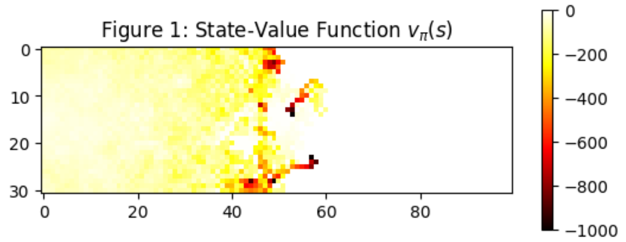
Part 2: Report:

In this project we have implemented Monte Carlo methods, SARSA algorithms and Q-learning to improve upon the agent's learning capabilities from just using dynamic programming, as done in the last assignment. For this assignment we used mountain lite.

First, we implemented MC-prediction to attain a $v_\pi(s)$ and $q_\pi(s,a)$ from the collection of trajectories given. Here we implemented it using a discount factor of 0.9 and with a first-visit approach. This is ensuring both that it converges to $v_\pi(s)$ and that the returns $G_\tau(s)$ are independent and identically distributed.



5. Yes, Monte Carlo (MC) policy improvement can be performed, our approached used an ε-greedy approach. Since the Monte Carlo methods are model free, we don't need to know the environment's dynamics, instead, these methods rely on learning directly from episodes of experience. In our mc improvement implementation it derives an improved policy by leveraging an estimated action-value function, Q. It does so by adopting an ε-greedy strategy. The question of whether the new policy we get is the best possible one is not easy to answer. The best or "optimal" policy depends on how well the Q values match up to reality. In theory, to get these Q values completely right, you'd need to try out every possible action many times in every possible situation, which isn't doable in real life. Because of this, and because the environment can be unpredictable, you might not ever get the Q values exactly right. So, we cannot say that we get the optimal policy, we get a suboptimal policy which can get better and closer to the optimal policy if we continue to repeatedly estimate Q values.

6. No, performing Monte Carlo (MC) control is not possible in this context due to several critical limitations. Firstly, the available data consists of fixed trajectories generated under SpaceY's proprietary policy (π), limiting it to only be possible to evaluate this existing policy without the capacity for active exploration or policy improvement. Secondly, the constraint that all episodes end after a maximum of 500 steps due to finite battery life imposes a predefined limit on episode length, potentially affecting the accuracy of return estimates for state-action pairs. Lastly, the inability to generate new, exploratory data under varying policies restricts the iterative process of MC control, which is crucial for accurately estimating the action-value function (Q) and deriving an improved policy. Given these constraints, effectively implementing MC control to determine an optimal policy is not achievable with the data and conditions provided by SpaceY.

7. Our starting policy for the robot was a random policy. Choosing a random policy as the starting strategy for navigating the mountain lite environment is motivated by the need for unbiased exploration. This approach ensures that the robot samples a wide variety of states within the environment without prior assumptions, crucial for gathering diverse data on terrain dynamics.

9. We implemented MC-control and plotted the trajectory it found starting in state (15,0) and as seen below in Figure 2 it does not seem to learn the optimal policy. the first challenge is that it keeps going in circles after the first state for a while before reaching a state that does lead it further out. However, an

even bigger issue is the way it ends. The trajectory ends by going from state s' to s'' which leads to state s' again. When the agent reaches a pair of states like these that just keep going back and forth between them, it will be stuck to do this for the rest of the agent's battery life. This happened already at step 23. This indicates we need a better approach to deal with this challenge.

11. We compared Monte Carlo (MC) and SARSA(0) algorithms using cumulative average rewards over 1000 episodes as a measure of reward maximization efficiency. The analysis revealed the MC-derived policy outperformed the SARSA-derived policy, with average rewards of -83.494 versus -253.141, indicating MC's better effectiveness in this scenario, eg. the specified random seed: 10.

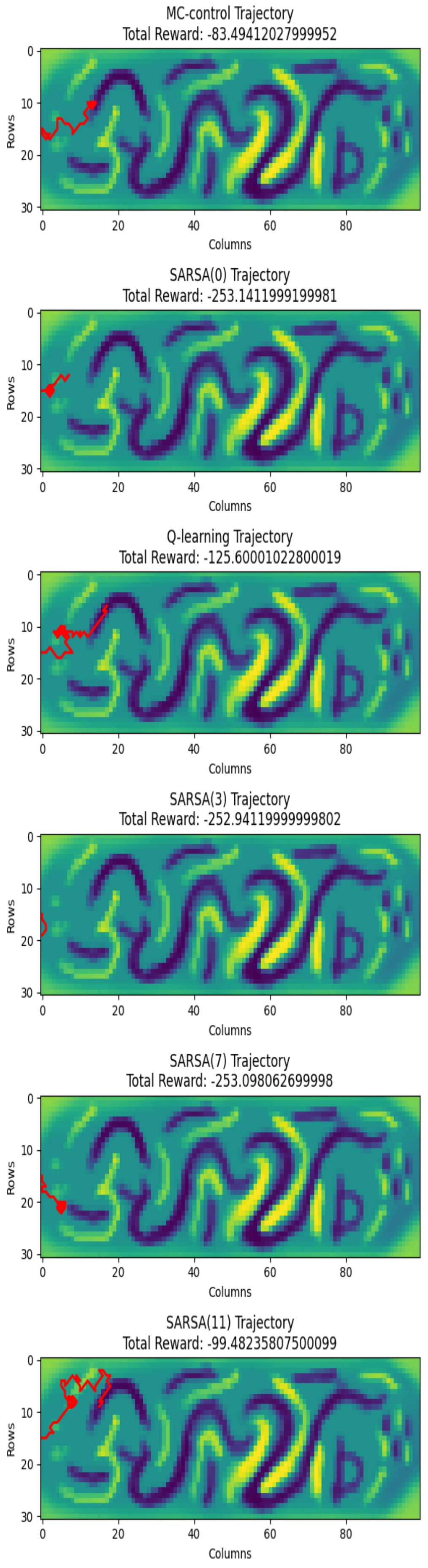
13. In our comparison of SARSA(0) and SARSA(n), we evaluated performance using cumulative average rewards. SARSA(n) had the different n values of 3, 7, and 11, revealing total rewards of -252.94, -253.09, and -99.48, respectively. SARSA(n=11) demonstrated best performance, suggesting it approximated the optimal policy more closely than the other variants.

SARSA(0) focuses on immediate outcomes, while SARSA(n) extends this by considering future consequences over n steps, leading to a more accurate estimates. SARSA(n)'s consideration of extended action sequences offers a significant advantage in environments where future outcomes heavily influence the optimal strategy, as highlighted by SARSA(n=11)'s results.

15. Q-learning resulted in a cumulative reward of -125.60. this is better than SARSA(0), SARSA(3) and SARSA(7), but worse than MC-control and SARSA(11). In general Q-learning should converge to the optimal policy more so than the others given a deterministic environment, which this is.

All in all, MC-control resulted in the best policy according to the rewards, even though this task is more so suited for SARSA and Q-learning. We believe this is due to the use of randomness in the epsilon greedy, as well as the exploration of random starting positions, that could have resulted in this instance yielding better results for the MC-control.

Figure 2:
Trajectories of different algorithms


MC-control Trajectory
Total Reward: -83.49412027999952
SARSA(0) Trajectory
Total Reward: -253.1411999199981
Q-learning Trajectory
Total Reward: -125.60001022800019
SARSA(3) Trajectory
Total Reward: -252.94119999999802
SARSA(7) Trajectory
Total Reward: -253.098062699998
SARSA(11) Trajectory
Total Reward: -99.48235807500099
Rows
Columns