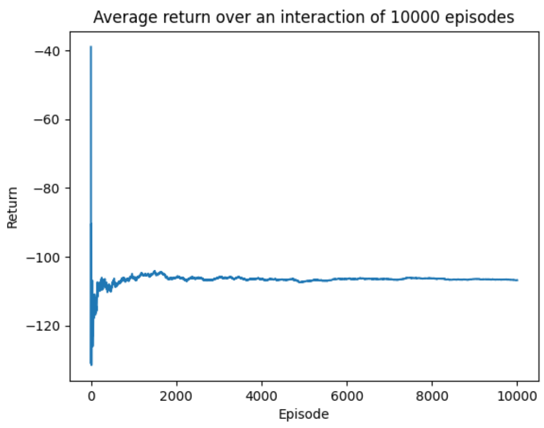
Bartosz Bogucki r1032166

1.1

The final task experiment is conducted using a 5x5 version of the GridWorld environment. The agent in the top-left corner of the grid, and the goal is positioned in the bottom-right corner. The agent incurs a penalty of -1 for every step taken, including invalid moves that result in no movement. A RandomAgent, which selects actions uniformly at random, is used. The average return over 10 000 episodes is as follows:



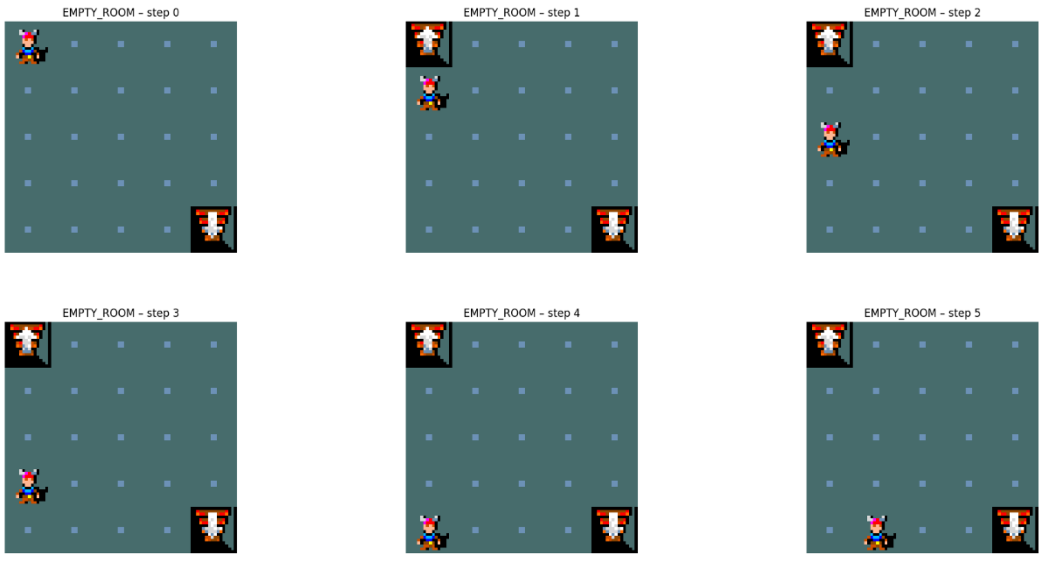
The average return begins with high variance and gradually converges to around -106.8 return after 10 000 episodes. The initial variance is due to the stochastic nature of random exploration and the small number of episodes collected during this phase. After around 2 000 episodes, the return starts to converge, not because the agent is improving - the RandomAgent does not learn or adapt, so that returns do not improve - but because more data has been collected, which smooths out fluctuations in the results. Visualising the agent's first ten steps in a sample episode illustrates its erratic movement across the grid:

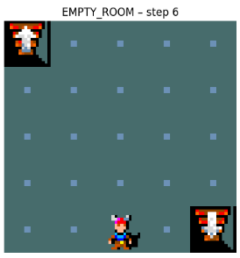
Start	**Step 1**	**Step 2**	**Step 3**	**Step 4**	**Step 5**	**Step 6**	**Step7**	**Step8**	**Step 9**	**Step 10**
A . . . . G	. A . . . . G	. A . . . . G	. . . . . . A . . . . G	. . . . . A . . . . G	. . . . . A . . . . G	. . . . A . . . . G	. . . . A . . . . G	. . . . A . . . G	. . . . A . . G	. . . . A . . . G

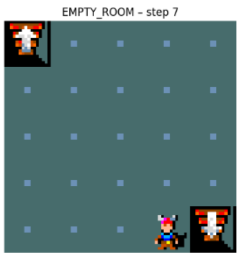
As a result, the performance remains consistently poor throughout the experiment.

1.2

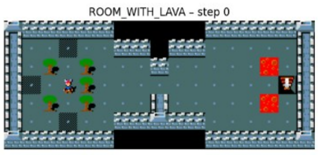
In this task, the FixedAgent moves downwards until it cannot move any further, and then moves to the right. This agent was tested in two environments: EMPTY_ROOM and ROOM_WITH_LAVA.

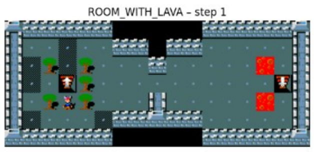


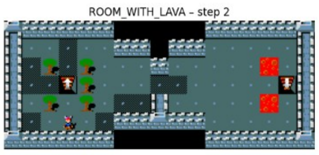


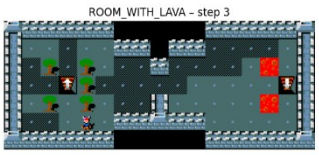


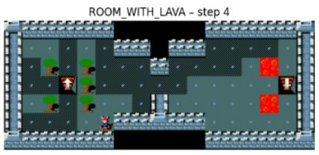
As expected, the FixedAgent successfully reached the goal in 8 steps in the EMPTY_ROOM environment - since there are no obstacles and moving downwards until it could not move any further, and then turning right, led directly to the goal.

