Reinforcement Learning Assignment - Report

Saptarshi Chakrabarti - r0775439

June 1, 2024

1 Environments

1.1 Grid-World Environment

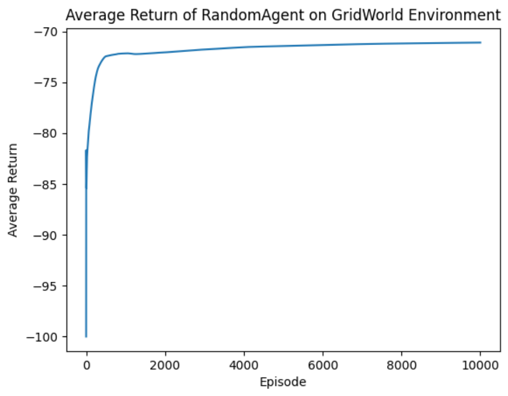

Figure 1: Evolution of Average Return for RandomAgent in GridWorld over 10000 steps.

A variable size (n × m) Grid-World environment was developed using the `gym.Env` class. The agent operates with four actions: up, down, right, and left, receiving a reward of -1 per action. The agent's starting position is (0, 0) and the goal position is (n-1, m-1). The environment provides a simple string representation through the `gym.Env.render` function.
For the RandomAgent, we implemented a class that selects actions randomly with equal probability. To facilitate the interaction, an `RLTaskLearning` class was created to return a list of average returns $\hat{G}_k$ for all episodes k. The `visualize_episode` method renders each step of an episode.
In a 5 × 5 Grid-World environment, the RandomAgent was evaluated over 10000 episodes. The evolution of average return over these episodes is illustrated in Plot 1. Initially, the agent's average return decreased rapidly due to its lack of a strategic approach, stabilising around -72 after 3500 episodes.

1.2 MiniHack Environments

We utilised four MiniHack environments:
EMPTY_ROOM, CLIFF, ROOM_WITH_LAVA, and ROOM_WITH_MONSTER.

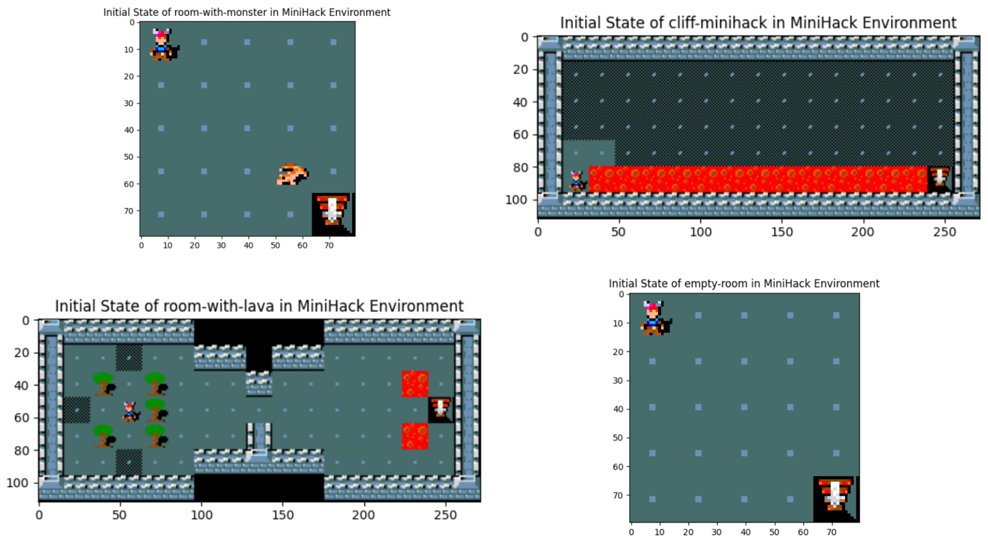
Initial State of room-with-monster in MiniHack Environment
Initial State of cliff-minihack in MiniHack Environment
Initial State of room-with-lava in MiniHack Environment
Initial State of empty-room in MiniHack Environment

Figure 2: The four MiniHack environments utilised in this project.

