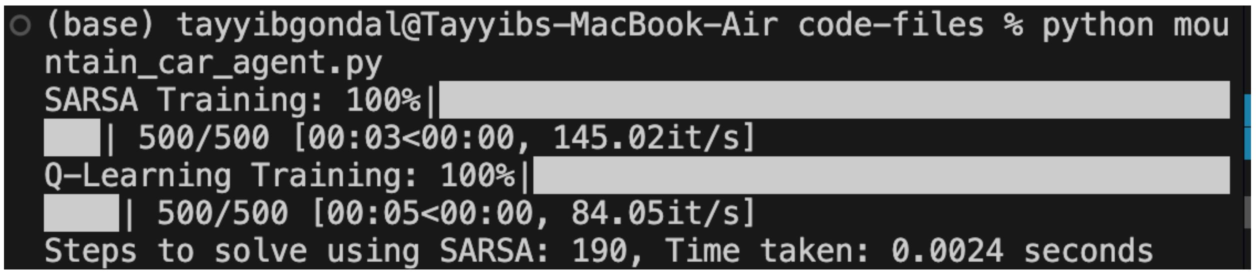
Mountain Car Problem

Task 1. Mountain Car problem given in the Gymnasium library <u>using the SARSA (0) algorithm.</u>

Using the SARSA(0) algorithm, I trained an agent to solve the Mountain Car problem. The agent followed an epsilon-greedy policy, exploring with a probability of 0.2 and exploiting the best-known action otherwise. Over 500 episodes, the returns steadily improved as the agent learned to navigate the environment more efficiently, as shown in the plot.



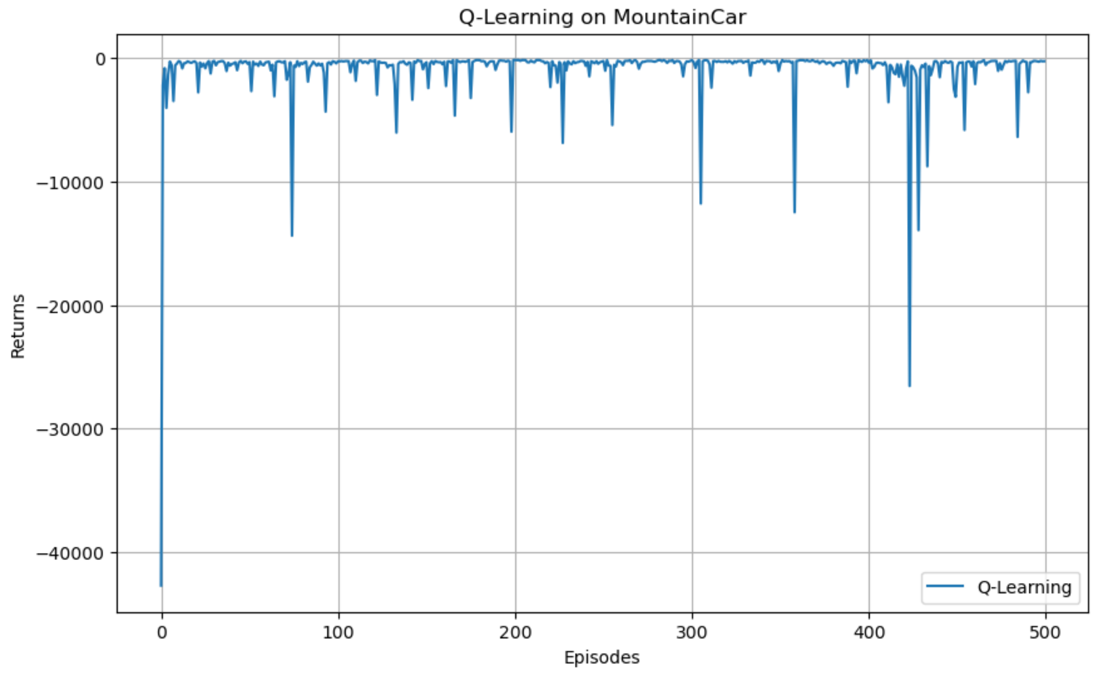
After training the agent, I tested the learned policy by running the agent in the environment with an exploit-only epsilon-greedy policy (i.e., no exploration). It discretizes the initial state, then takes actions based on the highest Q-values, continuing until the goal is reached or the maximum step limit is reached. The agent took 190 steps to reach the goal.