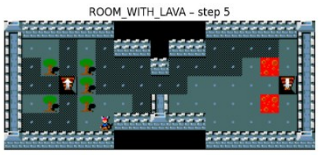


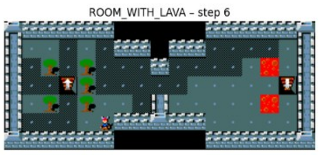


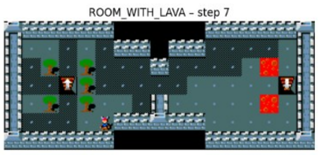


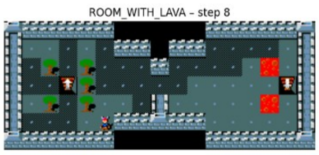


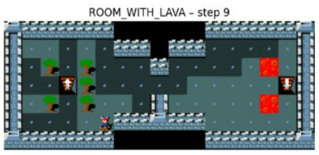


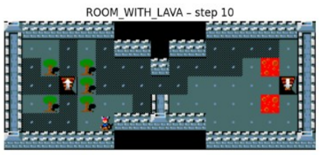








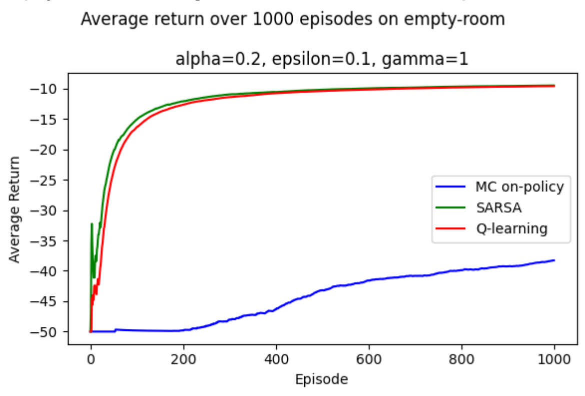


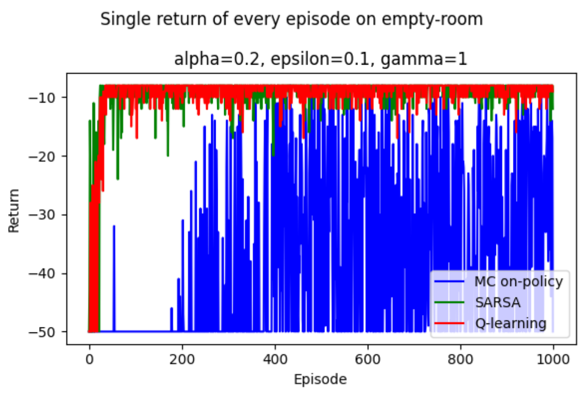


In the ROOM_WITH_LAVA environment, the FixedAgent moves downwards in accordance with its fixed policy until it is unable to do so, successfully moving downwards for two steps and then moving to the right. However, it then encounters a wall blocking the path to the right. From step 5 onwards, the agent becomes stuck against this obstacle, repeatedly attempting to move into the wall. Consequently, the agent's position remains unchanged from steps 5 to 10. This behaviour is expected, as the FixedAgent is a non-learning, deterministic agent that does not respond dynamically to obstacles.

2.1

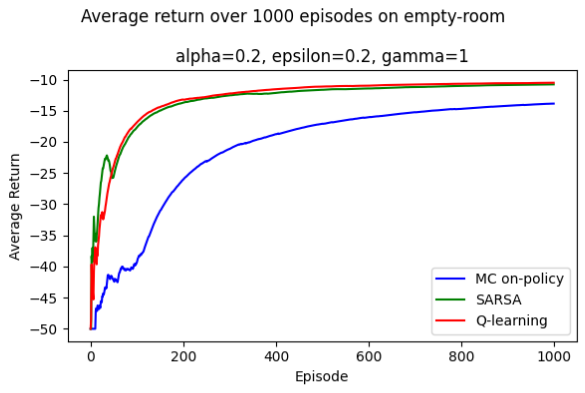
The experiments shown include average return curves, calculated as in Task 1.1, as well as individual returns for each episode. We evaluate Monte Carlo On-policy First-Visit, Temporal Difference On-policy (SARSA), and Temporal Difference Off-policy (Q-learning). All of them explore with an ε-greedy behaviour policy. Note that both first-visit and every-visit Monte Carlo methods should converge to the optimal policy in the limit, but their learning curves can differ due to how returns are sampled. MC is implemented with the sample mean, so there is no explicit step-size and α does not matter. For SARSA and Q-learning α is the usual TD learning rate. The first three experiments are conducted on an empty room, for which we always use a discount factor of $\gamma = 1$ because an empty room has rather short episodic horizons with no obstacles. The first experiment was conducted over 1 000 episodes on an empty room, using $\alpha = 0.2$, $\varepsilon = 0.1$, and $\gamma = 1$:

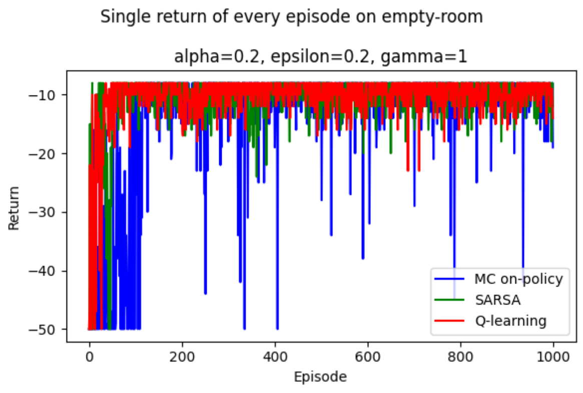




Both SARSA and Q-learning converge much faster than Monte Carlo On-policy. After 1 000 episodes, SARSA and Q-learning achieve average returns close to the optimal value of –8, albeit with some variance shown in a single return plot due to exploration, as ε is above 0. The MC agent occasionally discovers the optimal path but remains well below optimal on average after 1 000 episodes about -40, indicating that 1 000 episodes are insufficient for reliable convergence for MC in this setting, so that MC requires more episodes to converge effectively. The low exploration rate emphasizes exploitation once a promising policy emerges, which benefits the incremental updates of TD methods. Since MC waits until episode end to update, it requires more episodes to accumulate accurate return estimates. Also, although SARSA and Q-learning both follow the same ϵ-greedy behaviour policy, SARSA's on-policy update incorporates the exploratory action, whereas Q-learning replaces it with a greedy action in its TD target, thereby adding extra variance via the max operator. Consequently, Q-learning exhibits higher variance than SARSA with respect to the individual returns of each episode.

