Katholieke
Universiteit
Leuven

**Department of
Computer Science**

INDIVIDUAL REPORT

Reinforcement Learning
Professor Giuseppe Marra

Lorenzo Viano (r0928156)
Academic year 2024–2025

Task 1: Environments

Grid-World encoding (Task 1.1)

The Grid-World environment was implemented as a custom `gym.Env` class, supporting flexible grid sizes (n, m). The agent is initialised at $(0, 0)$, while the goal is positioned at the bottom-right corner $(n-1, m-1)$. It can move up, down, left, or right, with boundary checks to prevent leaving the grid. Each step incurs a fixed penalty of -1 and episodes terminate upon reaching the goal. The `render` method provides a human-readable visualization of the grid, clearly marking the agent and goal locations. This implementation fully adheres to the `gym.Env` interface, ensuring compatibility with RL pipelines.

Random Agent exploration (Task 1.1)

A natural baseline for evaluating agent performance is the `RandomAgent`, which chooses each move direction uniformly at random, ignoring both its position and the goal. This simple baseline is commonly used in reinforcement learning experiments, as it provides a baseline reference against more sophisticated learning-based agent. Since each step in GridWorld incurs a -1 reward until the goal, the running average return $\hat{G}_k$ converges to a negative value reflecting the typical number of steps required to reach the goal under blind exploration. The random policy maximizes exploration but performs no exploitation, which explains the consistently poor average return observed. As shown in the left plot of Figure 1, the high variability in $\hat{G}_k$ during the first few thousand episodes gradually diminishes, and by around episode 4,000 the learning curve has fully flattened since new episodes no longer affect the average return. At $k = 10{,}000$, we measure $\hat{G}_{10{,}000} \approx -106.8$, indicating that, on average, the `RandomAgent` takes about 107 moves to reach the goal.. The right plot in Figure 1 presents the first ten moves of on example rollout, where the agent wanders aimlessly before reaching the goal.

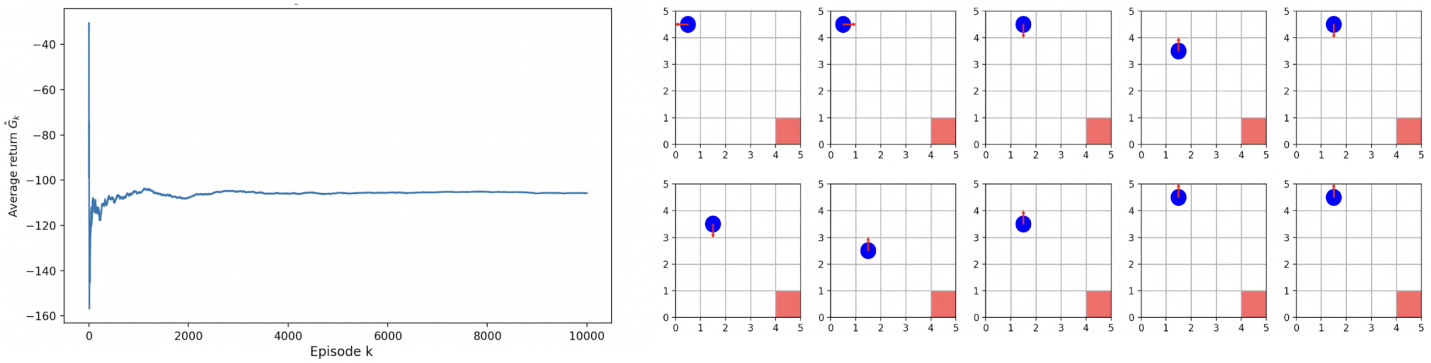


Figure 1: RandomAgent performance in the 5X5 empty Grid-World environment. Left: Cumulative average return $\hat{G}_k$ over episodes. Right: Example trajectory of the agent (blue cricle) towards the goal (red square).

Fixed Agent Exploration (Task 1.2)

For this task, a `FixedAgent` was developed by extending the `commons.AbstractAgent` interface. This agent follows a deterministic policy: it moves down until reaching the grid's bottom, then moves right for the rest of the episode. Unlike the `RandomAgent`, the `FixedAgent` must continuously account for the environment's state, ensuring it does not attempt moves beyond the grid's edges or into obstacles. Figures 2 and 3 illustrate the agent's trajectory in the `EMPTY_ROOM` and `ROOM_WITH_LAVA` environments, respectively. In the `EMPTY_ROOM`, the agent successfully reaches the goal by following its prescribed path while in `ROOM_WITH_LAVA` it gets trapped against a wall barrier after several moves, demonstrating the limitations of a fixed, non-adaptive policies.

Figure 2: Visualisation of FixedAgent predetermined trajecotry in Empty Room Enviroment

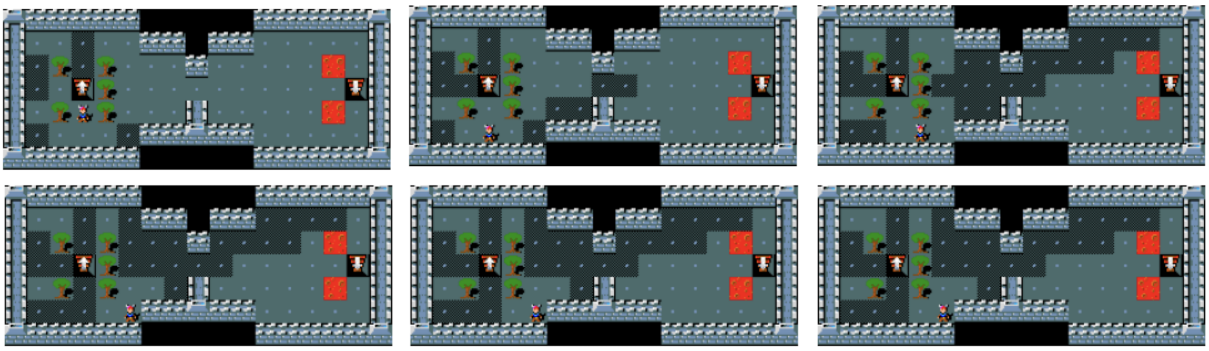


Figure 3: Visualisation of FixedAgent trajecotry in room with lava enviroment

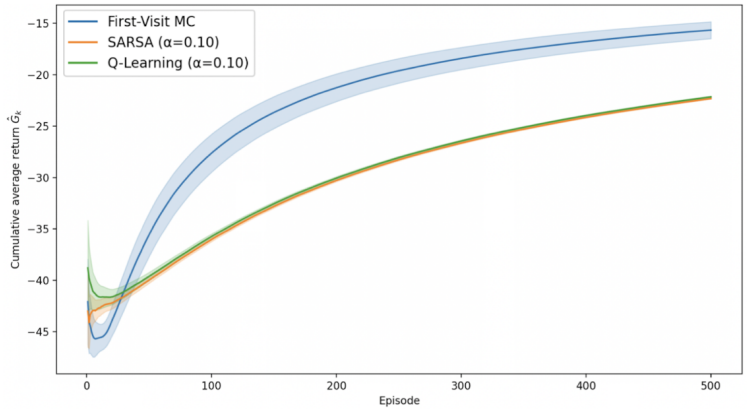
Task 2: Learning

Empty Room Environment (tasks 2.1)