A FixedAgent was designed to execute a fixed movement in all of these environments: the agent would keep going down, and if it could not go down any further, it would go right. In the case of the EMPTY_ROOM and ROOM_WITH_MONSTER environment, it reached the goal position in 10 steps. In CLIFF, it did not move at all since both actions were impossible in this environment. In ROOM_WITH_LAVA, it moved for 4 steps and got stuck due to no more possible actions.

2 Learning Agents

This section focuses on the implementation of various learning agents and their evaluation in different environments. The goal is to develop and test Monte Carlo, SARSA, and Q-learning agents using ϵ-greedy policies for exploration, followed by some experiments to analyse their performance. Additionally, we explore the convergence of SARSA and Q-learning in a specific environment and implement a Dyna-Q agent to demonstrate the benefits of incorporating planning into the learning process.

2.1 Learning Agents

In this section, we aim to evaluate the performance of various reinforcement learning algorithms in different Minihack environments. Specifically, we investigate the effect of exploration and learning rate parameters on the Monte Carlo, SARSA, and Q-learning agents. The experiments are conducted in the four MiniHack environments specified in the previous section. Our objective is to understand how the choice of epsilon (ϵ) for exploration and alpha (α) for learning rate influences the average return of each agent over a maximum of 3000 episodes. An initial experimental setup included running the algorithms for 10000 steps, but as they converged early enough, no meaningful inference could be drawn from observing plots that were stretched out over the x-axis. Although, the episode number could have been kept higher to observe the convergence of the algorithms with greater clarity, the time of each run would be significantly high considering the code is sweeping through 72 combinations of parameters each time. The experimental setup is detailed below.

For each environment and each agent, we varied the following parameters:

- Epsilon (ϵ) values: 0.5, 0.9
- Alpha (α) values: 0.1, 0.5, 0.9

2.1.1 Results

Effect of ϵ and α on Monte Carlo

For the Monte Carlo agent, we observe the impact of the parameters on the performance in the following two environments: EMPTY_ROOM and CLIFF.

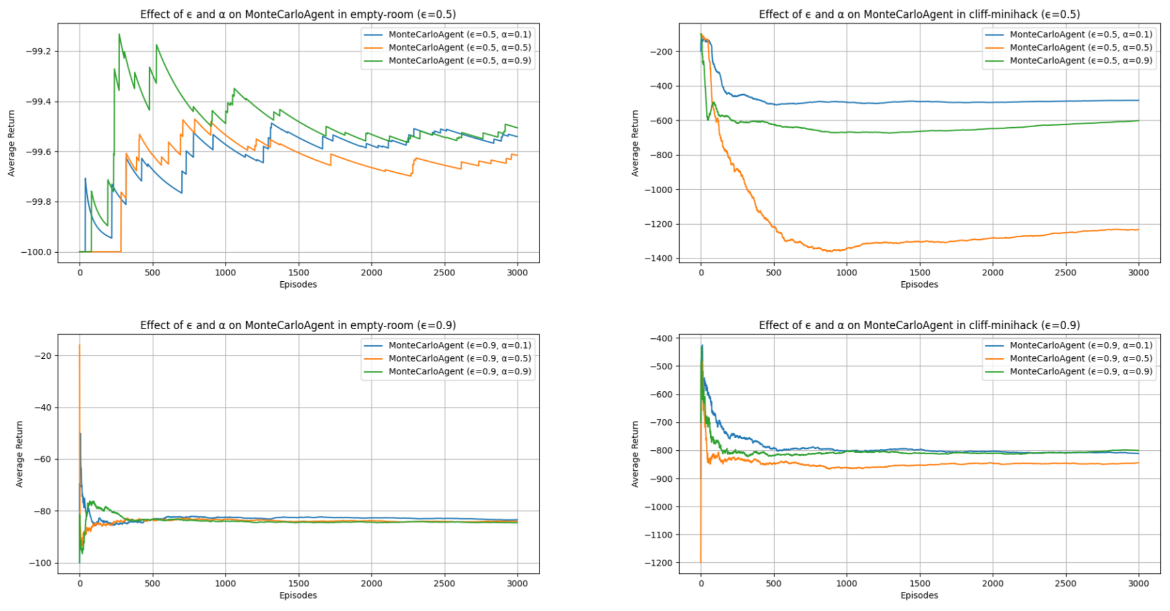

Figure 3: ϵ and α varied for Monte Carlo on EMPTY_ROOM and CLIFF.

