ZHAW School of Engineering
Technikumstrasse 9
8400 Winterthur

Reinforcement Learning

Graded Lab 1

Supervisor: Ricardo Chavarriaga

Authors: Anja Edelmann
edelmanj@students.zhaw.ch

Cyrill Gurtner
gurtncyr@students.zhaw.ch

Submitted: November 1st, 2024

Graded Lab 1 - MC & Q-Learning

Monte Carlo with Random Policy

Parameters

total_episodes

Specifies the number of episodes, or complete sequences of state-action pairs, for which the agent will interact with the environment.

map_size

Determines the size of the grid map used in the Frozen Lake environment. A value of 5 means a 5x5 grid, while 11 means an 11x11 grid.

seed

Sets the random seed for the environment to ensure that the experiment is replicable.

is_slippery

Defines whether the environment is "slippery" (i.e., actions may have non-deterministic outcomes) or deterministic (i.e., actions lead directly to the intended state).

proba_frozen

Sets the probability that a tile on the map is "frozen" (i.e., safe for the agent to step on) rather than a hole.

Missing Parameters

In this Monte Carlo setup with a random policy, gamma (discount factor) and epsilon (exploration rate) are not required.

gamma

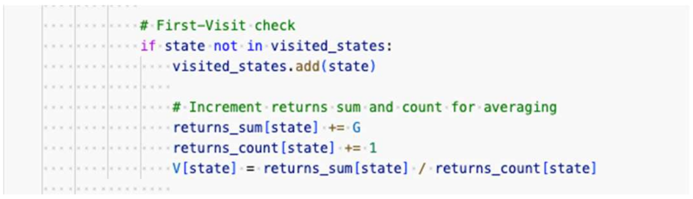
Since each episode in the Frozen Lake environment ends upon reaching a terminal state (goal or hole), discounting future rewards with gamma is not necessary. Each return is calculated from the beginning of an episode to its end without discounting, as there is no incentive to favor earlier rewards over future rewards in this episodic setup.

epsilon

In an epsilon-greedy policy, epsilon would control the balance between exploration and exploitation. However, this setup employs a fully random policy that does not rely on learned values to make decisions, so epsilon does not apply. All actions are equally probable without considering the value of choosing one action over another, making epsilon unnecessary.

First Visit vs. Multiple Visit

In our Monte Carlo approach, we opted for the first-visit method because it provides a straightforward yet robust way of estimating the value function by focusing on the first time each state is visited in an episode. This approach simplifies both the implementation and the computation of state values, as we only record the first occurrence of each state within a single episode, making it unnecessary to track multiple visits. In a sparse-reward environment like Frozen Lake, this choice is especially relevant because rewards are typically only encountered at the episode's conclusion.



```
            # First-Visit check
            if state not in visited_states:
                visited_states.add(state)

                # Increment returns sum and count for averaging
                returns_sum[state] += G
                returns_count[state] += 1
                V[state] = returns_sum[state] / returns_count[state]
```

Figure 1: Implementation First-Visit

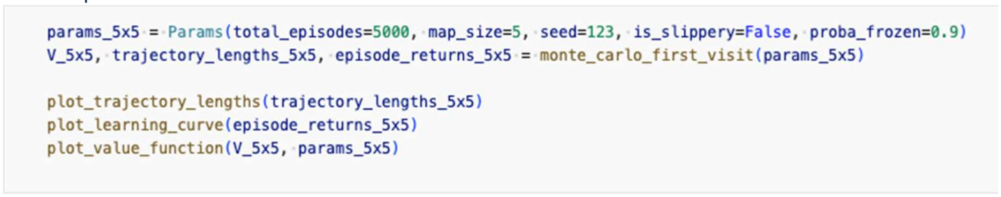
The first-visit method is well-suited for convergence efficiency since it reduces the variability in state-value estimates that could arise from multiple visits to the same state within a single episode. Given that we use a random policy here, our agent is likely to visit a wide array of states in each episode, often re-encountering states during a single run. Without a prioritized selection of states, using first-visit Monte Carlo allows us to collect stable, unbiased estimates of state values that converge more predictably over episodes, helping to avoid the high variance that an every-visit approach might introduce.

Hyperparameter Optimization

For the random policy, there are no hyperparameters to tune. The random policy simply selects actions uniformly from the available action space without any dependence on learned values or exploration rates.

In a random policy, the agent's actions do not depend on learned information or environmental feedback. This lack of dependency means we have no parameters to adjust, as the policy is fixed by design.

5x5 Map Evaluation



```
params_5x5 = Params(total_episodes=5000, map_size=5, seed=123, is_slippery=False, proba_frozen=0.9)
V_5x5, trajectory_lengths_5x5, episode_returns_5x5 = monte_carlo_first_visit(params_5x5)

plot_trajectory_lengths(trajectory_lengths_5x5)
plot_learning_curve(episode_returns_5x5)
plot_value_function(V_5x5, params_5x5)
```

Figure 2: 5x5 Random Policy Map Parameters

We initialize the Params with the following parameter values:

total_episodes=5000 We generate 5'000 episodes to evaluate a **map_size=5**. In addition we set **is_slippery=False** to have an output as deterministic as possible.

To give the agent a higher chance to explore the environment without constantly falling into holes, which could terminate the episode early, we set the **proba_frozen**=0.9.

Length of Trajectories per Episode

In the first plot, titled "Length of Trajectories per Episode", we observe the trajectory length (number of steps) taken by the agent in each episode throughout the training process.

