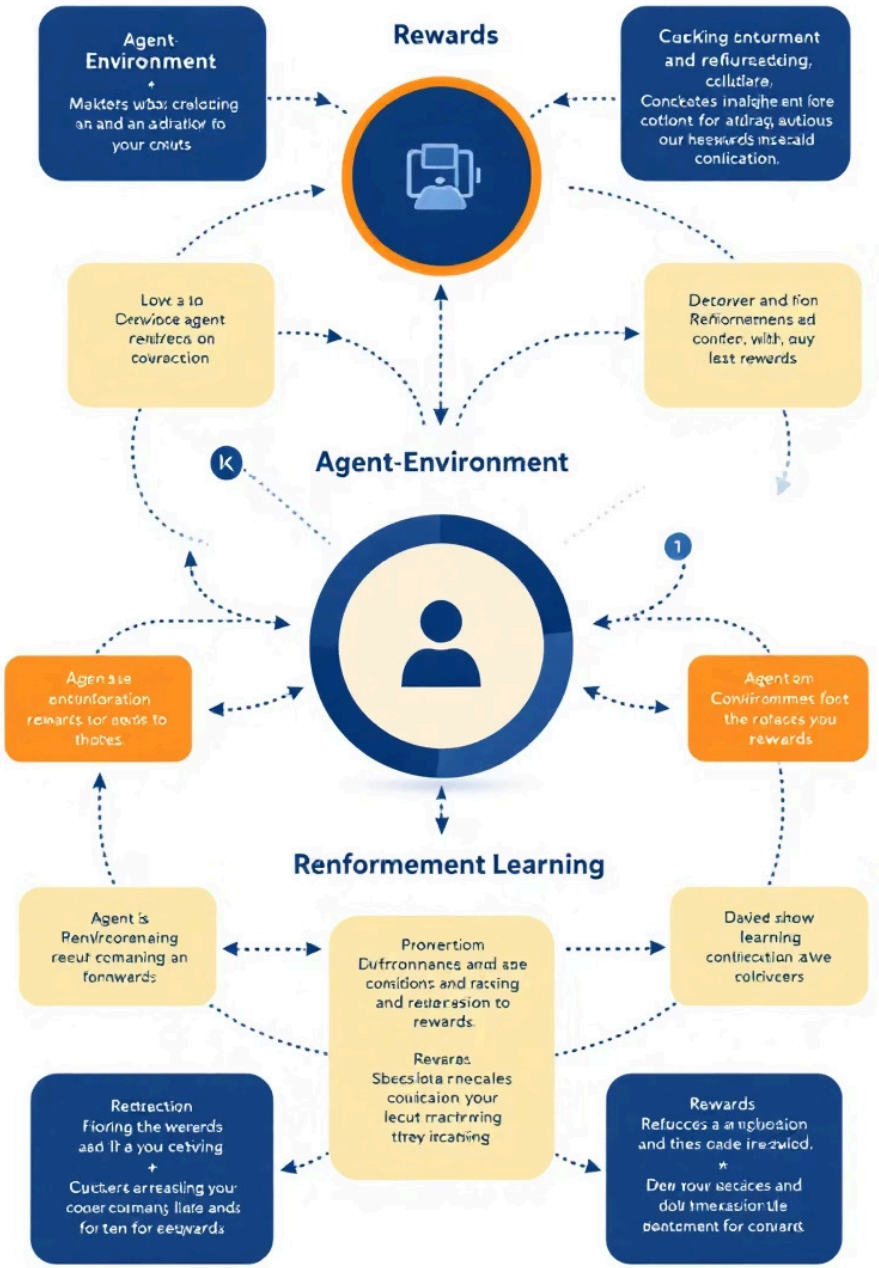
My First AI Project: Building a Reinforcement Learning Agent from Scratch

I'm thrilled to share my journey creating a Q-learning agent. This project marks my first hands-on AI experience.

