Honors Machine Learning

DYNAMIC PROGRAMMING
in Reinforcement Learning

Jaxon Ham & Ruth Walters

Reward *and* Return

- **reward**: the feedback that the agent receives from interaction with the environment
  - R_{t+1} is the immediate reward
  - R_{t+2} , R_{t+3} , … R_{t+n} are the subsequent n rewards
- **return**: the cumulative reward obtained over the duration of an episode
- **discount factor**: (γ) the extent to which future rewards are valued

Reward *and* Return

The return G_t at time t is calculated from the immediate return and the weighted sum of all subsequent returns:

Weighted sum:

$$G_t \stackrel{\text{def}}{=} R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots$$
$$= \sum_{k=0} \gamma^k R_{t+k+1}$$

Recursive:

$$G_t = R_{t+1} + \gamma G_{t+1}$$
$$= r + \gamma G_{t+1}$$

Policy

- **policy**: $\pi(a \mid s)$ a function that determines the next action a to take given a specific state s
- **optimal policy**: $\pi_*(a \mid s)$ the policy that yields the highest return
- In reinforcement learning, agents start with a random policy that they update to maximize return as they gain experience interacting with their environment

Value Function

- **value function**: function that measures the favorability of each state based on the return G_t

The value function $v_\pi(s)$ of state s represents the expected return of a policy, π:

$$v_\pi(s) \stackrel{\text{def}}{=} E_\pi[G_t \mid S_t = s]$$
$$= E_\pi\left[\sum_{k=0} \gamma^{k+1} R_{t+k+1} \mid S_t = s\right]$$

Action Value Function

- **action value function**: the expected return G_t of action $A_t = a$ when the agent is in state $S_t = s$

The action value function $q_\pi(s, a)$ of state s and action a represents the expected return of a policy, π for a and s

$$q_\pi(s,a) \stackrel{\text{def}}{=} E_\pi[G_t \mid S_t = s, A_t = a]$$

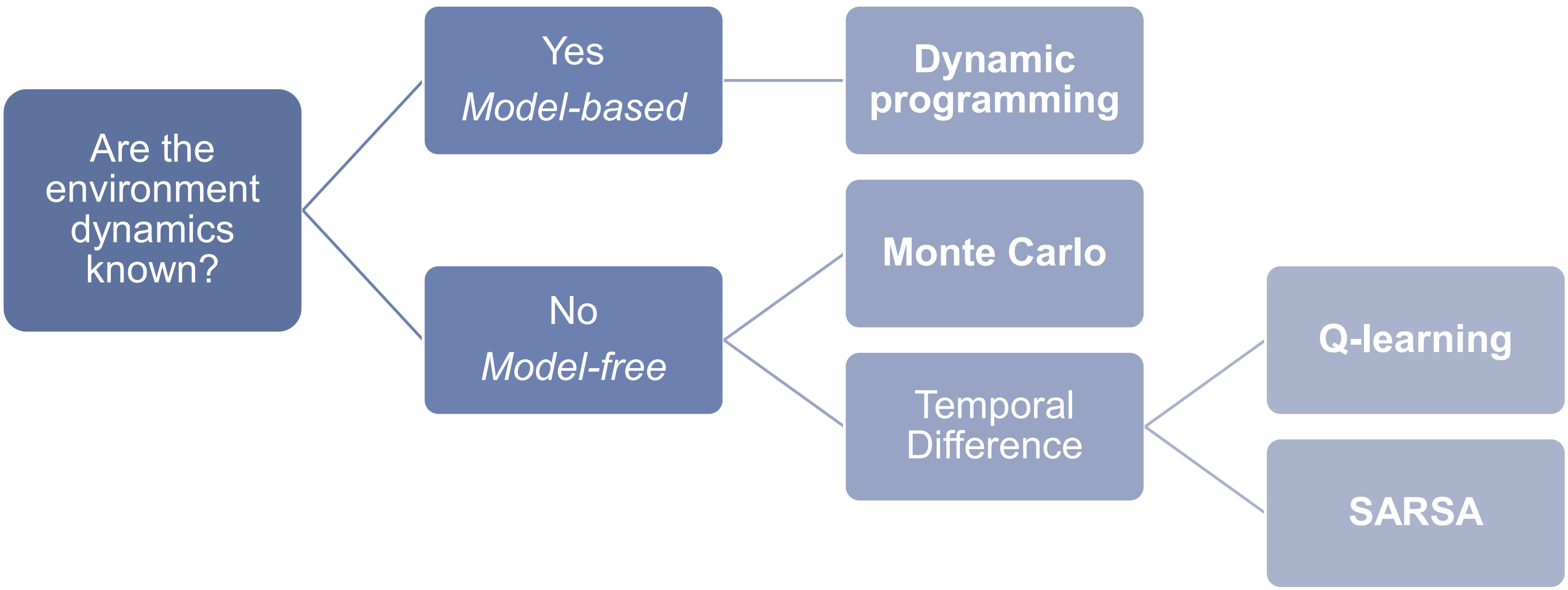
$$= E_\pi\left[\sum_{k=0} \gamma^{k+1} R_{t+k+1} \mid S_t = s, A_t = a\right]$$