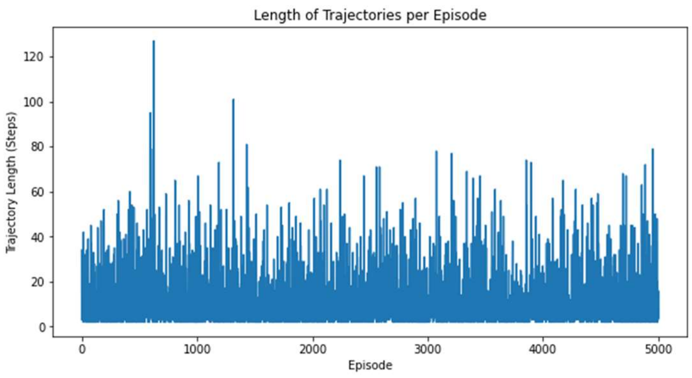


Figure 3: 5x5 Random Policy Length of Trajectories per Episode

The variation in trajectory lengths is high, indicating that the random policy leads the agent to reach the terminal states (either the goal or a hole) at differing paces. Some episodes are short, suggesting that the agent either quickly finds the goal or falls into a hole, while other episodes are longer, showing that the agent sometimes explores more before reaching an endpoint. The overall spread reflects the randomness in path selection, as there is no policy guiding the agent toward shorter or more efficient paths.

Learning Curve (Cumulative Reward over Episodes)

The plot, titled "Learning Curve (Cumulative Reward over Episodes)", shows the cumulative rewards accumulated by the agent over the course of 5'000 episodes.

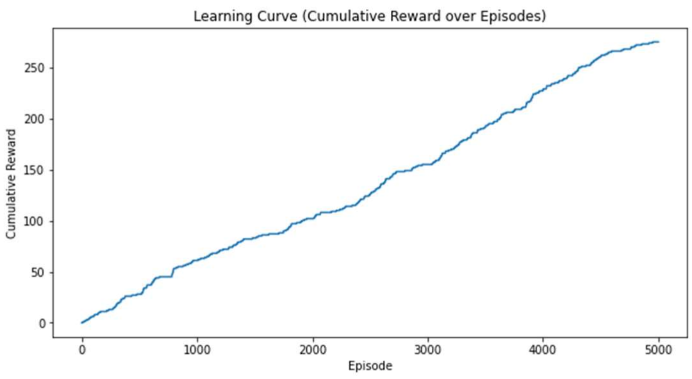
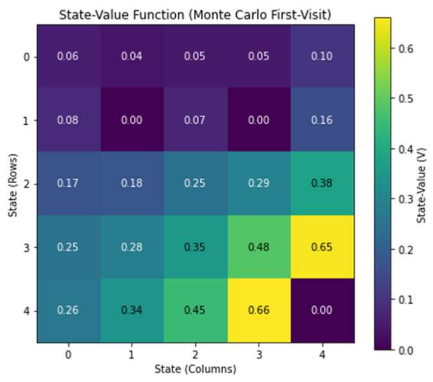


Figure 4: 5x5 Random Policy Learning Curve

The steady upward trend indicates that, despite the randomness of the policy, the agent is gradually accumulating positive rewards over time, which likely occur when it reaches the goal in the Frozen Lake environment. While the random policy is not optimized for reward maximization, the plot demonstrates that through repeated episodes, the agent does succeed in reaching the goal often enough for rewards to steadily increase.

State-Value Function (Monte Carlo First-Visit)

The third plot, "State-Value Function (Monte Carlo First-Visit)", provides a heatmap representation of the state-value estimates for each position in the 5x5 grid, with brighter colors indicating higher estimated values.

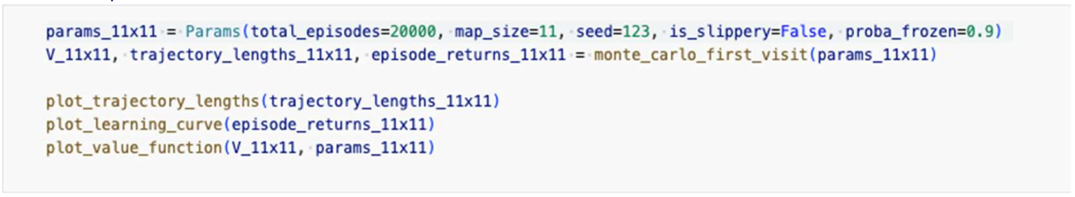


State (Rows)	0	1	2	3	4
0	0.06	0.04	0.05	0.05	0.10
1	0.08	0.00	0.07	0.00	0.16
2	0.17	0.18	0.25	0.29	0.38
3	0.25	0.28	0.35	0.48	0.65
4	0.26	0.34	0.45	0.66	0.00

Figure 5: 5x5 Random Policy State-Value Function

Higher values tend to be concentrated near the goal state (bottom-right corner), reflecting that these states, on average, yield higher returns due to their proximity to the goal. This map shows that the random policy enables the agent to explore the state space sufficiently to learn approximate values across the grid. However, without a directed policy, the values are relatively low across the board, except for states close to the goal where the chance of a successful path is higher.