My agent navigates a grid environment to reach goals. It learns optimal paths through exploration and feedback.

by Michal Uhrínek



Reinforcement Learning Fundamentals

Key Concepts

- Agent: The learner/decision maker
- Environment: World the agent interacts with
- State: Current situation
- Action: What the agent can do

Q-Learning Basics

- Learning optimal action values
- Balancing exploration vs. exploitation
- Incremental improvement through feedback

Reward System

The agent receives feedback through rewards. Higher rewards reinforce beneficial behaviors.

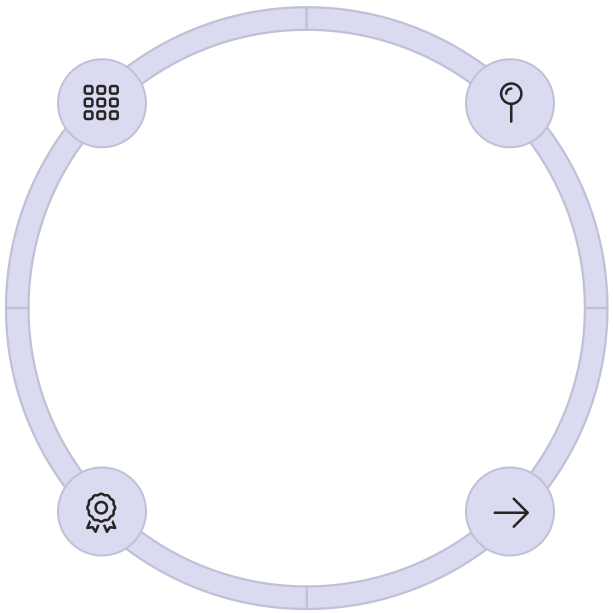
Project Architecture: The Grid Environment

