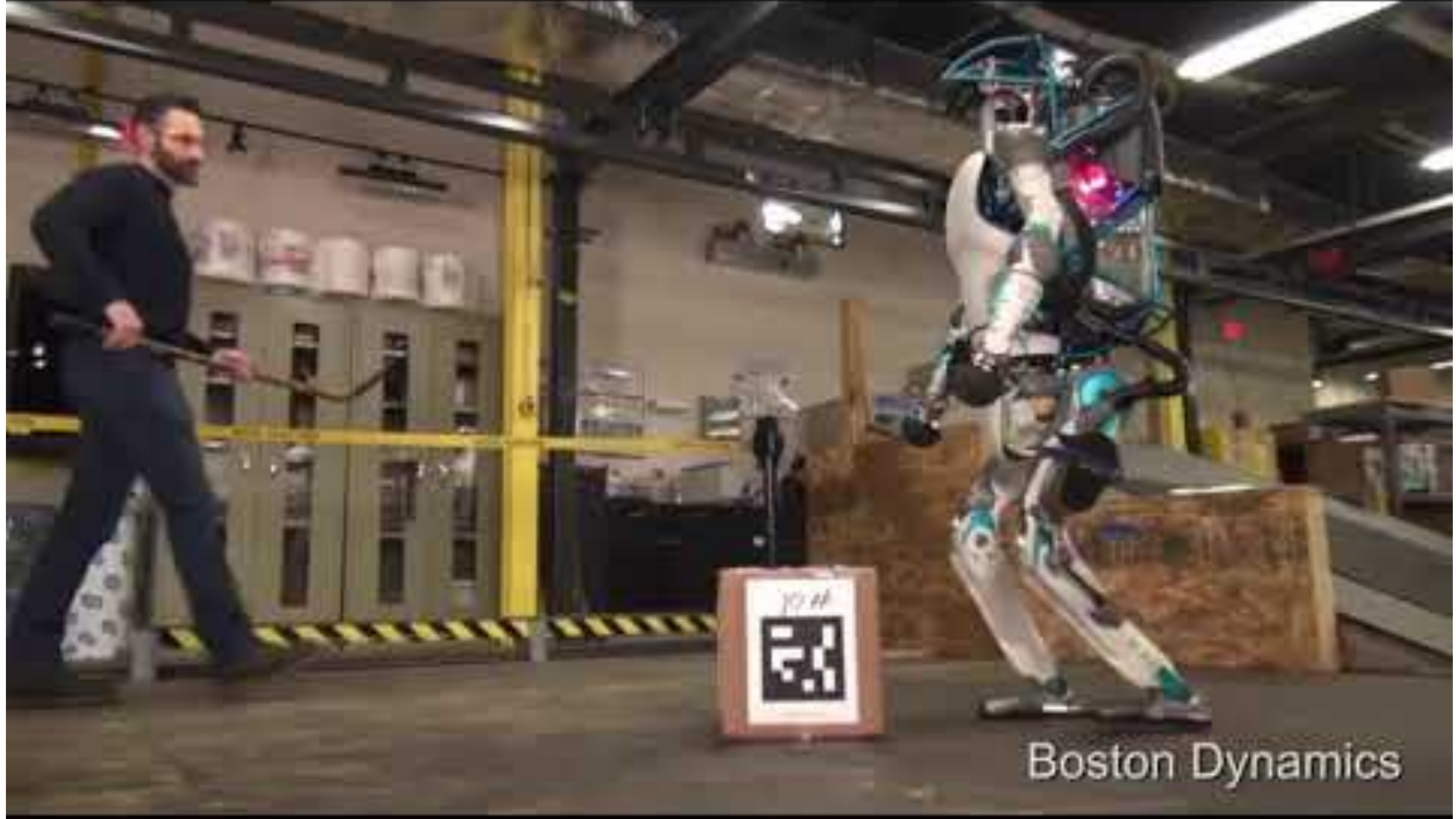


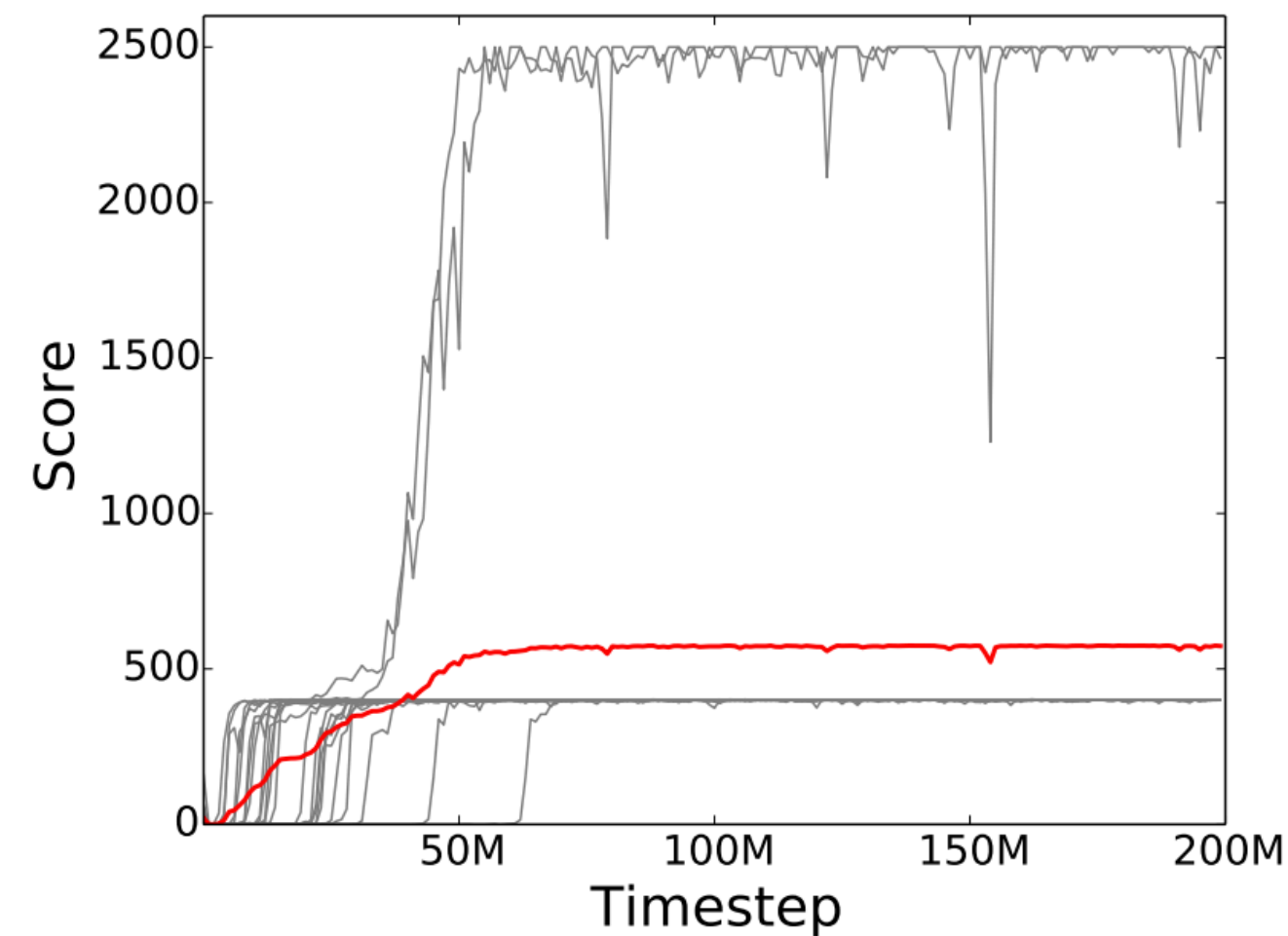
On-policy Prediction with Function Approximation

CMPUT 655
Fall 2022

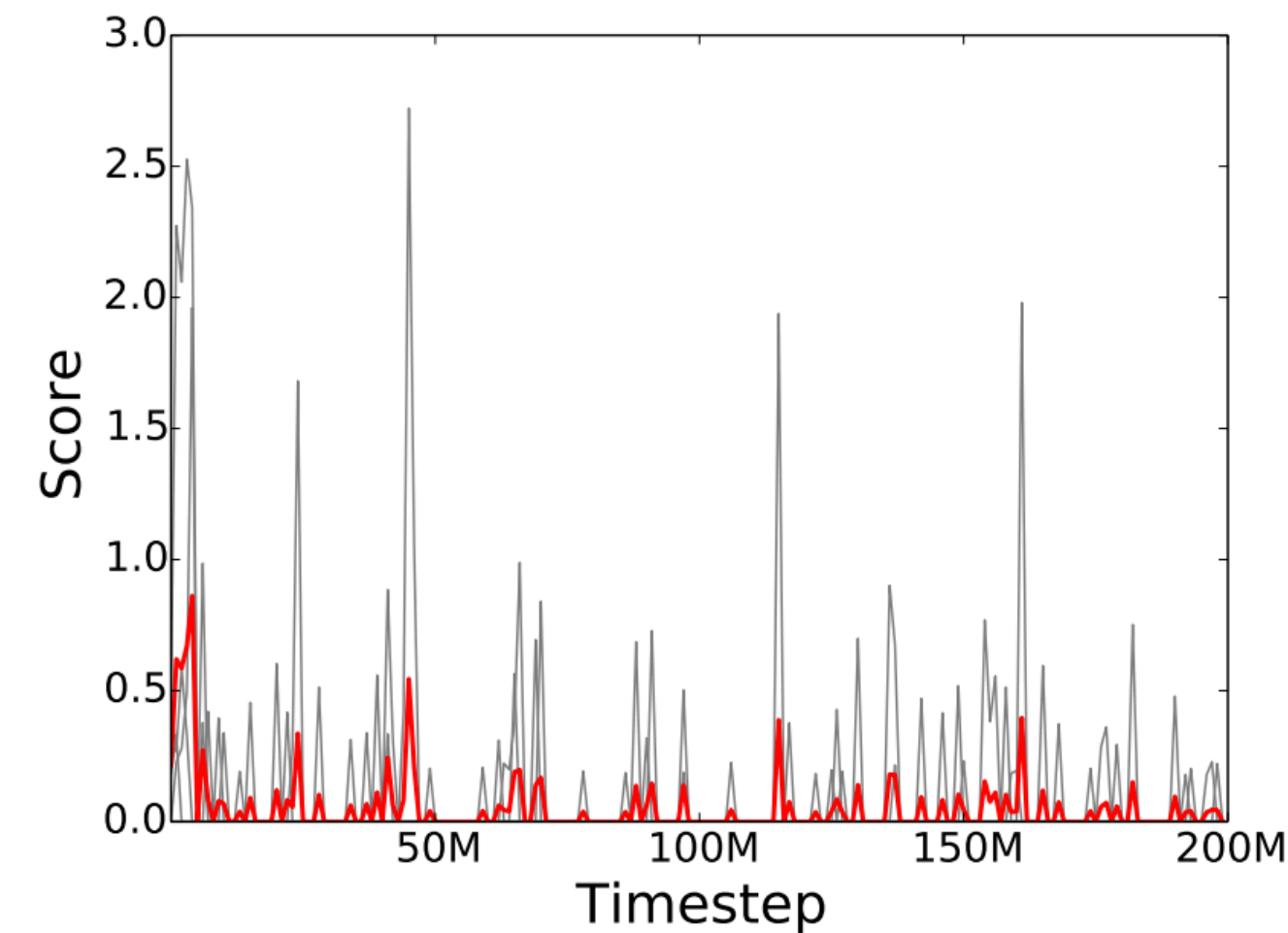


Boston Dynamics

Linear FA vs Deep RL!