In the EMPTY_ROOM environment, it is observed that the Monte Carlo agent performs optimally when configured with a high exploration rate (ϵ) and a low learning rate (α). Generally, higher values of ϵ result in improved performance across various learning rates. Notably, agents with a higher ϵ value exhibit faster convergence, which can be attributed to the extensive exploration fostering a more comprehensive understanding of the environment.

The high exploration rate facilitates extensive data collection, ensuring that the agent explores a wide range of state-action pairs. Concurrently, a low learning rate contributes to the stability of value function updates, preventing excessive fluctuations due to the frequent exploratory actions. This equilibrium between exploration and learning rate is critical for optimal performance, as it enables the agent to assimilate new information effectively without overreacting to the inherent variability introduced by high exploration.

In the CLIFF environment, it is observed that a lower epsilon (ϵ = 0.5) coupled with a lower learning rate (α = 0.10) outperforms the higher epsilon variant of the agent. This observation can be attributed to the specific dynamics of the CLIFF environment, which contains obstacles necessitating a balance between exploration and exploitation. While exploration is essential to navigate and understand the environment, a certain level of exploitation is crucial to avoid the pitfalls and maximise rewards. Furthermore, over the course of 3000 episodes, Monte Carlo agents with an epsilon of 0.5 and varying alpha values (0.1, 0.5, and 0.9) exhibit convergence to distinct values. Notably, the Monte Carlo agent with an epsilon of 0.9 and an alpha of 0.9 surpasses its lower epsilon counterpart. This indicates that, although a higher exploration rate generally yields better initial learning, the combination of a high learning rate and high exploration rate can lead to superior long-term performance in environments where optimal strategies are complex and require extensive exploration.

Effect of ϵ and α on SARSA

For the SARSA agent, we examine the impact of the exploration rate (ϵ) and learning rate (α) on performance in two distinct environments: EMPTY_ROOM and ROOM_WITH_LAVA. The latter environment presents additional complexity due to the presence of obstacles.

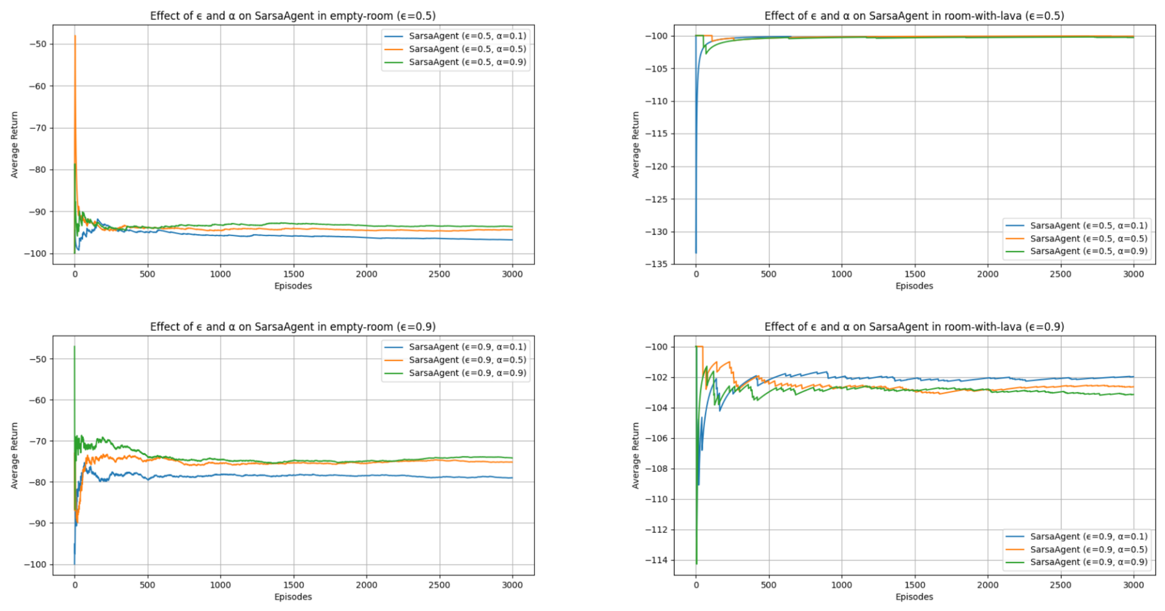

Figure 4: ϵ and α varied for SARSA on EMPTY_ROOM and ROOM_WITH_LAVA.