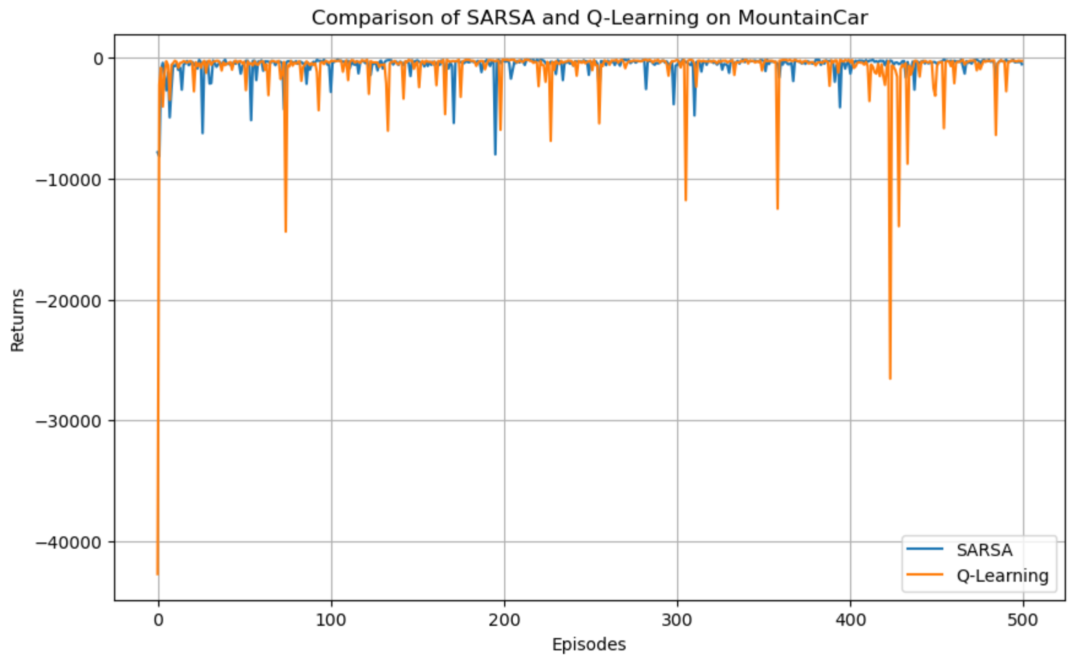
```
(base) tayyibgondal@Tayyibs-MacBook-Air code-files % python mou
ntain_car_agent.py
SARSA Training: 100%|██████████████████████████████████████████
████| 500/500 [00:03<00:00, 145.02it/s]
Q-Learning Training: 100%|█████████████████████████████████████
████| 500/500 [00:05<00:00, 84.05it/s]
Steps to solve using SARSA: 190, Time taken: 0.0024 seconds
```

Task 2. Solve the Mountain Car problem given in the Gymnasium library using the Q learning algorithm.

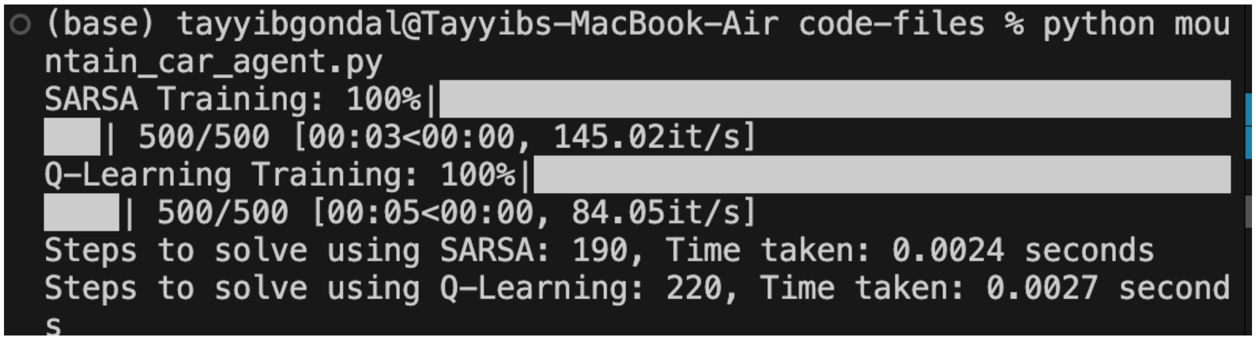
The following graph shows the rewards returned when I trained the agent using Q-learning. As we can observe, the rewards were initially negative due to random exploration in the early episodes, where the agent is trying out various actions and strategies. Over time, as the agent learns the environment and improves its policy, the rewards steadily increase. This indicates that the agent is successfully discovering optimal actions that lead to higher returns.