Grid World

10×10 grid with customizable obstacles

Reward Structure

- +1 for reaching goal
- -0.1 for each step
- -1 for hitting obstacle

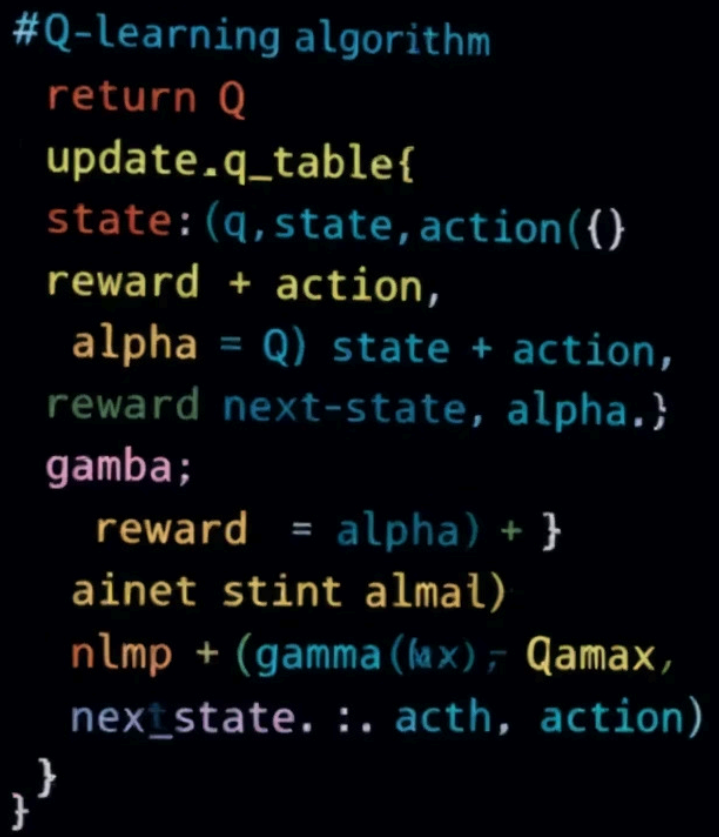


State Space

Represented as (row, column) coordinates

Action Space

Four possible moves: up, down, left, right



Q-Learning Implementation

Initialize Q-Table

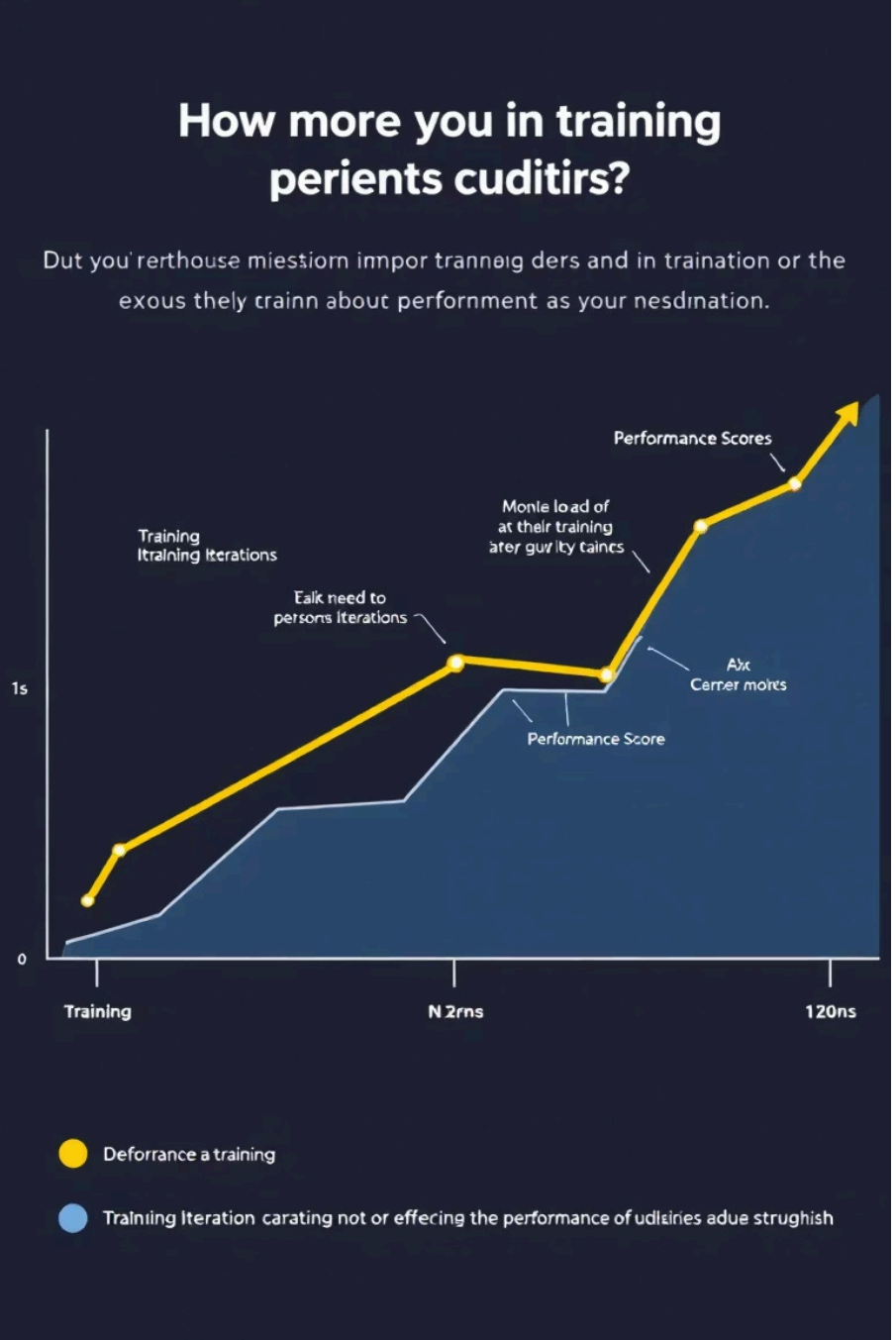
Set all Q-values to zero, creating a blank slate for learning

Q-Value Update Rule

$Q(s,a) = Q(s,a) + \alpha[R + \gamma \max Q(s',a') - Q(s,a)]$

Key Parameters

- Learning rate (α): 0.1
- Discount factor (γ): 0.9
- Exploration rate (ε): 0.1, decaying



