Course 3, Module 1
On-policy Prediction with Approximation

CMPUT 397
Fall 2020

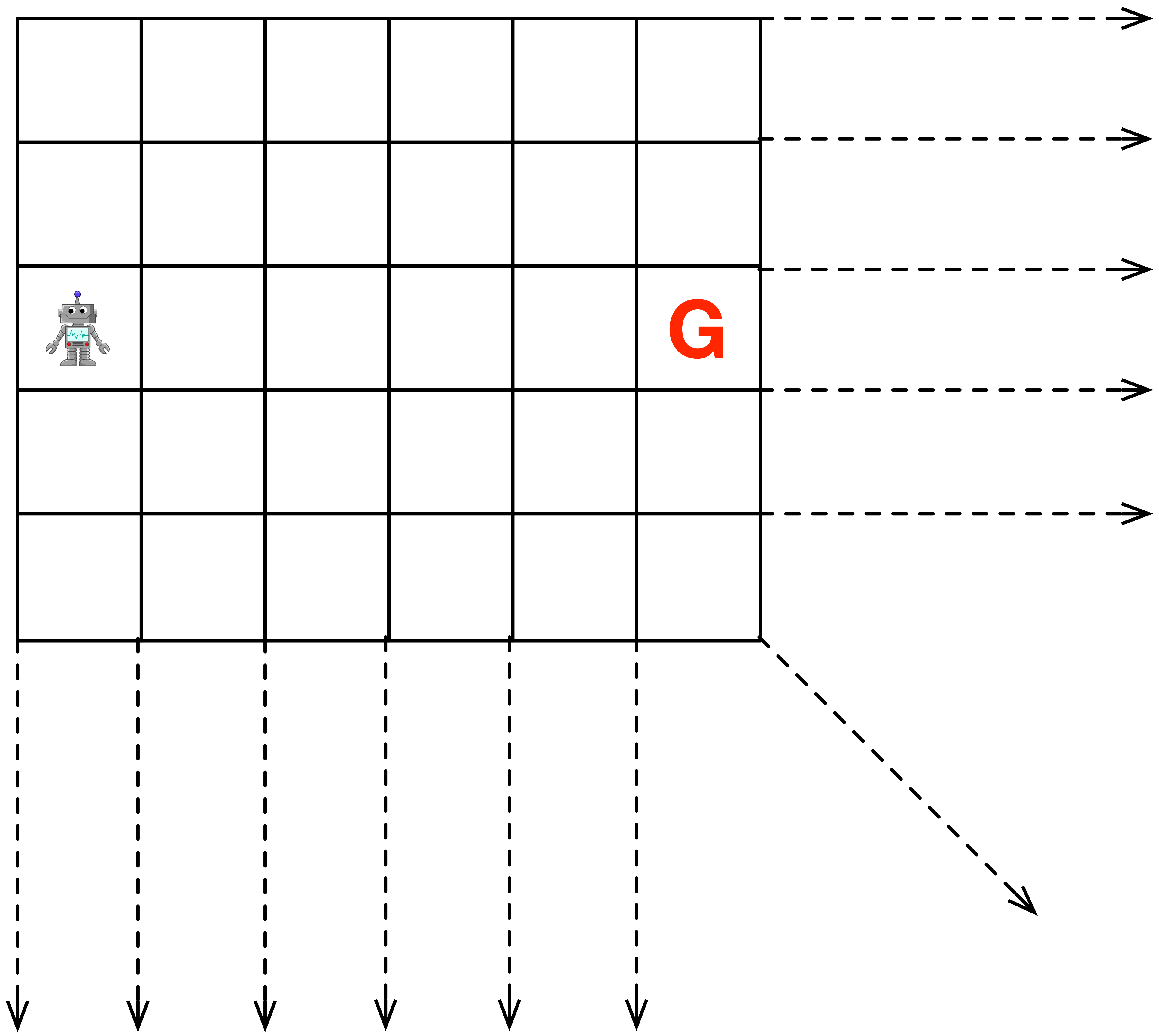
Announcements

- Discussion session this week
- Reading week next week
- Midterm when you come back from reading week
- I’ve posted a practice midterm, on eClass

