Report Of Lunar Landing Project

a. Overview of DQN

Deep Q-Learning(DQN) is a deep learning algorithm that is a combination of reinforcement learning and q-Network that bring better performance on q-learning with Target Network and replay buffer. Its main idea is using neural networks to approximate the Q-value function Q(s,a) using Temporal-difference(TD) learning. By using a replay buffer, the DQN is sufficient to learn from past transitions and using a separate target network to perform a stabilized learning process.

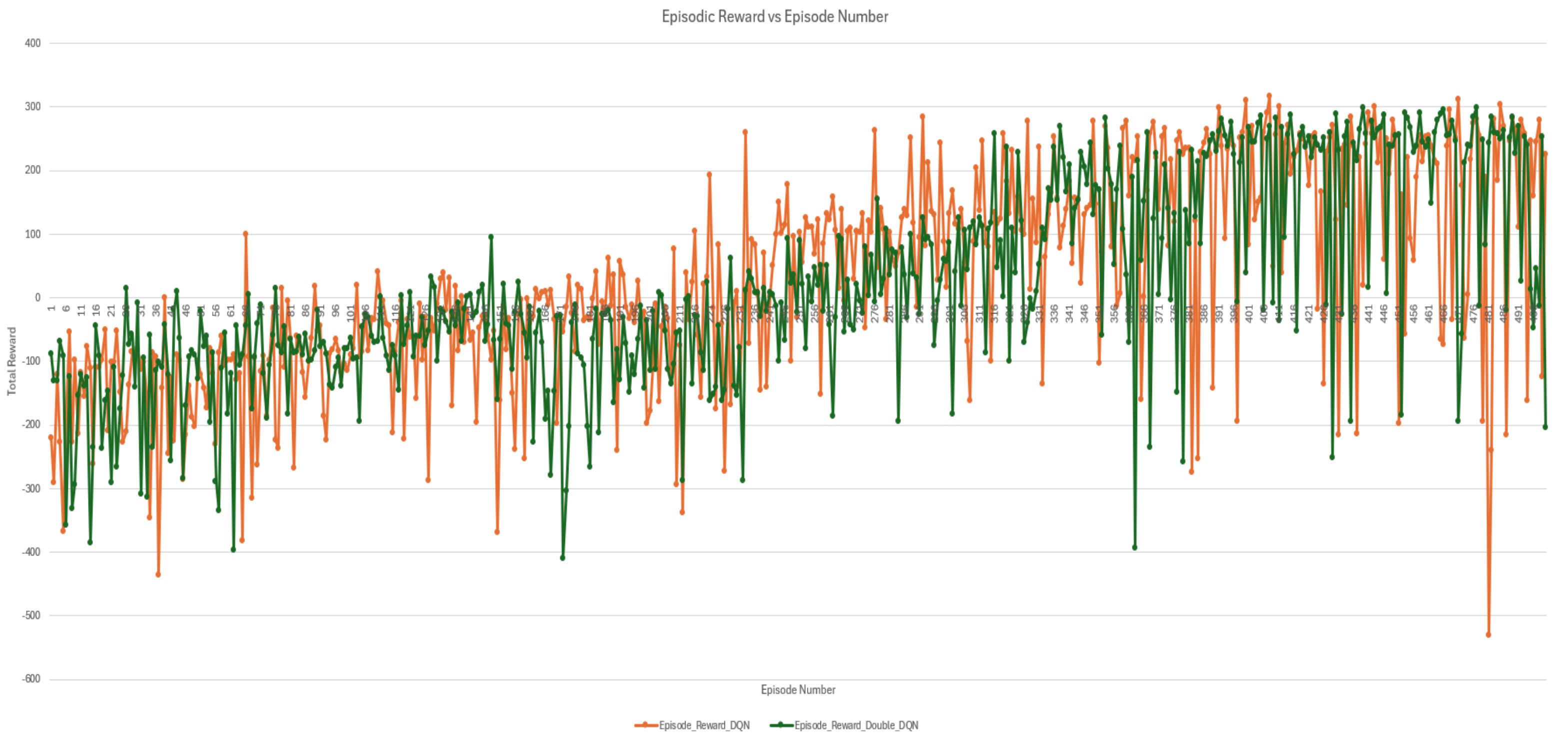
In the Lunar Landing Project, we implemented a DQN agent that landed a spacecraft on a simulated lunar surface with four actions: firing left engine, right engine, main engine and doing nothing. During each episode the agent will get the rewards or punishment based on its landing position and distance to the destination. After designing numbers of the episodes, the spacecraft successfully landed on the designated destination.

b. Description of chosen extension

Double DQN (Double Deep Q-Network) is an enhanced extension of the original DQN algorithm designed to address a key limitation: the tendency to overestimate action-value (Q-value) estimates. In standard Q-learning and DQN, the use of the max operator for both selecting and evaluating actions often leads to overly optimistic value estimates, which can result in suboptimal policies. Double DQN tackles this problem by decoupling the selection and evaluation steps. Specifically, it uses the online network to select the action and the target network to evaluate its value. This modest yet effective change significantly reduces overestimation bias and improves learning stability and policy quality. Empirical results—including those from this project—demonstrate that Double DQN often outperforms the original DQN, achieving more accurate value estimates and better overall performance, all without increasing computational complexity.

