REPORT

Mehmet Emin Meydanoğlu

090250756

Q-Learning Maze Solver

Summary

In this project, I developed a Q-Learning agent that can solve unseen mazes. The initial approach only memorized the training maze and completely failed on new ones (0% success). By iteratively improving the state representation, I achieved **93% success** on unseen mazes, without using any deep learning.

---

1. Q-Learning

Q-Learning is a reinforcement learning algorithm where the agent learns through trial and error. The agent records which action is best in each state in a table (Q-Table).

```
Q(state, action) += α × [reward + γ × max(Q(next_state)) - Q(state, action)]
```

This is the update formula. Here;

- **α (learning rate):** How fast to learn (0.1)
- **γ (discount):** How much to value future rewards (0.99)
- **ε (exploration):** How often to try randomly (initially 1.0, then decreases to 0.05)

---

2. Maze Design

Mazes are matrices containing 0, 1, 2, 3 where:

- **0:** Empty cell
- **1:** Obstacle (no pass)
- **2:** Start position of the agent

- **3:** Goal position

Every maze used by the agent both in training and in evaluation is guaranteed to be solvable. This is done by verifying the maze created with BFS algorithm before using it.

---

3. Agent Improvements

3.1 The Problem: A Memorizing Agent

When I first built the agent, I used coordinates (x, y) as states. The agent learned things like: "When I'm at row 3, column 5, I should go up." The agent was just memorizing the solution to one specific maze rather than learning how to navigate mazes in general.

When I tested this agent on new, randomly generated mazes, it achieved **0% success**. This is because the coordinates don’t give any true information about the obstacles, paths, or where the goal is.

3.2 Update 1: Vision

I replaced coordinates with perception-based states:

- **Wall detection:** Which of the 4 directions are blocked? (16 combinations)
- **Goal direction:** 8 compass directions (N, NE, E, SE, S, SW, W, NW)

State space: 16 × 8 = **128 states**

Result: 31% success. The agent started generalizing.

3.3 Update 2: Adding Memory

I noticed the agent was getting stuck in loops (local minima), repeatedly moving back and forth between two positions. It had no memory of where it had just been.

I fixed this by adding the agent's last action to its state representation. Now the agent knows: "I see these walls, the goal is northeast, and I just came from the left."

State space: 128 × 5 = **640 states** (5 = 4 directions + starting state)

3.4 Update 3: Reward Shaping

The original reward structure only gave feedback at the goal. Long horizon rewards make it hard for RL agents to learn what actions exactly gets the rewards. I added intermediate rewards to guide the agent more effectively:

- **+0.5** for moving closer to the goal
- **-0.3** for moving away from the goal

This helped the agent learn faster by providing immediate feedback on whether its actions were productive.

3.5 Update 4: Extended Vision

I extended the agent's vision from 1 square to 2 squares in each direction. This allows the agent to spot dead ends before entering them and plan ahead.

State space: 3,240 states

Test Results

Here are some results I have recorded while testing the updates. Agent was able to generalize up to %93 percent.

Version	States	Success	
Coordinate (Memorizer)	100	0%	
Vision	128	31%	
+Memory	640	73%	
+Reward Shaping	3,200	90%	
+Extended Vision	**3,240**	**93%**	

4. Code

```
q-learning-agent/
├── main.m          # trains or loads agent, runs tests
├── agents/         # Saved trained agents (.mat files)
└── src/
```

```
    ├── create_maze.m    # Maze generator with BFS path verification
    ├── encode_state.m   # State encoder (vision + direction + memory)
    └── q_learning.m     # Training loop with live visualization
```

Key Functions

create_maze.m: Generates random mazes. Obstacle density is configurable. Every maze is checked with BFS to make sure there's a valid path from start to goal.

encode_state.m: Takes what the agent sees and turns it into a single number (state index). It combines:

- Vision in 4 directions (open, blocked at distance 1, or blocked at distance 2)
- Goal direction (one of 8 compass directions)
- Last action (so the agent doesn't immediately reverse)