11x11 Map Evaluation



```
params_11x11 = Params(total_episodes=20000, map_size=11, seed=123, is_slippery=False, proba_frozen=0.9)
V_11x11, trajectory_lengths_11x11, episode_returns_11x11 = monte_carlo_first_visit(params_11x11)

plot_trajectory_lengths(trajectory_lengths_11x11)
plot_learning_curve(episode_returns_11x11)
plot_value_function(V_11x11, params_11x11)
```

Figure 6: 11x11 Random Policy Map Parameters

The only parameter we will change, is **total_episodes=20000** and **map_size=11**. We do this, so the agent can have more episodes to explore a bigger map.

Comparison of the length of Trajectories per Episode to 5x5

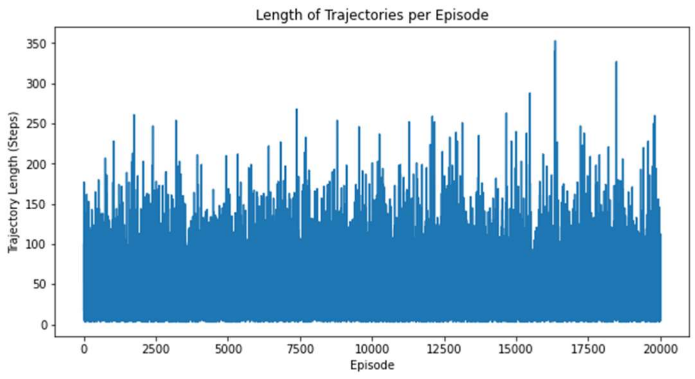


Figure 7: 11x11 Random Policy Length of Trajectories per Episode

We see a significant difference between the 11x11 and the 5x5 map. In the 11x11 environment, the trajectory lengths are generally longer, with more episodes reaching high step counts, sometimes exceeding 300 steps. This is due to the larger grid, which increases the number of states the agent must navigate, making it more likely to wander extensively before reaching a terminal state. In contrast, the 5x5 environment has a more constrained space, so episodes tend to have shorter trajectories. The larger map requires the agent to explore a more complex environment, often resulting in longer paths to completion.

Comparison of the Learning Curve to 5x5

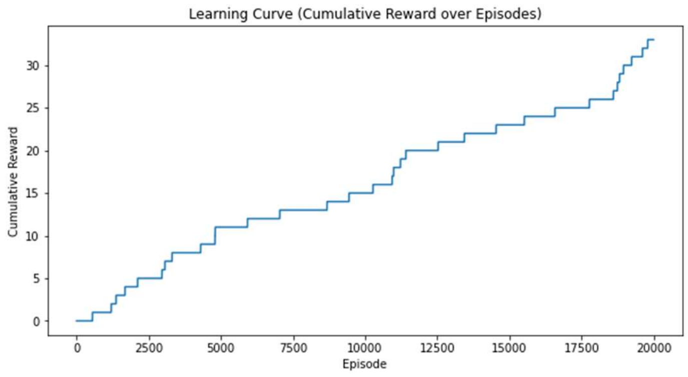


Figure 8: 11x11 Random Policy Learning Curve

The growth in cumulative reward is much slower in the 11x11 environment compared to the 5x5. With 20'000 episodes, the agent accumulates only around 30 cumulative rewards, whereas in the 5x5 environment, it accumulated over 250 in 5'000 episodes. This slower accumulation reflects the increased difficulty in successfully reaching the goal in the larger environment with a random policy.

Comparison of the State-Value Function to 5x5

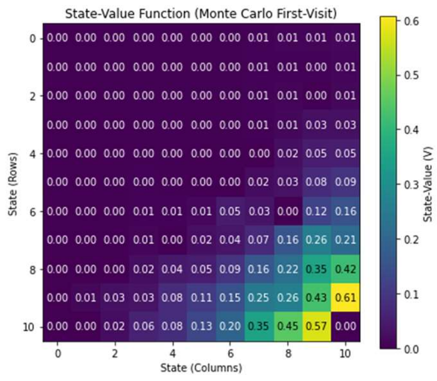


Figure 9: 11x11 Random Policy State-Value Function

The larger grid results in fewer successful episodes reaching states farther from the goal, leading to limited updates for these states and overall lower values. In comparison, the 5x5 environment shows a more extensive spread of higher values because the smaller grid allows for more frequent visits to states across the map, improving the average returns observed throughout the space.

Conclusion

In this experiment, we implemented a random policy to estimate the state-value function on two different grid sizes, 5x5 and 11x11, using a Monte Carlo first-visit approach.

Is the agent able to estimate the value-function correctly?

The agent is able to approximate the value function to some extent, but the accuracy of these estimates is limited, especially in the larger 11x11 grid. In the 5x5 environment, where the agent explores the entire state space more frequently, it develops a reasonable approximation of the state values, particularly in states closer to the goal.

In the 11x11 environment, however, the value estimates are less accurate and more clustered around lower values. This is due to the increased number of states, which limits the frequency with which each state is visited under a random policy. With fewer successful trajectories reaching the goal, many states are updated infrequently or inconsistently, leading to an incomplete and less accurate estimate of the value function.

Thus, while the agent can approximate the value function in a smaller environment, the accuracy and reliability of this approximation decreases significantly in larger environments with a random policy.

Is the agent able to learn the right policy?

No, the agent is not able to learn the right policy with a random action-selection strategy. Since it follows a purely random policy, the agent lacks any mechanism to prioritize actions that might lead it closer to the goal. In both grid sizes, the random policy results in aimless exploration without guidance, leading to long, often inefficient trajectories, and a low probability of consistently reaching the goal.

In the 5x5 environment, the agent does occasionally reach the goal, which allows for a modest accumulation of rewards, but it still lacks any directional preference or strategic planning.