(a) Sarsa(λ) + Blob-PROST



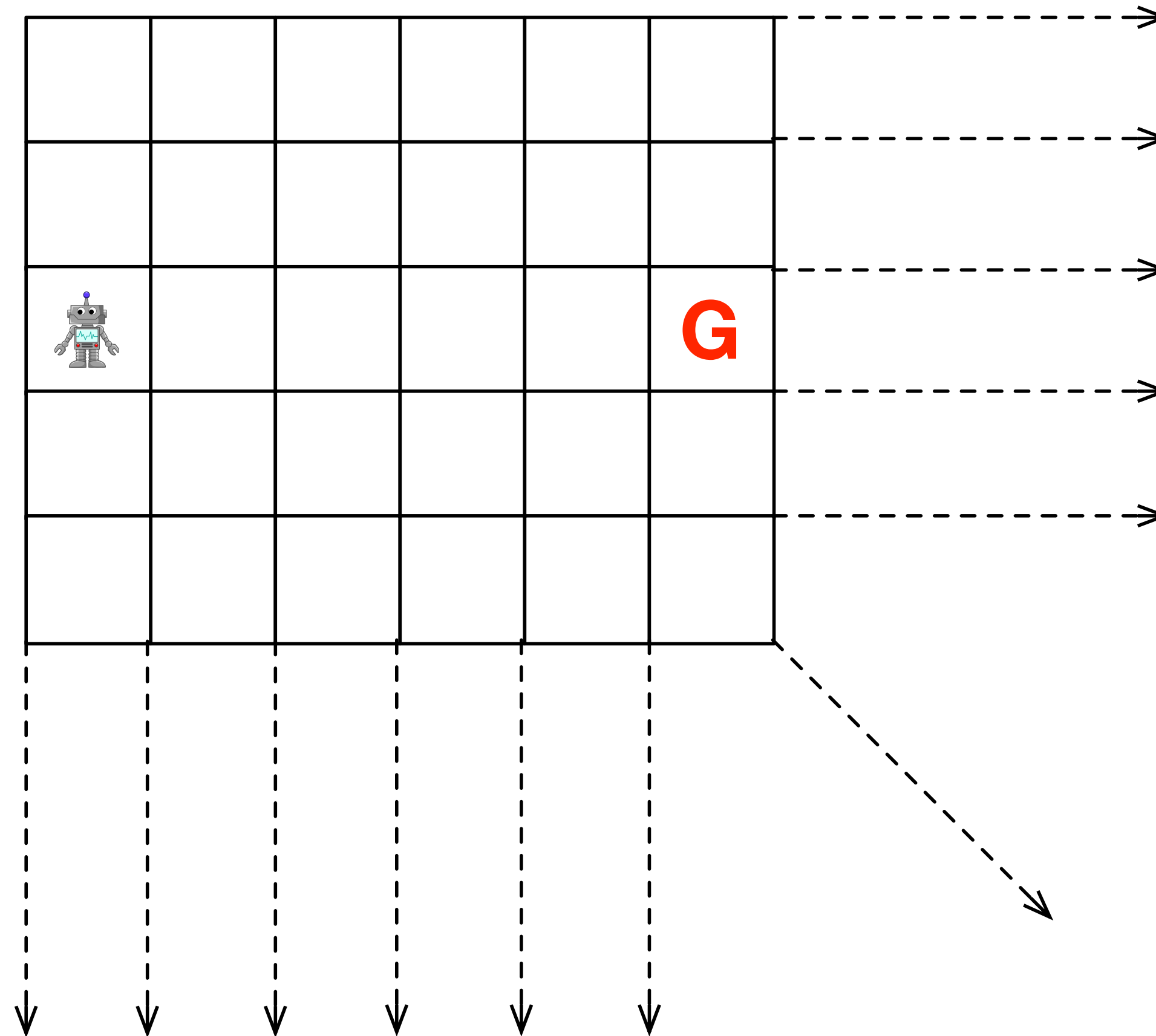
(b) DQN

Figure 4: Comparison between learning curves of DQN and Sarsa(λ) + Blob-PROST in MONTEZUMA'S REVENGE. Notice the y-axes are not on the same scale. Each point corresponds to the average performance over the last one million episodes. Grey curves depict individual trials. The red curve depicts the average over all trials.

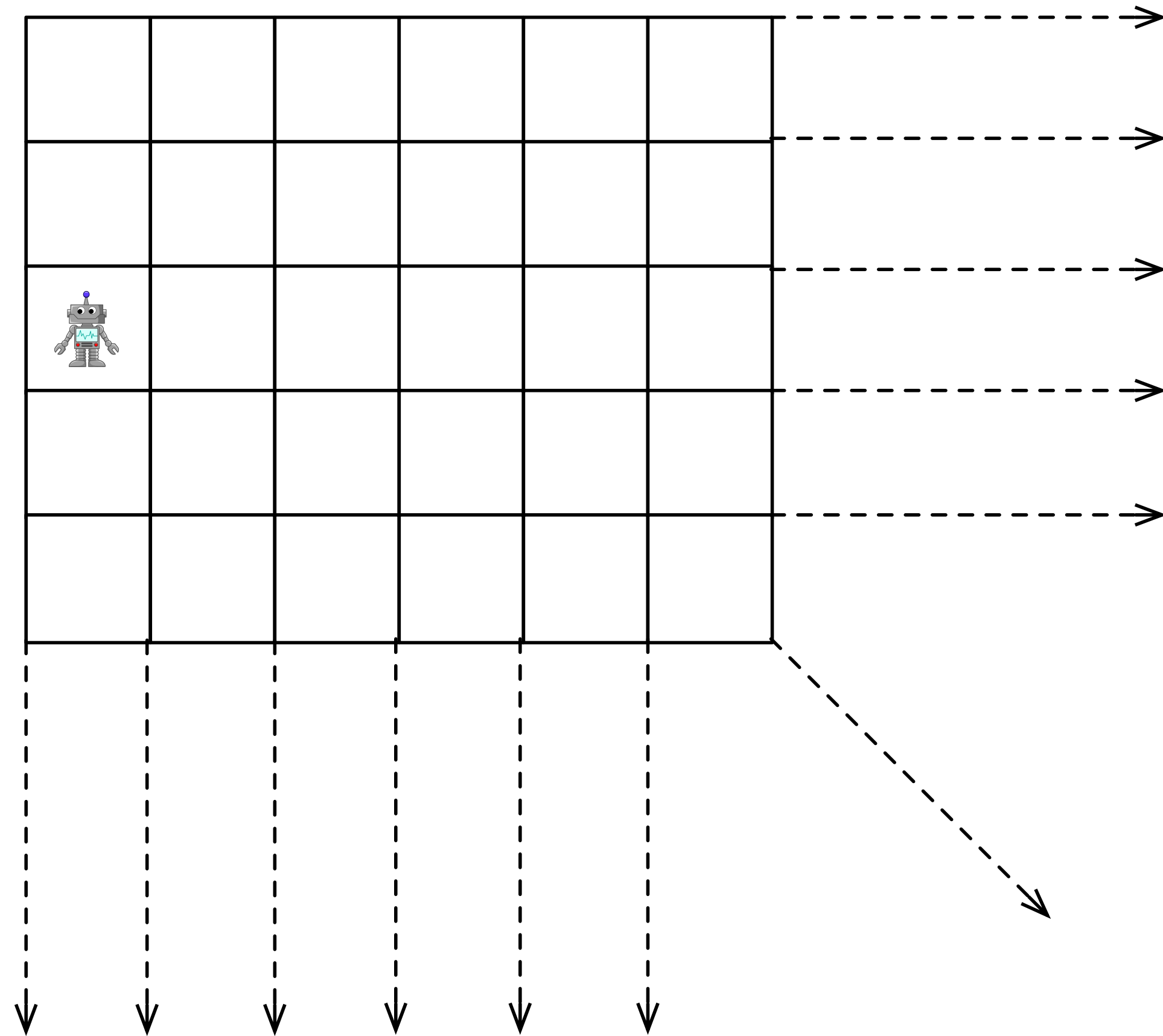
Tabular is not better!