In the EMPTY_ROOM environment, the SARSA agent with a higher learning rate ($\alpha = 0.9$) consistently outperforms those with lower learning rates ($\alpha = 0.5, 0.1$) for both epsilon values ($\epsilon = 0.5$ and $\epsilon = 0.9$). Notably, agents with a higher epsilon value ($\epsilon = 0.9$) demonstrate superior performance compared to their lower epsilon counterparts. In all scenarios, the agents converge relatively early, around 500 episodes, although they converge to slightly different average return values.

Conversely, in the ROOM_WITH_LAVA environment, agents with a lower epsilon ($\epsilon = 0.5$) perform better, and after approximately 500 episodes, all agents with different learning rates but the same epsilon ($\epsilon = 0.5$) converge to nearly identical values. However, for agents with a higher epsilon ($\epsilon = 0.9$), those with a lower learning rate perform better than those with higher learning rates.

This observed behaviour can be contrasted with that of Monte Carlo-based agents. Unlike SARSA, which updates its action-value estimates using the current state-action pair and the next state-action pair (hence being on-policy), Monte Carlo methods update value estimates based on complete episode returns, making them more suited for environments where the reward structure and state transitions are less deterministic. The SARSA agent's on-policy nature makes it sensitive to the actual exploration-exploitation balance, particularly in environments with obstacles, such as ROOM_WITH_LAVA. In such environments, effective navigation requires not only exploring new actions but also consistently avoiding negative states like lava, which is facilitated by a more conservative exploration strategy (lower ϵ) and a slower, more stable learning rate (lower α).

Effect of ϵ and α on Q-learning

For the Q-learning agent, we analyse the impact of the exploration rate (ϵ) and learning rate (α) on performance in two distinct environments: EMPTY_ROOM and ROOM_WITH_MONSTER. The latter environment includes obstacles, adding complexity to the agent's learning task.