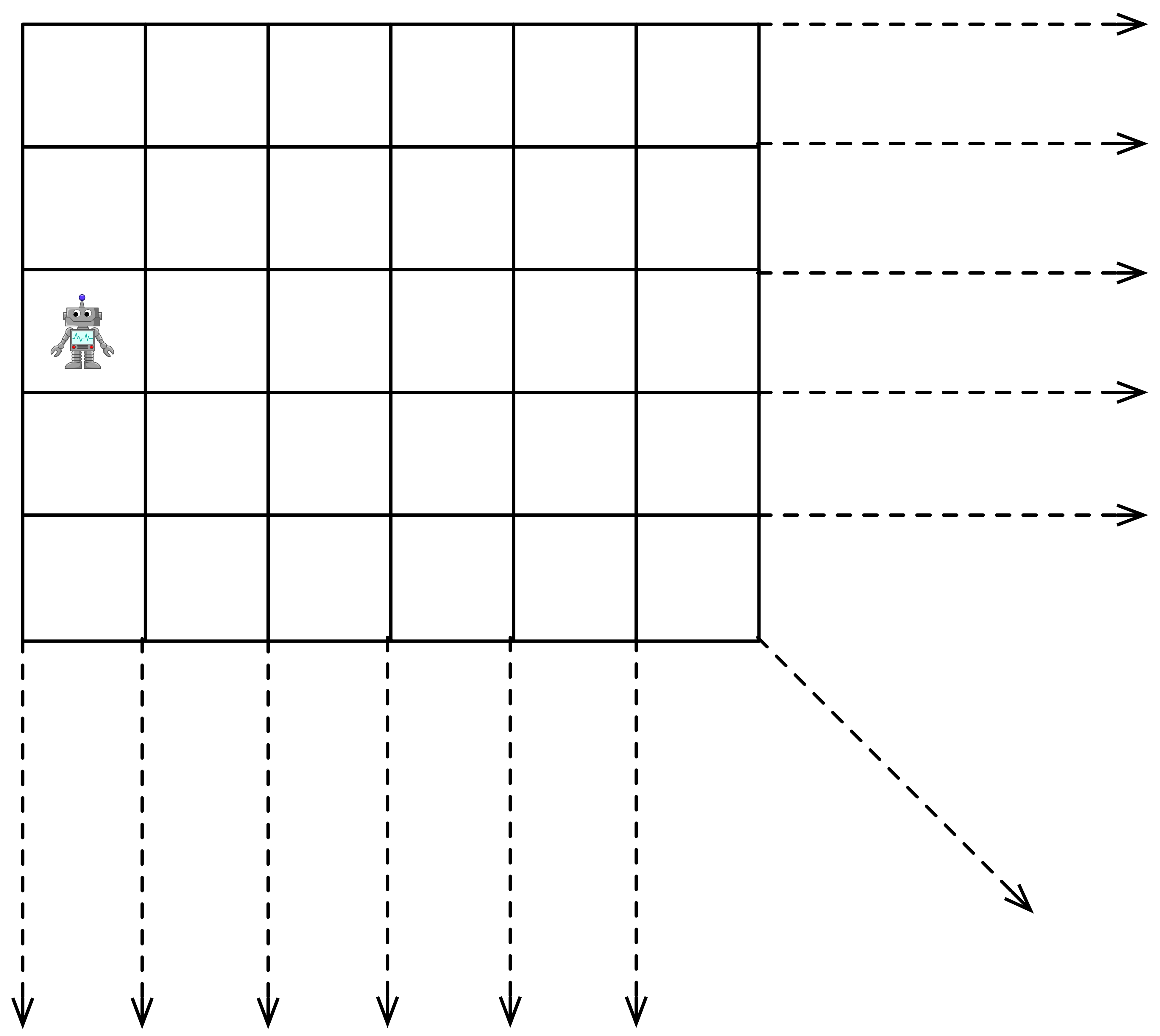
Moving to Approximation

- Our goal remains the same, as in Course 1 and Course 2
- But now we cannot represent value functions perfectly
    - because the space is too big
- Course 3 is about how to extend our algorithm to approximate value functions