**We do FA, not just because we
must... it's useful**

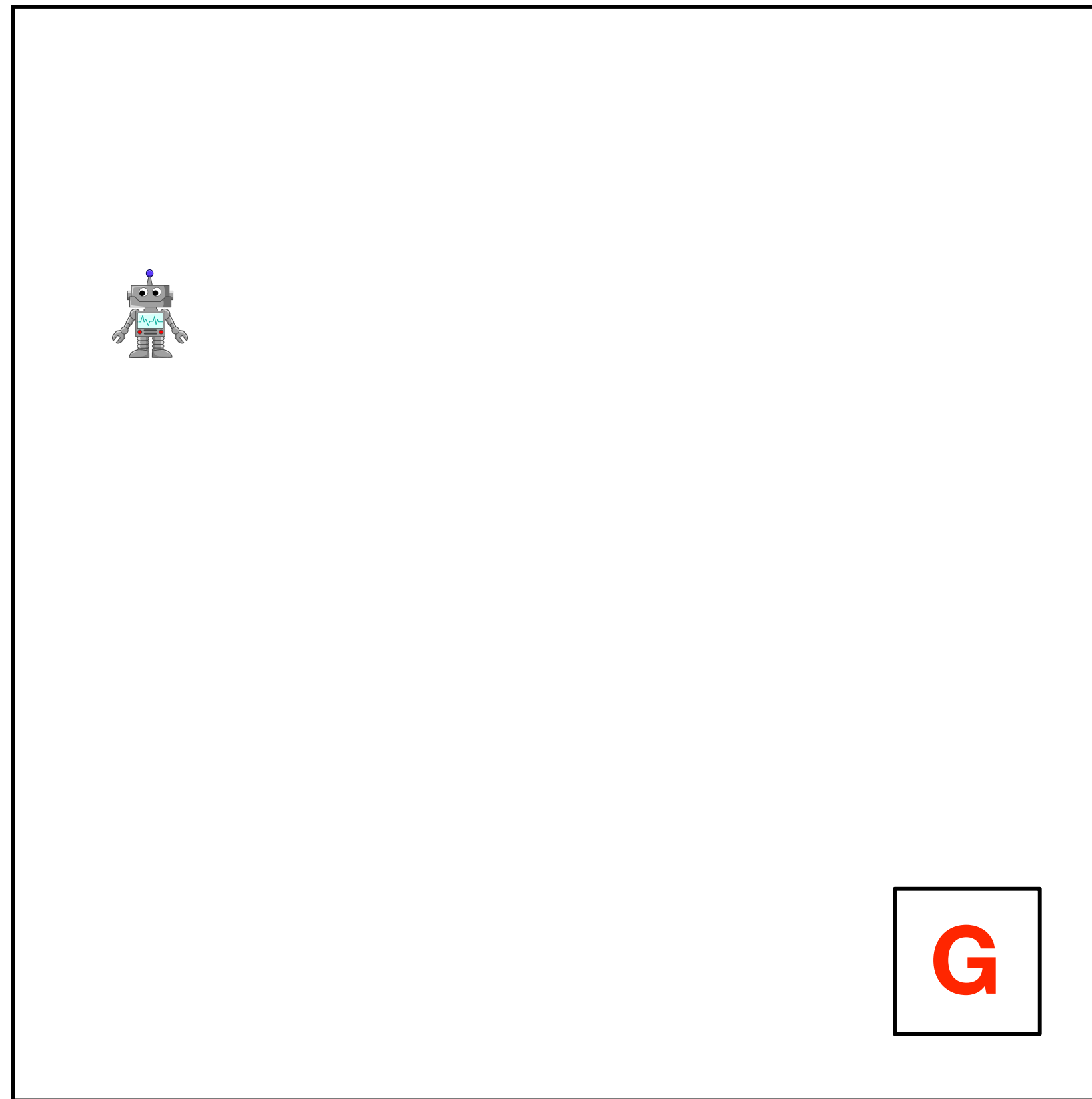
Imagine a huge state space



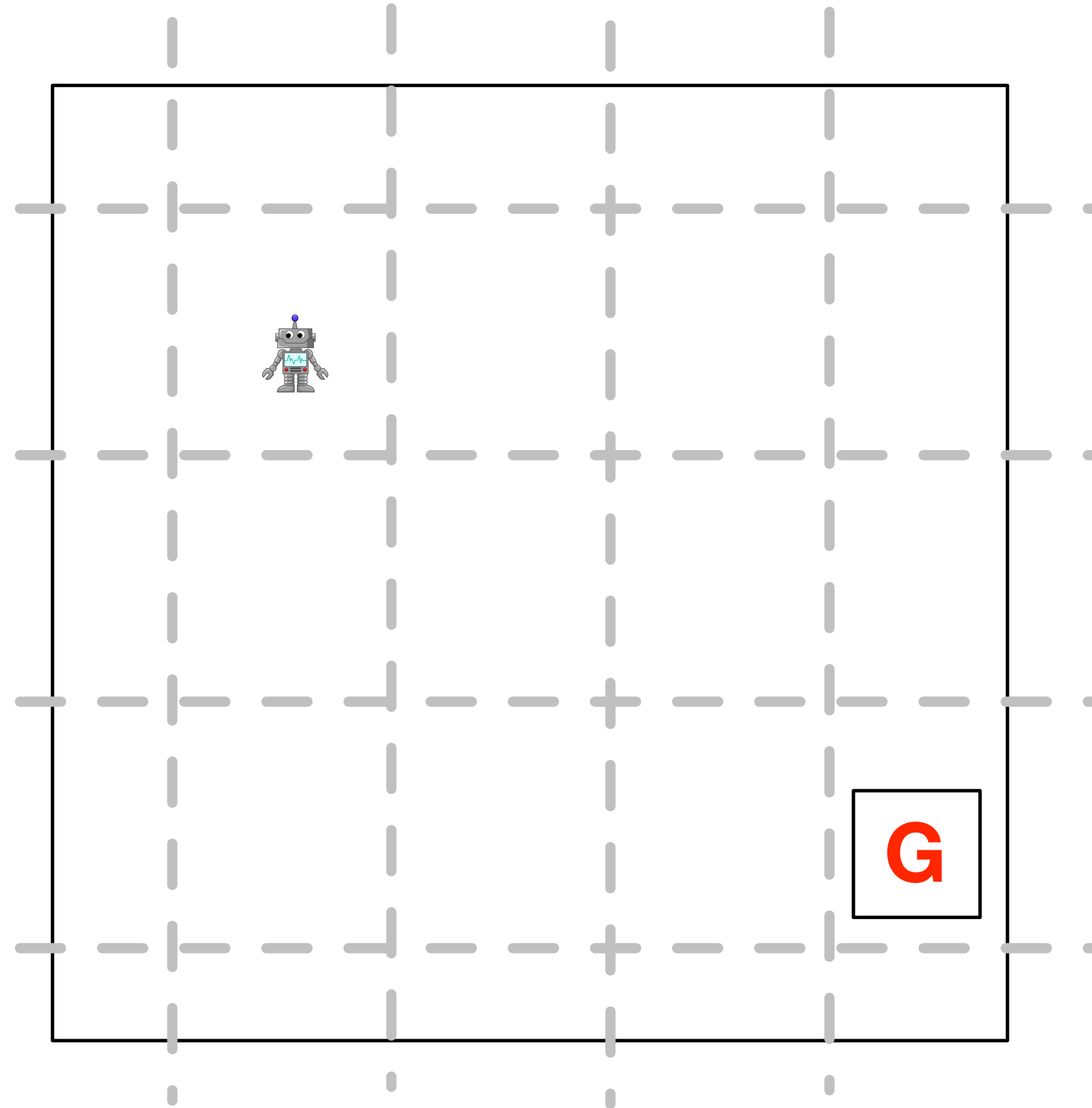
Imagine a huge state space



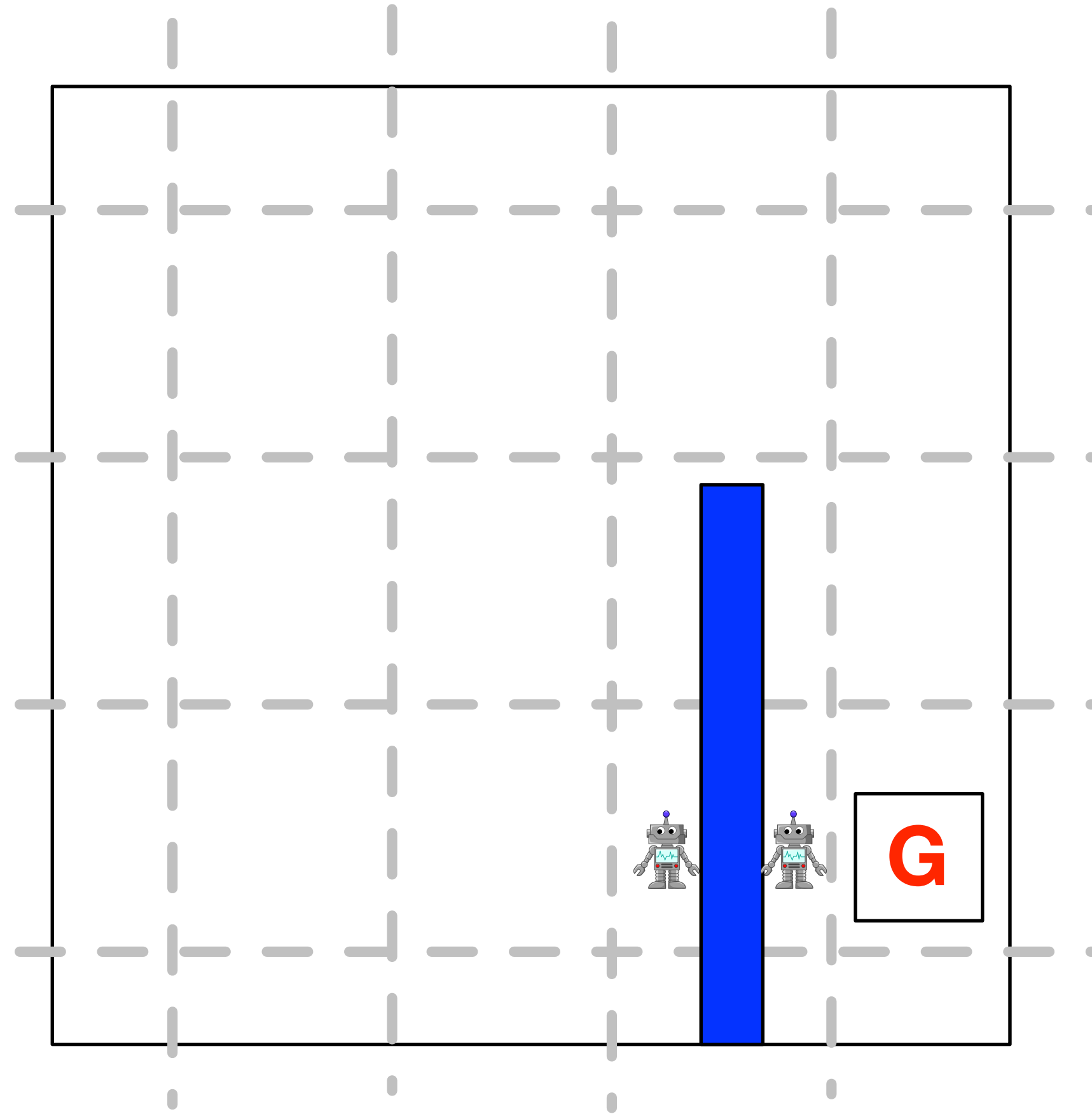
Imagine a continuous state space



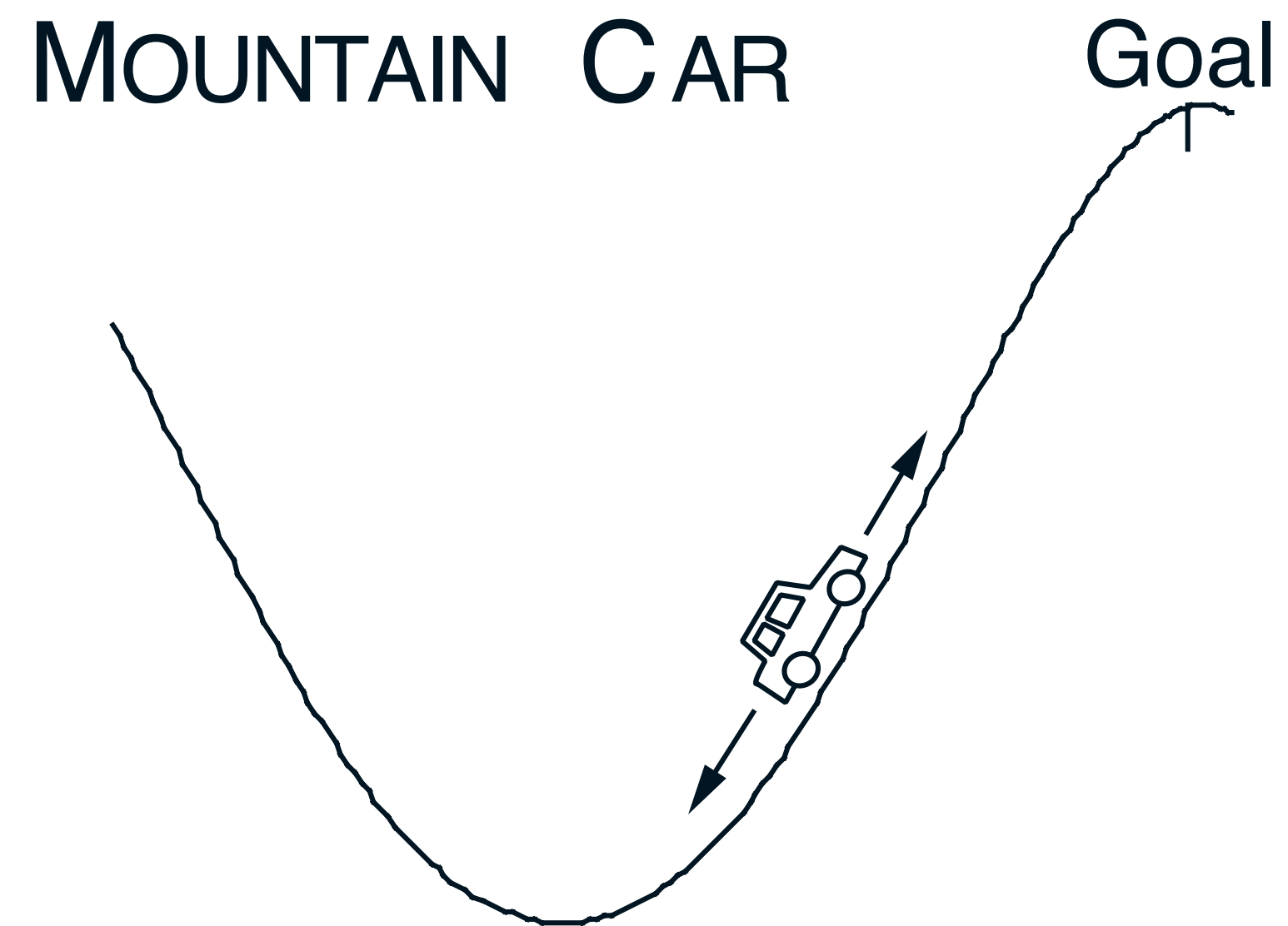
Imagine a continuous state space



Imagine a continuous state space



Another continuous state domain