For an initial comparison of Monte Carlo (MC), Q-learning (QL), and SARSA algorithms, we chose the `EMPTY_ROOM` environment due to its simplicity, which allows for clearer insights into algorithm-specific characteristics and convergence behaviors. Theoretically, all three methods should converge to an optimal policy given infinite episodes, provided that $\varepsilon > 0$. However, practically, the choice of exploration rate ε significantly affects each algorithm's convergence speed and reliability. To examine these effects, we set the discount factor $\gamma = 1$ and the learning rate $\alpha = 0.1$. We then conducted an experiment where, for each $\varepsilon \in \{0.01, 0.1, 0.3, 0.5, 0.8, 0.9\}$, we trained 100 independent agents per learning algorithm over 500 episodes each, subsequently evaluating their performance greedily (with $\varepsilon = 0$) to determine whether the known optimal return of –8, corresponding to the shortest-path policy in this environment, was achieved. Recording the proportion of agents that learned the optimal shortest-path policy within 500 episodes provides a direct measure of the exploration–exploitation trade-off, illustrating how smaller values of ε may lead to overly slow convergence, while excessively large ε values may prevent the agent from reliably settling on the optimal policy within the allotted episodes.

The simulation results (Table 1) indicate that both SARSA and Q-learning consistently converge toward one of the optimal policies within 500 episodes across varying ε values, showing relative insensitivity to changes in exploration rates. Conversely, Monte Carlo's performance is significantly impacted by the choice of ε: optimal convergence occurs at approximately $\varepsilon = 0.5$, while performance markedly deteriorates at higher exploration rates ($\varepsilon \geq 0.9$) due to excessive exploration limiting effective exploitation. Similarly, extremely low values of ε (e.g., $\varepsilon = 0.01$) limit exploration to the extent that certain state–action pairs remain undersampled, yielding unreliable Q-value estimates trapping the agent in suboptimal paths.

(a) $\varepsilon = 0.1$, MonteCarlo succeed



(b) $\varepsilon = 0.9$, MonteCarlo failed

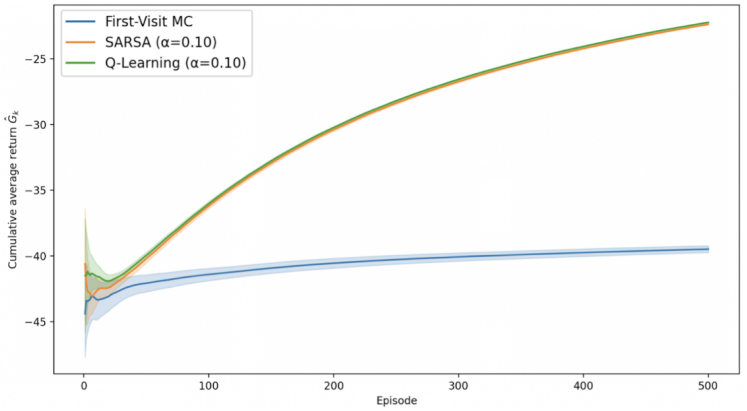


Figure 4: Cumulative average return (mean ± SEM across 10 seeds) of MC, SARSA, and Q-learning in EMPTY_ROOM comparing Monte Carlo at $\varepsilon = 0.1$ (left) and $\varepsilon = 0.9$ (right). For low ε, MC outperforms TD methods due to its efficient and exact value propagation in a small, deterministic state space, despite higher variance across seeds. At high ε, however, excessive exploration degrades MC's overall performance.

Algorithm	$\varepsilon = 0.01$	$\varepsilon = 0.1$	$\varepsilon = 0.3$	$\varepsilon = 0.5$	$\varepsilon = 0.8$	$\varepsilon = 0.9$
Monte Carlo	79	95	99	100	98	69
SARSA	85	97	93	96	94	93
Q-Learning	82	97	100	100	100	100

Table 1: Number of agents (out of 100) achieving the optimal policy in EMPTY_ROOM after 500 episodes, for each algorithm and ε value.

Cliff Environment (tasks 2.1 and 2.2)

In the `CLIFF` environment, distinct behavioral patterns emerge among the three RL methods. Monte Carlo experiences significant challenges in this environment. The large penalty of -100 incurred from falling off the cliff introduces high variance in episode returns. This high variance hampers the stability and speed of MC's convergence, making it difficult for the agent to accurately estimate reliable state–action values within a reasonable number of episodes. Consequently, even tuning of the fixed exploration parameter ε yields limited improvement: reducing ε lowers the frequency of cliff falls (and thus cuts variance), but it also makes the agent more likely to settle on a suboptimal detour, highlighting the limits of using constant ε. Empirical results indicate that the MC agent often becomes stuck without ever reaching the goal, repeatedly choosing safe yet suboptimal actions to avoid the severe cliff penalties. Very occasionally, due to favorable random initializations, the agent may successfully discover a safer, albeit longer, path along the top edge, as visualized in plot (a) in Figure 5.

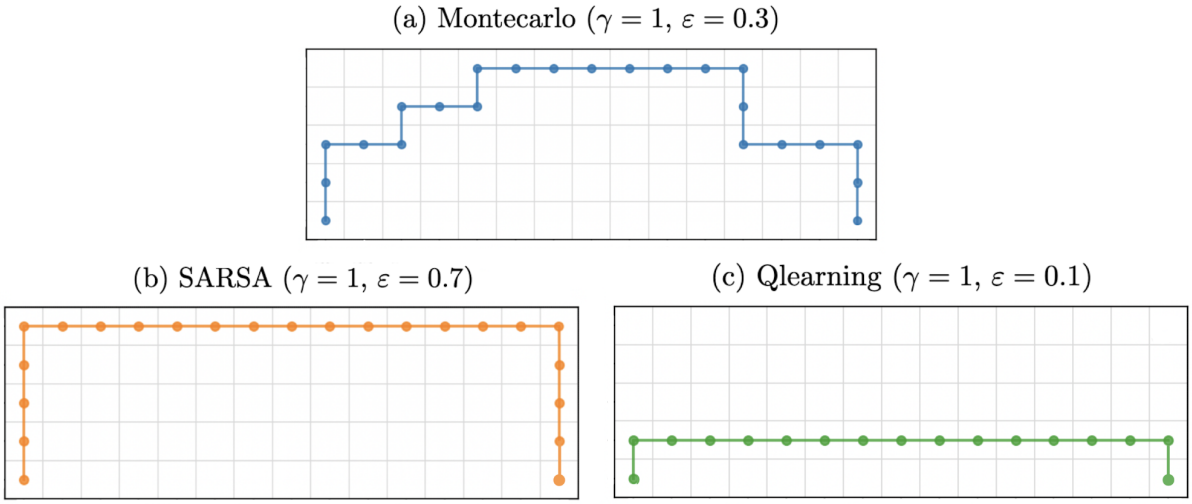


Figure 5: Greedy path after traning for Montecarlo (a), Sarsa (b) and Qlearning (c) in Cliff environment

