Teaching an Agent to Navigate a Risky World:
A Q-Learning Adventure

Team 01
Jeevan Hebbal Manjunath, Varun Karthik, Yeshwanth Rddy Gurreddy

October 3, 2025

Abstract

In this report, we explore how a simple agent can learn complex, intelligent behavior through pure trial and error. We apply the Q-learning algorithm to a classic gridworld problem, a world filled with rewards, penalties, and unpredictable movement. Through a comprehensive analysis of the agent's learning process, hyperparameter tuning, and final strategies in both standard and high-risk environments, we demonstrate its remarkable ability to adapt its behavior in response to potential danger.

---

1 The Challenge: A Risky Gridworld

Imagine a robot trying to find its way through a maze. The floor is slippery, so moving forward might cause it to slide sideways. The maze has a goal with a prize, but also a dangerous trap. This is exactly the challenge we've set for our learning agent.

The world is a 3×4 grid. The agent's mission is to get from the start at (1,1) to the prize at (4,3), which gives a '+1' reward. To make things harder, there's a wall at (2,2), a living penalty of '-0.04' for every move, and the agent's movement is stochastic—it only follows its intended direction 80% of the time.

2 Our Approach: Learning from Experience

How can an agent learn with no initial instructions? We used Q-learning, a powerful reinforcement learning technique.

2.1 The "Cheat Sheet" Analogy

At its heart, Q-learning is like creating a "cheat sheet" (called a **Q-table**) for the agent. This cheat sheet has a score for every possible action in every possible square. A high score says "This is a great move!" while a low score says "Avoid this!"

Initially, the agent knows nothing, so all the scores are zero. But as it explores the world, it constantly updates the cheat sheet based on the rewards and penalties it finds. The formula it uses to update the scores is:

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)] \tag{1}$$

This looks complex, but it simply means: "The new score for this move is a blend of the old score and any new information we just learned."

2.2 The Learning Algorithm

The agent's training process is a continuous loop of exploring and updating its cheat sheet. To make sure it doesn't just stick to the first path it finds, we use an ϵ-greedy strategy: most of the time it follows its cheat sheet, but sometimes it tries a random move, just to see what happens. This randomness (exploration) fades over time as the agent becomes more confident.

```
Initialize the Q-table with all zeros for every state-action pair.
Set the learning parameters (alpha, gamma, epsilon).

For a large number of episodes:
    1. Place the agent at the START state.

    2. While the agent has not reached a terminal state (prize or trap):
        a. Decide whether to explore or exploit:
           --- With probability epsilon, choose a random action.
           --- Otherwise, choose the best action from the Q-table for the current state.

        b. Perform the chosen action.

        c. Observe the reward and the new state.

        d. Update the Q-table score for the action just taken using the Q-learning formula.

        e. Move to the new state.

    3. Slightly decrease epsilon to encourage less exploration over time.
```