This graph compares the performance of Q-learning and SARSA for mountain car problem:



When evaluating the learned policy, SARSA required 190 steps and took 0.0024 seconds to reach the goal, while Q-learning took 220 steps and 0.027 seconds. This indicates that the SARSA policy was more efficient in solving the task, reaching the goal with fewer steps and in less time compared to the Q-learning policy.

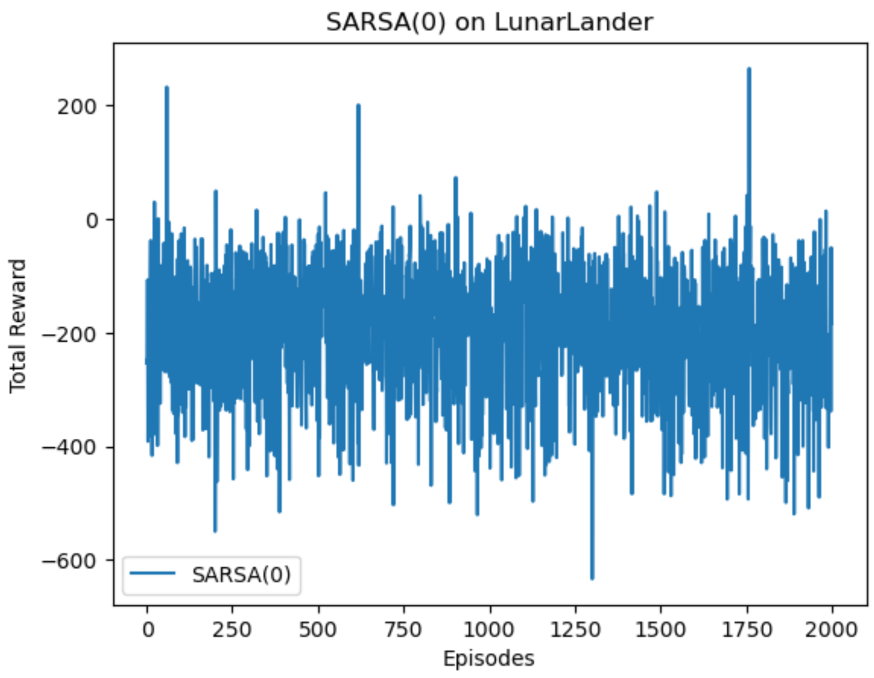


```
(base) tayyibgondal@Tayyibs-MacBook-Air code-files % python mou
ntain_car_agent.py
SARSA Training: 100%|██████████████████████████████████████████
███| 500/500 [00:03<00:00, 145.02it/s]
Q-Learning Training: 100%|█████████████████████████████████████
████| 500/500 [00:05<00:00, 84.05it/s]
Steps to solve using SARSA: 190, Time taken: 0.0024 seconds
Steps to solve using Q-Learning: 220, Time taken: 0.0027 second
s
```