In contrast to Monte Carlo, temporal-difference methods perform significantly better in this environment. The off-policy *Q-learning* algorithm consistently converges to the shortest and theoretically optimal path adjacent to the cliff. By updating state–action values based on the maximal expected value of subsequent actions, Q-learning inherently promotes an optimal yet risky policy. However, due to its greedy off-policy and ε-greedy exploration, Q-learning often selects cliff-adjacent actions during training, resulting in numerous cliff falls and large negative returns early on. Nevertheless, despite these intial setbacks, Q-learning eventually learns to consistently follow the optimal cliff-hugging path, as shown in plot (c) in Figure 5. Finally, *SARSA* balances exploration and exploitation effectively. Unlike Q-learning, SARSA updates its Q-values based explicitly on the next action selected by its current ε-greedy policy. This means the algorithm directly experiences the consequences of risky moves, rapidly adjusting values for actions leading toward the cliff. As a result, SARSA quickly learns to prefer a safer route, consistently choosing a longer detour along the top edge of the grid (Figure 5). Thanks to its incremental and lower-variance updates, SARSA converges significantly faster than Monte Carlo

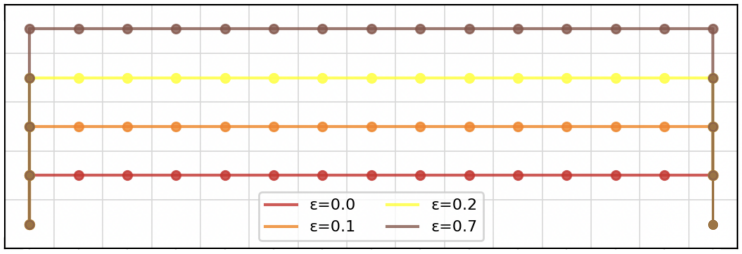


Figure 6: SARSA trajectories after training for different ε

and exhibits stable, risk-averse behavior throughout learning. Additionally, the exploration parameter ε substantially influences SARSA's learned policy: higher ε encourages exploration near the cliff, reinforcing the value of safer paths. Experiments show that at $\varepsilon = 0.7$, SARSA reliably settles on the safest path, maintaining maximum distance from the cliff. Conversely, lower values of ε gradually align SARSA's behavior closer to the riskier optimal path, with $\varepsilon = 0$ essentially reducing SARSA to the equivalent behavior of Q-learning. Figure 6 provides visual evidence of these trajectories for varying ε, illustrating how exploration rates shape learned policies.

Epsilon Decay in Cliff Environment (tasks 2.2)

In order to assess the impact of a linear epsilon-decay schedule on the learning algorithms on the Cliff environment, several experiments were conducted with different values of ε. Performance trends were visualized using a sliding-window average over 50 episodes to reduce the influence of early, highly negative returns.

In the Monte Carlo setting (Figure 7) the linear decay from ε=0.3 to ε=0.1 initially suffers significant negative returns due to frequent cliff falls. However, as exploration gradually diminishes, this schedule surpasses both 0.1 and 0.3 fixed-ε alternatives, ultimately achieving higher average returns. This occurs because early exploration sufficiently populates state-action pairs, allowing accurate first-visit value estimates, while the later reduction in randomness prevents further costly falls. Despite these findings, the environment still presents a great challenge for MC agent, as shown by the very low rewards achieved even after 3000 episodes.

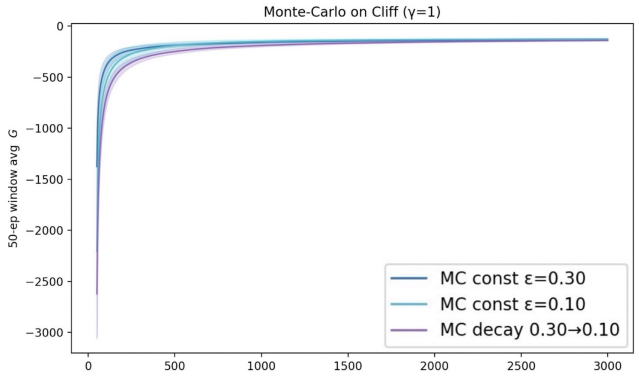


Figure 7: Moving return curves for MC in Cliff

In contrast, for SARSA, constant ε=0.1 maintains the highest performance as it achieves a balance that avoids frequent cliff falls while still occasionally sampling risky states near the cliff, thereby finding shorter routes. With the linear-decay schedule, the agent initially spends the first 100–150 episodes with a relatively high exploration rate ($\varepsilon \approx 0.25$), resulting in frequent and costly cliff falls, as seen by its initial low learning curve in the right plot in Figure 8. These early negative experiences cause SARSA to update the Q-values of cliff-hugging actions to be overly pessimistic, significantly underestimating their true long-term value. Once ε decays to lower values, the agent rarely revisits these risky states, and as a result, these pessimistic Q-values persist without correction. As a result, the policy becomes biased toward the long, safe detour, leading to better performance than the always-high $\varepsilon = 0.30$ baseline, but never reaching the level of the constant $\varepsilon = 0.10$ agent, which avoids both frequent cliff falls and excessive pessimism in its value estimates (see path for $\varepsilon = 0.10$ in Figure 6).

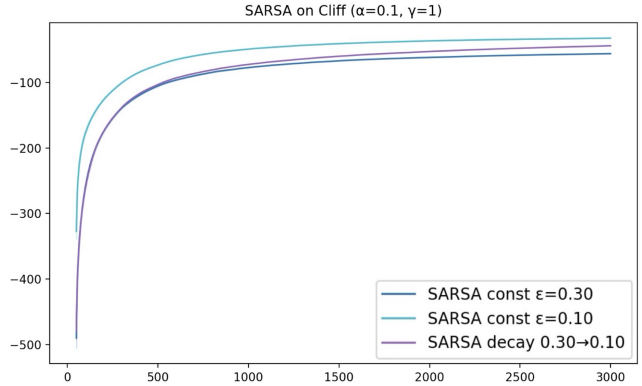


Figure 8: Moving return curves for SARSA in Cliff

Finally, as also shown in the previous section's Figure 5c, Q-learning with a fixed exploration rate of $\varepsilon = 0.10$ consistently finds the shortest possible path, walking right on the hedge of the cliff. This configuration also achieves the highest overall returns curve due to a favorable balance between exploiting the optimal policy and minimal exploratory penalties. Conversely, the linear decay schedule ($\varepsilon = 0.30 \rightarrow 0.10$) remains slightly penalized by early heavy exploration, incurring frequent high-cost events early in learning, which depresses its performance throughout training (Figure 9). Finally, with a constant higher exploration rate of $\varepsilon = 0.30$, the behavior policy never ceases to incur severe penalties, continually stepping into lava cells and producing persistent -100 rewards, underscoring how excessive exploration can severely degrade realized returns, even when near-optimal action values are learned internally.