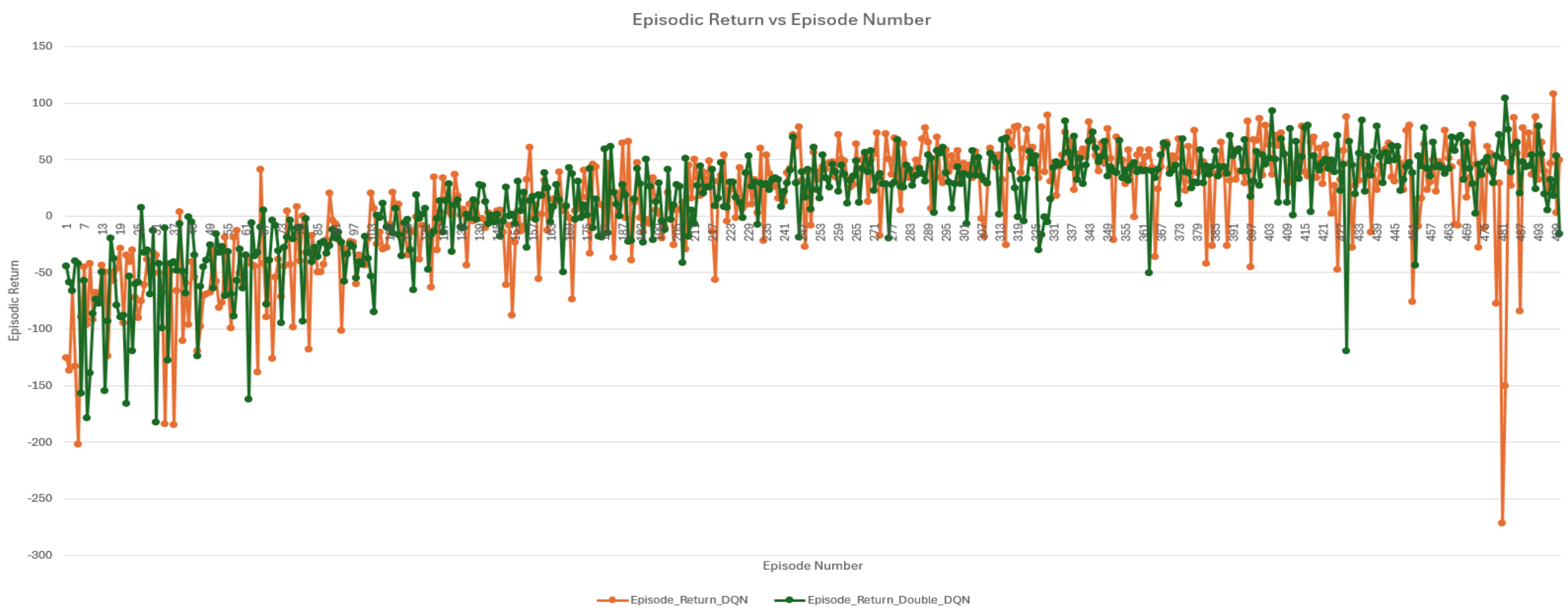
i. Include plots of metrics during training