$$p_{t+1} \doteq \text{bound}[p_t + \dot{p}_{t+1}]$$

$$\dot{p}_{t+1} \doteq \text{bound}[\dot{p}_t + 0.001A_t - 0.0025 \cos(3p_t)]$$

$$\cancel{V(s)} \approx v_{\pi}(s)$$

1.71

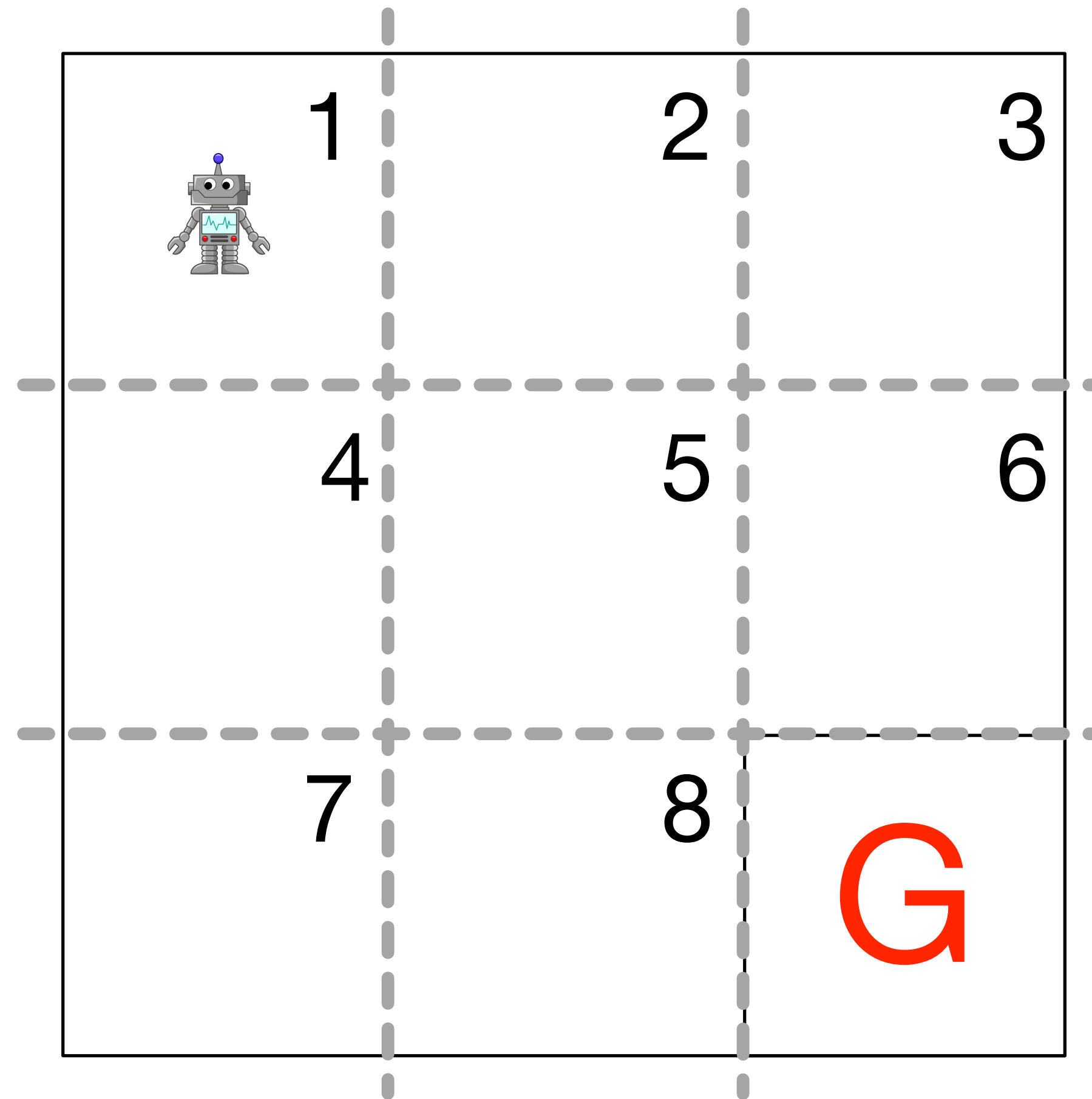
$\mathbf{w} \in \mathbb{R}^d$, *e.g.*, $\mathbf{w} =$
 parameter
 vector

$$\begin{bmatrix} 2.1 \\ 0.01 \\ -1.1 \\ 1.2 \\ -0.1 \\ 0.01 \\ 4.93 \\ 0.5 \end{bmatrix}, \quad \mathbf{x}(s) = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