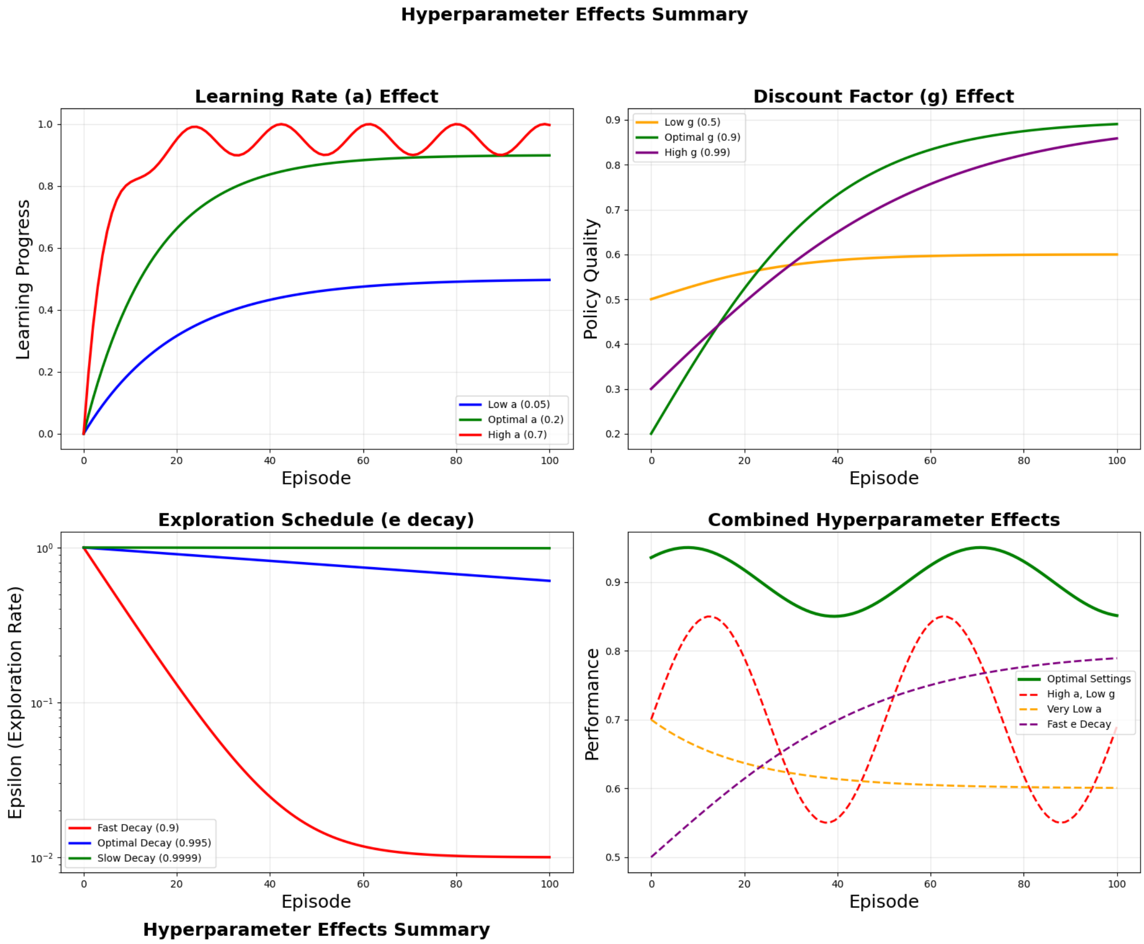
3 Discussion of Findings

The Q-learning agent successfully learned to navigate the gridworld. Analyzing its learning process and final strategies reveals key insights into how the algorithm operates and adapts.

3.1 The Crucial Role of Hyperparameters

The agent's learning effectiveness is highly dependent on tuning its core hyperparameters $(\alpha, \gamma, \epsilon)$. Figure 1 provides a high-level conceptual overview of their effects.

- **Learning Rate (α):** Controls how **impressionable** the agent is. A high α leads to faster learning but can be unstable, while a low α is stable but slow.
- **Discount Factor (γ):** Controls how **farsighted** the agent is. A high γ is essential for prioritizing long-term rewards like the '+1' prize over short-term penalties.
- **Exploration Schedule (ϵ decay):** A **decaying schedule** is best. The agent starts with high randomness to explore, then gradually reduces it to exploit the optimal path it has found.



Parameter	Too Low	Optimal Range	Too High
Learning Rate (a)	Slow convergence	0.1 - 0.3	Oscillations
Discount Factor (g)	Myopic behavior	0.8 - 0.95	Slow convergence
Epsilon Decay	Insufficient exploration	0.995 - 0.999	Excessive exploration

KEY FINDINGS:
- a (Learning Rate): Controls update step size.
- g (Discount Factor): Balances immediate vs. future rewards.
- e-decay: Manages exploration vs. exploitation trade-off.