Summary

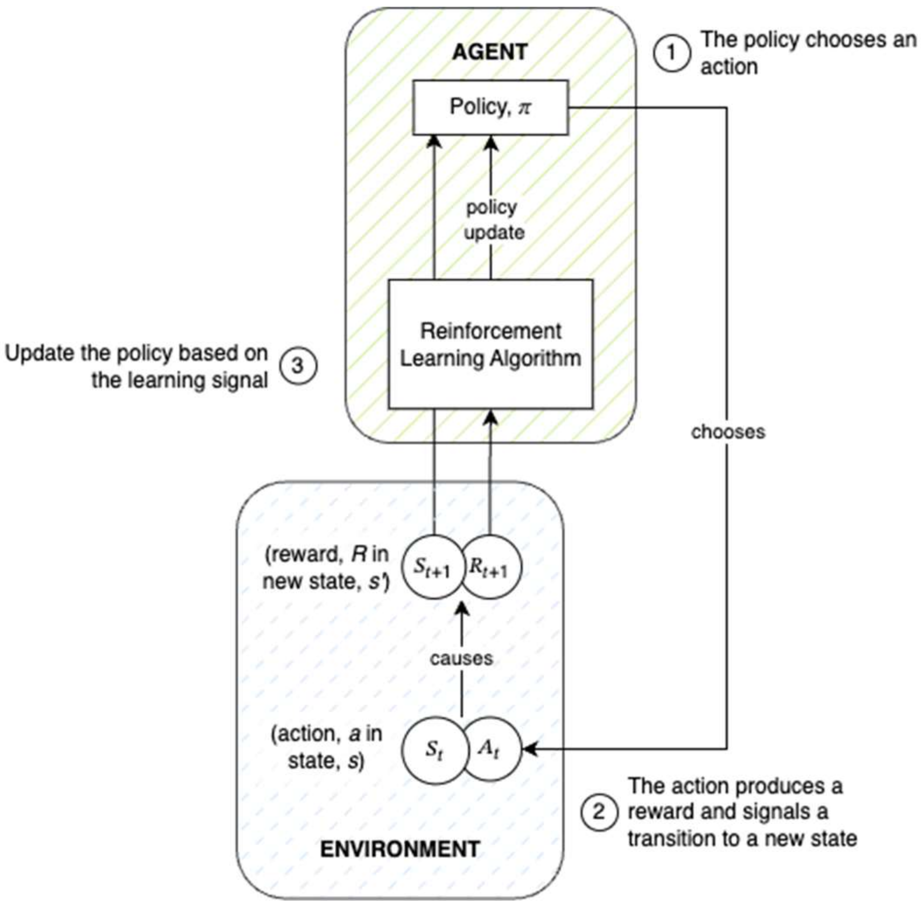
1. The **policy** decides an action
2. Each action produces a **reward**, which prompts the transition from one state to another. A reward is a *consequence* of an action.
3. The **return** is the stream of increasingly discounted future rewards. The return is the *cumulation* of rewards.
4. The **value function** is the expected return from a state under the policy
5. The **action-value function** is the expected return from a state after choosing a specific action and following the policy

Reinforcement Learning Algorithms

When the **dynamics** of the environment are **known**, a learning task can be solved with dynamic programming