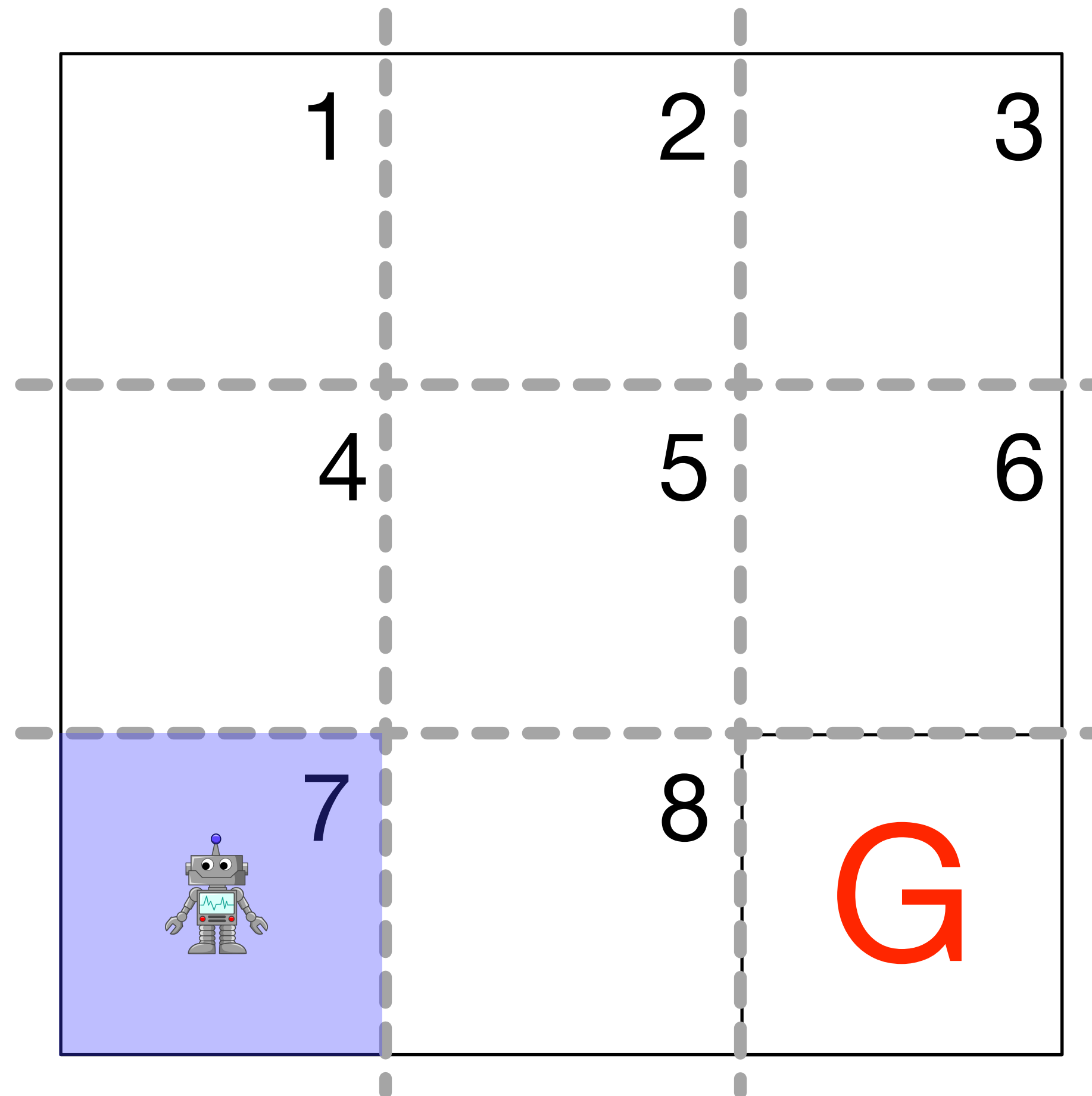
feature
 vector

$\mathbf{x} : \mathcal{S} \rightarrow \mathbb{R}^d$

Let's look at a simple state aggregation

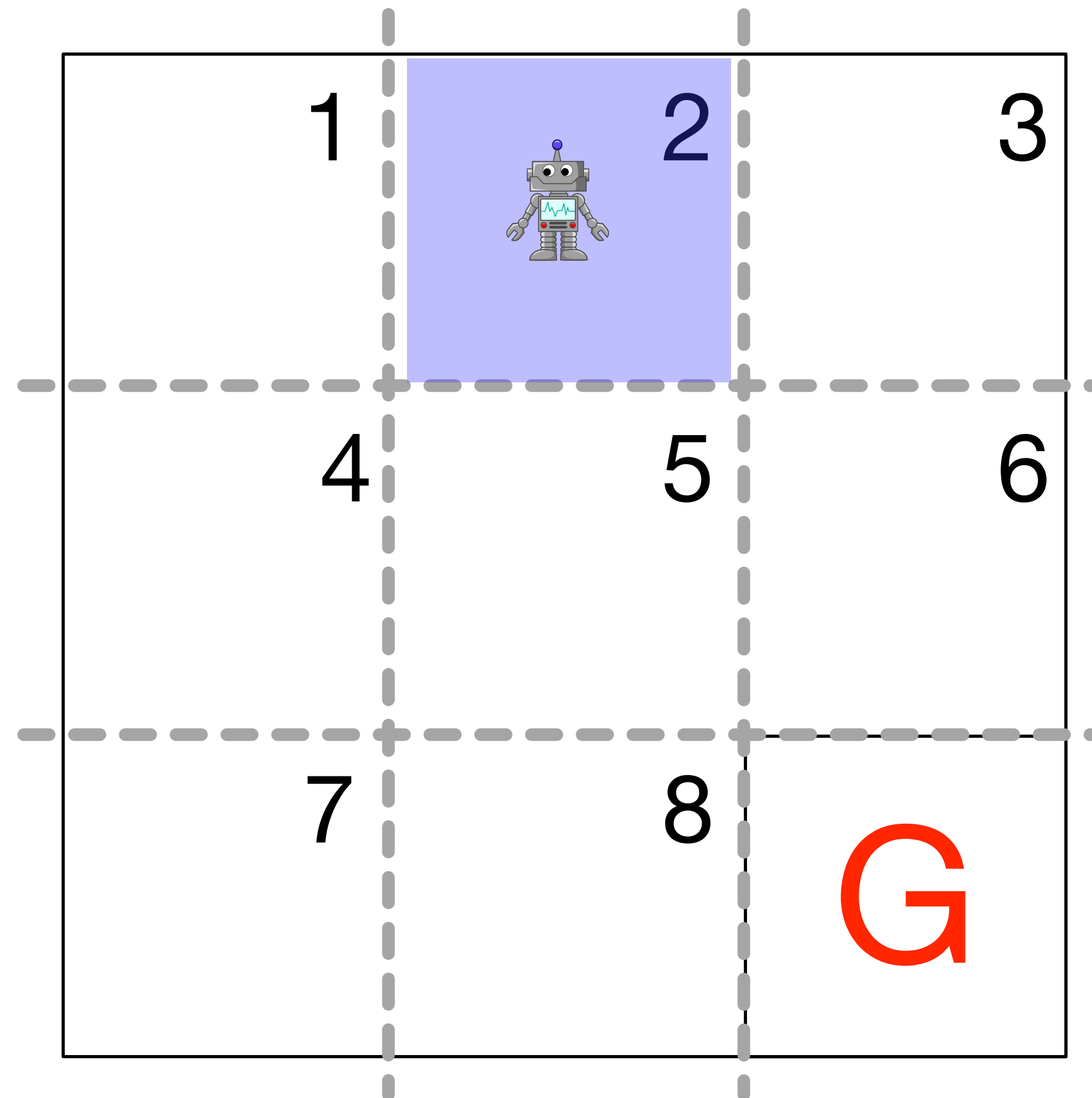


Here is the feature vector if the agent is somewhere in the bottom left



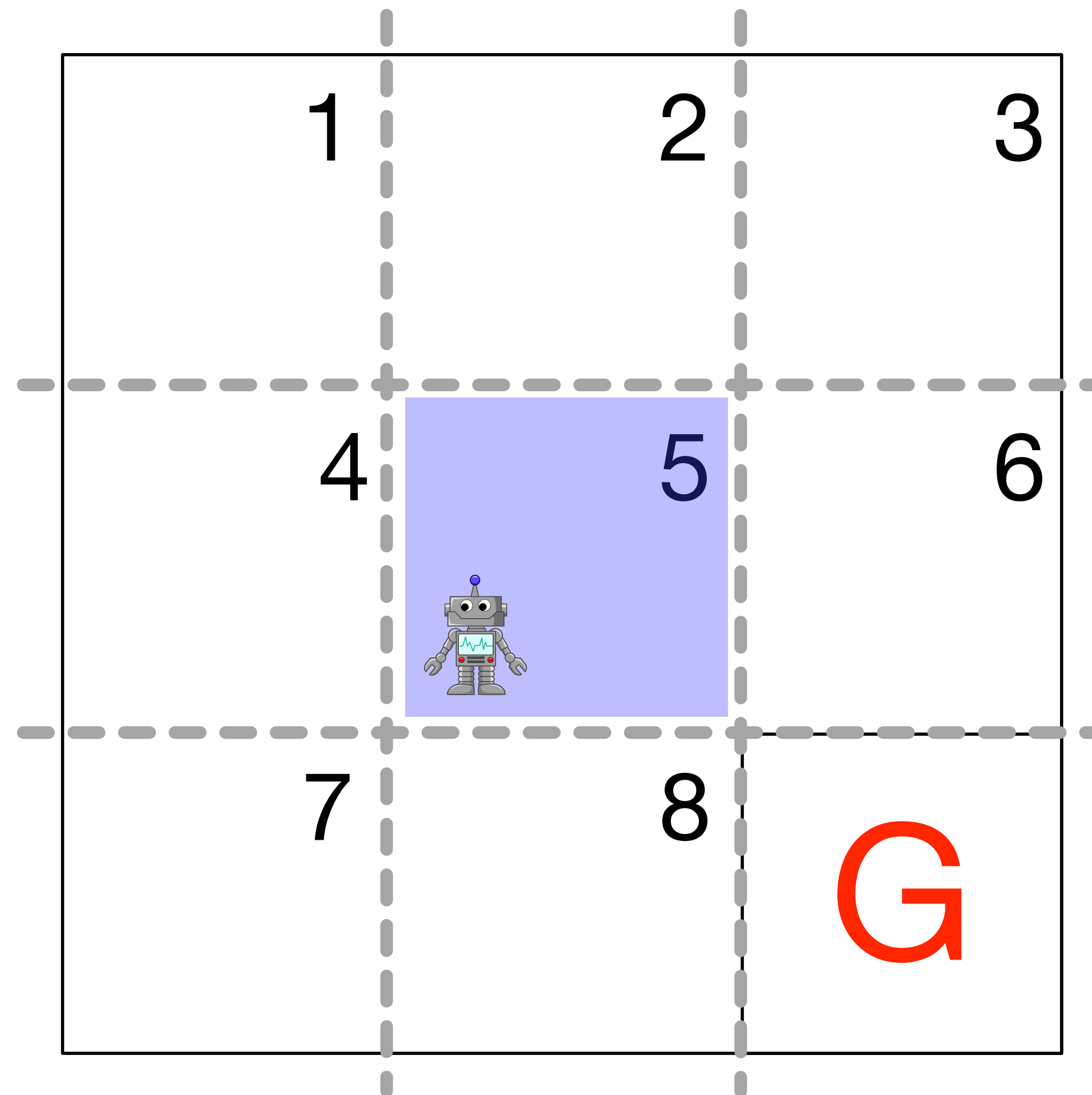
$$\mathbf{x}(s) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

What would the feature vector be if the agent was somewhere in the top middle?



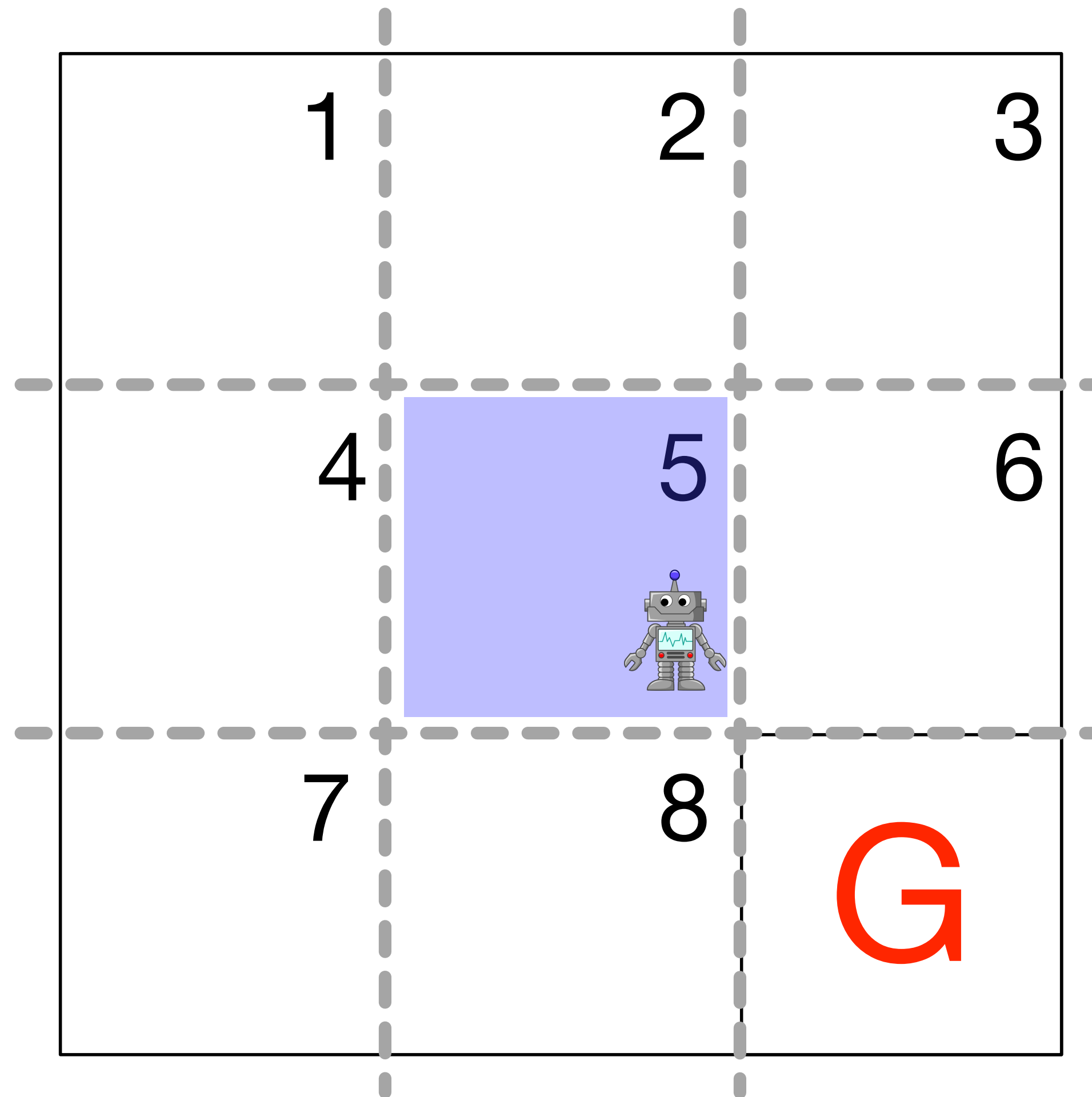
$$\mathbf{x}(s) = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

What would the feature vector be if the agent was in the middle?



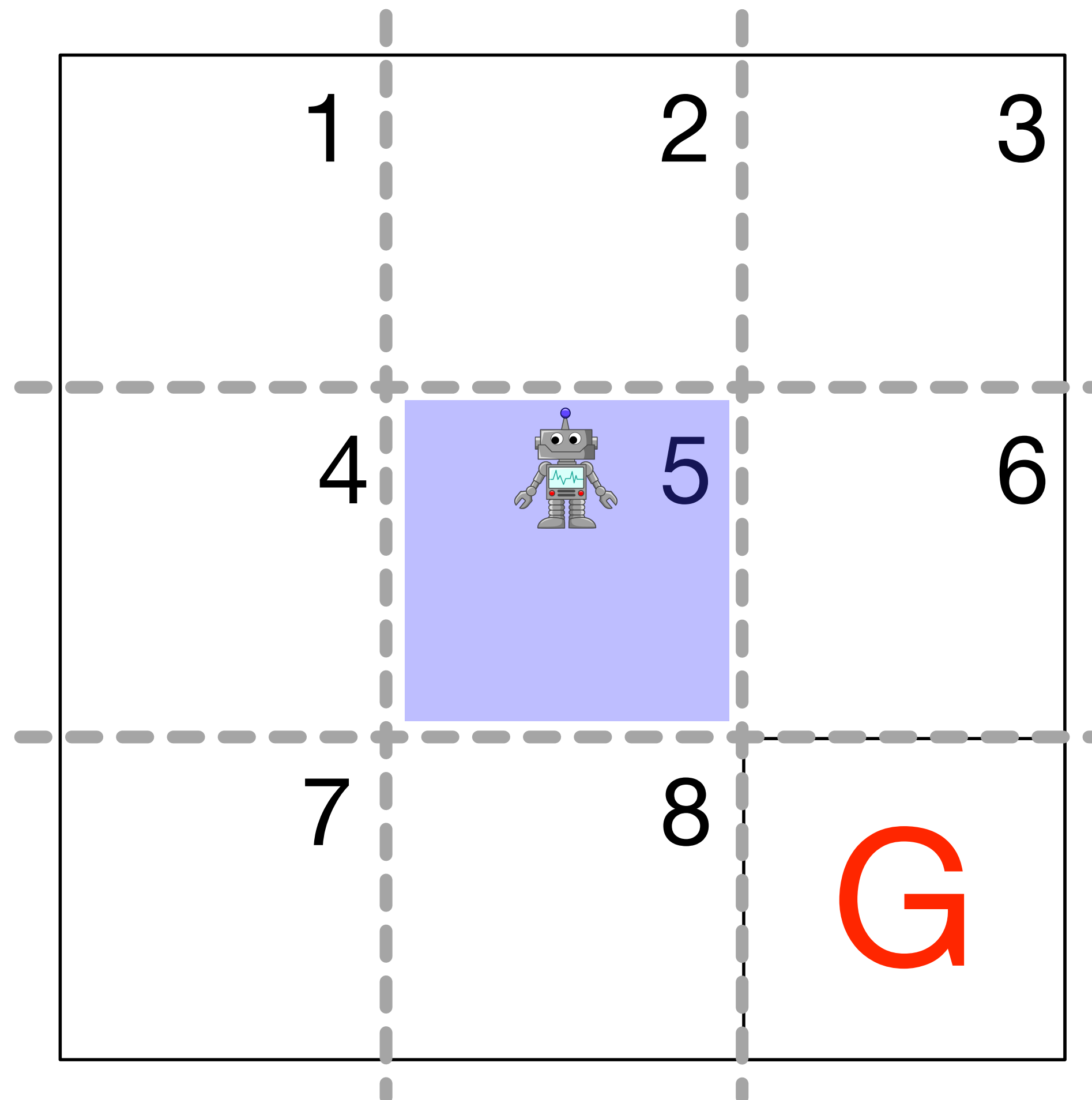
$$\mathbf{x}(s) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

How about here?