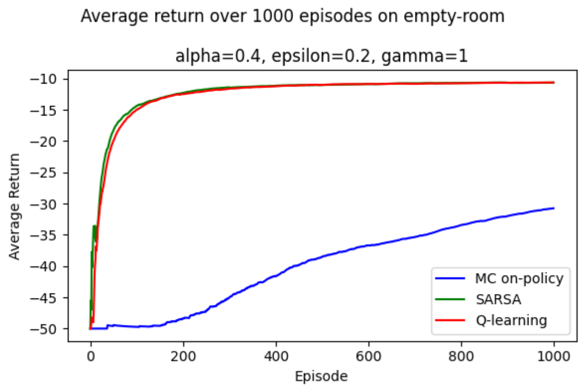
The second experiment has the same configuration as the first one, but using an increased ε of 0.2:

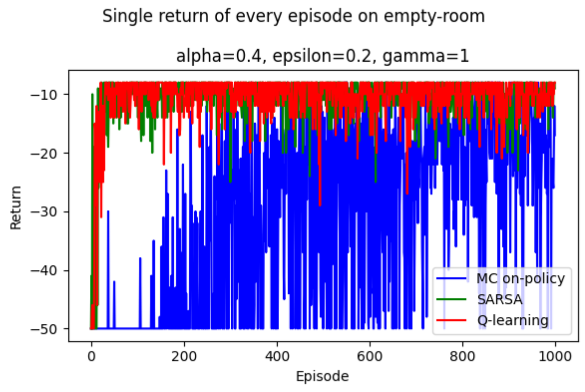




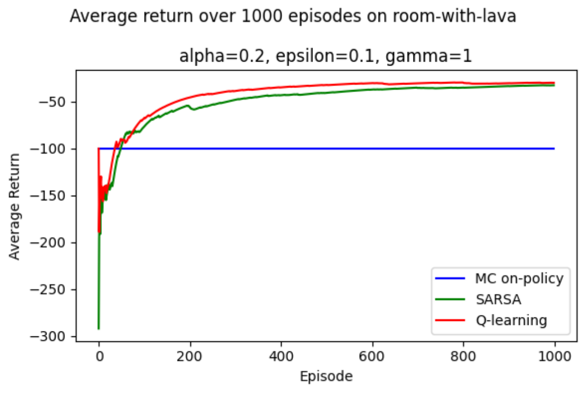
With a higher ε of 0.2, MC On-policy achieves a significantly higher final average return compared to the previous setting with $\varepsilon = 0.1$. This is because increased exploration allows it to discover the optimal path earlier and its final running average return rises to around –15, though it still has not fully converged. SARSA and Q-learning still achieve a final average return close to –8, but because they continue to take exploratory actions more frequently than in the previous experiment, the late episodes are noticeably more varied. This highlights the classic trade-off: while MC benefits from greater exploration due to its high variance, TD methods can afford to lower ε once they are near optimal. A constant ε encourages continued exploration, which benefits MC early on but prevents full exploitation later, leading to persistent variance. Also, higher variance can be observed for MC than for TD because MC uses noisy

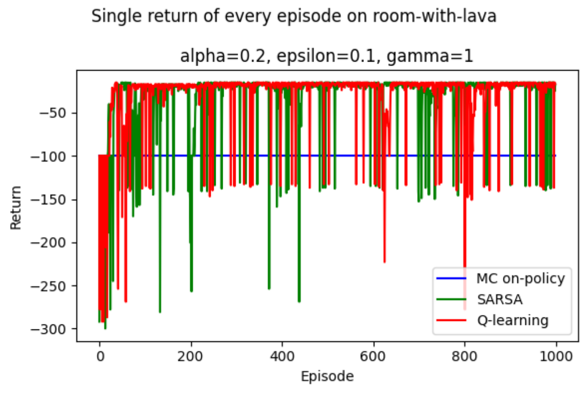
full-episode returns, which can fluctuate widely. In contrast, TD bootstraps one-step estimates, averaging out reward noise and propagating information incrementally.
The third experiment has the same configuration as the second one, but using an increased α of 0.4 for SARSA and Q-learning:

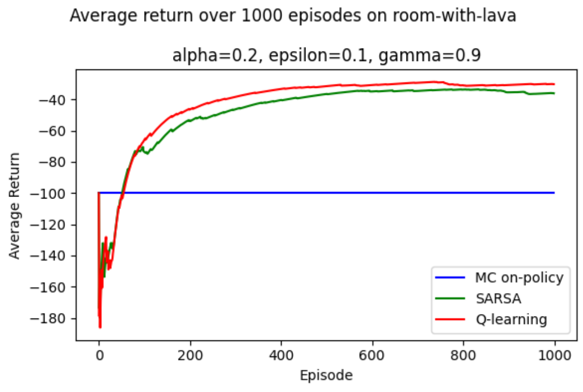


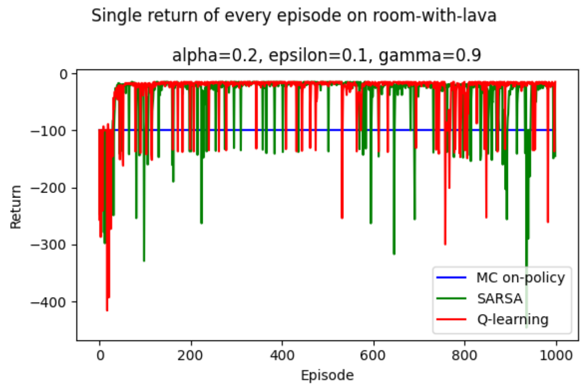


With a higher learning rate, SARSA and Q-learning exploit new information more aggressively. For both methods it results in a faster rise of average return and faster convergence. As long as ε is sufficiently high to ensure adequate exploration, the increased α helps TD methods exploit the optimal policy more efficiently. This leads to faster improvement in performance, though potentially at the cost of stability if α were pushed too high. MC's trajectory changes only through randomness in the learning process and ε-greedy action selection, since it has no α.
The next two experiments are conducted on the room with lava using all three models over 1000 episodes. The first run uses parameters α = 0.2, ε = 0.1, and γ = 1. The second run keeps α and ε the same, but reduces the discount factor to γ = 0.9:





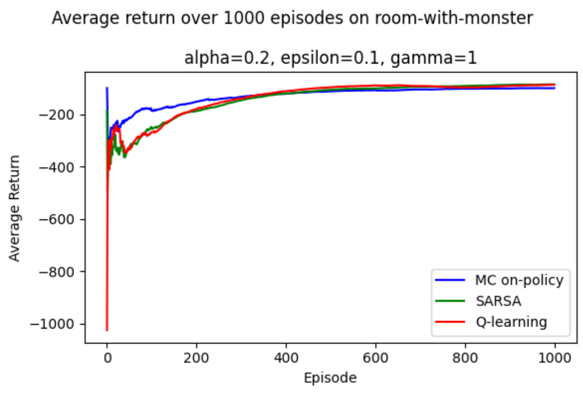


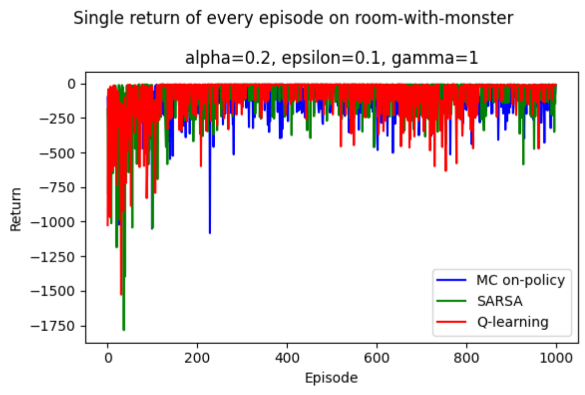


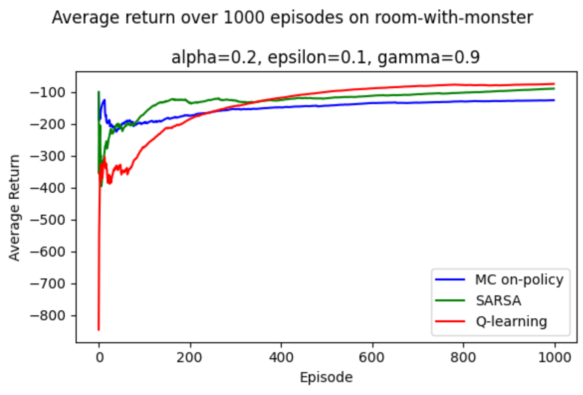
Monte Carlo On-policy never achieves return better than –100 in either episode. Because each episode can terminate via timeout or an automatic abort, MC on-policy primarily observes full-episode returns around –100. Since it updates only at the end of each episode, it never learns a policy that safely navigates to the goal - instead, it discovers that abortingincurs a smaller penalty than risking a lava fall. Since MC updates only from the complete return at episode end, it keeps reinforcing that –100 and never accumulates enough successful trajectories to discover the true goal-reaching policy. As a result, MC on-policy converges to repeatedly aborting, rather than attempting to step toward the goal.

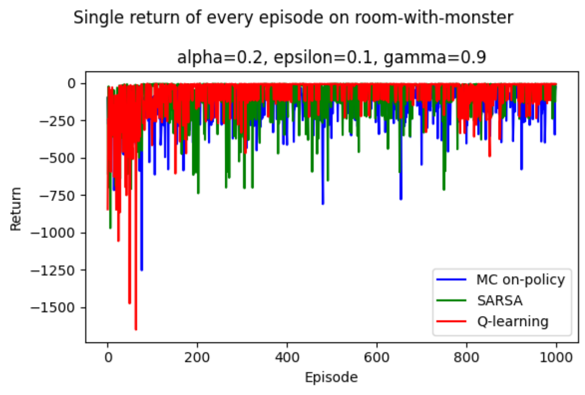
SARSA and Q-learning, by contrast, learn from every individual step. The moment −100 is given by stepping into the lava they push a large negative TD error back to the action that caused it, discouraging that move. Both SARSA and Q-learning discovered an optimal path and steadily pushed their average return toward the optimal value, dipping into lava when ε-greedy exploration forced a random step. When γ was lowered from 1 to 0.9 the two TD curves rose even faster, so that the convergence is faster. This is expected because a lower γ causes the agents to favor shorter-term rewards, effectively encouraging quicker paths to the goal rather than overly exploring.

A similar trend regarding a discounting factor is observed in the room with a monster. However, Monte-Carlo on-policy now is able to find optimal path on room with monster. With a lower γ, we again see faster learning and earlier convergence for all models, since the focus shifts toward more immediate returns. However, the presence of the monster introduces additional stochasticity - safe path can be ruined by a random collision, which spikes the single return variance. Nevertheless, SARSA and Q-learning show more stable convergence due to their ability to incorporate intermediate feedback, whereas MC remains more volatile:

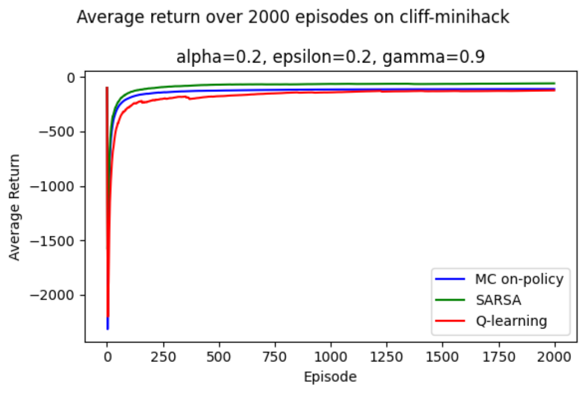


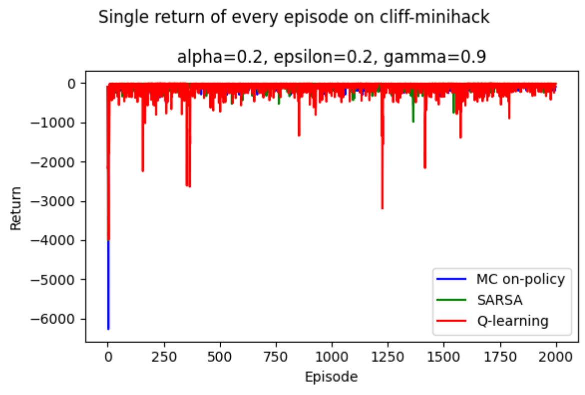






The final experiment is conducted on a cliff environment over 2 000 episodes for each learning agent, with the following parameters: α = 0.2, ε = 0.2, and γ = 0.9:

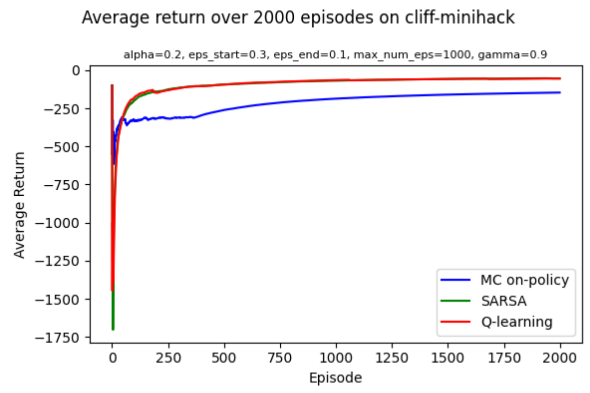


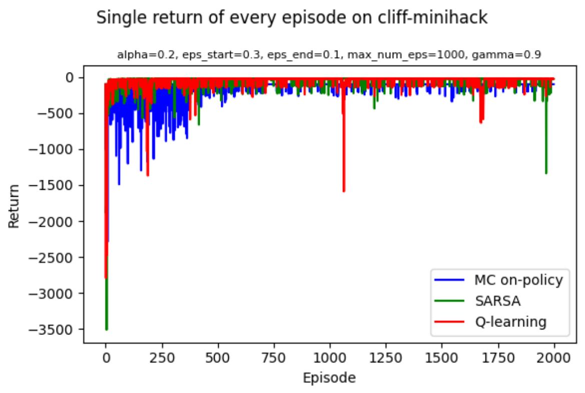


Simirarly to the results on the room with lava, in the cliff environment the Monte-Carlo on-policy learner never achieve a single return of above –100. On the other hand, SARSA and Q-learning is able to find a good path, but converge to different strategies. SARSA achieves a best return of -17, while Q-learning reached a maximum single return of -15 over 2 000 episodes. This difference can be attributed to the more aggressive nature of Q-learning. Q-learning's off-policy max target aggressively values the shortest route close to the lava, whereas SARSA's on-policy updates keep it farther away, trading extra steps for a much smaller chance of accidental death due to ε-greedy exploration when close to the lava. Thus, SARSA achieves a much lower variance, which leads to a faster rise in the average return and, consequently, faster convergence.

2.2

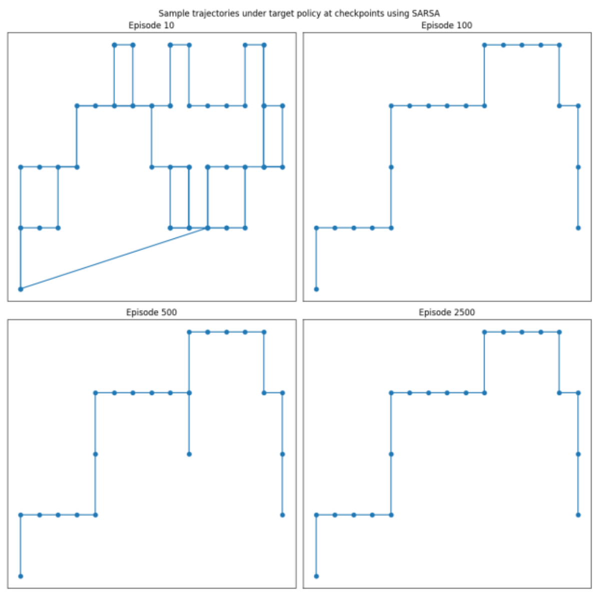
Using a linear decay for ε enables control the balance between exploration and exploitation over time. In our case, we used an initial value of ε_{start}=0.3 and a final value of ε_{end}=0.1, with the decay applied over the first 1000 episodes. This means ε decreases linearly from 0.3 to 0.1 during the first 1000 episodes, and then remains constant at 0.1 for the remaining episodes. The experiment was conducted on the Cliff environment over 2000 episodes, with α = 0.2 and γ = 0.9:

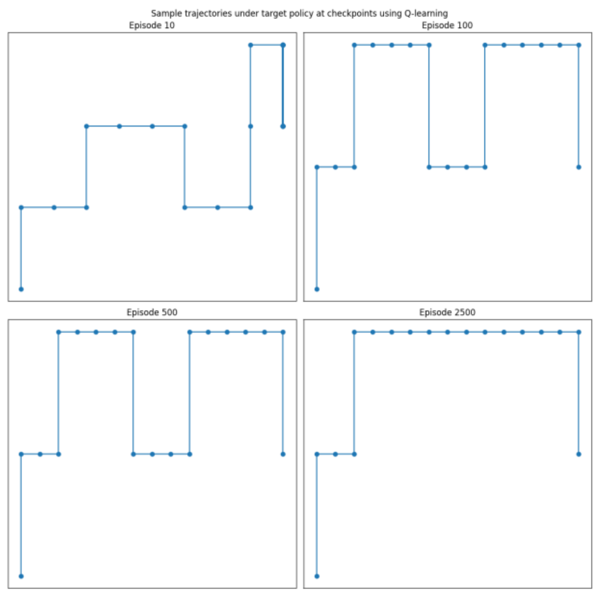




As discussed in Task 2.1, when using a constant ε of 0.2, SARSA and Q-learning converge to different strategies. This is because Q-learning, being off-policy, updates using greedy action values, and is therefore more aggressive - often taking shorter, riskier paths closer to the cliff. In contrast, SARSA, as an on-policy method, tends to avoid risky actions encountered during ε-greedy exploration, resulting in safer, longer paths. When using ε-decay, we gain more control over the convergence process. During the first handful of episodes, the higher ε close 0.3 promotes wider exploration, allowing Q-learning to discover that longer, safer paths away from the cliff can lead to better long-term rewards. As ε decreases, and it falls below 0.2 after 500 episodes, both TD methods begin to exploit their learned policies more than for the constant ε of 0.2. This results in a faster rise in average return and faster convergence for both SARSA and Q-learning, particularly for Q-learning. Notably, with this decay schedule, the average returns for SARSA and Q-learning become much closer than for the constant ε case, and both agents learn to avoid the lava of the cliff. Q-learning, despite its greedy updates, adapts to safer strategies earlier, achieving a maximum single return of -19, indicating that it has learned to take a longer but safer route. If ε were lowered even further, e.g. ε_{end}=0.0001, both agents would further reduce late-stage randomness and might eventually discover that edging very close to the cliff is not risky, potentially adopting an even shorter route. This highlights the importance of tuning ε-decay not only to guide learning but also to influence the risk preference of the resulting policy.

To further illustrate the behaviour, sample trajectories of each method's target policy at different learning stages are plotted. This is done on the cliff environment over 3 000 episodes, with ε starting at 0.30 and linearly decaying to 0.05 over the first 100 episodes (so from episode 101 onward ε remains fixed at 0.05). We use α = 0.1 and γ = 0.99. The checkpoints that we visualise are episodes 10, 100, 500 and 2,500. When inspecting the plots, please keep in mind how the environment looks, i.e. remember that the Cliff grid has only four rows, so moving up twice in the graphic actually places the agent on the second-to-bottom row, even if it appears near the top of the plot.



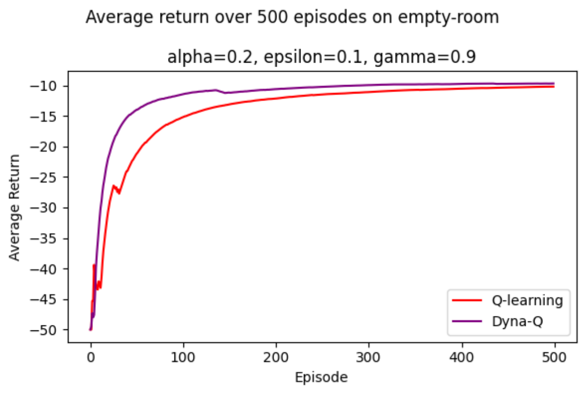


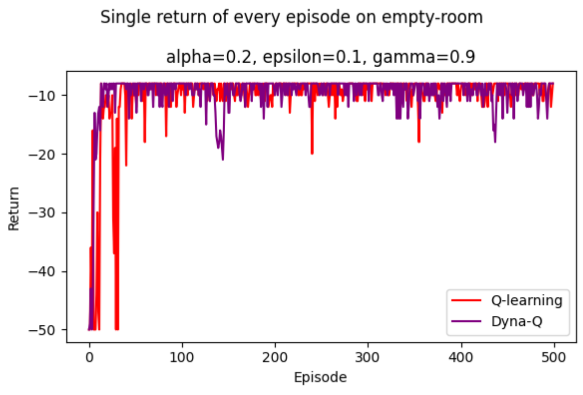
At episode 10, both agents exhibit mostly random behavior, with SARSA occasionally stepping into the lava due to high ε and still exploratory behavior. By episode 100, exploration has already decreased significantly, and SARSA begins to show a more conservative and efficient path to the goal. Q-learning at this point still tends to select slightly riskier paths moving more close to the lava due to its greedy updates. From episode 500 onwards, SARSA's behavior stabilizes, showing little change in trajectory, as ε has already reached its minimum of 0.05. Q-learning, however, continues to adapt, and by episode 2 500, it exhibits a safer trajectory than at episodes 100 or 500, avoiding lava more effectively.

As mentioned earlier, decaying ε helps control the pace of convergence. In this case, with a rapid decay over only 100 episodes, convergence is relatively fast. Moreover, ε-decay influences the risk preference of the final policy - early high exploration allows agents to discover safe alternatives, while lower ε later on enables exploitation of the learned, often safer, strategy.

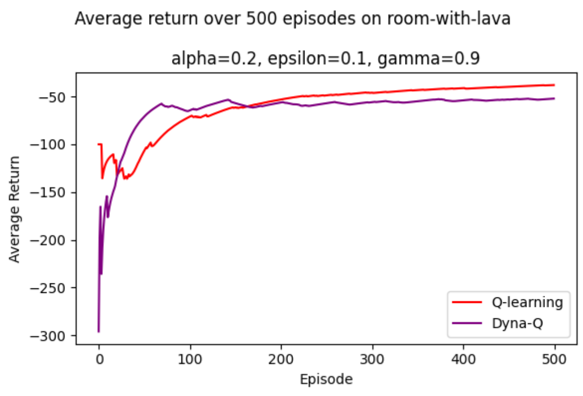
2.3

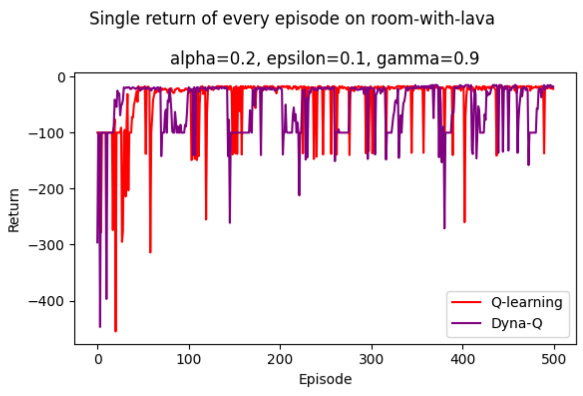
A Dyna-Q agent enhances standard Q-learning by incorporating background planning. In addition to taking real actions and updating its Q-table based on observed transitions, it continually samples from its learned model and performs simulated Q-updates. These extra updates accelerate the propagation of value information, meaning that Dyna-Q often converges to a good policy in fewer real episodes than standard Q-learning, especially in deterministic or low-variance environments. In our case, Dyna-Q uses 10 planning steps - each real step is followed by 10 simulated updates drawn randomly from the model. In each experiment, we run 500 episodes with α of 0.2, a constant ε of 0.1 and γ of 0.9. The first experiment is in an empty room:





The empty-room environment is fully deterministic—no stochastic monsters or lava penalties. It can be seen that Dyna-Q achieves a faster rise in average return and converges faster than regular Q-learning. This is thanks to the ten extra planning updates per step in Dyna-Q, which let it propagate the optimal return from the goal back to all upstream state-action pairs more quickly than plain Q-learning. Thus, in the empty room, Dyna-Q outperforms Q-learning by converging faster achieveing higher average reurns over episodes. The second experiment is done on the room with lava:



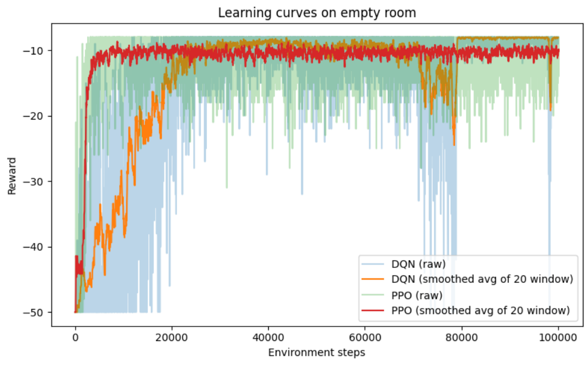


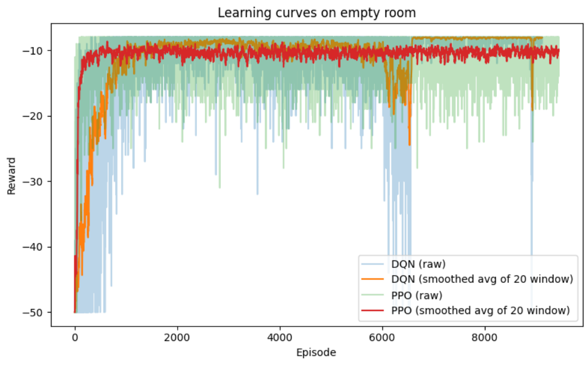
The room with lava environment is still deterministic, but due to the presence of lava and an ε value greater than 0, there is a risk of stepping into lava. Although Dyna-Q learns that stepping into lava is undesirable, the combination of an imperfect model due to limited real-world experience and aggressive planning can sometimes result in suboptimal transitions. Specifically, in the first 200 or so episodes, Dyna-Q propagates overly optimistic (or overly cautious) estimates for lava-adjacent states before it has witnessed enough instances of falling into lava. Consequently, it may learn an apparently attractive path with a single return of around –15, or conversely, an excessively cautious policy that avoids large portions of the room. Around episode 200, Dyna-Q's average return actually dips below that of regular Q-learning as the agent is still refining its model of the lava penalty and occasionally gets stuck stepping into lava or being away from the lava and the goal repeatedly, producing episodes with returns near –100 or lower. Only after accumulating sufficient real lava-fall experiences does the model correctly penalise those transitions. Meanwhile, standard Q-learning, which updates solely from real experience, learns a safer policy (e.g. a single return of –17) and therefore achieves a higher average return after 500 episodes. Dyna-Q may eventually outperform Q-learning, but only once the simulated-planning phase 'unlearns' its initial optimistic or pessimistic assumptions about moves adjacent to lava. It would be useful to potentially test further different ε values, since a lower ε would likely improve Dyna-Q's performance relative to Q-learning over time.

3

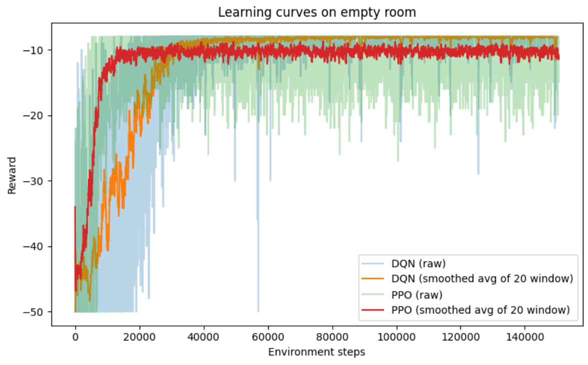
Two deep reinforcement learning agents - a Deep Q-Network (DQN) and an actor-critic method using Proximal Policy Optimization (PPO) - were implemented. Instead of a cumulative average return, we compute a smoothed average over a 20-episode window and also report raw episode returns. We always use an MLP policy, so no convolutional layers. Note that DQN is more sensitive to hyperparameter selection than PPO, therefore we select learning rate, replay buffer size, target-network update frequency, and exploration schedules, very carefully. Two runs are shown on an empty room – an initial one and an adjusted final one. The initial experiment was conducted on 100 000 episodes, using the following configuration settings (with all other parameters left at their default values):

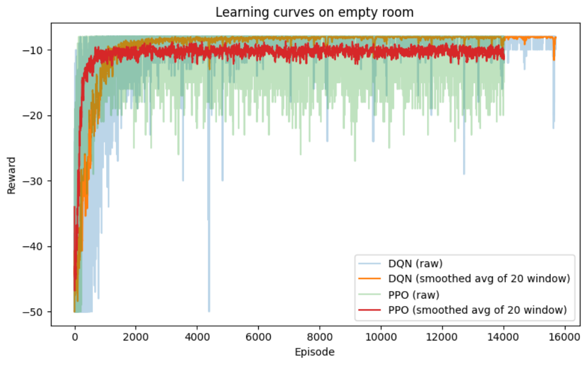
DQN (initial run)		**PPO (initial run)**	
Parameter	**Value**	**Parameter**	**Value**
learning_rate	$1*10^{-4}$	learning_rate	$1*10^{-4}$
buffer_size	25 000	n_steps	256
learning_starts	100	batch_size	64
batch_size	32	n_epochs	10 (default)
train_freq	2	gamma	0.99 (default)
gradient_steps	2	gae_lambda	0.95 (default)
gamma	0.99 (default)	ent_coef	0.0 (default)
target_update_interval	1 000		
exploration_fraction	0.2		
exploration_initial_eps	1.0		
exploration_final_eps	0.01		
policy_kwargs	net_arch=[32, 32]		