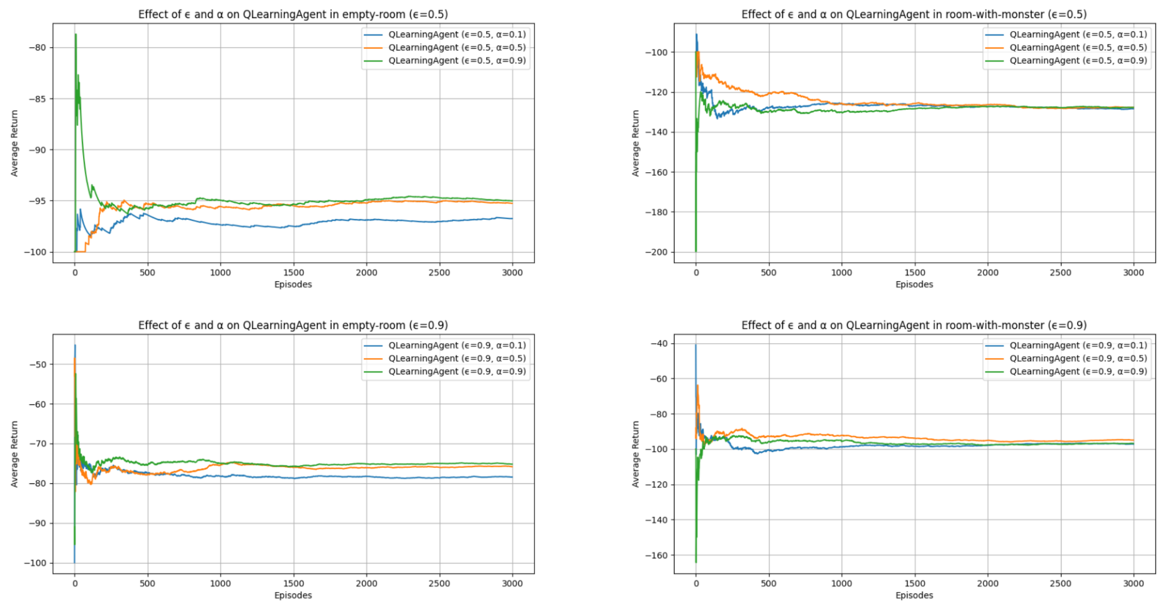

Figure 5: ϵ and α varied for Q-learning on EMPTY_ROOM and ROOM_WITH_MONSTER.

In the EMPTY_ROOM environment, Q-learning agents with a high learning rate ($\alpha = 0.9$) consistently outperform those with lower learning rates ($\alpha = 0.5, 0.1$) for both epsilon values ($\epsilon = 0.5$ and $\epsilon = 0.9$). Notably, agents with a higher epsilon value ($\epsilon = 0.9$) demonstrate superior performance compared to their lower epsilon counterparts. In all scenarios, the agents converge relatively early, around 1000 episodes, although they converge to slightly different average return values.

In the ROOM_WITH_MONSTER environment, agents with a higher epsilon ($\epsilon = 0.9$) perform better, and after approximately 2000 episodes, all agents with different learning rates but the same epsilon ($\epsilon = 0.5$) converge to nearly identical values. For agents with a higher epsilon ($\epsilon = 0.9$), those with an intermediate learning rate ($\alpha = 0.5$) perform better than those with either lower ($\alpha = 0.1$) or higher ($\alpha = 0.9$) learning rates, which converge to similar but lower values.

The behaviour of the Q-learning agent can be contrasted with both Monte Carlo and SARSA agents. Unlike the Monte Carlo method, which updates value estimates based on complete episode returns, Q-learning is an off-policy temporal difference method that updates value estimates using the maximum reward of the next state. This difference allows Q-learning to converge more quickly and effectively in environments with clear optimal paths, such as EMPTY_ROOM, where high exploration rates (ϵ) combined with high learning rates (α) expedite learning.

Compared to SARSA, Q-learning's off-policy nature enables it to potentially achieve higher rewards by always considering the optimal next action, rather than the actual next action taken. This can be advantageous in complex environments like ROOM_WITH_MONSTER, where optimal strategies need to account for both exploration to avoid obstacles and exploitation to maximise rewards. However, in highly dynamic or unpredictable environments, this can lead to instability if the learning rate is too high, as observed with Q-learning agents with $\alpha = 0.9$ in ROOM_WITH_MONSTER.

Q-learning agents benefit from a higher epsilon and moderate learning rate in environments with obstacles, balancing exploration and stable learning. This differs from the more conservative approach needed for SARSA agents and the episodic update mechanism of Monte Carlo agents, each adapting differently to the environment's challenges and the agent's exploration-exploitation strategy.