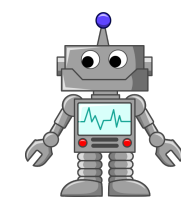
$$\mathbf{x}(s) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Or here?

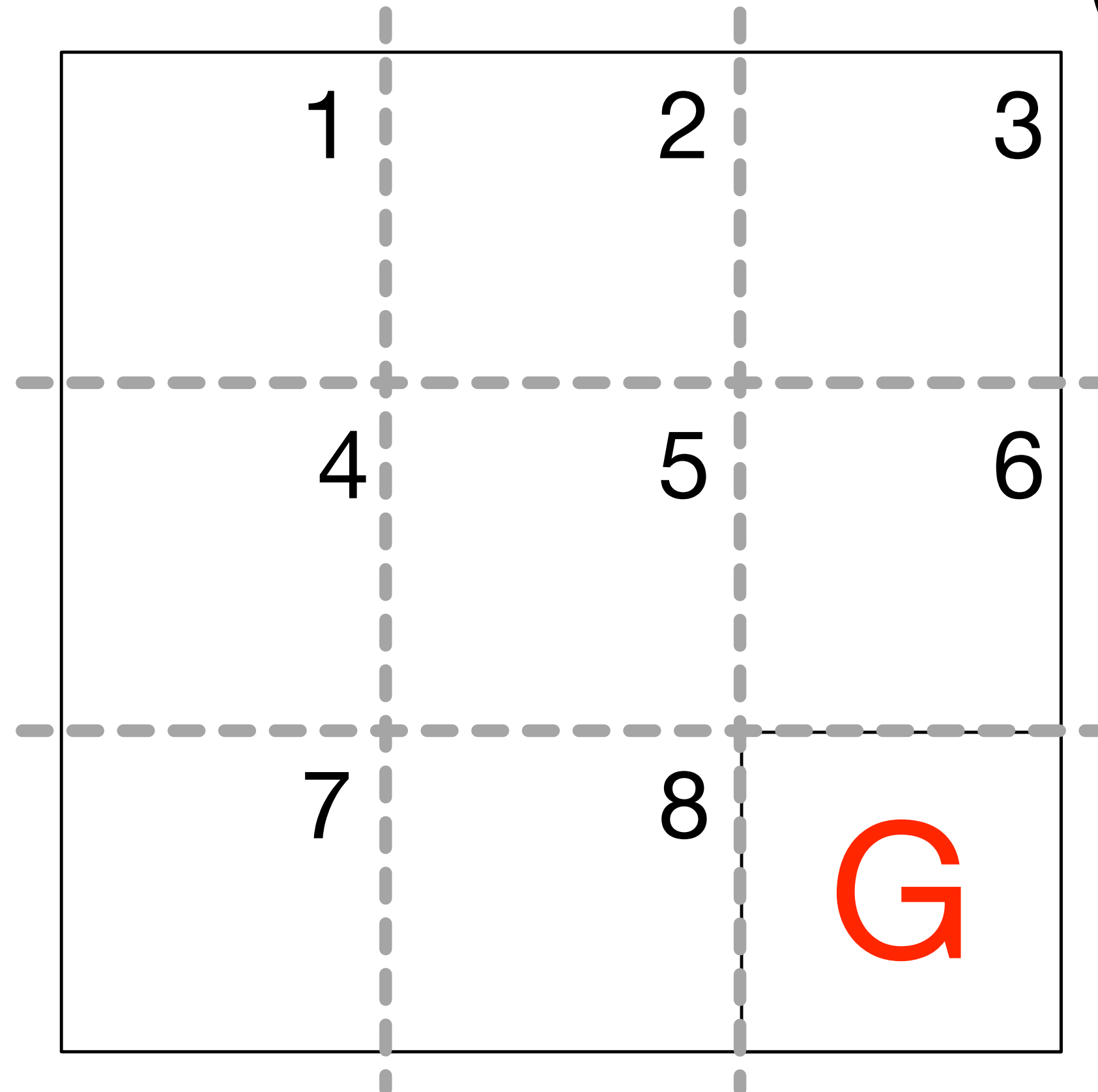


$$\mathbf{x}(s) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

What would the feature vector be if the agent was at this point?

 = (.8, .8)

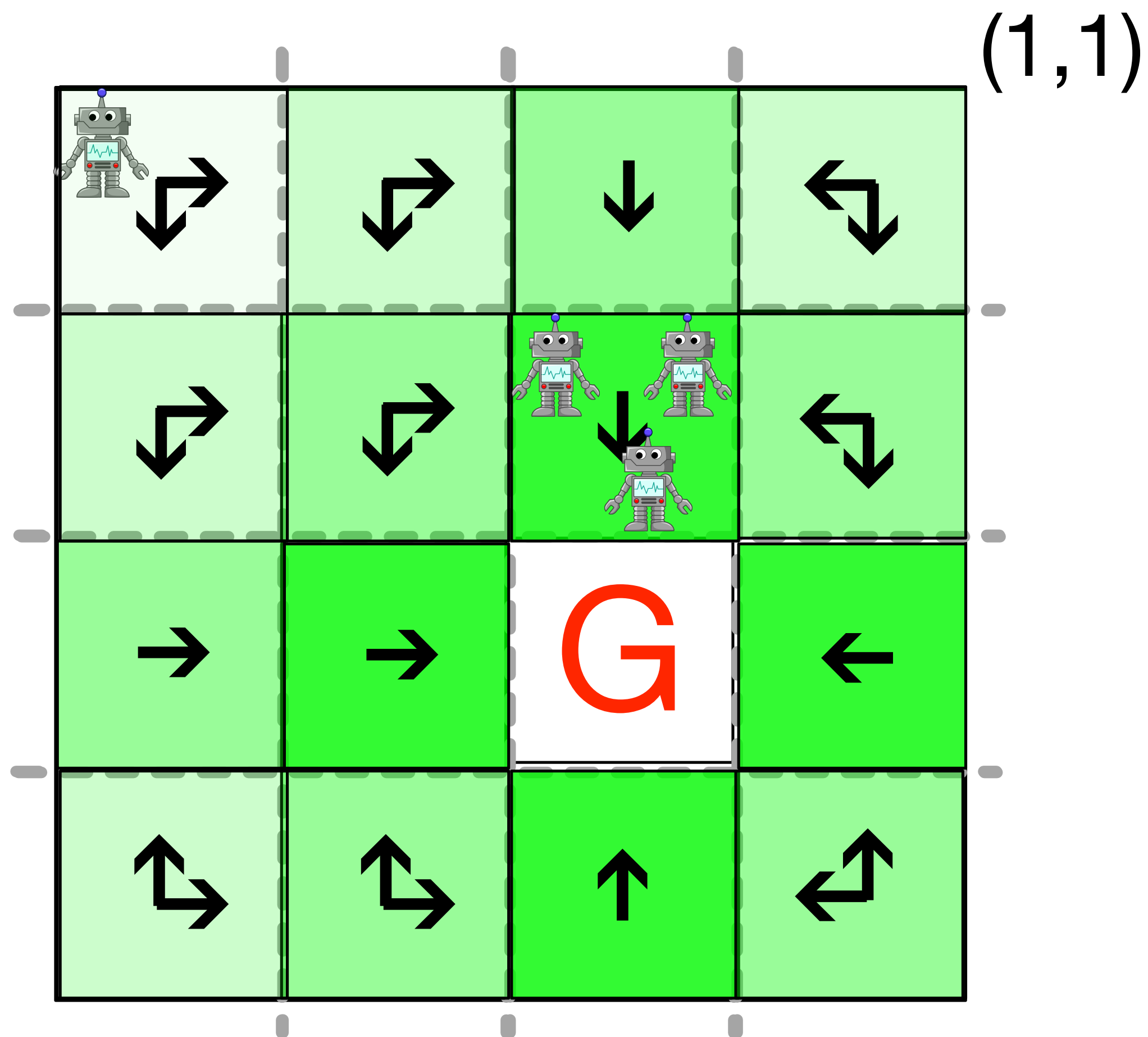
(1,1)



$$\mathbf{x}(s) = \mathbf{x}(\text{robot}) = \mathbf{x}(0.8, 0.8) =$$

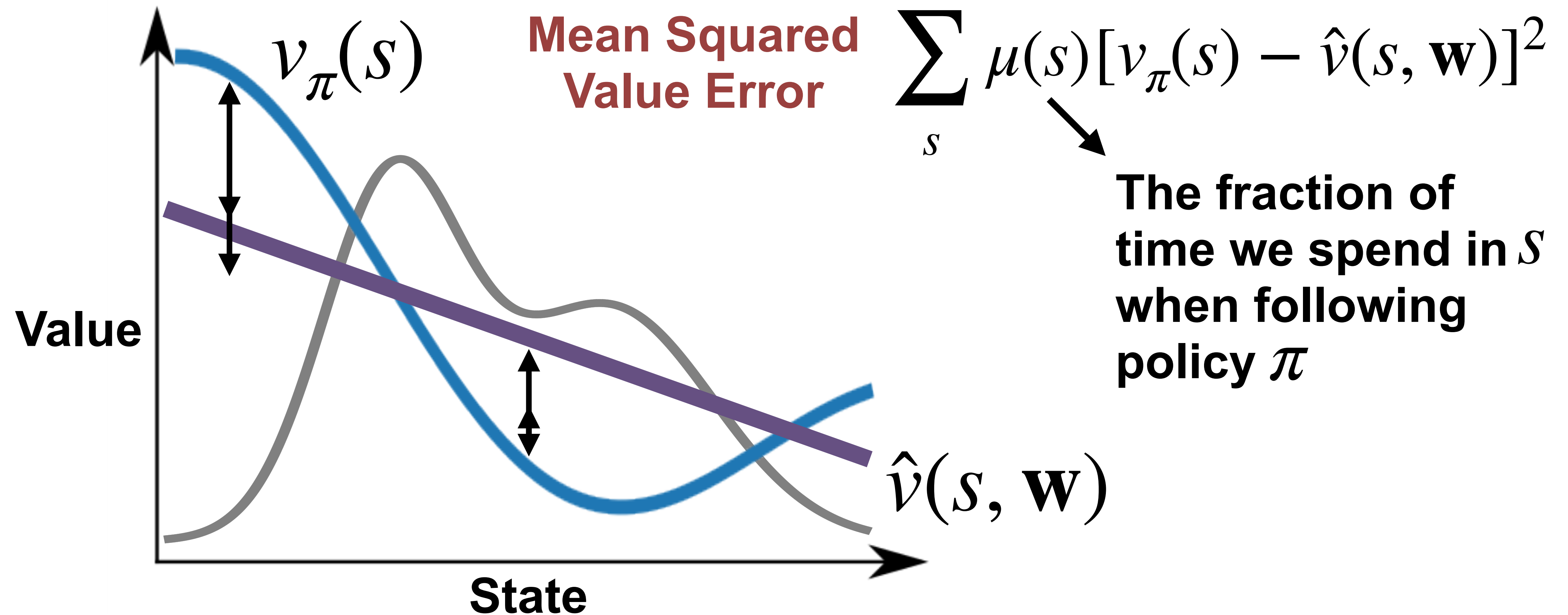
$$\begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

What might the final estimate of the value function look like with this state aggregation?