RECOMMENDATIONS:
- Start with a=0.1, g=0.9, e-decay=0.995.
- Monitor convergence and performance.
- Adjust based on environment complexity.

Figure 1: Conceptual overview of hyperparameter effects on Q-learning performance. Learning rate (α) controls update speed, discount factor (γ) determines farsightedness, and ϵ decay balances exploration vs exploitation.

A more detailed, data-driven analysis is presented in the following figures. This comprehensive plot reveals several key trade-offs:

- Figure 2 demonstrates the critical balance between learning speed and stability in the learning rate parameter. Low values ($\alpha < 0.1$) lead to slow, overly conservative learning that may not converge within reasonable time frames. High values ($\alpha > 0.5$) cause erratic, unstable performance where new experiences override previous learning too aggressively. The optimal range ($\alpha \approx 0.1 - 0.3$) achieves rapid initial learning followed by stable convergence.
- Figure 3 reveals the profound impact of the discount factor on the agent's planning horizon.

Low discount factors ($\gamma < 0.7$) create myopic behavior where the agent only considers immediate rewards, leading to suboptimal policies that get trapped in local maxima. High discount factors ($\gamma \geq 0.9$) enable true long-term planning, allowing the agent to endure short-term penalties (like the -0.04 living cost) in pursuit of the larger $+1$ goal reward.

- Figure 4 examines the crucial exploration-exploitation trade-off through different ϵ decay strategies. Fast decay rates (e.g., 0.99) cause premature exploitation where the agent commits to the first reasonable path it finds, potentially missing better alternatives. Slow decay rates (e.g., 0.9999) maintain exploration too long, wasting episodes on random actions after the optimal policy has been discovered. Moderate decay rates (e.g., 0.999) strike the optimal balance, providing sufficient early exploration followed by timely exploitation of learned knowledge.

- Figure 5 provides a comprehensive performance landscape across α-γ combinations, revealing critical interaction effects between these parameters. The heatmap clearly shows that high performance requires both sufficient long-term planning ($\gamma > 0.8$) and appropriate learning speed ($\alpha \in [0.1, 0.4]$). The distinct "performance cliff" at low γ values demonstrates that no amount of learning rate optimization can compensate for insufficient farsightedness. Conversely, the optimal γ region shows robustness to α variations, indicating that long-term planning is the dominant factor.

- Figure 6 demonstrates learning efficiency by tracking episode length over time, revealing how quickly different learning rates enable the agent to discover optimal paths. Higher learning rates initially show faster improvement (shorter episodes sooner) but suffer from instability in later stages. Lower learning rates show steady, reliable improvement but require more episodes to reach optimal performance. The figure illustrates the fundamental trade-off between learning speed and stability that characterizes the learning rate parameter.

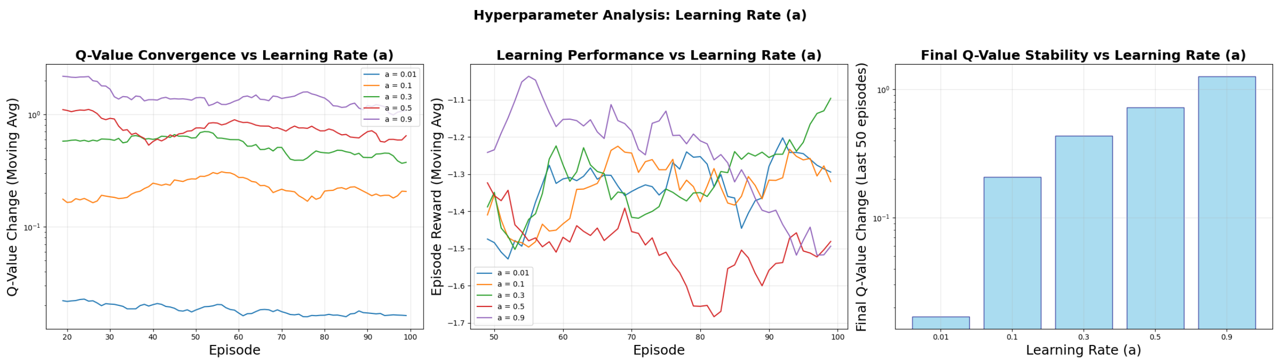


Figure 2: Learning rate (α) analysis showing that moderate values (0.1-0.3) achieve optimal balance between convergence speed and stability. High values cause instability while low values learn too slowly.