Plot 1: Episodic Reward vs Episode Number



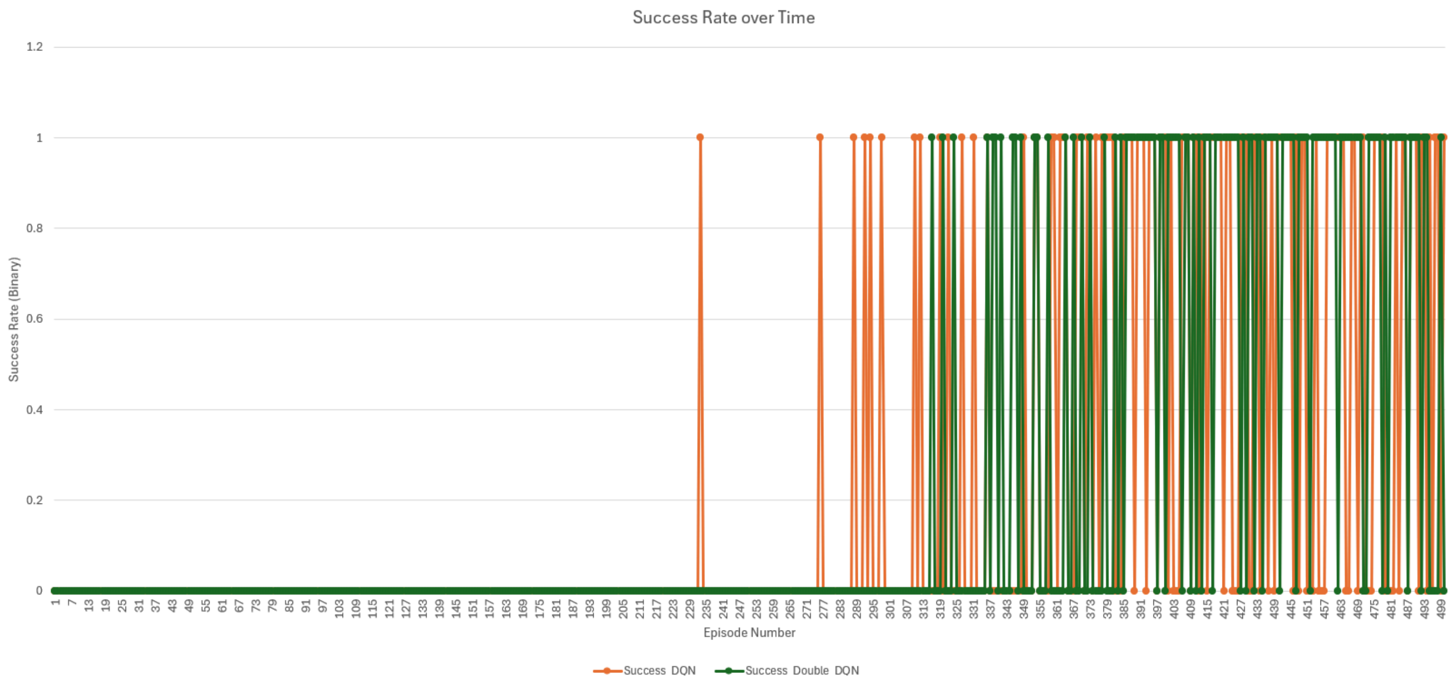
Insight: When further analyzing the data, we can distinguish between the orange line (vanilla DQN) and the green line (double DQN). At first glance across all 500 tests, double DQN's results for episodic rewards are more tightly compact than the results of the vanilla DQN. Both plots of data move similarly across the tests as we view the graph.

Plot 2: Episodic Return vs Episode Number



Insight: According to the data above, we can yet again analyze the two plots of data, the orange line representing the data for vanilla DQN and the green line representing the data for double DQN. Like the data for Episodic Reward, we can examine Episodic Return and see that double DQN is more tightly compacted together across the 500 episodes compared to vanilla DQN.

Plot 3: Success Rate over Time (binary per episode)



Insight: The success rate of DQN (the line in orange) and double DQN (the line in green) can be observed across the 500 tests. We can see that DQN begins succeeding in a plethora of episodes way before the first time double DQN begins to succeed. However, according to the data we can see that the success rate of double DQN is more frequent than that of vanilla DQN and is overall more tightly compact.

iii. Include a table of metric averages for last 100 episodes

Table of Metric Averages:

Metric	DQN (Vanilla)	Double DQN
Average Episodic Reward	161.14	186.59
Average Return	34.41	44.42
Success Rate (%)	60.78%	75.49%

Insight: This table of averaging the metrics is observed across the last 100 episodes of the training sessions for both vanilla DQN and the extension double DQN. According to the table, we can see that the average values of all three metrics for double DQN happen to be higher than the recorded averages of the metrics for vanilla DQN. Most importantly, we can see that the success rate percentage of double DQN is overall higher by about 15% then vanilla DQN.

d. Contributions of each team member

Ryan Wilkerson - modified the vanilla DQN program into the chosen extension double DQN. Recorded results for both vanilla DQN and double DQN and inputted the results into the three plots and the averages table. Filled in section C of the report showing the collected plots and the table as well as the insight into each one.

Hanqing Tao - contributed the project by taking parts in reports and the initial structures, also with meeting arrangements.

Jack Chen - Implementing the core DQN architecture, including the Q-network, replay buffer, and agent logic. also integrated the training logger for performance tracking and maintained the main training loop.

Boyd Bouck - Contributed mainly to the training loop in main.py, ensuring proper execution order of instructions and correct formatting.

Quang Phan - Set up the initial structure for the project by creating folder and early version of the files. Check and correct the vanilla DQN program and refine how the epsilon value decay during training by shifting it out of the update() function to apply once per episode for better training behavior

e. Link to your GitHub repository

https://github.com/CoffeGlory/TCSS-435-Castle.git