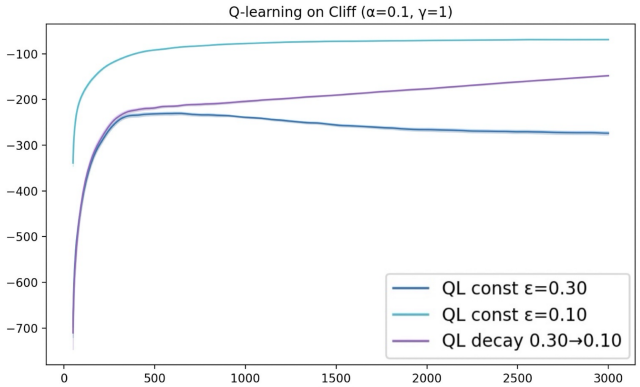


Figure 9: Moving return curves for Qlearning in Cliff

For this task, we also analyzed the evolution of the agents' greedy trajectories over training (Figure 10). The illustration reveals fundamental differences in how the three algorithms adapt in the cliff-walking environment. In the early stages (50 and 70 episodes), Monte Carlo with linearly decaying ε ($\varepsilon = 0.30 \rightarrow 0.10$) remains near the starting area, showing slow and indecisive progress. This illustrates Monte Carlo's sample inefficiency: since it only updates state–action values at episode end, and most early episodes end with the agent falling off the cliff, the agent collects little useful experience, and its policy remains highly exploratory. By 500 episodes, Monte Carlo begins to make progress, but its path is still suboptimal and does not reach the goal.
In contrast, SARSA with a fixed $\varepsilon = 0.7$ rapidly discovers and consistently follows a safe but longer route along the top of the cliff, converging already after 500 episodes to a risk-averse policy that minimizes the chance of falling. Finally, Q-learning, with $\varepsilon = 0.1$, demonstrates fast and decisive learning. Even after just 70 episodes, the agent's greedy policy hugs the cliff edge and, aside from a small upward jitter, moves nearly straight to the goal, quickly converging to the optimal path and by episode 500 it has fully converged to the shortest path.

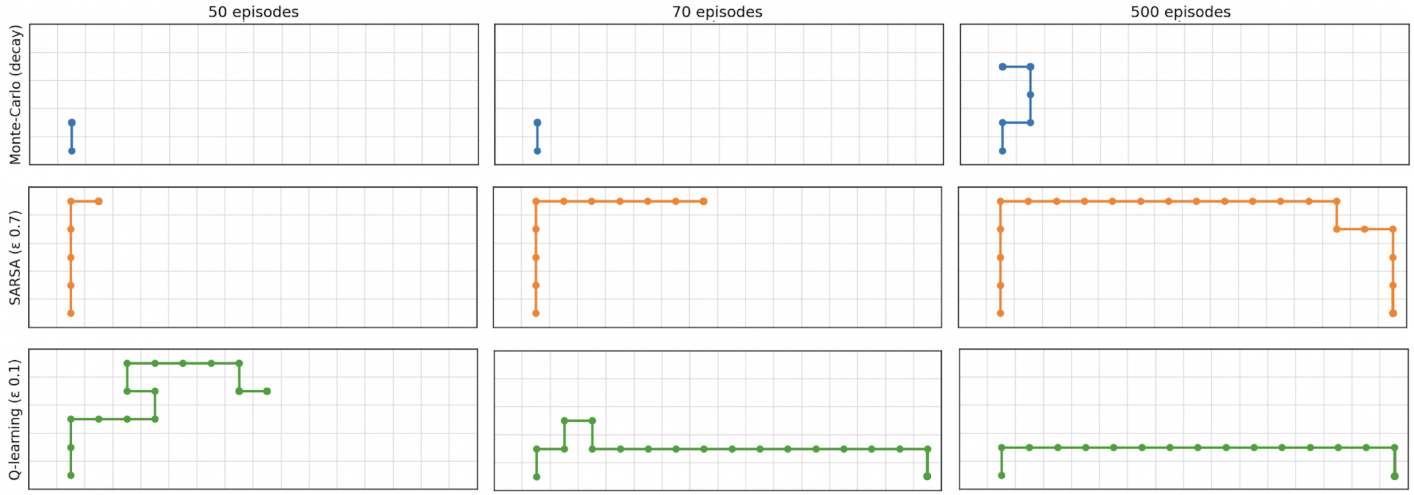


Figure 10: Greedy paths after 50, 70, and 500 episodes for Monte Carlo (top row, using linear ε-decay), SARSA (middle row, $\varepsilon = 0.7$), and Q-learning (bottom row, $\varepsilon = 0.1$) in Cliff environment.

Room With Lava (tasks 2.1)

In the `ROOM_WITH_LAVA` environment, specific experimental configurations were conducted to evaluate the performance of Monte Carlo, Q-learning, and SARSA algorithms. The discount factor γ was set to 1 and α to 0.1. Cumulative learning curves (left plot Figure 11) were generated by plotting the cumulative average of the mean returns across ten random seeds. Each episode's return was first averaged across seeds and then cumulatively averaged to smooth out stochastic fluctuations and clearly highlight convergence behaviors. Additionally, moving average curves (right plot Figure 11) were also plotted to better visualize recent performance trends without the highly negative impact of early exploration episodes. Furthermore, Table 2 summarizes both the agents' average per-episode returns from episode 100 to 2000 and their final path rewards after 2000 episodes, evaluated under a greedy policy ($\epsilon = 0$). Results are reported as the mean and standard deviation across seeds, acknowledging that varying training trajectories result in slightly different learned Q-tables.

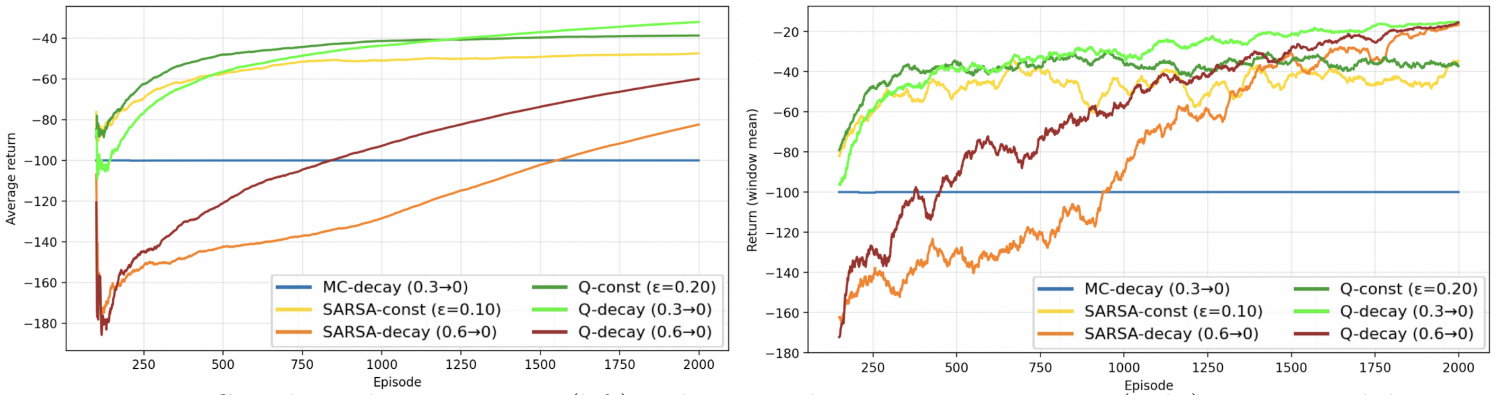


Figure 11: Cumulative learning curves (left) and 50-episode moving return curves (right) in room with lava