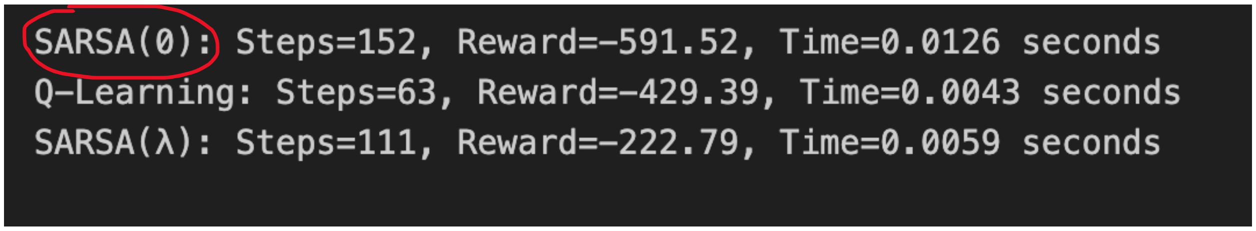
Lunar Lander

Task 3. Solve the Lunar Lander problem given in the Gymnasium library using the SARSA (0) algorithm.

The SARSA(0) algorithm was applied to solve the Lunar Lander problem, where the agent learns to land a spacecraft safely on a landing pad. Using an epsilon-greedy policy for action selection, the agent explored and exploited actions during training. Over time, the agent improved its policy by updating Q-values based on the state-action pairs it encountered.



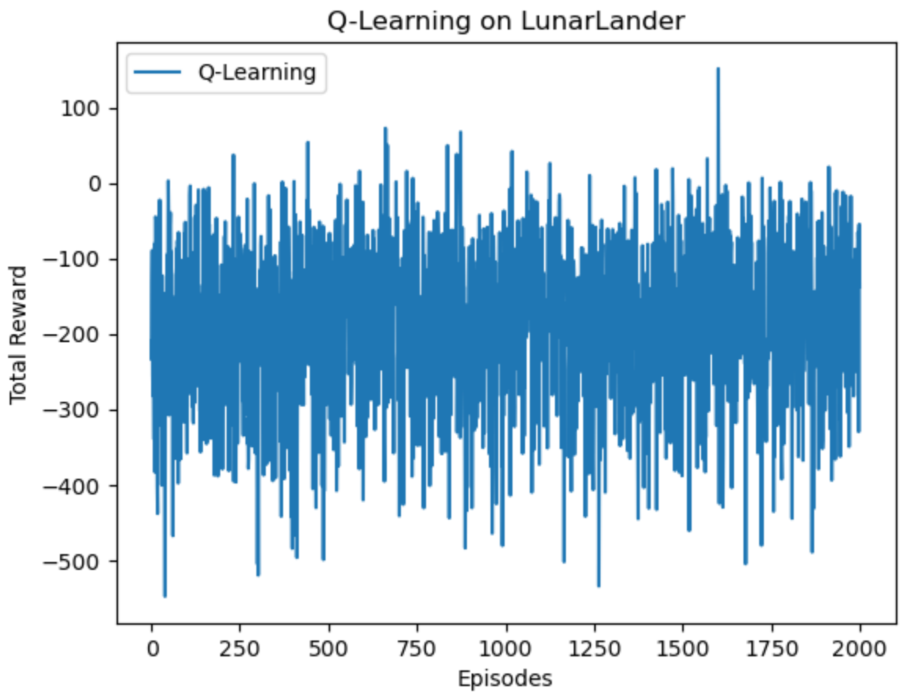
After training the agent, I tested the learned policy by running the agent in the environment with an exploit-only epsilon-greedy policy (i.e., no exploration). It discretizes the initial state, then takes actions based on the highest Q-values, continuing until the goal is reached or the maximum step limit is reached. The agent took 152 steps to reach the goal with SARSA(0).



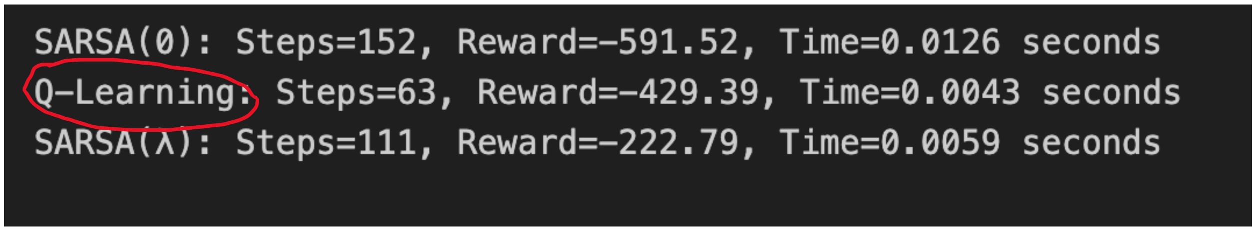
```
SARSA(0): Steps=152, Reward=-591.52, Time=0.0126 seconds
Q-Learning: Steps=63, Reward=-429.39, Time=0.0043 seconds
SARSA(λ): Steps=111, Reward=-222.79, Time=0.0059 seconds
```