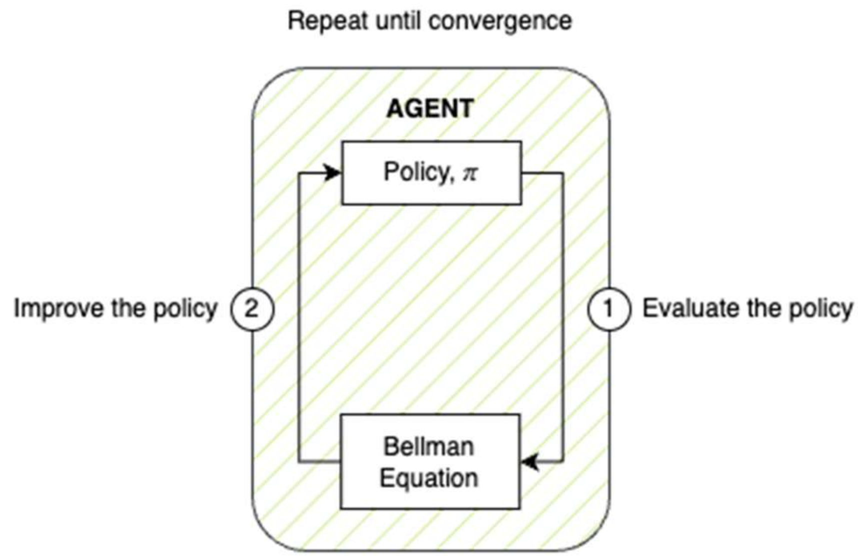


Model-Free methods



Model-Based methods,

e.g. dynamic programming



Dynamic programming models don't have to interact with their environment

the Bellman Equation

The value function for a state s can be given as a function of its subsequent state, s':

$$v_\pi(s) = \sum_{a \in \hat{A}} \pi(a \mid a) \sum_{s' \in \hat{S},\ r \in \hat{R}} p(s', r \mid s, a)[r + \gamma\ v_\pi(s')]$$

Assumptions *in* Dynamic Programming

1	2
The environment dynamics are known	The agent's state has the Markov property

- **Markov property**: a property of an agent's state in which **the next action and reward depend *only* on the current state and choice of action** made at the current moment (time step)

Objectives *of a* Dynamic Programming Model

1 **Prediction Task**
Obtain the true-state value function, $v_\pi(s)$, of a policy π via **policy evaluation**

2 **Policy Improvement Task**
Find the optimal value function, $v_*(s)$, of a policy π via **generalized policy iteration**

Value Iteration

Policy Evaluation

1. Start at $v^{\langle 0 \rangle}(s)$, with zero values for each state
2. At each iteration, update the values for each state using the Bellman equation

$$v^{\langle i+1 \rangle}(s) = \sum_a \pi(a \mid s) \sum_{s' \in \hat{S},\ r \in \hat{R}} p(s', r \mid s, a)[r + \gamma\ v^{\langle i \rangle}(s')]$$

- As i increases towards infinity, the value function $v^{\langle i+1 \rangle}(s)$ will converge to the true state value function
- Since the dynamics of the environment are known, there is no need to interact with the environment

Policy Iteration

- Once the value function $v_\pi(s)$ has been determined for the current policy π, the value function can be used to improve that policy
- The **goal** is to find a new policy π' that, **for each state** s would yield **higher or at least equal value** relative to the current policy

$$v_{\pi'}(s) \geq v_\pi(s) \quad \forall\ s \in \hat{S}$$

Generalized Policy Improvement

Iterate

1. Compute the action-value function $q_\pi(s, a)$ for each state s and each action a
2. For each $q_\pi(s, a)$, compare the value if the next state s' if action a was selected

Evaluate

Compare the action a_* that produces the highest state value to the action selected by the current policy

Update

If the action suggested by the current policy is worse than the action suggested by the action value-function, update the policy

Value Iteration

- **value iteration**: the process of selecting the best action out of all possible actions by combining policy evaluation and policy improvement steps
- Combines policy evaluation and policy iteration
- The updated value for $v^{\langle i+1 \rangle}(s)$ is maximized by choosing the best action out of all possible actions