In particular, the Monte Carlo method notably struggled, failing to converge toward a viable policy. This issue primarily arose because the Minihack environment halted interaction during episodes where the agent became trapped in repetitive loops, typically after approximately 100 steps. In such scenarios, remaining stationary provided a relatively higher return (around -100) compared to exploring and frequently encountering lava, so the Monte Carlo agent learned to stand still or repeat actions, stalling learning progress toward the goal.

Agent's Algorithm	Avg Return	Greedy Return
MC-decay ($0.3 \to 0$)	-100.0 ± 0.0	-100.0 ± 0.0
SARSA-const ($\epsilon = 0.10$)	-47.4 ± 0.5	-59.3 ± 13.6
SARSA-decay ($0.6 \to 0$)	-82.4 ± 1.0	-16.6 ± 0.7
Q-const ($\epsilon = 0.20$)	-38.7 ± 0.3	-15.0 ± 0.0
Q-decay ($0.3 \to 0$)	-32.0 ± 0.3	-15.0 ± 0.0
Q-decay ($0.6 \to 0$)	-60.0 ± 0.6	-15.0 ± 0.0

Table 2: Performances room with lava (episodes 100–2000, 10 seeds). Both mean return and final greedy policy return are reported as mean ± standard error across 10 seeds.

In contrast, Q-learning showed strong performance, especially with an exploration strategy decaying from $\epsilon = 0.3$ to 0. This strategy outperformed both a constant $\epsilon = 0.2$ and a more exploratory ϵ decay starting at 0.6. The ϵ-decay from 0.3 achieved an optimal balance, exploring sufficiently early to discover advantageous paths sooner while rapidly stabilizing and exploiting learned Q-values as ϵ decreased. Evidence from the cumulative learning plot showed that the ϵ-decay ($0.3 \to 0$) consistently remained above all other approaches from approximately episode 1250 onward, finishing with a notably higher cumulative average return. Similarly, the moving window average plot demonstrated that the ϵ-decay ($0.3 \to 0$) approach maintained superior performance from around episode 300 onward, significantly outperforming the ϵ-decay ($0.6 \to 0$) strategy, which only caught up much later around episode 1800. As shown in Table 2, the training average returns further underscored this, with the ϵ-decay ($0.3 \to 0$) strategy achieving the highest returns (-32 ± 0.3): while all Q-learning variants eventually found the same optimal greedy policy (around -15 steps), the $0.3 \to 0$ decay strategy discovered it more quickly, thus enjoying improved returns during the training phase.

For SARSA, the experimental findings were nuanced. While a constant $\epsilon = 0.1$ strategy initially appeared superior based on cumulative curves, the ϵ-decay (from $0.6 \to 0$) ultimately achieved better final performance. During the initial phase ($\epsilon \approx 0.6$), extensive exploration frequently resulted in encounters with lava, leading to early negative returns that significantly affected the cumulative average in Figure 11. However, once ϵ dropped below 0.2, updates became increasingly greedy, allowing Q-values to stabilize and guiding the agent towards a shorter, albeit riskier, path. In contrast, with ϵ fixed at 0.1, SARSA consistently anticipated random exploration, preferring safer but substantially longer paths away from the lava boundary. Consequently, although the cumulative average misleadingly favored the constant-ϵ strategy due to its permanent integration of early negative returns, moving window averages and the final greedy tests clearly indicated the superiority of the ϵ-decay approach. The results in Table 2 confirm this, with the ϵ-decay strategy achieving near-optimal final returns (-16 ± 0.7) compared to the constant-ϵ strategy's substantially longer and suboptimal returns with higher variance across seeds (-59 ± 13). Overall, Q-learning consistently dominated SARSA and Monte Carlo in both cumulative and moving window plots and in final greedy path performance, consistently finding the optimal path. Its off-policy bootstrap mechanism facilitated rapid learning and convergence to the optimal yet risky shortest path, underscoring its effectiveness in more challenging environments.

Room With Monster (tasks 2.1)

In `ROOM_WITH_MONSTER` environment, five reinforcement learning agents were compared: Monte Carlo with an ε-decay schedule, SARSA with both constant and decaying ε strategies, and Q-learning with constant and decaying ε. The agents were trained over 15,000 episodes across 10 random seeds. Performance was assessed using 50-episode rolling averages to illustrate short-term adaptation (Figure 12) and stacked-bar histograms (Figure 13) to visualize the distribution of outcomes categorized as "good wins" (returns greater than -10), "normal wins" (wins with returns smaller than -10), or failures (due to deaths during the episode or reached timeout). Q-learning with a constant ε of 0.10 demonstrated the fastest initial learning as shown in its average returns curve

in Figure 12. The moderate, fixed exploration rate allowed early discovery of efficient and safe paths, after which its off-policy bootstrapping rapidly propagated optimistic Q-values throughout the state-action space. Consequently, Q-learning quickly improved, achieving an 80% win rate within 3,000 episodes (Figure 13), after which performance plateaued with minimal further gains during the remaining training episodes.

SARSA, employing a constant ε of 0.15, also showed strong initial performance. The slightly higher exploration rate enabled effective early exploration, rapidly improving policy quality and achieving an approximately 80% win rate after about 4,500 episodes, but once again without achieving further gains during the remaining training episodes (Figure 13, bottom right graph). In contrast, SARSA with a decaying ε schedule from 0.60 to 0.05 initially performed poorly, due to extensive early exploration and frequent encounters with the monster, as shown by its initial low moving average and high failures rates. However, this extensive exploration phase effectively populated the Q-table with comprehensive state-action coverage. Once exploration diminished below $\varepsilon = 0.2$, the policy became highly effective, rapidly improving exploitation efficiency. By episode 15,000, this agent achieved the highest overall win rate, approaching 90%, surpassing all competitors despite its slower start (Figure 13, bottom left graph). Despite these strong overall win rates, SARSA methods consistently showed fewer "good wins" due to inherent risk aversion in their on-policy updates. Each action near the monster penalizes the subsequent Q-value estimations, encouraging slightly longer, safer trajectories to avoid the monster, differently from QL that occasionally confronted and killed it.