2.2 Convergence in CLIFF Environment

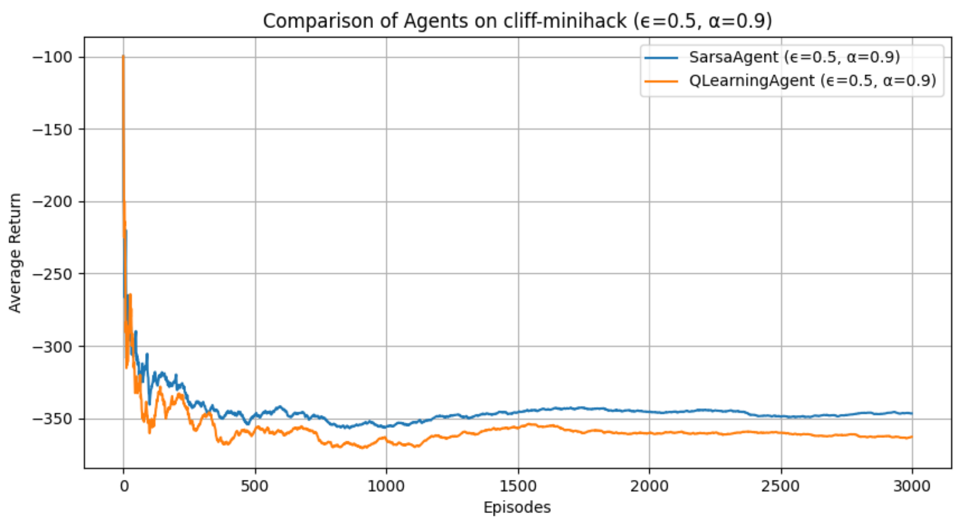

Figure 6: Performance of SARSA and Q-learning in CLIFF with a constant ϵ.

The convergence behaviour of SARSA and Q-learning was further examined in the CLIFF environment. Due to their different policy update mechanisms, SARSA converged to a safer policy, avoiding lava, while Q-learning occasionally opted for riskier paths to achieve higher rewards.

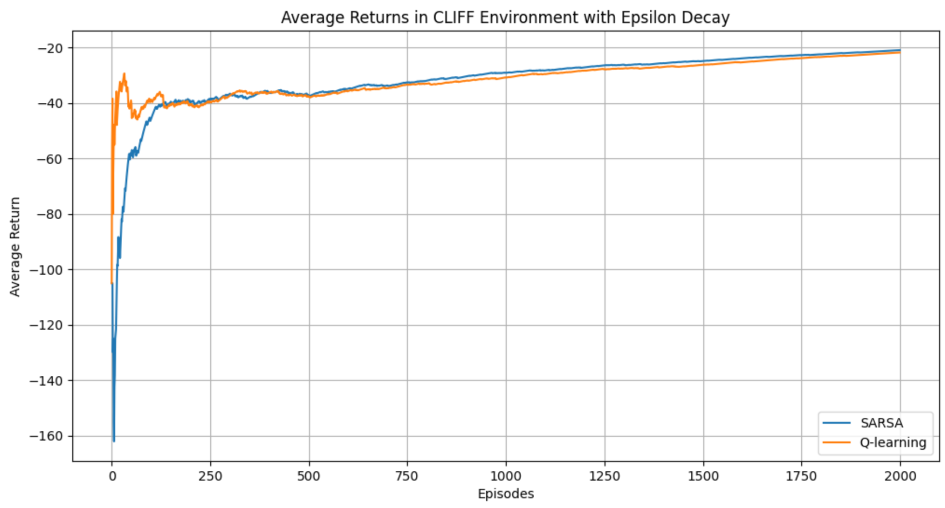

Figure 7: Performance of linearly decayed ϵ on SARSA and Q-learning in CLIFF.

In SARSA, the target policy remains the ϵ-greedy policy, promoting a balance between exploration and exploitation throughout the learning process. In contrast, Q-learning's target policy is always the greedy policy, driving the agent to pursue the highest estimated rewards, even if it involves taking greater risks initially.

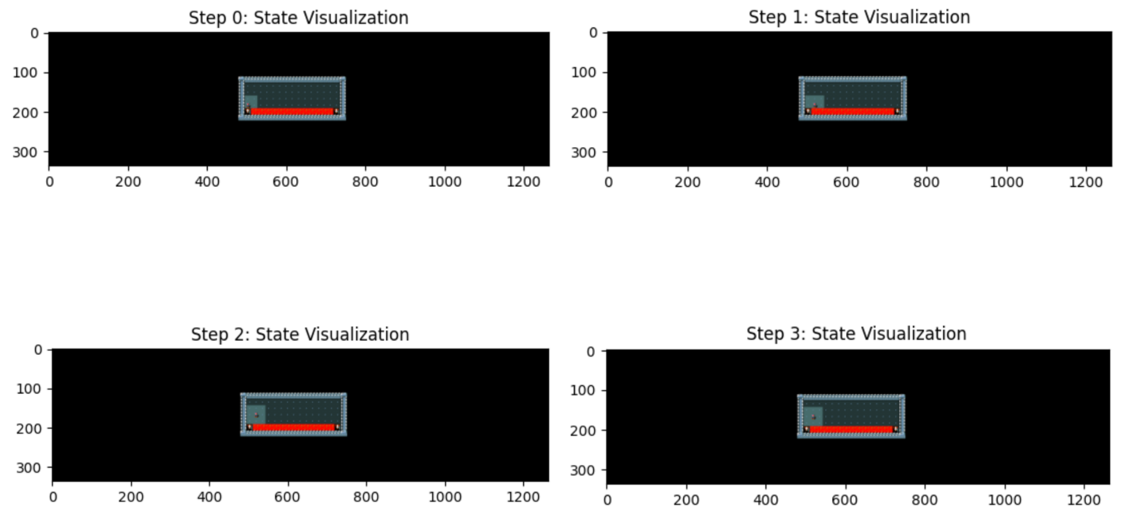
Step 0: State Visualization
Step 1: State Visualization
Step 2: State Visualization
Step 3: State Visualization