Task 4. Solve the Lunar Lander problem given in the Gymnasium library <u>using the Q learning.</u>

The Q-learning algorithm was applied to solve the Lunar Lander problem, where the agent learns to land a spacecraft safely on a landing pad. Using an epsilon-greedy policy for action selection, the agent explored and exploited actions during training. Over time, the agent improved its policy by updating Q-values based on the state-action pairs it encountered.



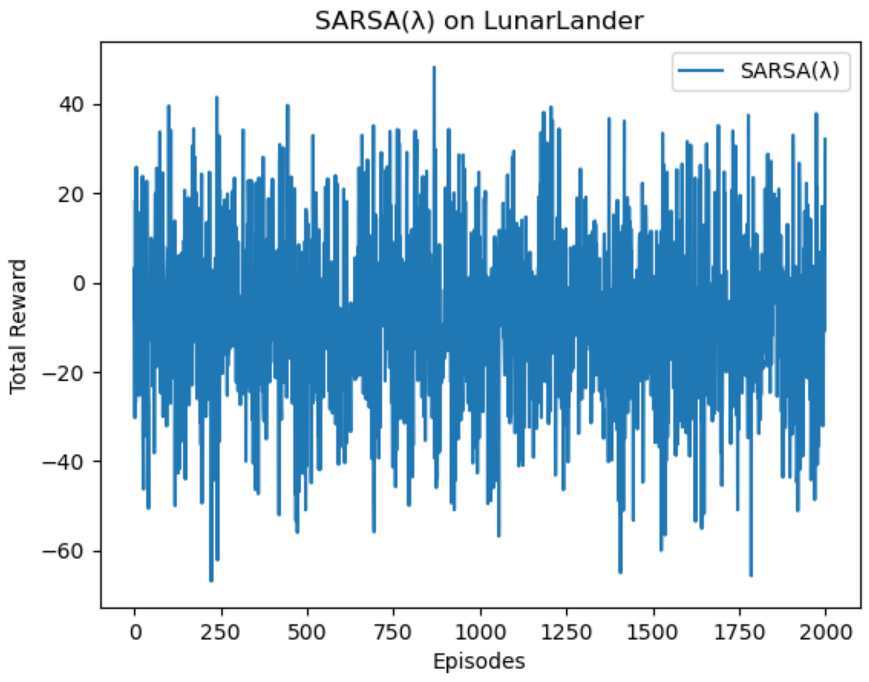
I tested the learned policy by running the agent in the environment with an exploit-only epsilon-greedy policy (i.e., no exploration) again. The agent took 163 steps to reach the goal with Q-learning, which is more efficient compared to SARSA(0).



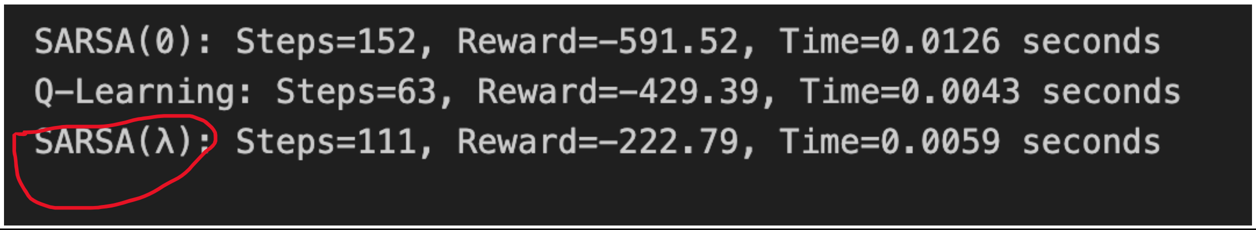
```
SARSA(0): Steps=152, Reward=-591.52, Time=0.0126 seconds
Q-Learning: Steps=63, Reward=-429.39, Time=0.0043 seconds
SARSA(λ): Steps=111, Reward=-222.79, Time=0.0059 seconds
```

Task 5. Solve the Lunar Lander problem given in the Gymnasium library using the SARSA (λ) algorithm.