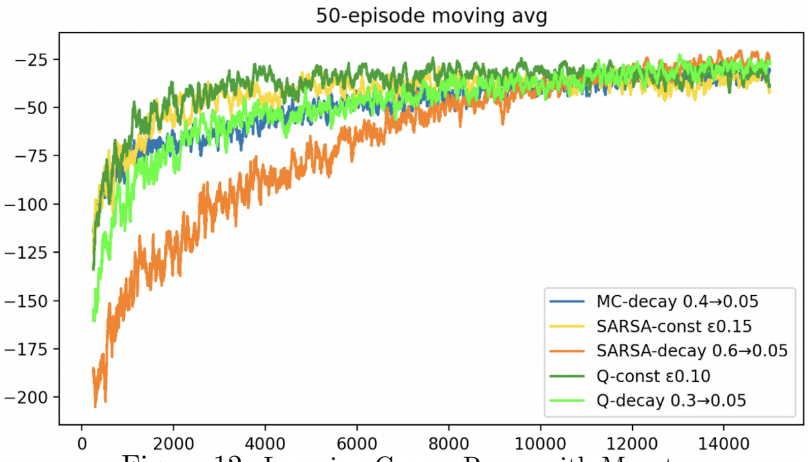


Figure 12: Learning Curves Room with Monster

Finally, Monte Carlo with ε-decay from 0.40 to 0.05 uniquely achieved the highest proportion of "good wins" (Figure 13 top right plot). Because MC updates are episodic and integrate complete returns, occasional successful short paths yield highly optimistic returns, strongly incentivizing risky but potentially optimal routes: MC yet incurred significantly more failures early in training due to risky explorations, leading to a slower overall increase in win rate compared to the other algorithms, highlighting its weakness in stochastic environments.

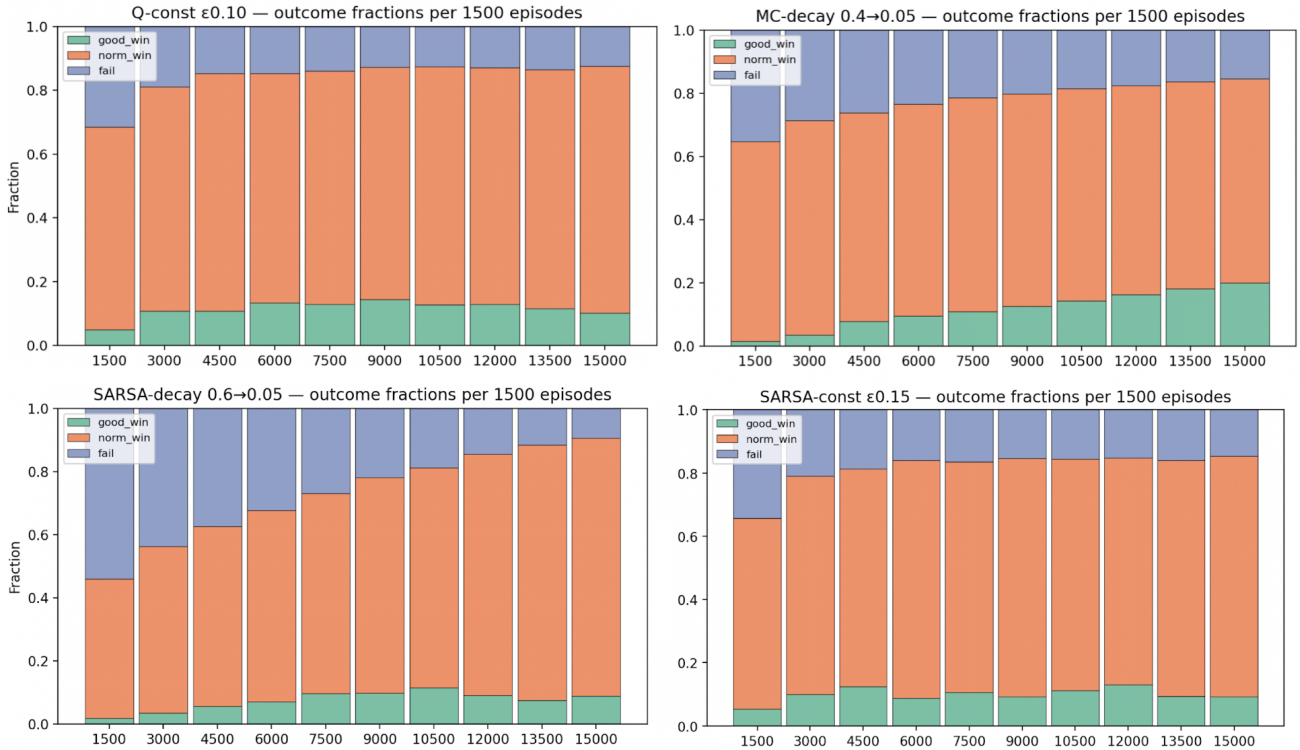


Figure 13: Stacked-bar histograms of outcomes fractions: good wins, normal wins, and failures

DynaQ algorithm (tasks 2.3)

In addition to the previously discussed algorithms, a Dyna-Q agent was developed and compared with traditional Q-learning in two distinct environments: the deterministic `CLIFF` environment and the stochastic `ROOM_WITH_MONSTER`. For both agents α and ϵ were set to 0.1 and Dyna-Q planning steps were set to 10.

In the deterministic `CLIFF` environment, the inclusion of Dyna-Q's internal planning significantly accelerated learning during the initial episodes (Figure 14). Specifically, Dyna-Q rapidly improved from an average return of approximately -150 to around -50 within the first 30 episodes, considerably outperforming Q-learning, which remained below -150 at the same stage. This early advantage stems from Dyna-Q's ability to perform additional simulated updates using an accurate internal model, quickly propagating knowledge of severe penalties such as the cliff's -100 penalty, enabling faster recognition and adherence to safer paths. However, as training progressed, Q-learning gradually approached the performance of Dyna-Q, consistent with theoretical predictions that both methods converge to the same optimal policy in expectation in deterministic finite Markov decision processes, differing primarily in convergence speed. Furthermore, Dyna-Q demonstrated narrower variance bands across runs with the ten different seeds, reflecting reduced run-to-run variability due to accurate internal planning. This result highlights the efficiency of Dyna-Q in deterministic environments: by leveraging accurate internal models for planning, Dyna-Q achieves several-fold faster learning than Q-learning while incurring only minimal additional computational cost for its simulated updates.