In summary, with a random policy, the agent can approximate the value function to a limited extent, especially in smaller environments, but it does not learn an effective policy to reach the goal efficiently.

Incremental Monte Carlo

Parameters

In addition to the random policy, we will add three hyperparameters.

gamma

The discount factor controls how future rewards are weighted compared to immediate rewards. A value close to 1, for instance gamma=0.99, gives near-equal weight to future rewards.

The gamma parameter helps the agent to evaluate returns by considering future rewards along with immediate ones. This is crucial for environments where the agent may need to take multiple steps before reaching a reward, as it encourages planning by assigning value to longer paths that may ultimately yield higher returns.

epsilon

This parameter dictates the exploration-exploitation balance in the epsilon-greedy policy. A higher epsilon encourages more random actions, while a lower epsilon emphasizes exploiting known high-value actions.

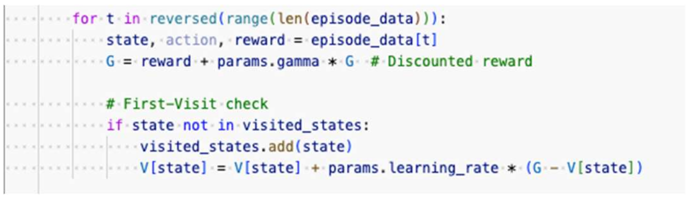
learning_rate

The learning rate controls the magnitude of each update to the state-value function. This incremental approach uses learning_rate to make gradual adjustments based on new returns observed from episodes.

The incremental Monte Carlo method applies each new experience as a small update to the current value estimate, rather than averaging returns as in non-incremental methods. A higher learning rate accelerates updates but risks introducing instability, while a lower rate yields smoother but slower convergence.

First Visit vs. Multiple Visit

Similar to our random policy, we use a first-visit approach in the incremental method to ensure comparable results between the two.

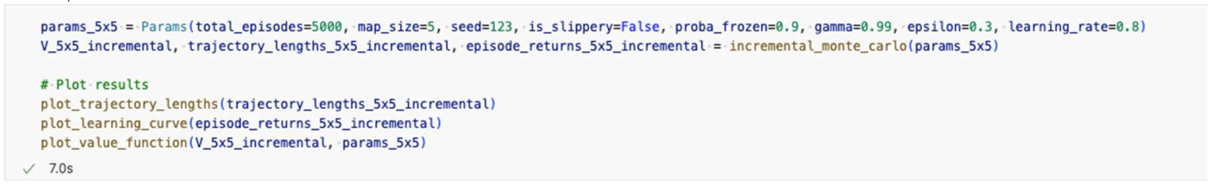


```
        for t in reversed(range(len(episode_data))):
            state, action, reward = episode_data[t]
            G = reward + params.gamma * G  # Discounted reward

            # First-Visit check
            if state not in visited_states:
                visited_states.add(state)
                V[state] = V[state] + params.learning_rate * (G - V[state])
```

Figure 10: Implementation First-Visit in Incremental Monte Carlo

Hyperparameter Optimization

5x5 Map



```
params_5x5 = Params(total_episodes=5000, map_size=5, seed=123, is_slippery=False, proba_frozen=0.9, gamma=0.99, epsilon=0.3, learning_rate=0.8)
V_5x5_incremental, trajectory_lengths_5x5_incremental, episode_returns_5x5_incremental = incremental_monte_carlo(params_5x5)

# Plot results
plot_trajectory_lengths(trajectory_lengths_5x5_incremental)
plot_learning_curve(episode_returns_5x5_incremental)
plot_value_function(V_5x5_incremental, params_5x5)
```

✓ 7.0s

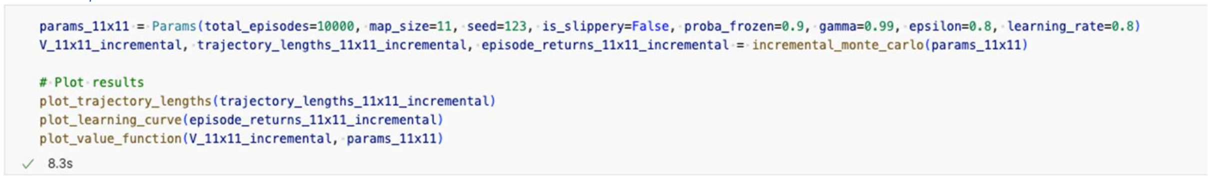
Figure 11: 5x5 Incremented Monte Carlo Hyperparameter Optimization

With **gamma=0.99**, the agent places a high value on future rewards, encouraging it to consider long-term benefits rather than focusing solely on immediate outcomes.

An **epsilon=0.3** allows the agent to balance exploration and exploitation; while it tends to favor actions that lead to higher-valued states, there is still a 30 % chance of taking a random action, which helps avoiding local optima and ensures broad state coverage.

The **learning_rate=0.8** enables the agent to adapt its value estimates quickly based on new experiences, accelerating the learning process, especially in the initial episodes.