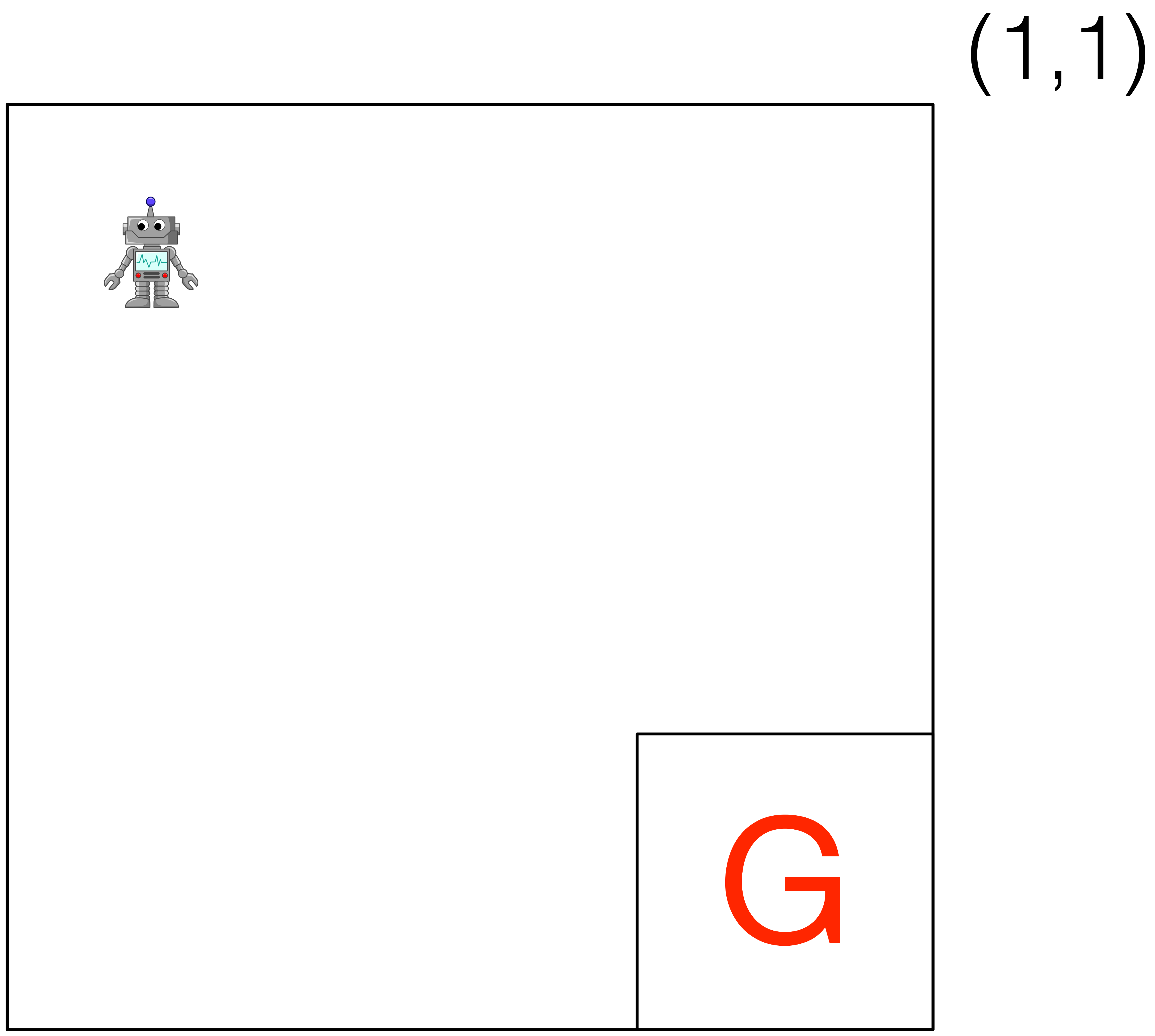
Imagine a huge state space



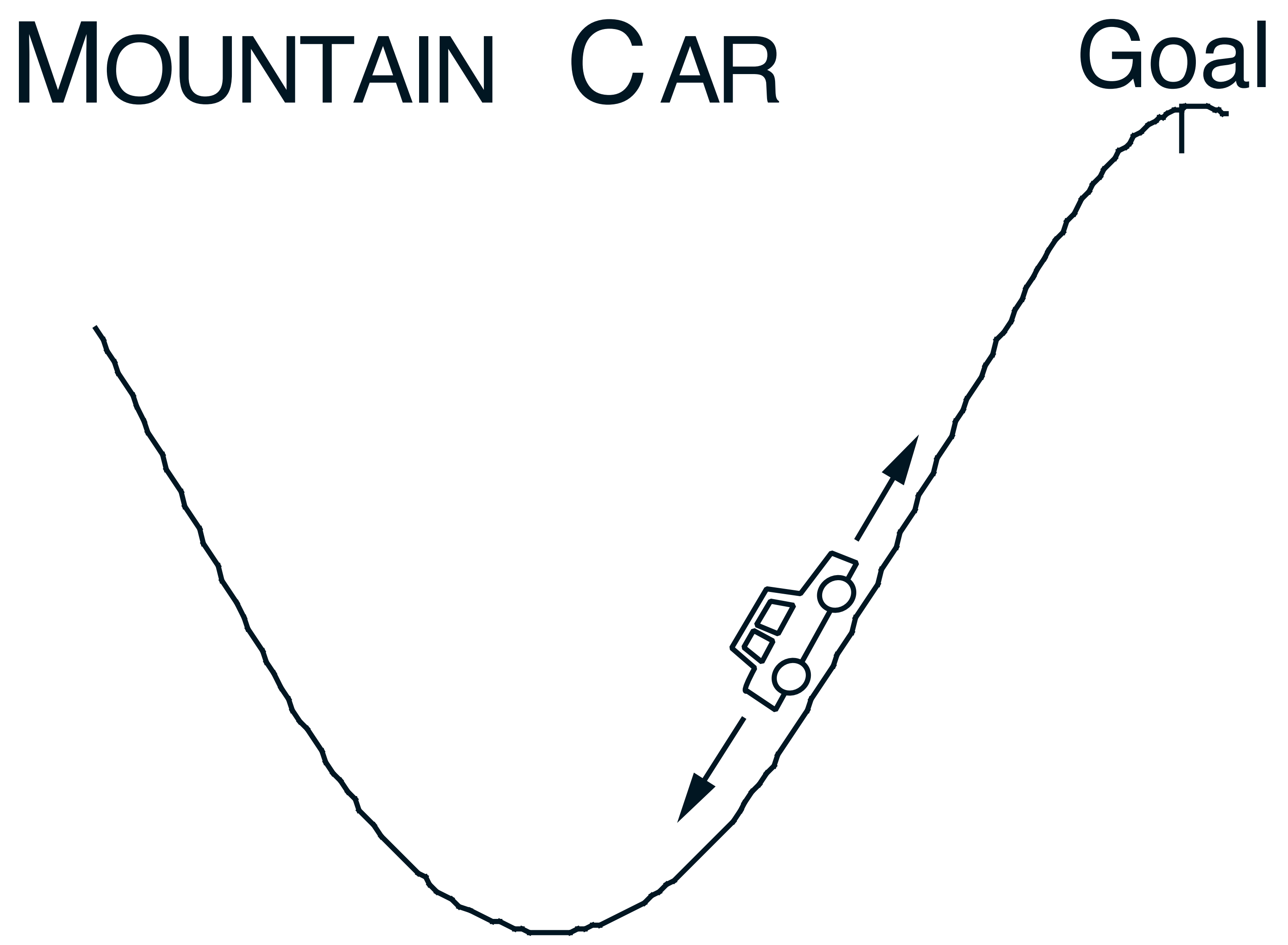
Imagine a huge state space



Imagine a continuous state space



Another continuous state domain



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

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

Video 1: Moving to Parameterized Functions

- **Using parameterized functions to represent value functions.** From tables of values to more general functions over states

- Goals:

  - Understand how we can use parameterized functions to approximate values.

  - Explain linear value function approximation.

  - Recognize that the tabular case is a special case of linear value function approximation

  - Understand that there are many ways to parameterize an approximate value function.

$$V(s) \approx v_\pi(s)$$

$$V(s) \approx v_\pi(s)$$

$$\cancel{V(s)} \approx v_\pi(s) \approx \hat{v}(s, \mathbf{w})$$

$$\cancel{V(s) \approx} \; v_\pi(s) \;\approx\; \hat{v}(s, \mathbf{w})$$

$$\mathbf{w} \in \mathbb{R}^d, \quad e.g., \quad \mathbf{w} = \begin{bmatrix} 2.1 \\ 0.01 \\ -1.1 \\ 1.2 \\ -0.1 \\ 0.01 \\ 4.93 \\ 0.5 \end{bmatrix}$$

parameter vector

transpose

$$\sout{V(s) \approx}\ v_\pi(s) \approx \hat{v}(s, \mathbf{w}) \doteq \mathbf{w}^\top \mathbf{x}(s)$$

inner product

$$\mathbf{w} \in \mathbb{R}^d, \quad e.g., \quad \mathbf{w} = \begin{bmatrix} 2.1 \\ 0.01 \\ -1.1 \\ 1.2 \\ -0.1 \\ 0.01 \\ 4.93 \\ 0.5 \end{bmatrix}$$

parameter vector

$$\cancel{V(s) \approx} \; v_\pi(s) \;\approx\; \hat{v}(s, \mathbf{w}) \;\doteq\; \mathbf{w}^\top \mathbf{x}(s)$$

transpose

inner product

$$\mathbf{w} \in \mathbb{R}^d, \quad e.g., \quad \mathbf{w} = \begin{bmatrix} 2.1 \\ 0.01 \\ -1.1 \\ 1.2 \\ -0.1 \\ 0.01 \\ 4.93 \\ 0.5 \end{bmatrix}, \qquad \mathbf{x}(s) = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

parameter vector

feature vector

transpose

$$\cancel{V(s)} \approx v_\pi(s) \approx \hat{v}(s, \mathbf{w}) \doteq \mathbf{w}^\top \mathbf{x}(s)$$

inner product

$$\mathbf{w} \in \mathbb{R}^d, \quad e.g., \quad \mathbf{w} = \begin{bmatrix} 2.1 \\ 0.01 \\ -1.1 \\ 1.2 \\ -0.1 \\ 0.01 \\ 4.93 \\ 0.5 \end{bmatrix}, \qquad \mathbf{x}(s) = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}, \quad \mathbf{x} : \mathcal{S} \to \mathbb{R}^d$$

parameter vector

feature vector

transpose — ith components

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

inner product

$$\mathbf{w} \in \mathbb{R}^d, \quad e.g., \quad \mathbf{w} = \begin{bmatrix} 2.1 \\ 0.01 \\ -1.1 \\ 1.2 \\ -0.1 \\ 0.01 \\ 4.93 \\ 0.5 \end{bmatrix}, \qquad \mathbf{x}(s) = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}, \quad \mathbf{x} : \mathcal{S} \to \mathbb{R}^d$$

parameter vector — feature vector

transpose

ith components

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

inner product

$$\mathbf{w} \in \mathbb{R}^d, \quad e.g., \quad \mathbf{w} = \begin{bmatrix} 2.1 \\ 0.01 \\ -1.1 \\ 1.2 \\ -0.1 \\ 0.01 \\ 4.93 \\ 0.5 \end{bmatrix}, \qquad \mathbf{x}(s) = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}, \quad \mathbf{x} : \mathcal{S} \to \mathbb{R}^d$$

parameter vector

feature vector

transpose

*i*th components

$$\cancel{V(s)} \approx v_\pi(s) \approx \hat{v}(s,\mathbf{w}) \doteq \mathbf{w}^\top \mathbf{x}(s) \doteq \sum_{i=1}^{d} w_i \cdot x_i(s) = 1.71$$