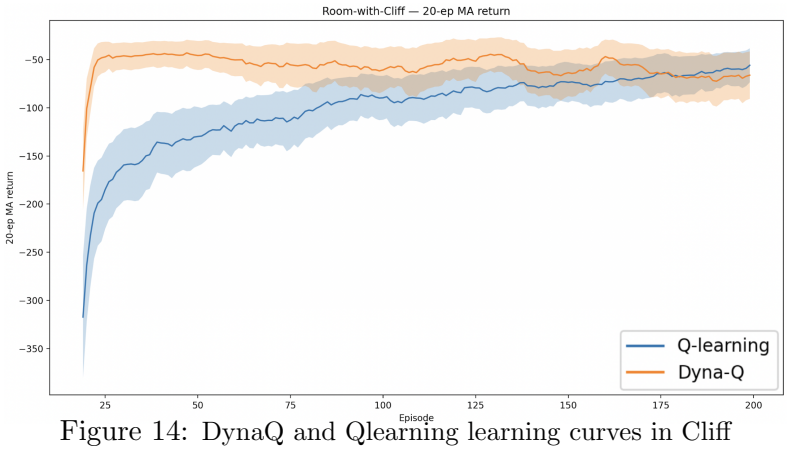


Figure 14: DynaQ and Qlearning learning curves in Cliff

Conversely, in the stochastic `ROOM_WITH_MONSTER` environment, Dyna-Q exhibited notably poorer performance compared to standard Q-learning. Over 15,000 episodes, the average return of Dyna-Q stabilized around -75, significantly worse than Q-learning's stabilization near -40 (Figure 15, left plot). Additionally, the variability in Dyna-Q's outcomes across seeds was substantially higher, as indicated by wider confidence intervals. Further histogram analysis reinforced these observations, revealing that Dyna-Q achieved a considerably lower overall success rate (approximately 70%) compared to Q-learning's success rate exceeding 85%, accompanied by a substantially higher incidence of failures (Figure 15, right plot). The reduced effectiveness of Dyna-Q in the stochastic environment is primarily due to inaccuracies inherent in its single-sample internal model. This simplistic model frequently stores suboptimal transitions resulting from stochastic monster movements, causing biased Bellman targets. With increased planning steps, these biases become magnified, continually reinforcing incorrect expectations and causing the agent to frequently mispredict the monster's position, resulting in increased failures. In contrast, Q-learning's model-free approach relies solely on unbiased real experiences, thus yielding better target estimations, ultimately leading to a more robust and reliable policy convergence in stochastic environments.

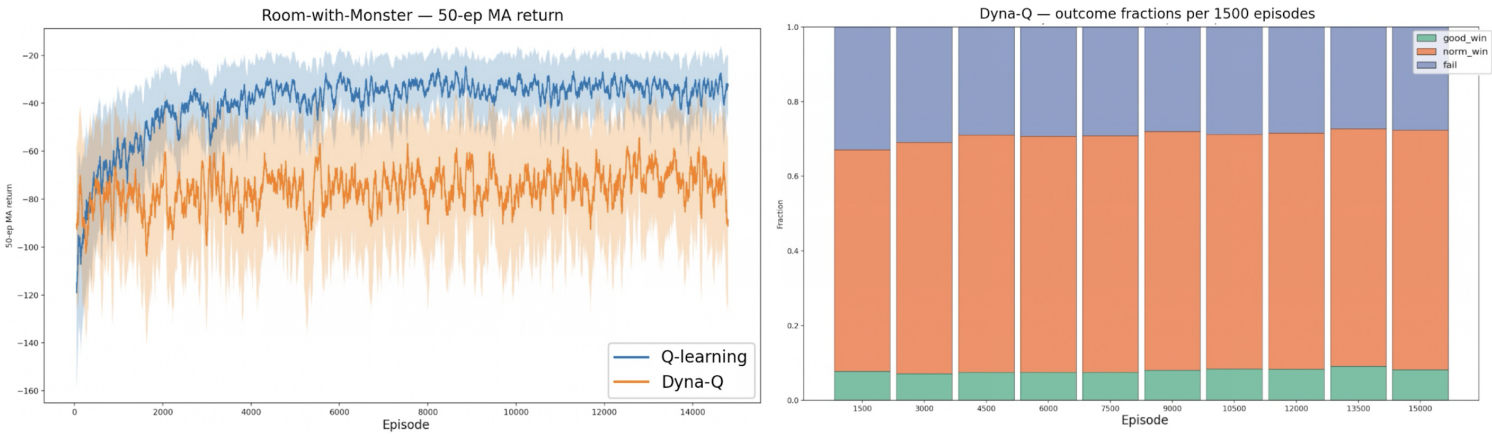


Figure 15: Room-with-Monster benchmark comparing Q-learning and Dyna-Q. Left: 50-episode moving-average return (mean ± SEM across 10 seeds). Right: Histograms of outcome fractions (good_win, norm_win, fail).

1.0 Task 3: Deep Reinforcement Learninging

Deep Reinforcement Learning in Empty Room (tasks 3)

In this final section of the report, we assess the performance of two deep reinforcement learning algorithms, Deep Q-Network (DQN) and Proximal Policy Optimization (PPO),across environments of increasing complexity and stochasticity: `EMPTY_ROOM` and `ROOM_WITH_MULTIPLE_MONSTERS`.

Firstly, the two algorithms were tested against traditional tabular Q-learning with epsilon decay in the empty room environment. This environment, characterized by a deterministic 5×5 empty grid, presents a simple and small discrete state space (approximately 25 states with four possible actions each). Observations for DQN and PPO were preprocessed by extracting a 3×3 egocentric crop around the agent, transforming each state into a fixed-size representation suitable for SB3's MlpPolicy. The evaluation involved plotting cumulative-average returns, restricted to the initial 2,000 episodes to highlight early learning behaviors (see Figure 16). Analysis of the cumulative-average returns revealed that traditional Q-learning quickly achieved near-optimal performance, rapidly improving from an initial random-policy average of approximately -50 to about -15 within the first 400 episodes, after which performance steadily increased. PPO also demonstrated quick improvement but still lagged behind Q-learning. In contrast, DQN showed significantly slower improvement, requiring substantially more samples, around 10,000 episodes with the experiment settings-, to approach comparable performance levels. It is important to highlight how this difference was accentuated by our experimental choices: specifically, DQN was configured with a relatively large two-hidden-layer MLP (256 units per layer) and infrequent target network synchronization (every 10,000 steps), both of which increase the function approximation burden and slow down convergence in simple tasks, emphasizing the approximation overhead. Using a smaller network architecture would likely reduce this gap.

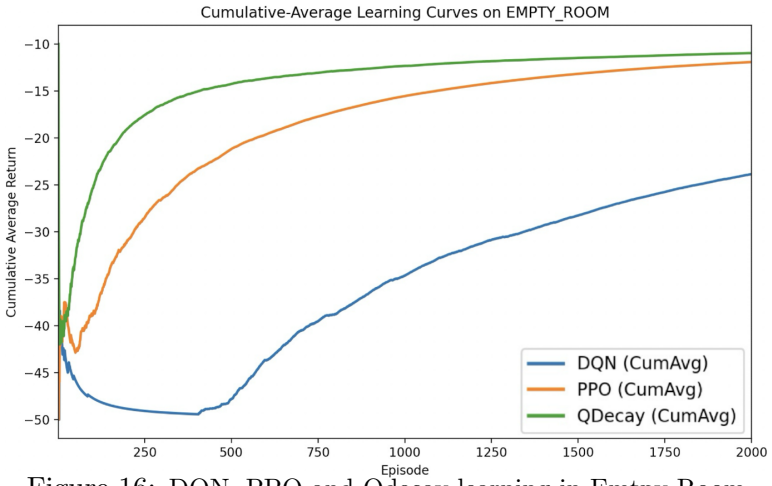


Figure 16: DQN, PPO and Qdecay learning in Emtpy Room

The superior early performance of tabular Q-learning in this environment is primarily due to its straightforward tabular approach, which allows precise implementation of exact Bellman-backup updates, efficiently converging to optimal Q-values once sufficient state-action pairs have been observed. In contrast, both DQN and PPO rely on function approximation via neural networks, introducing additional overhead and approximation errors that delay learning. This is particularly pronounced in DQN due to overestimation and instability in target updates while PPO's on-policy approach, using clipped policy-gradient updates, provided a more stable and sample-efficient learning process, enabling it to outperform DQN in early learning stages. Thus, in environments with small, deterministic state spaces, traditional tabular methods like Q-learning typically exhibit significant advantages in sample efficiency and early learning speed. In contrast, deep reinforcement learning models such as DQN and PPO introduce unnecessary complexity, which leads to slower convergence and greater instability, making them less efficient than precise tabular methods in very simple and non-stochastic environments.