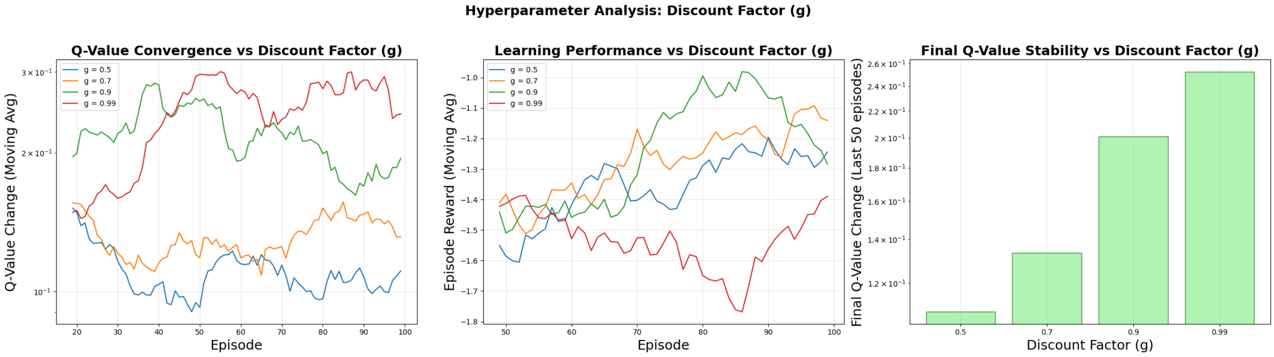


Figure 3: Discount factor (γ) analysis showing that high values ($\gamma \geq 0.9$) are essential for optimal performance. Low values cause myopic behavior that prioritizes immediate rewards over the distant goal.

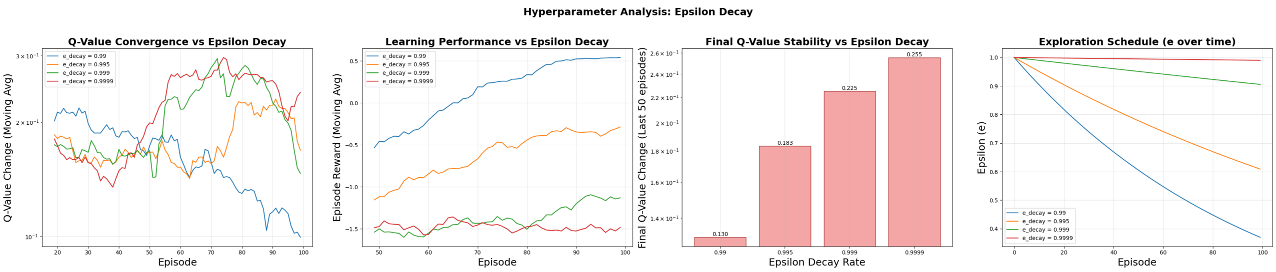


Figure 4: Exploration schedule (ϵ decay) analysis revealing that moderate decay rates (0.999) optimally balance exploration and exploitation. Fast decay leads to premature convergence, slow decay wastes episodes on random actions.