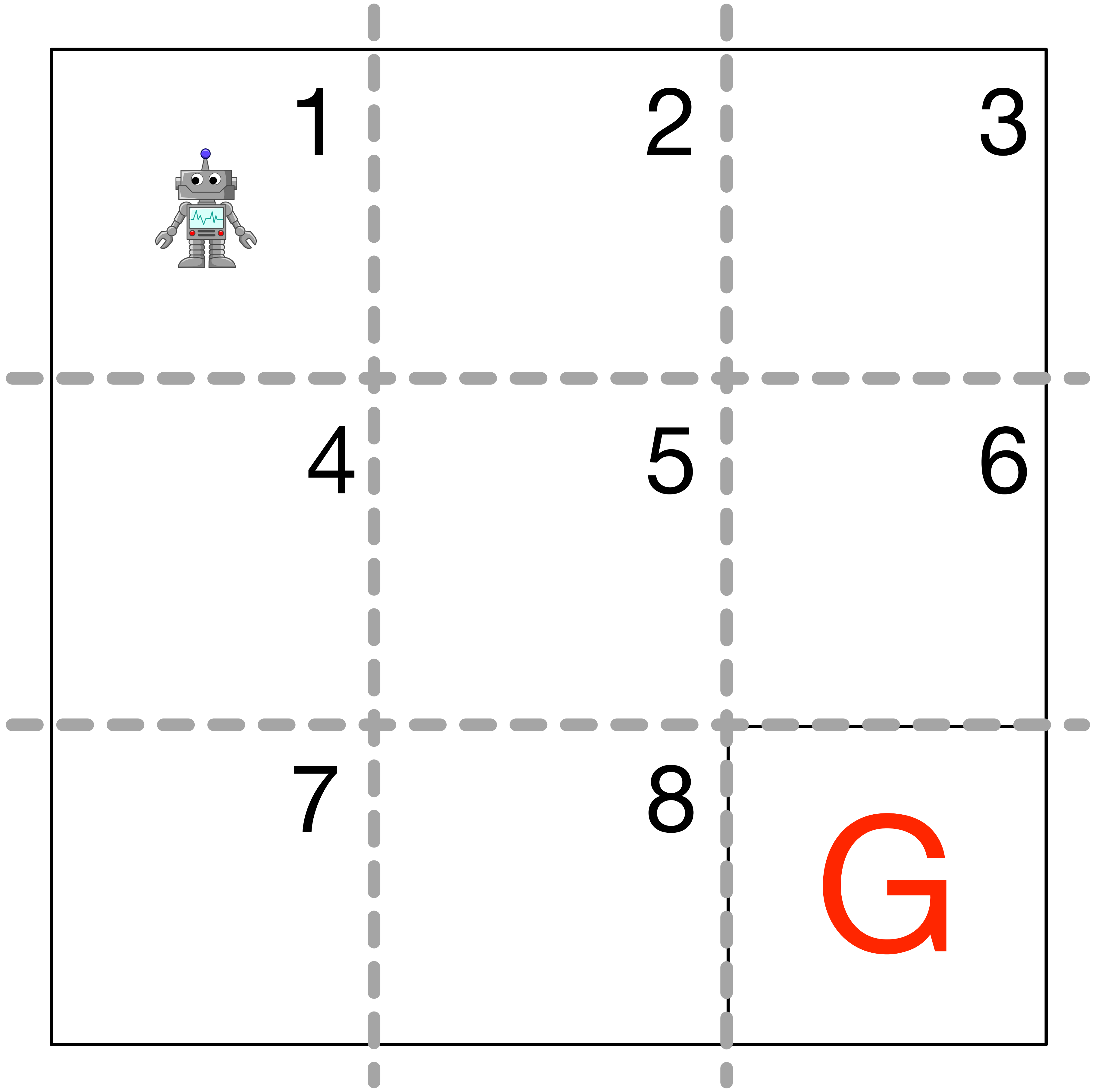
inner product

$$\mathbf{w} \in \mathbb{R}^d, \quad e.g., \quad \mathbf{w} = \begin{bmatrix} 2.1 \\ 0.01 \\ -1.1 \\ 1.2 \\ -0.1 \\ 0.01 \\ 4.93 \\ 0.5 \end{bmatrix}, \qquad \mathbf{x}(s) = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}, \quad \mathbf{x} : \mathcal{S} \to \mathbb{R}^d$$