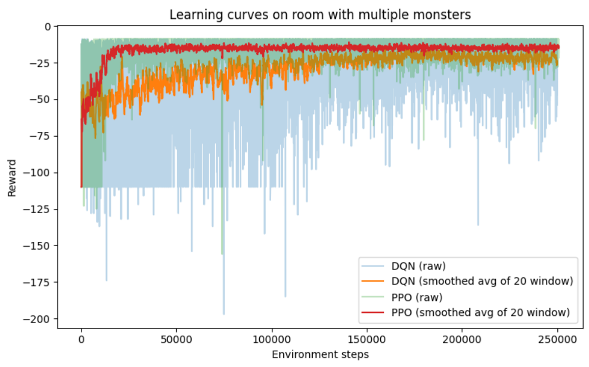
After 100 000 episodes on the empty room with these configurations, DQN achieved a mean return over the last 100 episodes of –8.10 with a standard deviation of 0.39. PPO achieved a mean return over the last 100 of –10.19 with a standard deviation of 2.51. In principle, a large replay buffer of 25 000, a target-network update interval of 1 000 steps, a relatively high discount factor of γ=0.99, and a low final ε=0.01 should stabilize DQN's learning. However, we still observed instability around episodes 70 000–80 000, DQN's smoothed return temporarily dropped before recovering. PPO learned more steadily, but its final mean returned was farther from the optimal of –8 due to its variance remained high. Because DQN remained unstable in that first run, we performed a second, more carefully tuned experiment. We reduced DQN's learning rate from $1*10^{-4}$ to $5*10^{-5}$ (keeping all other parameters the same), and we adjusted the PPO configuration to reduce variance. Specifically, PPO's learning rate remained $1*10^{-4}$, but we aimed to reduce variance by increasing the rollout length (n_steps) from 256 to 1 024, raising batch_size from 64 to 256, cutting number of epochs from 10 to 5, lowering γ from 0.99 to 0.95, reducing GAE λ from 0.95 to 0.90 (thus trading a bit more bias for lower variance), and adding a small entropy coefficient (ent_coef = 0.01) to encourage exploration to some extent. The second run is over 150 000 episodes, as we expect the learning process to take slightly longer with above-mentioned configurations:

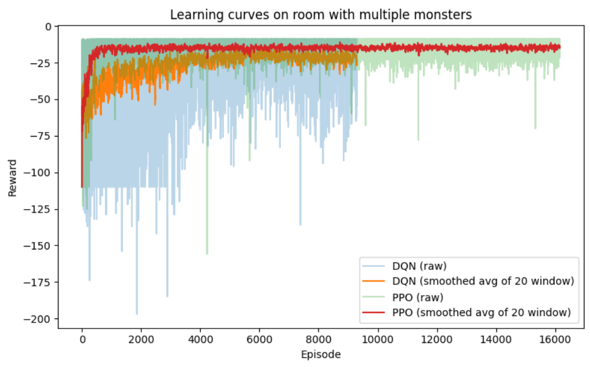




After 150 000 episodes on the empty room with these tuned hyperparameters, the mean return of DQN over the last 100 episodes was −9.10, with a standard deviation of 2.80. PPO's mean return over the last 100 episodes was -10.45, with a standard deviation of 2.53. Reducing DQN's learning rate eliminated the breakout episodes and made its smoothed average return more consistent. However, the final mean return was slightly worse than in the first run, potentially due to one poor episode. PPO's performance remained similar to the first run, with still very high variance. This suggests that PPO is less sensitive to hyperparameters. However, further tuning of the discount factor and GAE λ could potentially reduce its variance slightly, as a higher λ leads to lower bias and higher variance, while a lower λ leads to higher bias and lower variance. Overall, DQN outperformed PPO in the empty room, converging closer to the optimal return, maintaining lower variance and achieving a higher smoothed average return. However, PPO converged faster.

With the same configuration as for the second run of an empty room, this run is performed in a room containing multiple monsters over 250 000 episodes. This is because we expect a room containing multiple monsters to be more difficult to learn.





After 250 000 episodes, DQN's mean return over the last 100 episodes was −21.93 with a standard deviation of 11.57. PPO achieved a mean return over the last 100 episodes of −13.95 with a standard deviation of 4.48. In this stochastic, high-variance environment, PPO clearly outperformed DQN - its average return was much higher and its variance much lower. DQN's replay-buffer–based Q-updates struggled to keep up with randomly moving monsters, causing noisy Q-value estimates and slow convergence. PPO's on-policy actor-critic updates handled the environment's nonstationarity better, yielding faster convergence to a good policy. The DQN could potentially be tested further using more episodes.

Regarding the comparison between the tabular and deep versions of the Q-learner, in an easy environment such as an empty room where the grid is small and discrete, and the environment is deterministic, the tabular version is much better than Deep Q-learning (DQN) due to faster convergence and lower variance. Tabular Q-learning assigns an exact value to every visited state–action pair and converges to the optimal return of –8 in fewer episodes. Thereafter, it shows very small variance, mainly attributed to ε. Therefore, the advantages of the table version of the Q-learner are: perfect accuracy with no function-approximation error, extreme sample efficiency and robust stability, with only α, ε and γ needing tuning. It is therefore better suited to genuinely small state spaces. However, it is not scalable as the table grows linearly, so it breaks down as soon as the state space stops being toy-sized or becomes continuous. DQN flips that trade-off. Because a neural network generalises across similar states, DQN scales gracefully when the state space becomes huge or continuous - a slight shift in a monster's position does not require a new table row. This generalisation is precisely what makes tabular methods infeasible in rooms with multiple monsters, where every new monster arrangement constitutes a new state. The downside is that DQN is sample-hungry, computationally intensive, sensitive to hyperparameters and potentially unstable without replay buffers, target networks and careful learning rate selection. However, DQN enables large or continuous state spaces by sacrificing some exactness for function approximation, but this comes at the cost of stability and sample efficiency.