

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

Jeevan Hebbal Manjunath

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. By comparing the agent's learned 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 3x4 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.

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:

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

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 ability to learn effectively is highly dependent on the tuning of its core hyperparameters (α, γ, ϵ).

- **Learning Rate (α):** This controls how **impressionable** the agent is to new experiences. A high learning rate can lead to unstable learning, while a low rate can be too slow.
- **Discount Factor (γ):** This parameter controls how **farsighted** the agent is. A high value (e.g., 0.9) is essential here to motivate the agent to seek the long-term '+1' prize.
- **Exploration Schedule (ϵ):** A **decaying schedule** is best. The agent starts with high randomness to explore the grid, then gradually reduces it to exploit the optimal path it has found.

3.2 Convergence: Policy vs. Q-Values

In this problem, the **policy converges much earlier than the Q-values** do. The policy 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.

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 1. 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 1: 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 2. 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 2: 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](#).