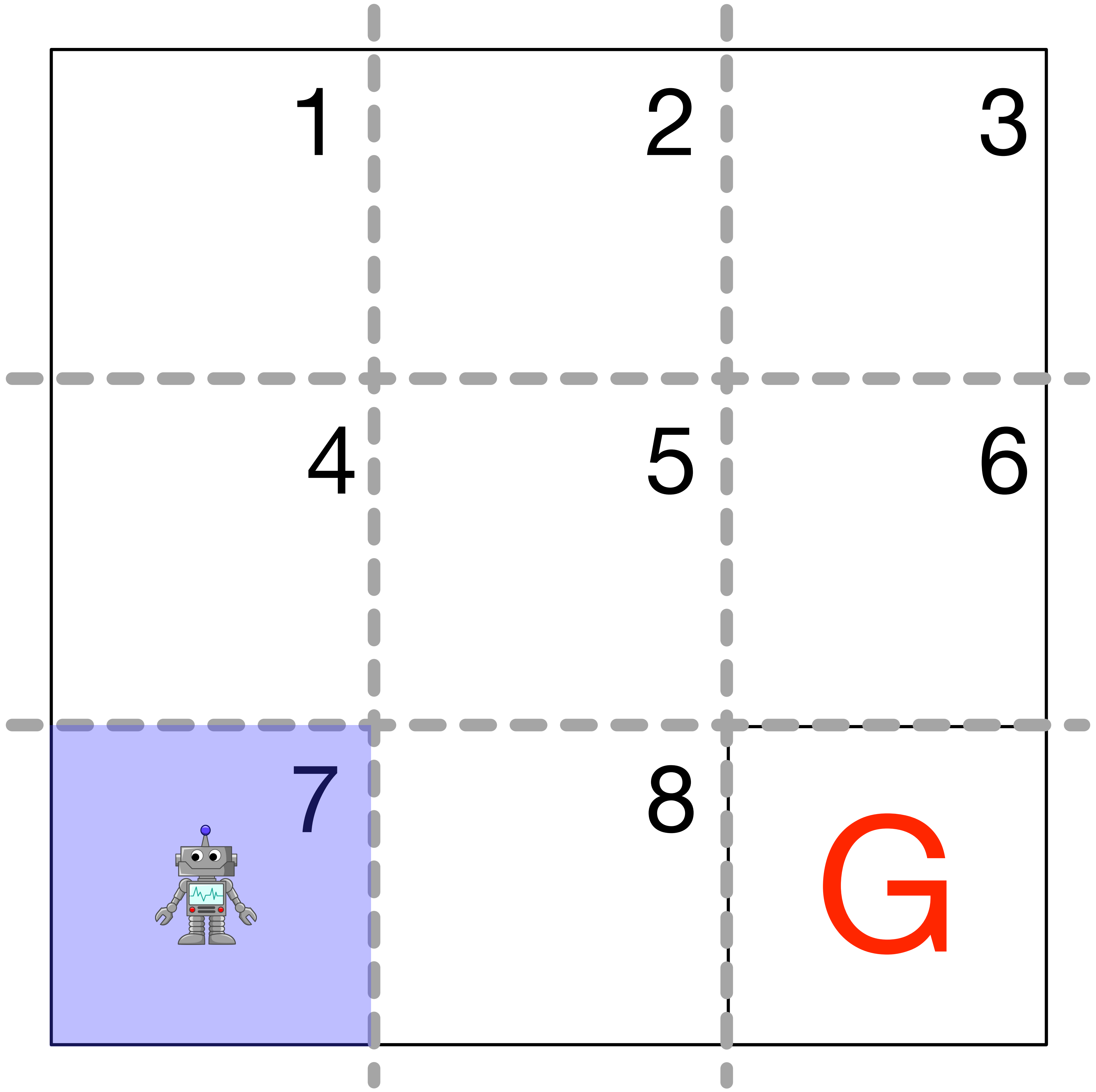
parameter vector

feature vector

Let's look at a simple state aggregation

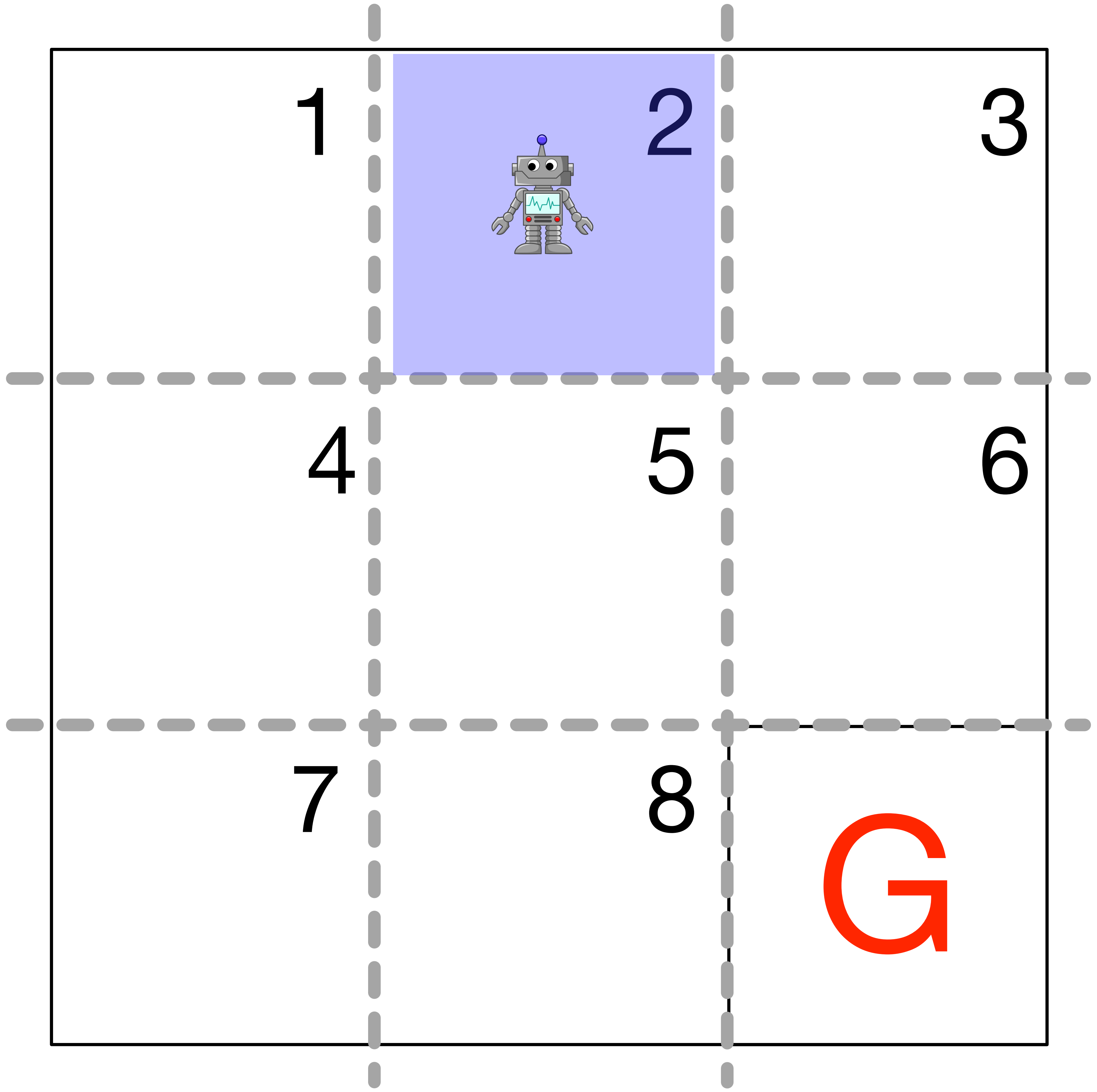


What would the feature vector be if the agent was somewhere in the bottom left?