11x11 Map



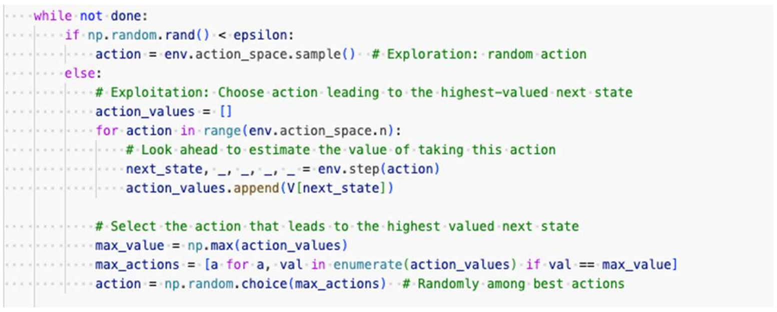
```
params_11x11 = Params(total_episodes=10000, map_size=11, seed=123, is_slippery=False, proba_frozen=0.9, gamma=0.99, epsilon=0.8, learning_rate=0.8)
V_11x11_incremental, trajectory_lengths_11x11_incremental, episode_returns_11x11_incremental = incremental_monte_carlo(params_11x11)

# Plot results
plot_trajectory_lengths(trajectory_lengths_11x11_incremental)
plot_learning_curve(episode_returns_11x11_incremental)
plot_value_function(V_11x11_incremental, params_11x11)
```

✓ 8.3s

Figure 12: 11x11 Incremental Monte Carlo Hyperparameter Optimization

We increased **epsilon=0.8** to encourage greater exploration in the larger environment. With a higher epsilon value, the agent is more likely to choose random actions (80 % of the time), which allows it to explore a wider range of states across the much larger state space of the 11x11 grid.

Epsilon-Greedy Functionality



```
    while not done:
        if np.random.rand() < epsilon:
            action = env.action_space.sample()  # Exploration: random action
        else:
            # Exploitation: Choose action leading to the highest-valued next state
            action_values = []
            for action in range(env.action_space.n):
                # Look ahead to estimate the value of taking this action
                next_state, _, _, _ = env.step(action)
                action_values.append(V[next_state])

            # Select the action that leads to the highest valued next state
            max_value = np.max(action_values)
            max_actions = [a for a, val in enumerate(action_values) if val == max_value]
            action = np.random.choice(max_actions)  # Randomly among best actions
```

Figure 13: Implementation of Epsilon-Greedy in Incremental Monte Carlo

Exploration

With probability epsilon, the agent explores by choosing a random action. This encourages the agent to visit a variety of states, which is especially important in large environments where it risks overlooking parts of the state space if it only exploits. Higher exploration allows the agent to gather diverse experiences, aiding in the discovery of potential paths to the goal.

Exploitation

When the random probability is greater than epsilon, the agent exploits its current knowledge by selecting the action that leads to the highest estimated value in the next state. This choice is based on the current value function, favoring actions that have previously shown higher returns.

5x5 Map Evaluation

Compare the length of Trajectories per Episode to the 5x5 with Random Policy

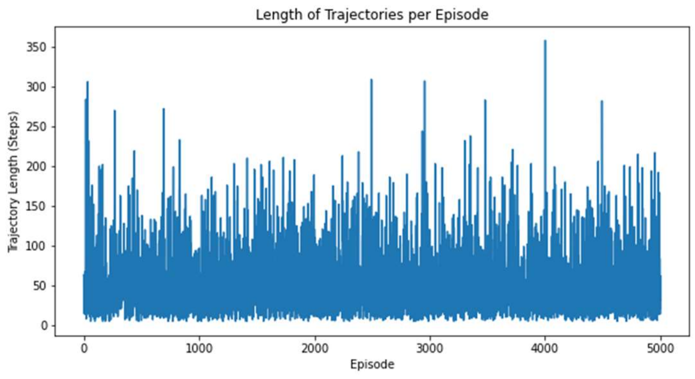


Figure 14: 5x5 Incremental Monte Carlo Length of Trajectories per Episode

The trajectory lengths for the incremental Monte Carlo method in the 5x5 environment appear more consistent and controlled compared to the random policy approach. While both methods exhibit a degree of variability in trajectory length, the incremental Monte Carlo agent, with its epsilon-greedy policy, has fewer episodes with very short or very long trajectories. This difference suggests that the incremental approach allows the agent to better navigate the grid, as it increasingly exploits learned values rather than purely exploring. In contrast, the random policy led to a wider spread of trajectory lengths, reflecting its lack of direction and dependence solely on chance.

Compare the Learning Curve to the 5x5 with Random Policy

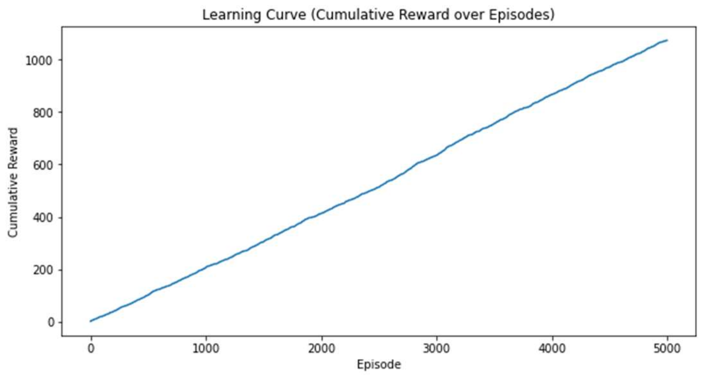


Figure 15: 5x5 Incremental Monte Carlo Learning Curve

The incremental Monte Carlo method shows a much faster and more substantial accumulation of rewards than the random policy. Over 5’000 episodes, the cumulative reward for the incremental agent reaches around 1’000, whereas the random policy’s cumulative reward was much lower over the same number of episodes. This marked improvement is due to the incremental agent's use of an epsilon-greedy policy, which allows it to gradually exploit higher-value states while still exploring.