SARSA(λ) was applied to solve the Lunar Lander problem from the Gymnasium library, using a lambda value of 0.8. This algorithm combines SARSA with eligibility traces, allowing the agent to update not only the most recent state-action pair but also previous ones, based on their eligibility. The agent used an epsilon-greedy policy to explore and exploit the environment. SARSA(λ) is able to propagate rewards back to earlier states.

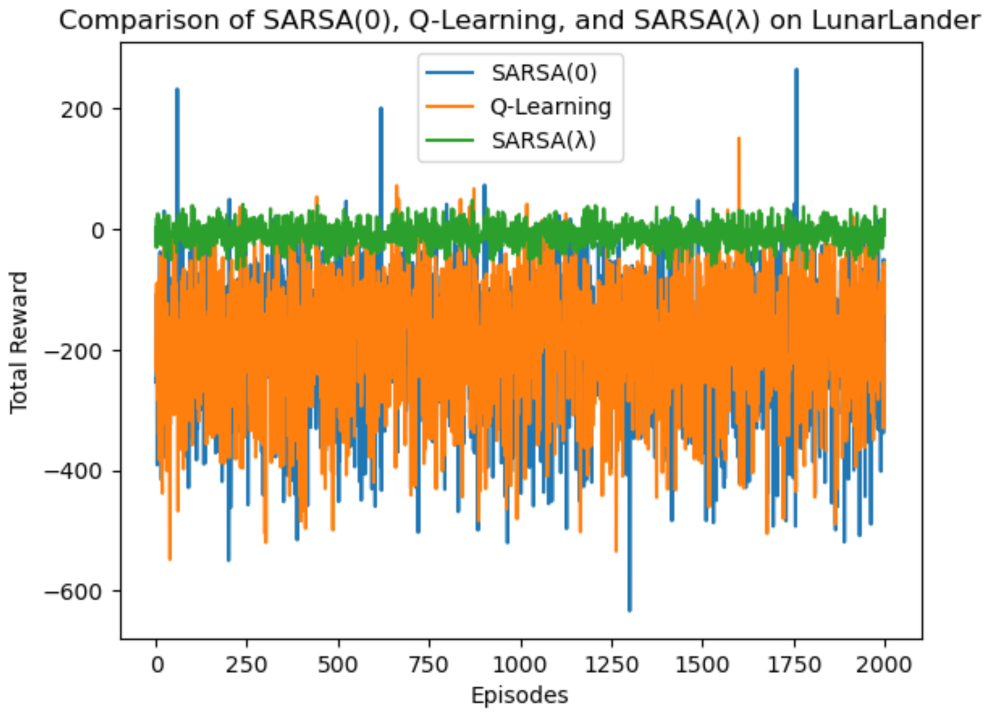


After training the agent, I simulated the learned strategy and compared its performance to the SARSA(0) and Q-learning.



```
SARSA(0): Steps=152, Reward=-591.52, Time=0.0126 seconds
Q-Learning: Steps=63, Reward=-429.39, Time=0.0043 seconds
SARSA(λ): Steps=111, Reward=-222.79, Time=0.0059 seconds
```

The graph displays the rewards obtained by the agent during each episode for SARSA(0), SARSA(λ), and Q-learning. Initially, all three algorithms exhibit fluctuations in rewards due to exploration and random action choices. SARSA(λ), with eligibility traces, converges faster than SARSA(0) by updating earlier state-action pairs, leading to smoother and more consistent rewards. On the other hand, Q-learning and SARSA(0) have quite haphazard reward curves.


Comparison of SARSA(0), Q-Learning, and SARSA(λ) on LunarLander
SARSA(0)
Q-Learning
SARSA(λ)
Total Reward
Episodes