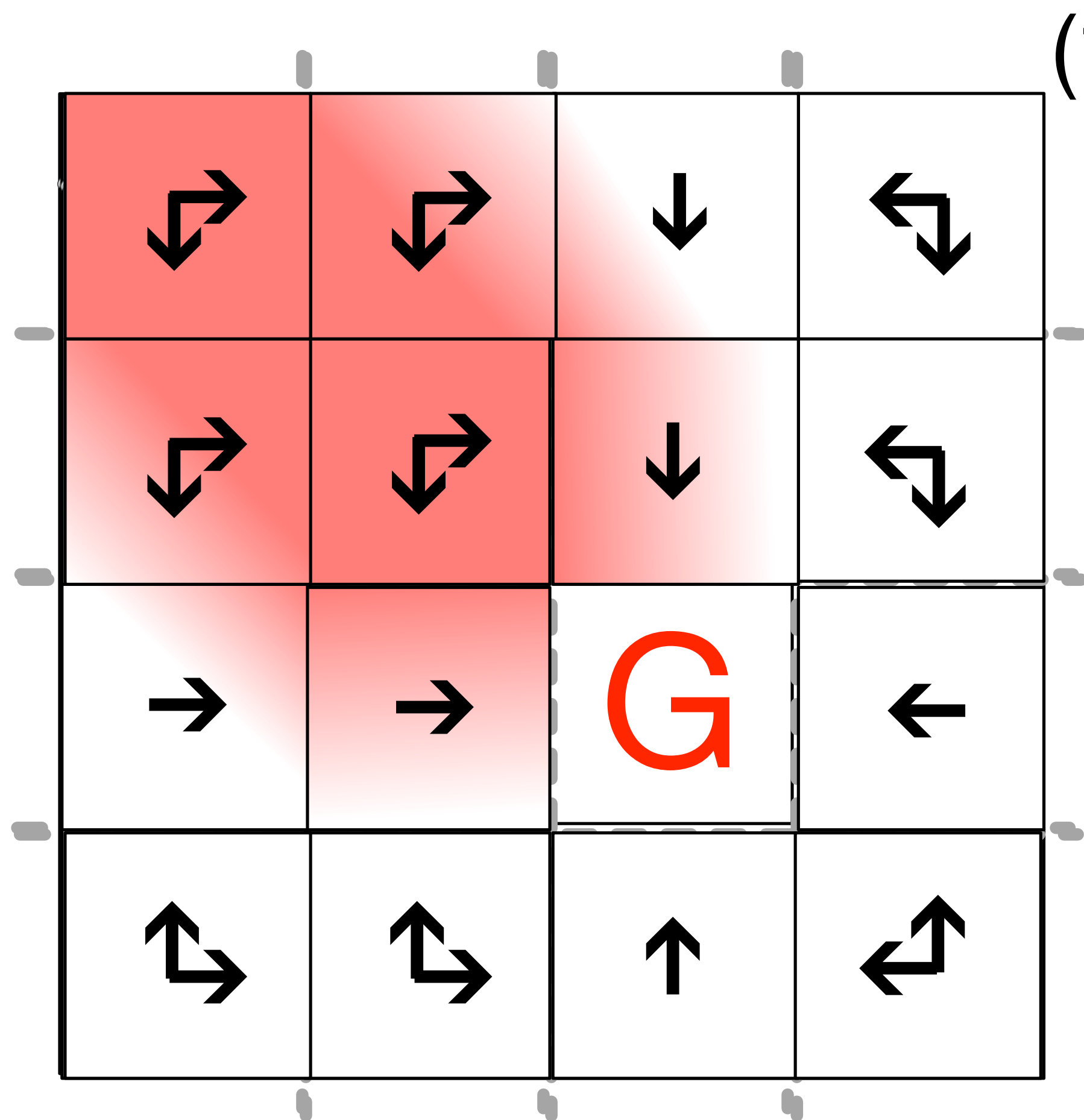
- $R = +1$ per step
- episodic, $\gamma = 1$
- agent starts in the top left corner
- $\pi =$ shortest path policy
- what should $\hat{v}(s, \mathbf{w})$ look like?

The Mean Squared Value Error Objective



Question: Why didn't we use the Value Error in the tabular setting?

What might the μ (proportion of time the agent spends in each state) look like with this state aggregation?



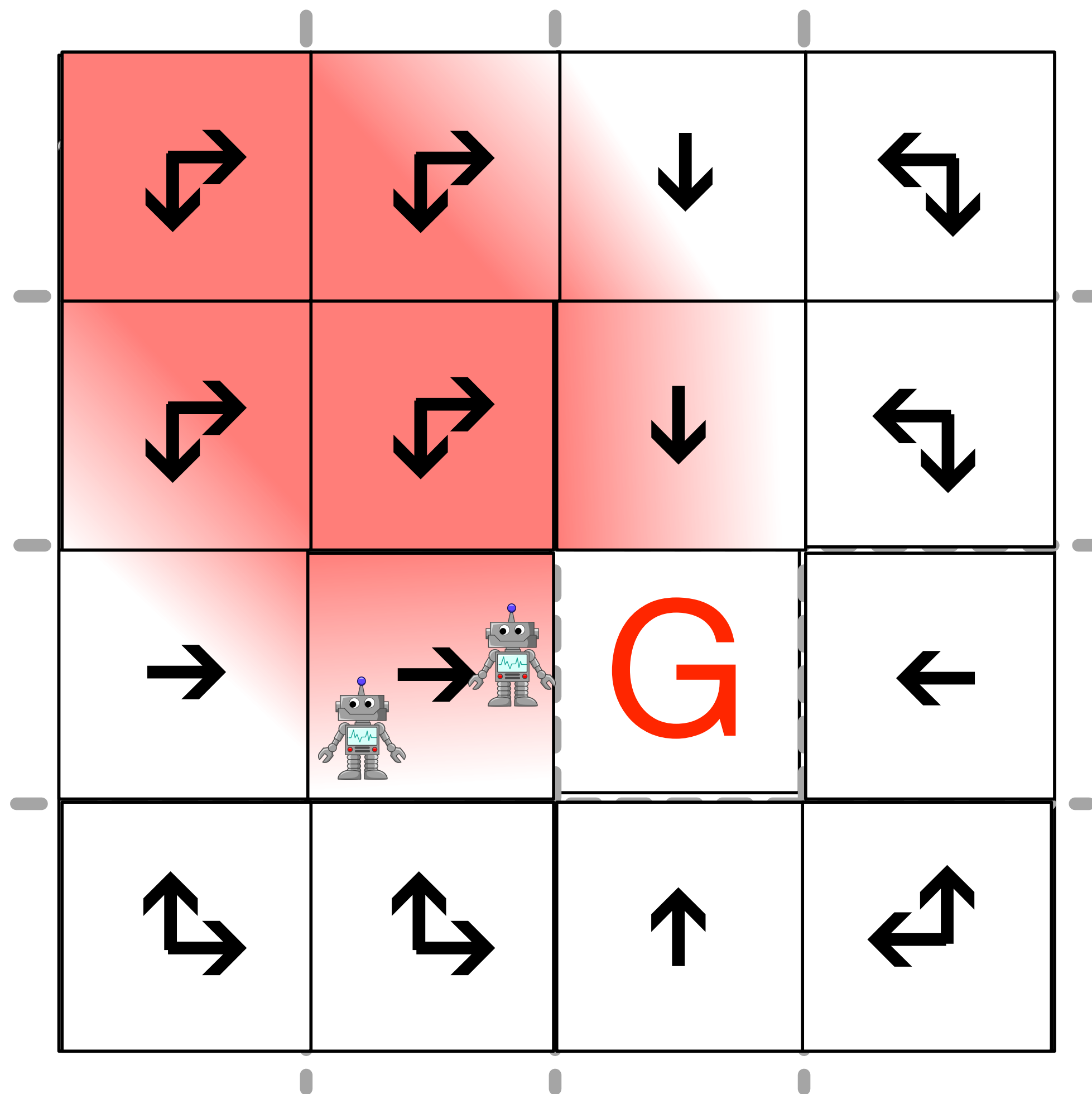
- $R = +1$ per step
- episodic, $\gamma = 1$
- agent starts in the top left corner
- π = shortest path policy

Mean Squared Value Error

$$\sum_s \mu(s) [v_\pi(s) - \hat{v}(s, \mathbf{w})]^2$$

The fraction of time we spend in s when following policy π

$\mu(s)$ Impacts how we update $\hat{v}(s, \mathbf{w})$

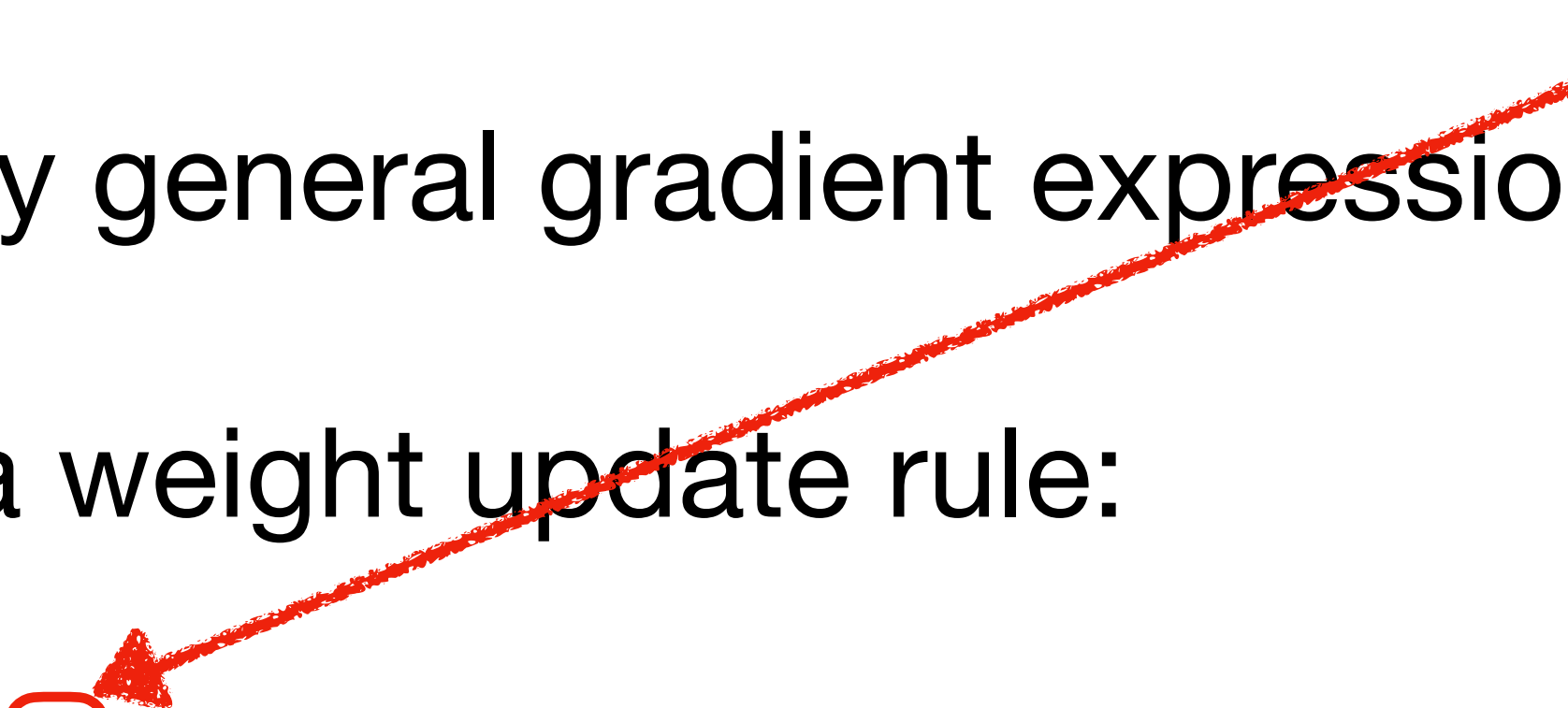


Mean Squared Value Error

$$\sum_s \mu(s) [v_\pi(s) - \hat{v}(s, \mathbf{w})]^2$$

The fraction of time we spend in S when following policy π

The usual recipe for gradient descent

1. Specify a function approximation architecture (parametric form of value function)
 2. Write down your objective function
 3. Take the derivative of objective function with respect to the weights
 4. Simplify general gradient expression for your parametric form
 5. Make a weight update rule:
 - $\mathbf{w} = \mathbf{w} - \alpha \text{ gradient}$
- 