This behavior enables it to reach the goal more frequently, leading to a steeper and more consistent rise in cumulative rewards

Compare the State-Value Function to the 5x5 with Random Policy

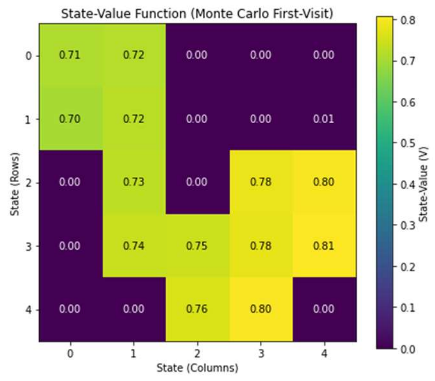


Figure 16: 5x5 Incremental Monte Carlo State-Value Function

The plot "State-Value Function (Monte Carlo First-Visit)" highlights the substantial difference in state-value estimates between the two approaches. With the incremental Monte Carlo method, the values near the goal are significantly higher, and the value function is well-defined across the grid. This shows that the agent has learned to differentiate between states based on their proximity to the goal.

The incremental approach's use of gamma, epsilon, and learning_rate has allowed the agent to learn and update its state-value estimates more effectively, resulting in a more accurate representation of the environment's structure and optimal paths.

11x11 Map Evaluation

Comparison of the length of Trajectories per Episode

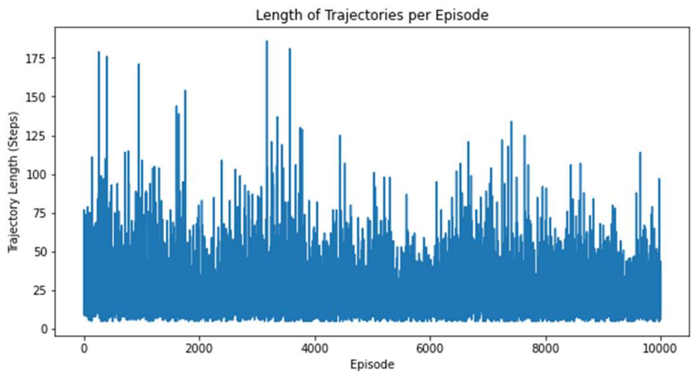


Figure 17: 11x11 Incremental Monte Carlo Length of Trajectories per Episode

The 11x11 incremental Monte Carlo setup shows a higher variability in trajectory lengths compared to both the 5x5 incremental and the 11x11 random policy. While the epsilon-greedy policy allows the agent to prioritize known high-value actions, the larger grid still leads to numerous long episodes, as the agent must navigate more states to reach a terminal state. The 5x5 incremental approach, in contrast, had shorter, more consistent trajectory lengths, reflecting the ease of navigating a smaller state space. Compared to the 11x11 random policy, the incremental approach in 11x11 is somewhat more controlled, though both exhibit episodes with extended lengths due to the vast grid size.

Comparison of the Learning Curve

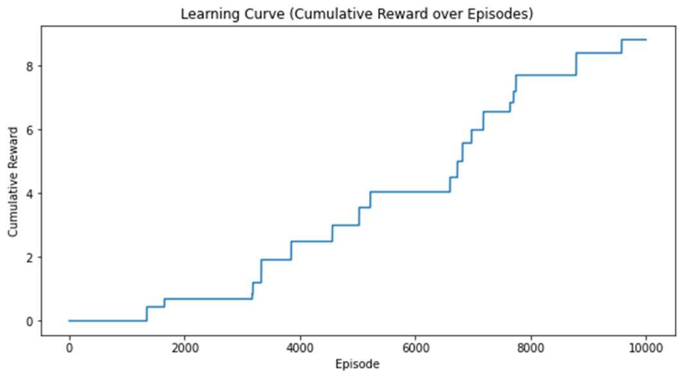


Figure 18: 11x11 Incremental Monte Carlo Learning Curve

We observe that the cumulative reward in the 11x11 incremental setup grows very slowly compared to the 5x5 incremental case, indicating that the agent reaches the goal much less frequently in the larger grid. However, it still performs better than the 11x11 random policy, where the cumulative reward barely rises due to the lack of a directed exploration mechanism. The epsilon-greedy policy in the 11x11 incremental approach allows for occasional exploitation of learned values, leading to a gradual increase in cumulative rewards, though it is still far slower than the steady growth seen in the 5x5 incremental approach.

Comparison of the State-Value Function

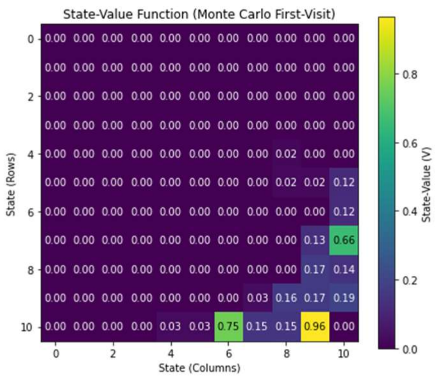


Figure 19: 11x11 Incremental Monte Carlo State-Value Function

The incremental approach in the 11x11 environment produces a value function with higher values concentrated near the goal, though with limited spread across the grid. This is an improvement over the 11x11 random policy, where the value function was minimally informative and showed little differentiation across states. However, the value distribution in the 11x11 incremental setup is still much sparser and less defined than in the 5x5 incremental environment, where values closer to the goal gradually increased in a more continuous gradient. The 11x11 incremental results indicate that the agent can start learning high-value paths near the goal, but the large state space limits its ability to fully propagate these values across the grid.