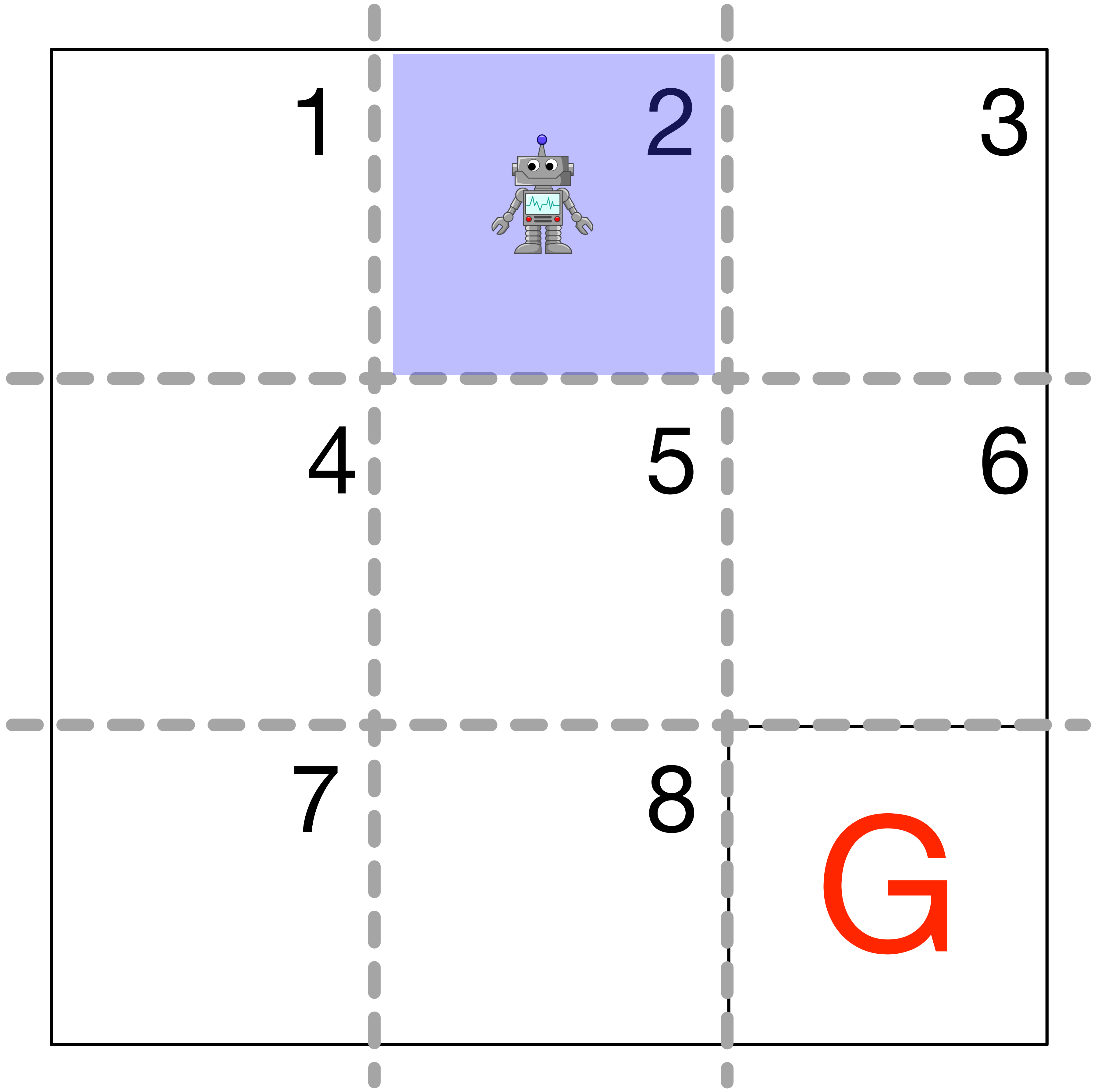


$$\boldsymbol{x}(\mathrm{s}) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