lets try out the recipe

1. Specify a function approximation architecture (parametric form of value function)

- We will use **State Aggregation**
 - so the **features** are always **binary** with only a single active feature that is not zero
 - the value function is a **linear function**
 - that is, we query the value function by a simple procedure:
 1. **query the features** for the current state
 2. take the inner product between the features and the weights

$$v_{\pi}(s) \approx \hat{v}(s, \mathbf{w}) \doteq \mathbf{w}^{\top} \mathbf{x}(s) \doteq \sum_{i=1}^n w_i \cdot x_i(s)$$

2. Write down your objective function

- We will use the value error

$$\begin{aligned}\overline{VE}(\mathbf{w}) &\doteq \sum_{s \in \mathcal{S}} \mu(s) [v_{\pi}(s) - \hat{v}(s, \mathbf{w})]^2 \\ &= \sum_{s \in \mathcal{S}} \mu(s) [v_{\pi}(s) - \boxed{\mathbf{w}^T \mathbf{x}(s)}]^2\end{aligned}$$

state
aggregation



3. Take the gradient of objective function with respect to the weights

$$\nabla \overline{V E}(\mathbf{w}) = \nabla \sum_{s \in \mathcal{S}} \mu(s) [v_{\pi}(s) - \mathbf{w}^T \mathbf{x}(s)]^2$$

3. Take the gradient of objective function with respect to the weights

$$\begin{aligned}\nabla \overline{VE}(\mathbf{w}) &= \nabla \sum_{s \in \mathcal{S}} \mu(s) [v_{\pi}(s) - \mathbf{w}^T \mathbf{x}(s)]^2 \\ &= \sum_{s \in \mathcal{S}} \mu(s) \nabla [v_{\pi}(s) - \mathbf{w}^T \mathbf{x}(s)]^2 \\ &= - \sum_{s \in \mathcal{S}} \mu(s) 2[v_{\pi}(s) - \mathbf{w}^T \mathbf{x}(s)] \nabla \mathbf{w}^T \mathbf{x}(s)\end{aligned}$$

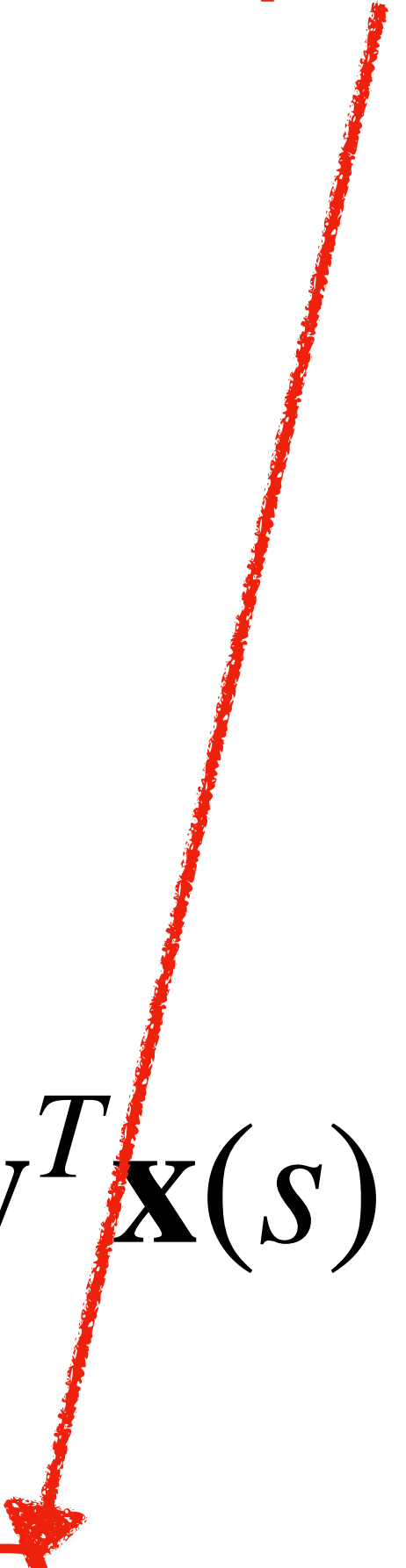
4. Simplify the general gradient expression to be specific for your parametric form

$$\nabla \mathbf{w}^T \mathbf{x}(s) = \mathbf{x}(s)$$

The gradient of the inner product is just \mathbf{x}

4. Simplify general gradient ...

gradient of
linear value function
approximation (state agg.)

$$\begin{aligned}\nabla \overline{VE}(\mathbf{w}) &= \nabla \sum_{s \in \mathcal{S}} \mu(s) [v_{\pi}(s) - \mathbf{w}^T \mathbf{x}(s)]^2 \\ &= \sum_{s \in \mathcal{S}} \mu(s) \nabla [v_{\pi}(s) - \mathbf{w}^T \mathbf{x}(s)]^2 \\ &= - \sum_{s \in \mathcal{S}} \mu(s) 2[v_{\pi}(s) - \mathbf{w}^T \mathbf{x}(s)] \nabla \mathbf{w}^T \mathbf{x}(s) \\ &= - \sum_{s \in \mathcal{S}} \mu(s) 2[v_{\pi}(s) - \mathbf{w}^T \mathbf{x}(s)] \mathbf{x}(s)\end{aligned}$$


5. Make weight update rule: $\mathbf{w} = \mathbf{w} - \alpha$ gradient

$$\nabla \overline{VE}(\mathbf{w}) = - \sum_{s \in \mathcal{S}} \mu(s) 2[v_{\pi}(s) - \mathbf{w}^T \mathbf{x}(s)] \mathbf{x}(s)$$



$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha 2[v_{\pi}(s) - \mathbf{w}^T \mathbf{x}(s)] \mathbf{x}(s)$$

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha [v_{\pi}(s) - \mathbf{w}^T \mathbf{x}(s)] \mathbf{x}(s)$$

Wait, Wait, Wait!! We don't have v_π

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha [\boxed{v_\pi(s)} - \mathbf{w}^T \mathbf{x}(s)] \mathbf{x}(s)$$

Let's replace it with something we do have!

Let's replace v_π with something we do have!

Let's call it's replacement U_t

Whatever we use in place of v_π , it should satisfy one criteria!

$$v_\pi(s) = \mathbb{E}[U_t]$$

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha[\boxed{v_\pi(s)} - \mathbf{w}^T \mathbf{x}(s)] \mathbf{x}(s)$$

Whatever we use in place of v_π , it should satisfy one criteria!

$$v_\pi(s) = \mathbb{E}[U_t]$$

We know one such replacement, that meets this criteria!

$$U_t \doteq G_t$$

A sample of the return!!

Since we are using sample returns we have a Monte Carlo algorithm!

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha[G_t - \mathbf{w}^T \mathbf{x}(s)]\mathbf{x}(s)$$

Monte Carlo Policy Evaluation for finding v_π

Semi-gradient TD(0) for estimating $\hat{v} \approx v_\pi$

Input: the policy π to be evaluated

Input: a differentiable function $\hat{v} : \mathcal{S}^+ \times \mathbb{R}^d \rightarrow \mathbb{R}$ such that $\hat{v}(\text{terminal}, \cdot) = 0$

Algorithm parameter: step size $\alpha > 0$

Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)

Loop for each episode:

 Initialize S

 Loop for each step of episode:

 Choose $A \sim \pi(\cdot | S)$

 Take action A , observe R, S'

$\mathbf{w} \leftarrow \mathbf{w} + \alpha [R + \gamma \hat{v}(S', \mathbf{w}) - \hat{v}(S, \mathbf{w})] \nabla \hat{v}(S, \mathbf{w})$

$S \leftarrow S'$

 until S is terminal

Question: What is different compared to Tabular TD(0)?

Discussion posts

- Book suggests VE objective might not be the best for control. Why? Alternatives?
 - Average reward + PG methods
- Chapter mentions partial observability. How does FA help with that? Do we need RNNs?
 - Maybe ... Imagine classical conditioning on the rabbit with eye pokes
- Why use function approx in tabular problems?
 - Generalization is good! 1000 state walk is small. Would FA be better?

- What's better linear FA or Deep NNs?
 - Case study in hearts; Case study in Atari; Case study in GO
- Do we have a true gradient descent TD
 - Yes! Is it better? No!
- I'd like to get some detailed reasons why we should use "semi-gradient method". For me, it feels like we are using an unstable and incomplete method for calculating gradients.
 - Is GD the ultimate algorithm?
- Equation 9.12 shows a closed form solution for the weights corresponding to the TD fixed point. Can we solve this equation to get the weights instead of doing (semi-)gradient descent? Would this be more computationally expensive?
 - The alg is called least squares TD or LSTD

- Would target nets help TD with FA?
 - Not that I have seen. Why are target nets more important for q-learning
- Where does μ come from? Can we design it?
 - The policy π . Remember we are doing on-policy policy evaluation
- State aggregation is defined as a form of "linear function approximation". Could we think of using a small bottle-neck layer in a neural network (as discussed in C3M2) as an example of non-linear state aggregation? Since it's a form of simplifying the representation space we have for the value function.
 - Could this then be said about all learned reps?
- Do people use state agg in practice?
 - More generally, do people do feature engineering?

- Does weight initialization make a big impact on perf
 - Yes. Linear we can do opt init. Non-linear: open research to find good inits
- What about RMSprop, Adam, and generally changing alpha based on learning?
 - Or having $\alpha[i]_t$
- Is μ still taken as the on policy distribution (fraction of time spent at s) if we are using a planning method relying on simulated experience? Because simulated experience is a finite collection of trajectories, wouldn't this lead to over-fitting to the first few encountered states early on?
 - Consider Dyna or ER
- Does TD with non-linear FA converge?
 - We have a counter example

- The state distribution ($\mu(s)$) is chosen to be the fraction of time spent in each state. If we think about catastrophic states, like the cliff in cliff-walking example, they have large negative rewards and control methods would quickly update the policy to not visit these states. This would reduce their μ value, but since these states have huge consequences we wouldn't want our estimates of their values to be wrong. Can we instead make μ dependent on the reward from a state? The more extreme the reward, the more important the state?
- Let's not confuse policy evaluation and control