Conclusion

Is the agent able to estimate the value function correctly?

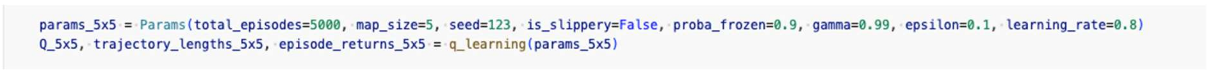
In the incremental Monte Carlo approach, the agent demonstrates a moderate ability to estimate the value function, particularly in smaller environments like the 5x5 grid. In this setting, the agent builds a clearer representation of state values, showing a gradual gradient toward the goal, which reflects its understanding of which states are more advantageous. This approximation is reasonably accurate due to the epsilon-greedy policy, which enables a balance between exploring new states and exploiting known high-value states. However, in the larger 11x11 grid, the agent's ability to estimate the value function diminishes. While it does recognize higher values near the goal, the vast state space and infrequent successful episodes prevent it from learning a fully accurate value distribution across all states.

Is the agent able to learn the right policy?

The incremental Monte Carlo method enables the agent to move toward a better policy than a random approach, especially in the 5x5 environment. With the epsilon-greedy policy, the agent is able to balance exploration and exploitation, occasionally prioritizing actions that lead to higher-value states. This allows it to develop a more directed policy that reaches the goal more frequently in the smaller grid. However, in the 11x11 environment, while the agent performs better than with a purely random policy, it still struggles to learn an optimal policy.

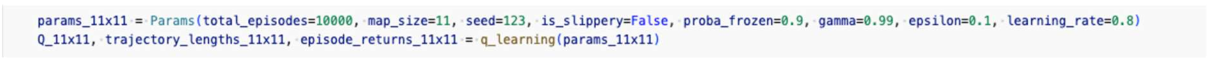
Applying Q-Learning

Hyperparameter Optimization



```
params_5x5 = Params(total_episodes=5000, map_size=5, seed=123, is_slippery=False, proba_frozen=0.9, gamma=0.99, epsilon=0.1, learning_rate=0.8)
Q_5x5, trajectory_lengths_5x5, episode_returns_5x5 = q_learning(params_5x5)
```

Figure 20: map_size=5 parameter selection



```
params_11x11 = Params(total_episodes=10000, map_size=11, seed=123, is_slippery=False, proba_frozen=0.9, gamma=0.99, epsilon=0.1, learning_rate=0.8)
Q_11x11, trajectory_lengths_11x11, episode_returns_11x11 = q_learning(params_11x11)
```

Figure 21: map_size=11 parameter selection

With epsilon=0.1, the agent will choose the action with the highest estimated value 90 % of the time, helping it follow the best-known paths to maximize rewards. This is beneficial in the Q-learning setup, where the agent is actively updating the Q-values based on past experiences and needs to exploit this learned information to improve its performance over time.

To ensure that the agent does not stop exploring, a 10 % chance remains. This small amount of exploration helps it avoid getting stuck in suboptimal policies and still allows it to discover potentially better actions or paths, especially in uncertain or partially learned areas of the environment.

Comparison of the length of Trajectories per Episode

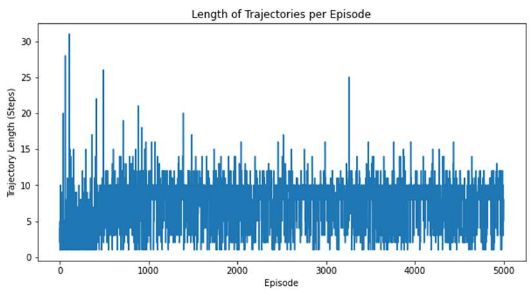


Figure 22: 5x5 Length of Trajectories per Episode

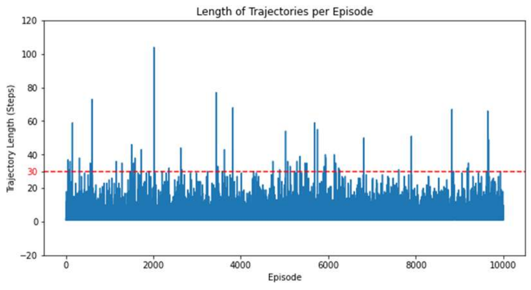


Figure 23: 11x11 Length of Trajectories per Episode

In the comparison of trajectory lengths for the 5x5 and 11x11 environments, we see a marked difference in the agent's behavior and the environment's complexity. In the 5x5 grid (left plot), trajectory lengths start with some variability but quickly stabilize to shorter episodes as the agent learns more efficient paths to reach the goal or a terminal state. This pattern suggests that the smaller state space allows the agent to converge on an effective strategy relatively quickly, leading to consistently shorter episodes.

In contrast, the 11x11 grid (right plot) initially exhibits much longer and more variable trajectory lengths, reflecting the larger state space and increased exploration needed. Over time, trajectory lengths decrease as the agent starts to find shorter paths, but they remain longer on average than in the 5x5 grid, indicating that the larger environment continues to pose a greater challenge for efficient navigation even after significant training.