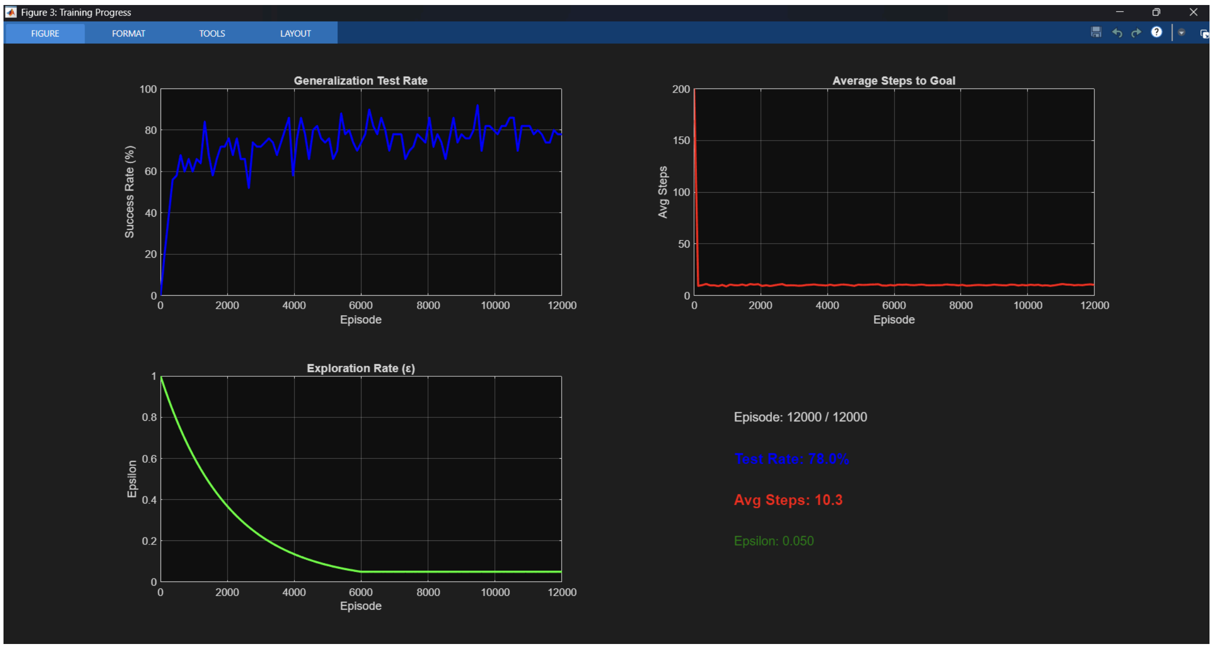
q_learning.m: Training loop. Each episode gets a fresh random maze. Agent explores with ε-greedy, Q-table gets updated. There's also a live plot that shows progress and runs periodic tests on unseen mazes.

main.m: Entry point. Either trains a new agent or loads a saved one. After that, it tests on random mazes and shows the results in a grid.

---

5. Example Training Results

Here is one training and evaluation log from a 10*10 maze setup, with 12.000 episodes. Agent was able to generalize near 80%, and had an average steps of 10.3, which means the agent is pretty efficient.

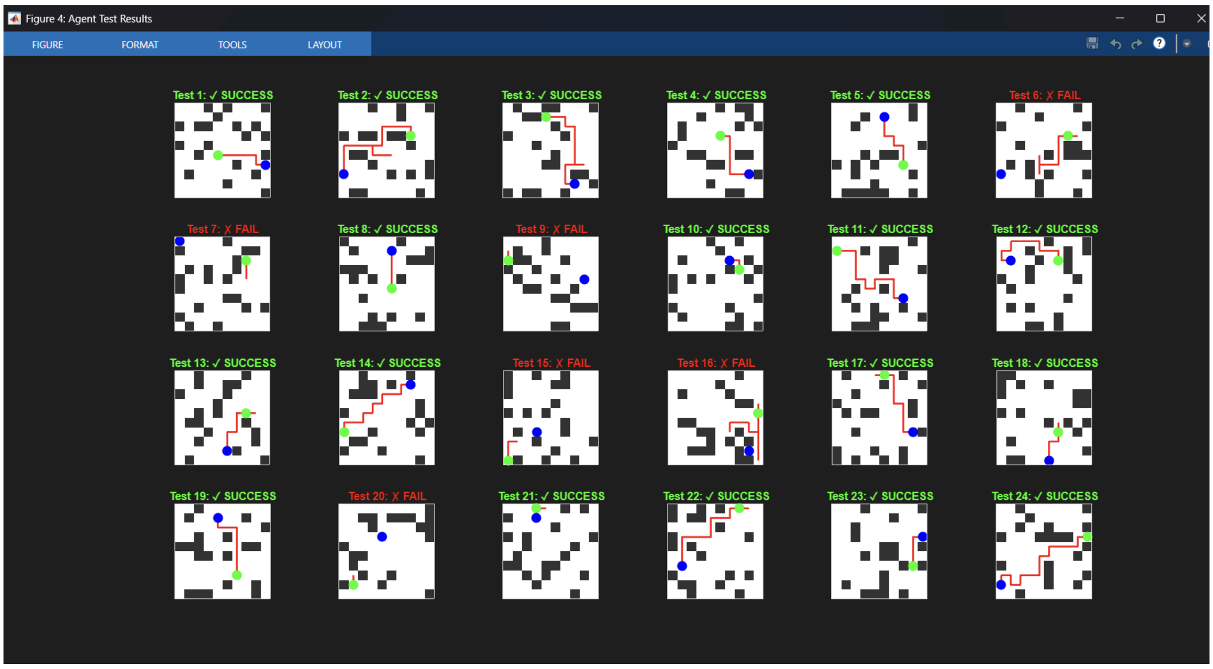


Episode: 12000 / 12000

Test Rate: 78.0%

Avg Steps: 10.3

Epsilon: 0.050



6. Takeaways

1. State representation matters more than the algorithm. I used the exact same Q-Learning algorithm throughout. The only thing I changed was how I represented states and rewards, and generalization rate moved from 0% to 93%.
2. Adding just the last action to the state gave a 42% improvement. It was enough to stop the agent from going back and forth between the same two

cells.