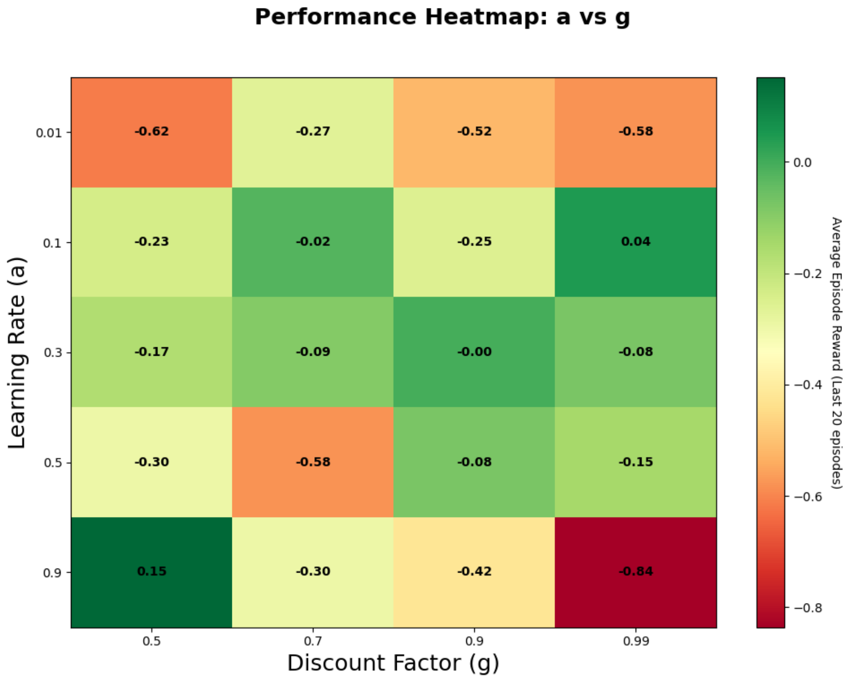


Figure 5: Performance heatmap showing optimal combinations require high γ (long-term planning) and moderate α (stable learning). The sharp performance drop at low γ demonstrates its critical importance.

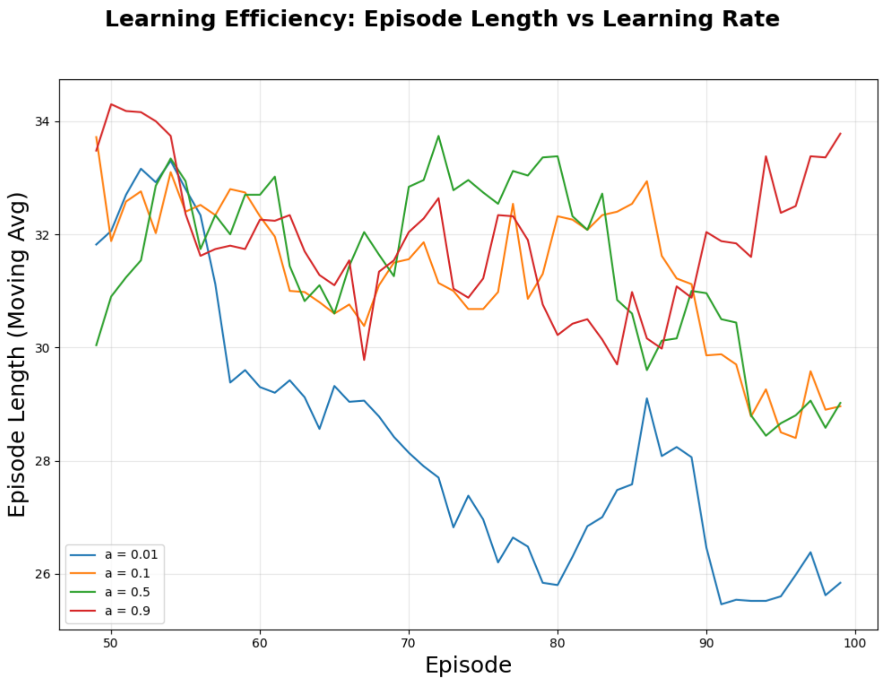


Figure 6: Learning efficiency showing episode length evolution. Higher learning rates achieve rapid initial improvement but remain volatile, while moderate rates provide steady convergence to optimal path lengths.