Comparison of the Learning Curve

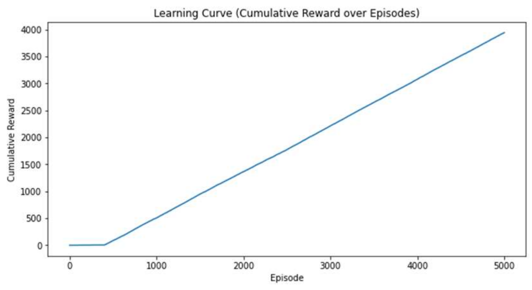


Figure 24: 5x5 Learning Curve

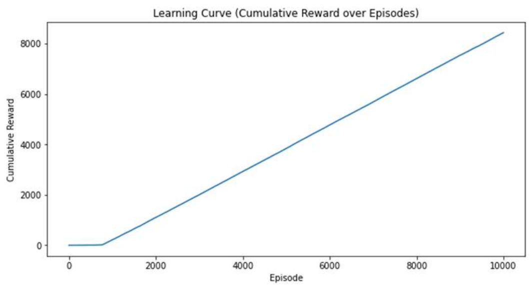


Figure 25: 11x11 Learning Curve

The learning curves for the 5x5 and 11x11 environments reveal differences in how quickly and effectively the agent accumulates rewards in each setting. In the 5x5 grid (left plot), the cumulative reward rises steadily and consistently, reaching around 4'000 by the end of 5'000 episodes. This linear growth indicates that the agent is able to learn and reinforce efficient paths to the goal, allowing it to accumulate rewards at a stable rate. In contrast, the cumulative reward in the 11x11 environment (right plot) also grows linearly but at a slower pace, reaching approximately 8'000 over 10'000 episodes. The slower growth in the larger environment reflects the increased complexity and difficulty in navigating a larger state space.

Comparison of the State-Value Function

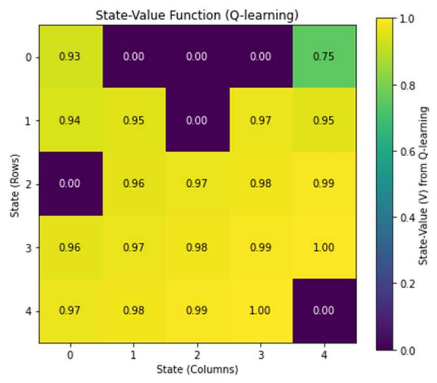


Figure 26: 5x5 State-Value Function

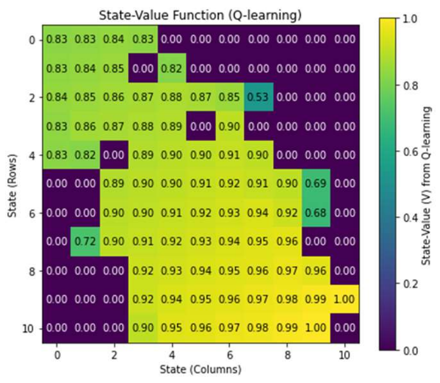


Figure 27: 11x11 State-Value Function

In the 5x5 grid (left plot), the state-value function is well-defined, with high values near the goal and a gradual gradient extending from the goal to surrounding states. This pattern shows that the agent has learned an effective policy, accurately estimating the values of states based on their proximity to the goal. The values are uniformly high across most reachable states, suggesting consistent and efficient navigation to the goal.

In the 11x11 grid (right plot), the state-value function also shows higher values around the goal, but the distribution is less uniform across the grid. Although the agent still assigns higher values to states near the goal, there are more pockets of lower values, especially in distant regions. This pattern suggests that the agent has not fully explored or learned all possible optimal paths, likely due to the increased complexity and size of the state space.

Conclusion

Is the agent able to estimate the value function correctly?

Yes, the agent is able to estimate the value function reasonably well with Q-learning, especially in the 5x5 environment. The Q-learning approach provides a clearer gradient of values leading to the goal, showing that the agent has developed an accurate understanding of state values relative to their proximity to the goal. In the larger 11x11 grid, the value function estimate is less consistent, with pockets of lower values in distant areas, indicating that the agent has some difficulty fully exploring the larger state space. However, even in this more complex environment, Q-learning provides a more reliable value estimate compared to both the random and incremental Monte Carlo policies.

Is the agent able to learn the right policy?

The agent effectively learns a near-optimal policy in the 5x5 environment using Q-learning, where it consistently finds efficient paths to the goal. This marks a substantial improvement over both the random policy, which has no structure, and the incremental Monte Carlo approach, which was slower and less directed in policy formation. In the 11x11 grid, while Q-learning produces a better policy than either the random or incremental Monte Carlo methods the agent sometimes struggles with the larger state space.

Discussion

In this assignment, three different reinforcement learning methods – random policy, incremental Monte Carlo, and Q-learning – were implemented and analyzed for their ability to estimate the value function and learn effective policies in a Frozen Lake environment. Each method showcases a different approach to the exploration-exploitation trade-off and offers unique insights into learning in environments with varying levels of complexity.

In summary, each method exhibits strengths and limitations. The random policy serves as a straightforward baseline, highlighting the inefficiencies of unguided action. Incremental Monte Carlo introduces a mechanism for gradual value estimation and policy improvement, showing the benefits of controlled exploration. Q-learning, with its rapid updates and focus on maximizing cumulative rewards, provides the most efficient approach among the three, demonstrating strong performance in both value estimation and policy learning. Each method illustrates a different aspect of reinforcement learning, from undirected action to structured updates, ultimately leading to more refined and goal-oriented behavior.