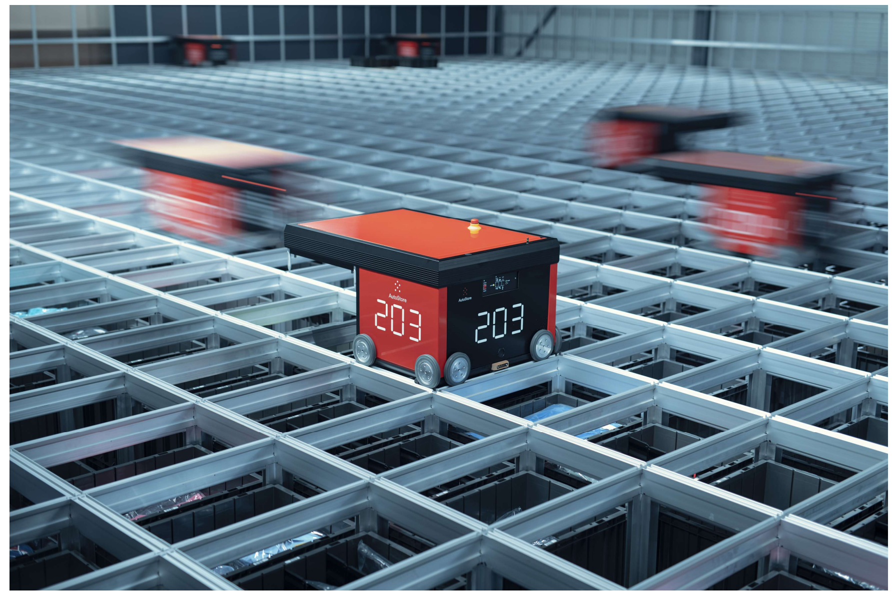
Value Iteration

- The value function $v^{\langle i+1 \rangle}$ for iteration $i + 1$ is updated based on the action that maximizes the weighted sum of the next state value and its immediate reward

$$v^{\langle i+1 \rangle}(s) = \max_a \sum_{s',\ r} p(s', r \mid s, a)[r + \gamma\ v^{\langle i \rangle}(s')]$$

- Compare to policy evaluation, which updates using a weighted sum of all actions

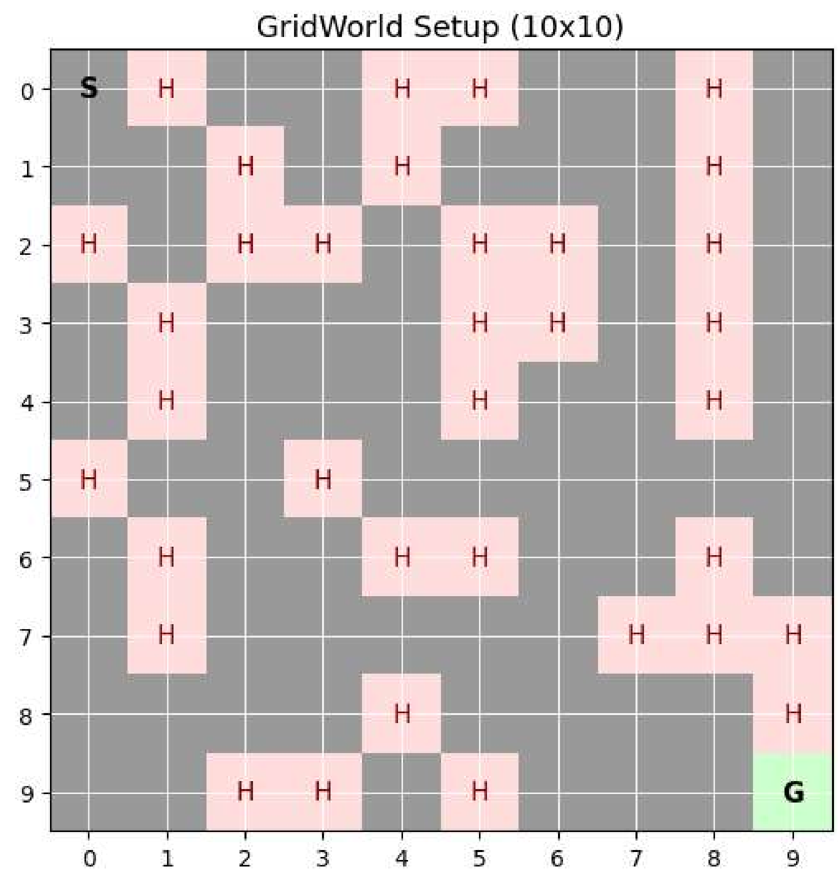
Example Context



Basic Concept: Warehouse Robots

Building the "GridWorld" Environment

- **Rewards (R):**
  - -1 per move (step cost)
  - -2 to -6 for hazards
  - +10 for reaching goal
- **Transition Model (P):** 10% slip chance → imperfect movement
- **Discount (γ = 0.97):** favors quicker paths