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

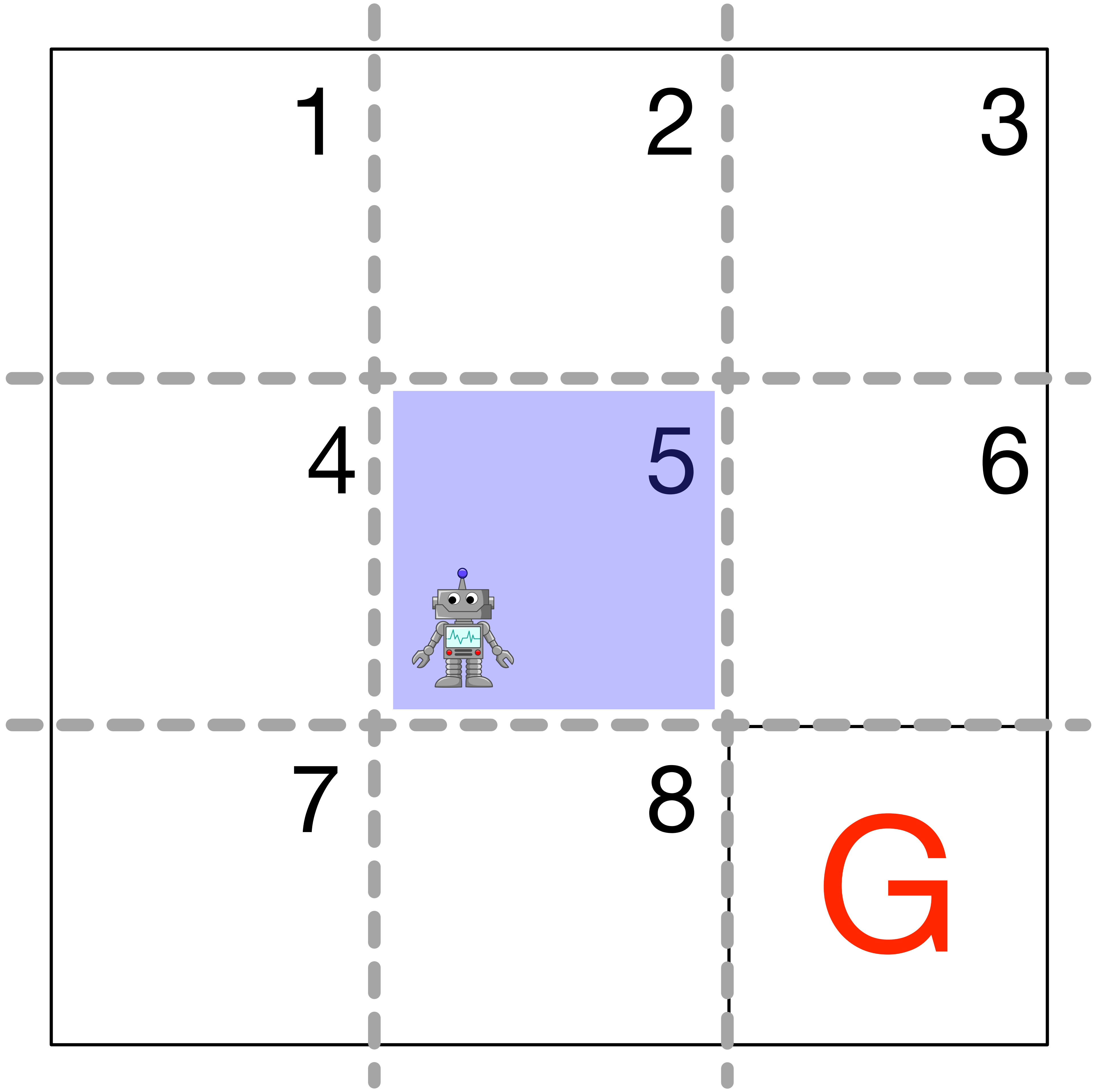


What would the feature vector be if the agent was somewhere in the top middle?



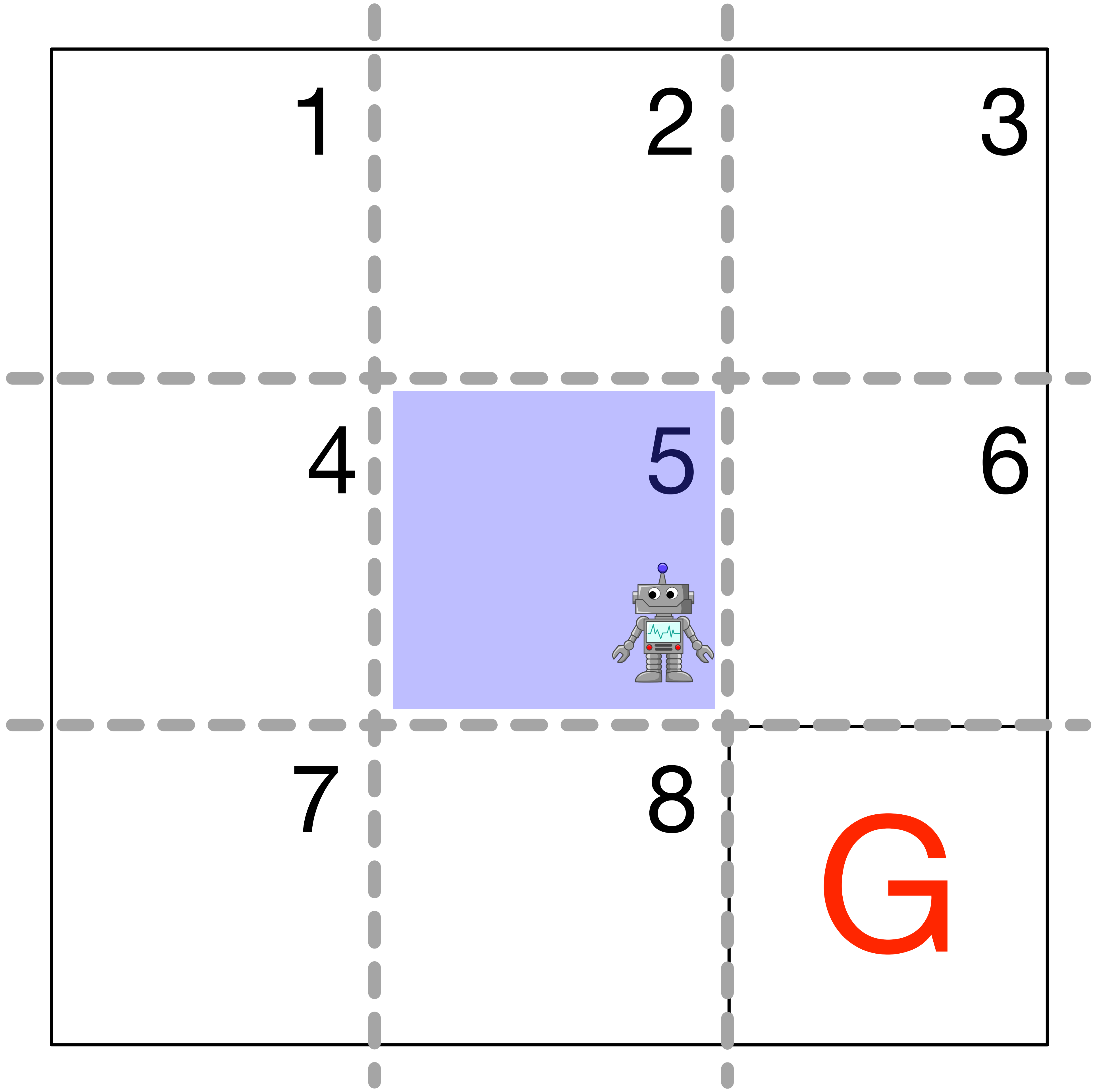
$$\boldsymbol{x}(s) = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