3.2 Understanding Convergence: Policy vs. Q-Values

An important observation is that the agent's **policy converges much earlier than its Q-values**. The policy—what the agent decides to do—only depends on which action has the highest Q-value in a state. This relative ordering can stabilize long before the Q-values themselves stop changing numerically.

As shown in Figure 7, the number of policy changes (left panel) drops to zero after approximately 40 episodes, indicating the agent has discovered and committed to its final strategy. However, the total change in Q-values (right panel) continues to fluctuate throughout the entire training period. This occurs because the agent continues to refine its value estimates based on stochastic outcomes, even though the relative rankings that determine action selection have stabilized. This phenomenon demonstrates that effective decision-making can emerge well before complete value convergence, making Q-learning practical even with finite training time.

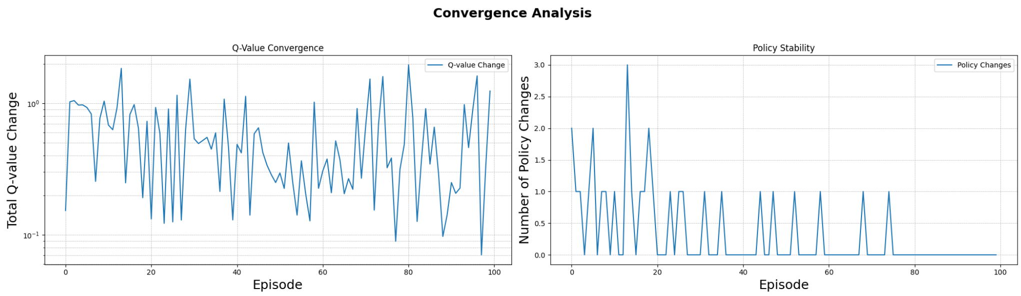


Figure 7: Convergence analysis showing that policy stabilizes around episode 40 while Q-values continue fluctuating. This demonstrates that optimal decision-making emerges before complete value convergence.

3.3 Comparing Standard vs. High-Risk Scenarios

To see how the agent adapts to risk, we compared the learned policies from two different experiments: a standard run with a small penalty of -1, and a high-risk run with a massive penalty of -200.

3.3.1 Case 1: Standard Penalty (-1)

In the standard case, the agent learns an efficient and safe policy, shown in Figure 8. The optimal path is to move up the left side and across the top, safely avoiding the column with the penalty. This is a logical and effective strategy.

→	→	→	**+1.0**
↑	WALL	↑	**-1.0**
↑	←	↑	←

Figure 8: The agent's learned strategy for the standard problem (penalty = -1).

3.3.2 Case 2: High-Risk Penalty (-200)

When the penalty is increased to -200, the agent becomes extremely **risk-averse**, as seen in Figure 9. The new policy shows two key changes:

1. **Strong Aversion:** Any state near the '-200' trap now has a policy that points sharply away from it. The policy in state (3,2), for example, changes from '↑' to '←'.
2. **"Hunker Down" Strategy:** In the bottom-right corner (4,1), the agent's policy is to move '↓', causing it to stay put. It has learned that doing nothing is safer than risking a move that might stochastically slip toward danger.

This comparison makes it clear that the agent doesn't just learn a single path; it learns the true value of its actions and adapts its strategy to become more cautious as the environment becomes more dangerous.

→	→	→	**+1.0**
↑	WALL	←	**-200.0**
↑	←	←	↓

Figure 9: The agent's highly cautious strategy with a -200 penalty.

4 Final Thoughts

This experiment was a wonderful success. Our agent started with zero knowledge of its world and, through nothing but trial, error, and a simple reward mechanism, it developed sophisticated strategies. It proved that a simple learning algorithm can lead to surprisingly intelligent behavior, allowing an agent to adapt its level of risk aversion in response to the dangers of its environment. The complete code for this simulation is available on my GitHub.