Figure 8: Trajectories of SARSA visualised.

To smooth the exploration process, a linear decay schedule for ϵ was implemented. Figure 8 and 9 illustrates sample trajectories at different learning stages for SARSA and Q-learning respectively. SARSA consistently avoided the lava from the beginning, while Q-learning initially explored more aggressive paths but eventually learned to steer clear of high-risk areas. Figure 7 compares the final policies of both agents, highlighting SARSA's conservative strategy against Q-learning's more exploitative approach.

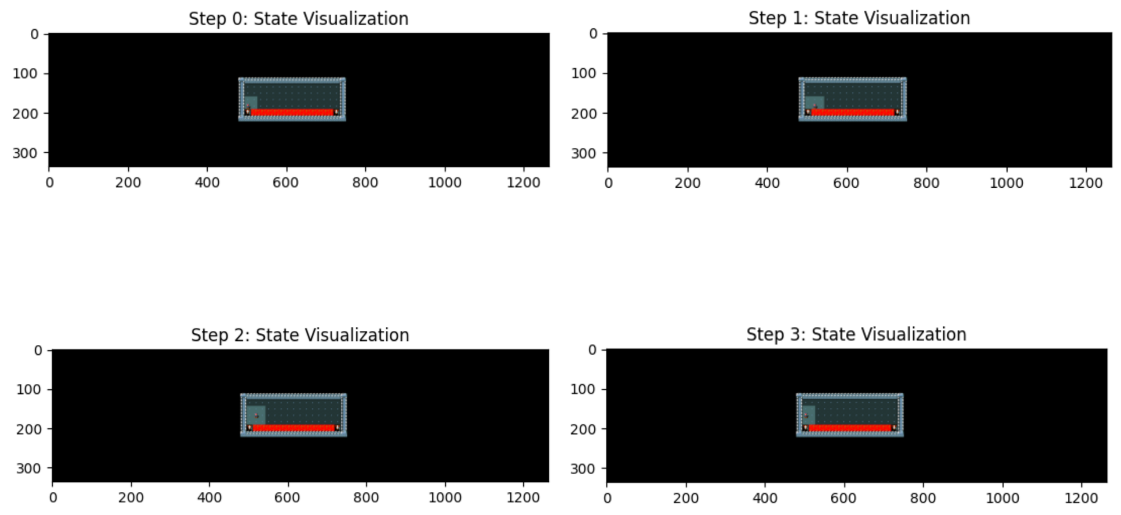
Step 0: State Visualization
Step 1: State Visualization
Step 2: State Visualization
Step 3: State Visualization

Figure 9: Trajectories of Q-learning visualised.

2.3 Dyna-Q Agent Implementation

A Dyna-Q agent, which incorporates background planning into the Q-learning algorithm, was implemented and evaluated. Both agents were subjected to the following experimental conditions:

- episodes: 2000
- $\epsilon = 0.5$
- $\alpha = 0.9$
- planning steps (for DynaQ) = 5
- environments: EMPTY_ROOM, ROOM_WITH_LAVA