Implementing the Bellman Equation

Bellman Optimality Equation: $$V(s) = \max_a \sum_{s',r} P(s',r|s,a)[r + \gamma V(s')]$$

In Code from value_iteration():

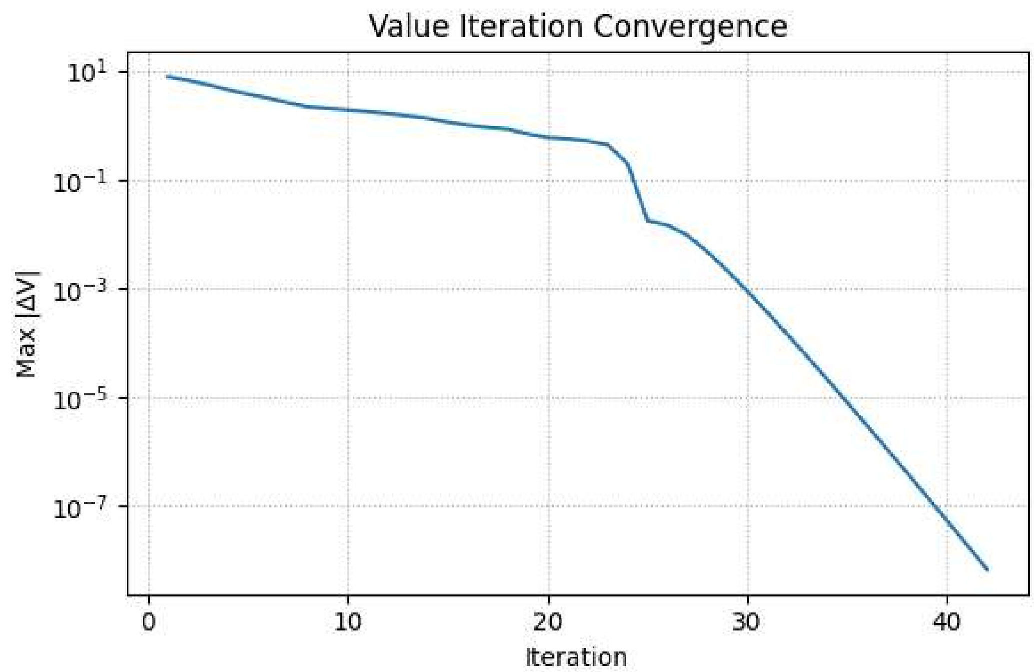
```
q = [sum(p * (r + gamma * V[s2])
         for p, s2, r, d in P[s][a]) for a in ACTIONS]
V[s] = max(q)
```

- *V(s)*: state's value
- *a*: action
- *s'*: next state
- *r*: immediate reward
- *P(s', r | s, a)*: transition probability