Training the Agent

Training Loop

Iterate over episodes until convergence criteria are met

Episode Parameters

Maximum 200 steps per episode to prevent infinite loops

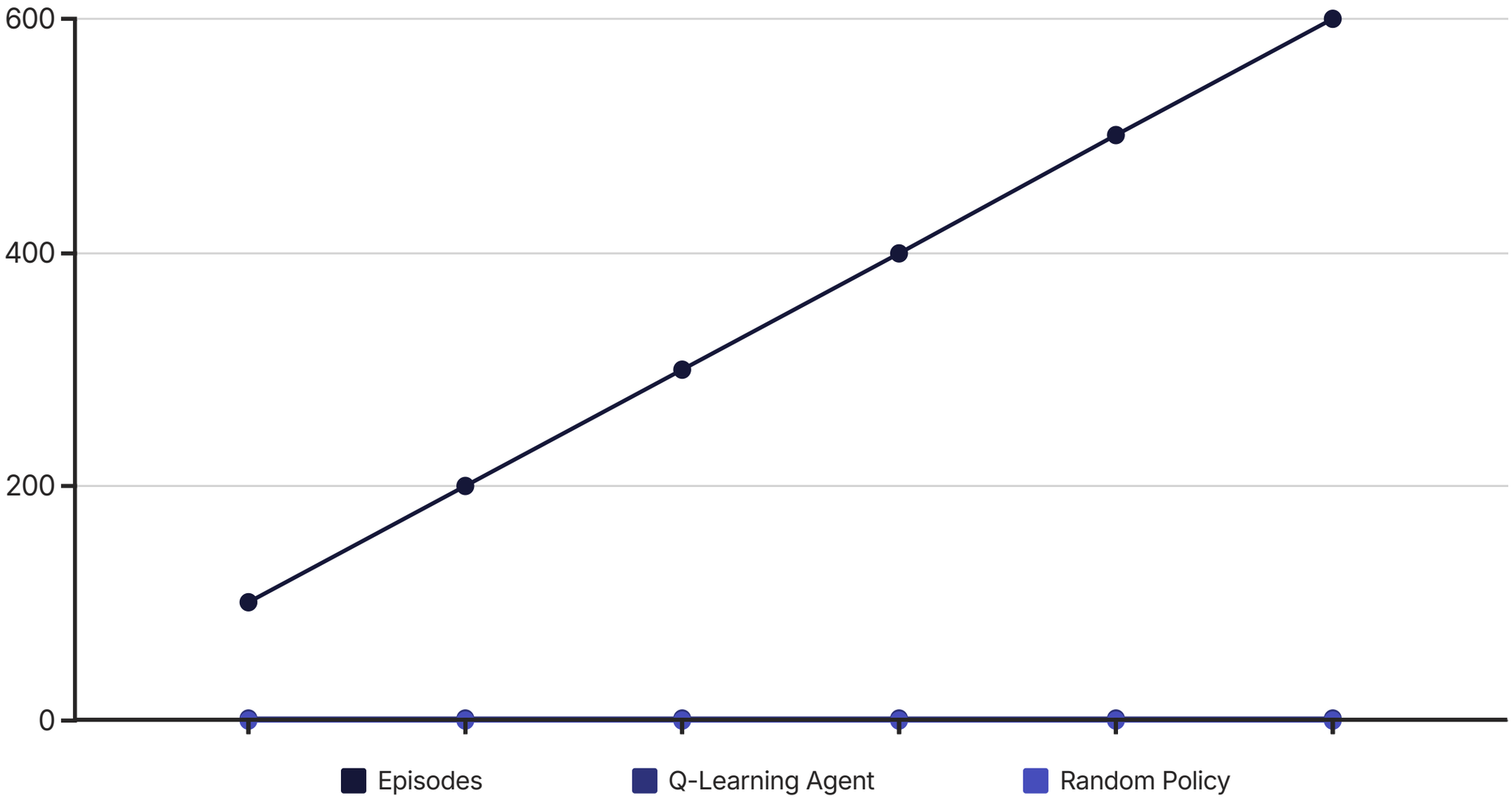
Convergence Criterion

Average reward over 100 episodes exceeds 0.8

Progress Tracking

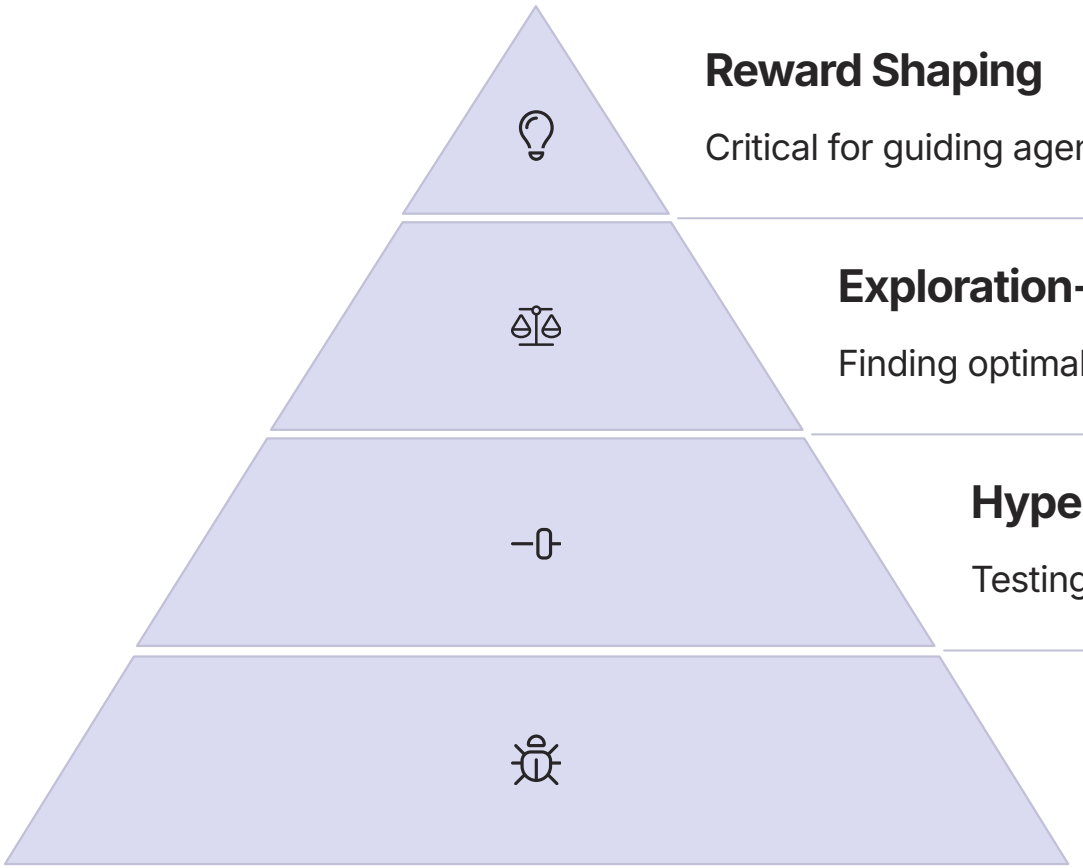
Visualize learning progress with reward vs. episode graphs

Results and Performance



The Q-learning agent significantly outperforms random policy. After training, the agent finds paths averaging just 12 steps.

Challenges and Lessons Learned



Reward Shaping

Critical for guiding agent learning

Exploration-Exploitation Balance

Finding optimal epsilon decay schedule

Hyperparameter Tuning

Testing various α, γ combinations

Local Optima Traps

Agent getting stuck in suboptimal paths

Future Improvements & Conclusion

Implement Deep Q-Networks

Use neural networks to handle more complex environments with continuous state spaces.

Explore Alternative Algorithms

Test SARSA and Policy Gradient methods for comparison.

Incorporate Visual Processing

Add convolutional networks to process visual environment data directly.

Reinforcement learning offers powerful, flexible approaches to complex problems. This project was just the beginning!