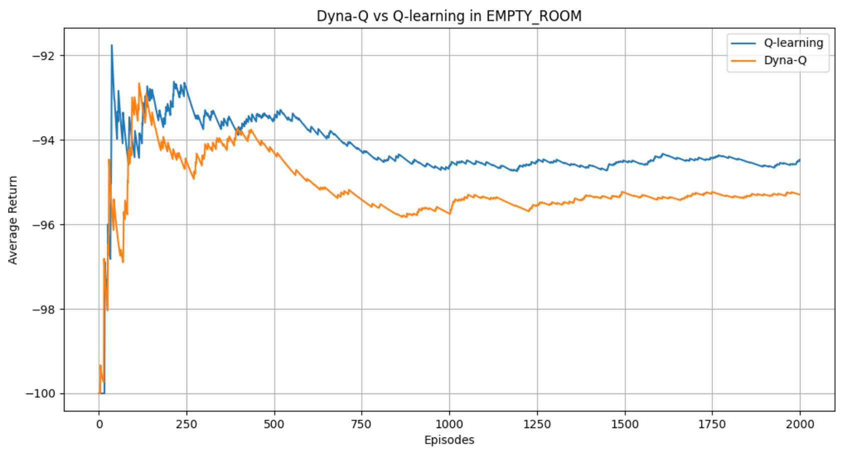

Figure 10: Comparison of Q-learning and Dyna-Q in EMPTY_ROOM.

In the EMPTY_ROOM environment, the Q-learning agent outperformed the Dyna-Q agent, although both agents did not converge to the same return. This deviation from the expected behaviour, where Dyna-Q should typically leverage its planning steps to achieve better or at least comparable performance, suggests that the benefit of additional simulated learning experiences was minimal in this straightforward environment. The simplicity of EMPTY_ROOM likely reduced the advantage gained from planning steps, which are more beneficial in complex environments with richer state transitions.

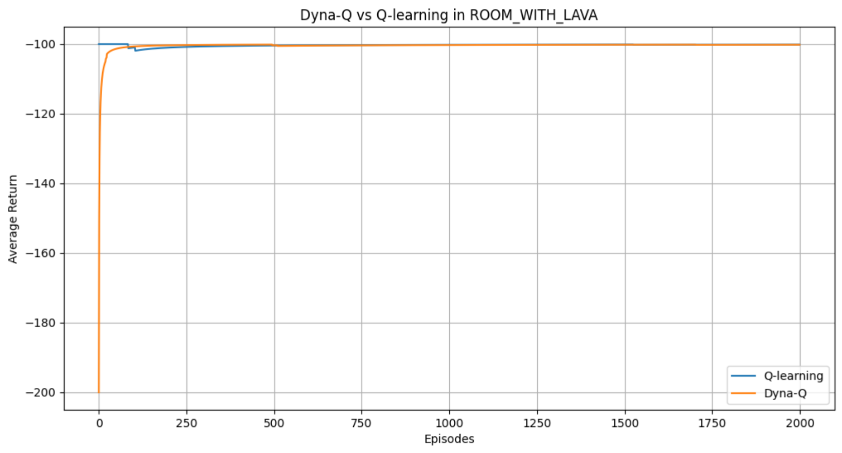

Figure 11: Comparison of Q-learning and Dyna-Q in ROOM_WITH_LAVA.

Conversely, in the ROOM_WITH_LAVA environment, which contains obstacles, both agents converged to the same values very early, around 250 episodes. This rapid convergence indicates that the additional planning steps in the Dyna-Q algorithm helped it quickly learn the optimal strategy to navigate the environment and avoid obstacles. The early convergence to similar values for both agents suggests that the immediate feedback from actual interactions was sufficient to learn effective policies, and the Dyna-Q agent's planning did not provide a significant additional benefit in this scenario.

Typically, Dyna-Q is expected to outperform or at least match the performance of Q-learning by utilising simulated experiences to reinforce learning. However, the observed behaviour in EMPTY_ROOM deviates from this expectation, possibly due to the environment's simplicity, where real interactions suffice for efficient learning. In contrast, the ROOM_WITH_LAVA environment demonstrated the expected advantage of early convergence, albeit without a significant performance gap, likely due to the efficacy of direct exploration and learning in environments with clear, immediate penalties for sub-optimal actions. Another possible issue could have also been the choice of the experimental parameters. A different set of parameters could have yielded better results.