State *s* → Action *a* → Next state *s'* → Reward *r* → Updated Value *V(s)*

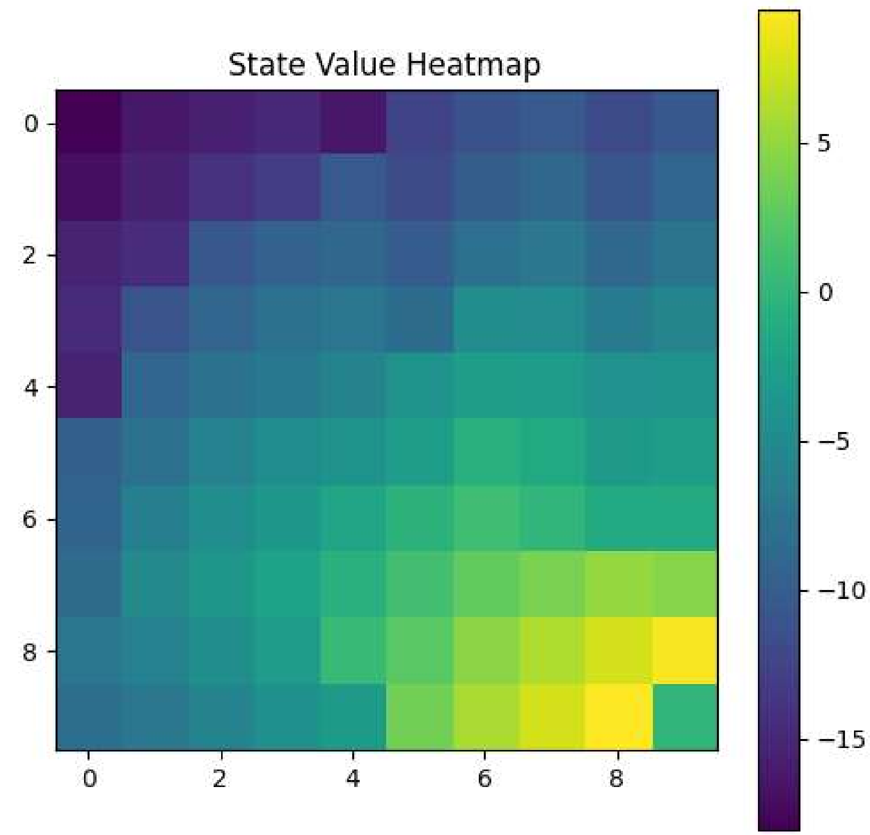
The Dynamic Programming Process

- **Initialize:** all V(s) = 0
- **Update:** apply Bellman equation repeatedly
- **Improve:** pick best action per state (greedy)
- **Converge:** stop when ΔV < threshold
- Combines:
  - **Policy Evaluation** (find Vπ)
  - **Policy Improvement** (make π greedy)



Value Heatmap: Learned State Values

- Each color = expected total reward *V*(s)*
- Brighter → higher value (closer to goal)
- Darker → lower value (hazards / farther away)
- DP propagates value <u>backward</u> from the goal



Policy Grid: Extract Optimal Actions

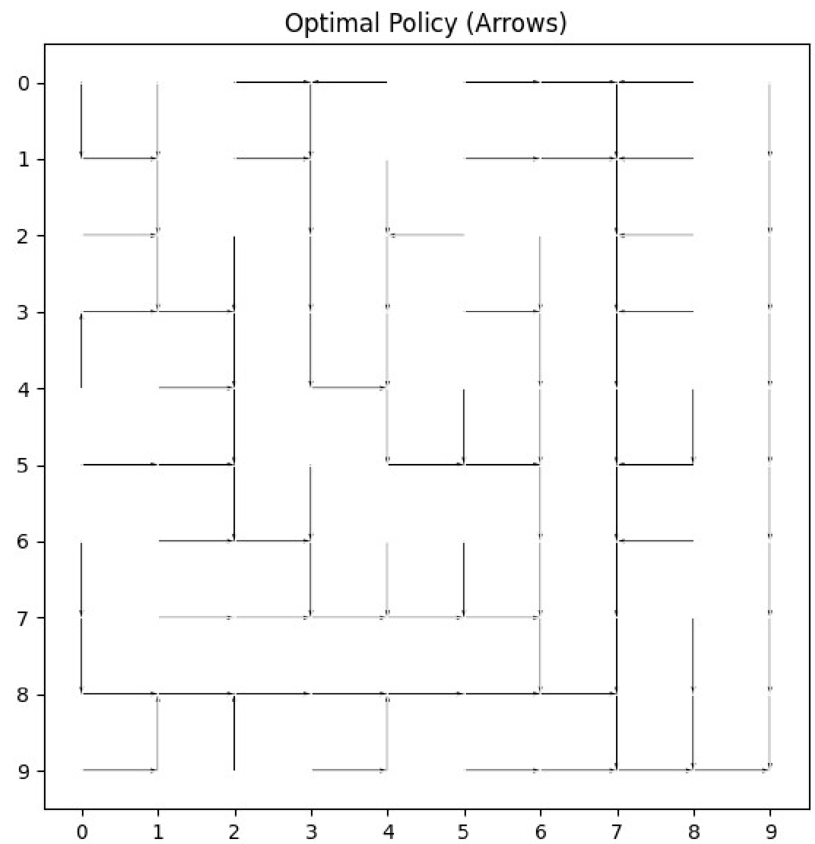
Policy Formula:

$$\pi^*(s) = \arg\max_a \sum_{s',r} P(s', r|s, a)[r + \gamma V^*(s')]$$

In Code greedy_policy_from_V():

- Arrows = best actions per state
- Hazards cause arrows to reroute

```
pi[s, np.argmax(q)] = 1
```



Rollout Simulation: Testing the Policy

- Simulates following π* from Start → Goal
- Each step:
  - Pick best action: *argmax(π[s])*
  - Random 10% slip chance
  - Add step/hazard/goal reward
- **Rollout path**: sequence of states
- **Total return**: sum of all rewards