Deep Reinforcement Learning in Room with Multiple Monsters (tasks 3)

The power of deep reinforcement learning algorithms becomes particularly evident in complex environments, such as the `ROOM_WITH_MULTIPLE_MONSTERS` scenario. To thoroughly evaluate the potential of these model we carefully tuned the hyperparameters of Deep Q-Network and Proximal Policy Optimization algorithms. For optimal performance, hyperparameters such as learning rate, batch size, exploration parameters, discount factor, and network architecture have been tuned systematically through Bayesian optimization. The final configurations used in the experiments can be seen in Table 3 for DQN and in Table 4 for PPO.

We evaluated these methods by plotting the moving-average returns over 100-episode windows across the first 20,000 episodes (Figure 17). Results from these plots highlighted clear distinctions among the approaches. PPO

Table 3: DQN hyperparameters

Param	Value
γ	0.95
Batch size	16
Learning starts	1000
τ	0.9
Target update int.	10,000
Expl. fraction	0.45
ϵ_{init}	0.55
ϵ_{final}	0.152
Policy arch.	MlpPolicy (2×256)

Table 4: PPO hyperparameters

Param	Value
γ	0.95
Batch size	32
n_steps/update	1024
n_epochs	10
Clip range	0.2
GAE λ	0.998
v_f coef	0.94
Target KL	0.018
Policy arch.	MlpPolicy (2×256)

demonstrated exceptional efficiency, rapidly improving performance and reaching near-optimal returns as early as episode 3,000. DQN, although eventually achieving comparable performance, required significantly more episodes, converging around episode 9,000. Q-learning showed markedly slower progress, never converging to the deep models returns and with a substantial oscillations across episodes. Such volatility is characteristic of tabular methods in stochastic environments, where extensive variations in state transitions due to random monster movements greatly hinder stable value convergence. Further supporting these findings, Figure 18 presents the comparative analysis of success rates (episodes completed without dying) alongside "good wins," defined as successful episodes with rewards greater than −10. This analysis underscores clearly the superior capabilities of deep reinforcement learning algorithms. PPO achieved a 100% success rate by episode 3,000, whereas DQN reached the same milestone near episode 7,500. Q-learning, constrained by its tabular representation, never reached such results, plateauing at an 80% success rate early on, but without being able to achieve clear improvements across successive episodes. Notably, beyond 15,000 episodes, both PPO and DQN reached similar percentages of "good wins," slightly exceeding 30%. PPO's rapid and superior convergence compared to DQN can be attributed to its on-policy updates, which effectively utilize current policy samples and stabilize training through policy clipping and advantage estimation.

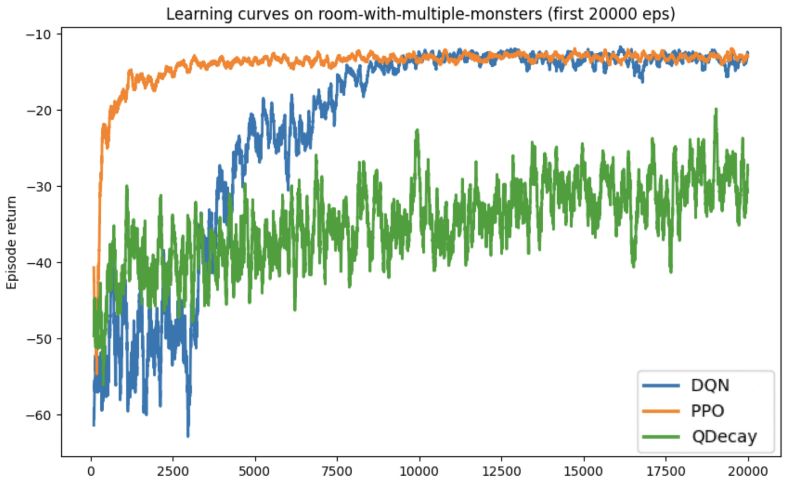


Figure 17: DQN, PPO and Qdecay learning in Monsters Room

These experiments illustrate the robustness and efficacy of deep reinforcement learning techniques in complex and stochastic environments, in contrast to their performance in very simple scenarios such as the empty room environment. Despite being inherently more complex, and harder to implement, tune, and train compared to tabular methods, deep learning methods, through their function approximation and generalization capabilities from limited experience, are able to effectively handle unseen states and stochastic variability. This allows them to significantly outperform traditional tabular methods in tasks that involve high-dimensional inputs, uncertainty, and large or continuous state spaces, as demonstrated in these experiments.

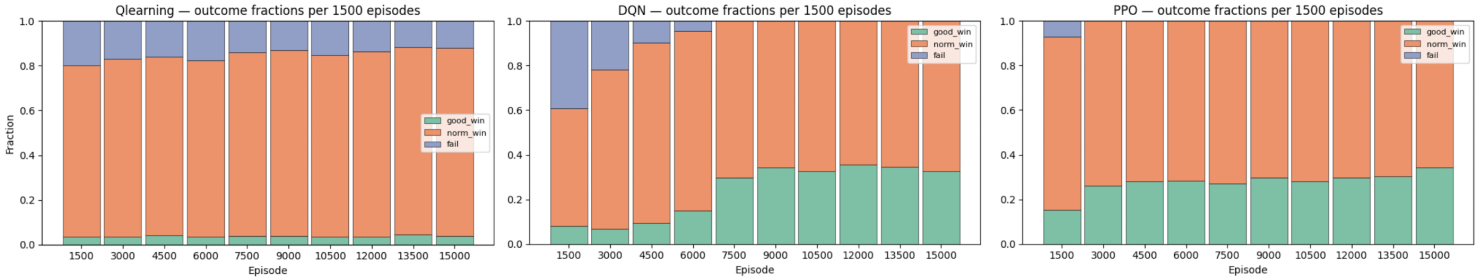


Figure 18: Stacked-bar histograms of outcomes fractions (good wins, normal wins, and failures) in ROOM_WITH_MULTIPLE_MONSTER for Q-learning (left), DQN (center) and PPO (right)