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



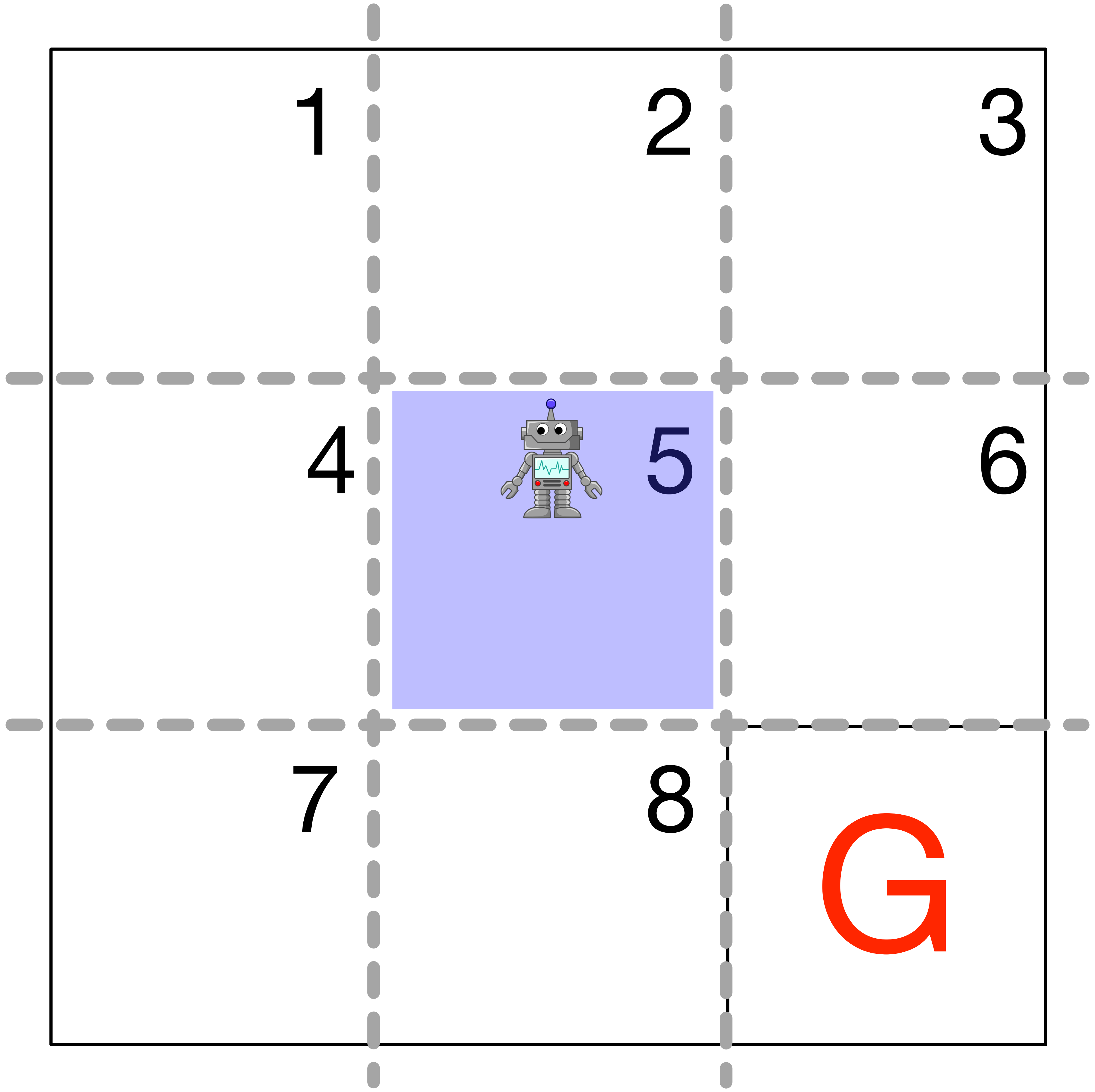
$$\boldsymbol{x}(s) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

How about here?



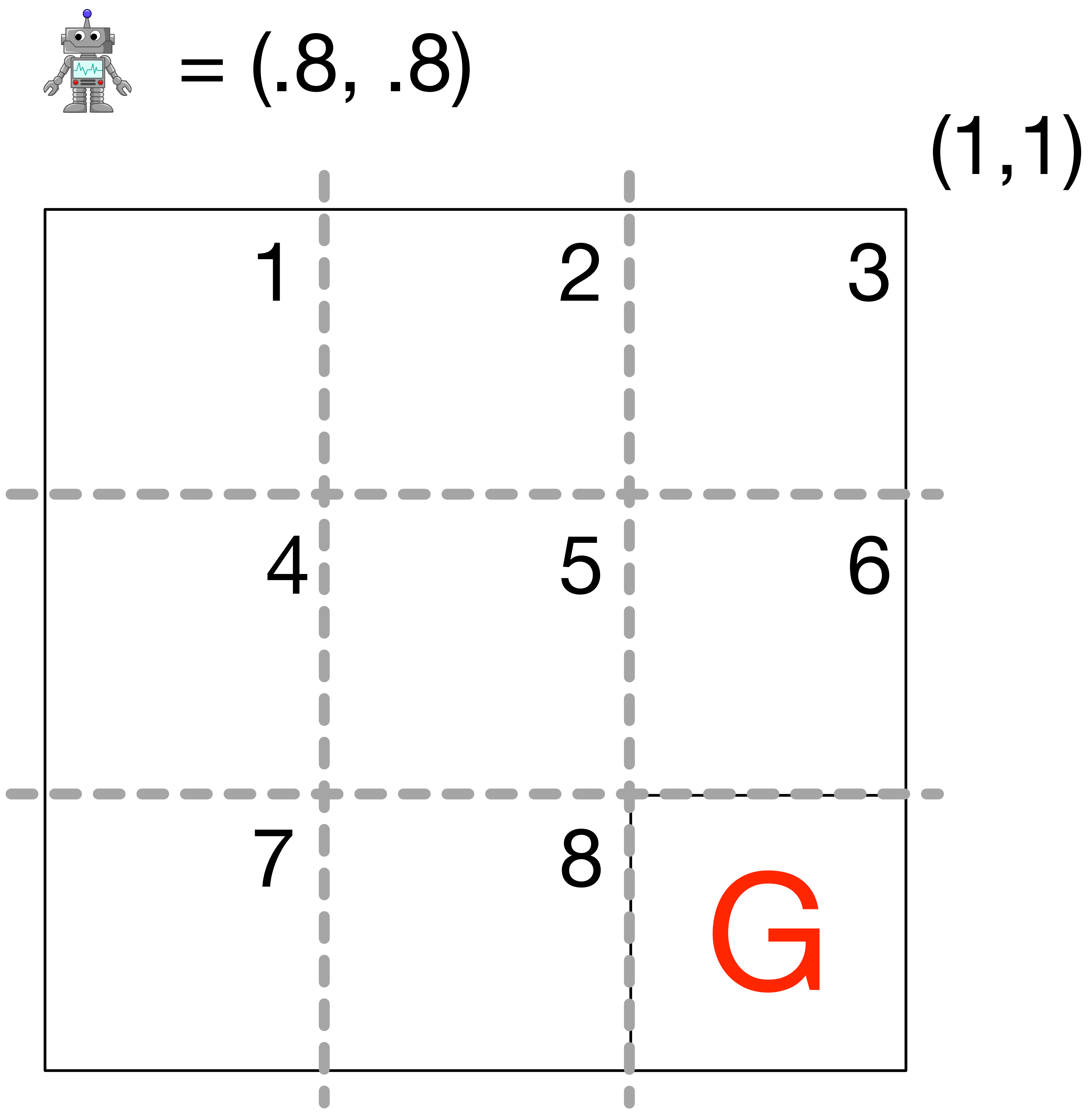
$$\boldsymbol{x}(s) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Or here?

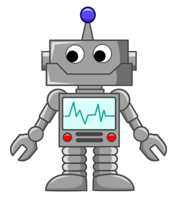


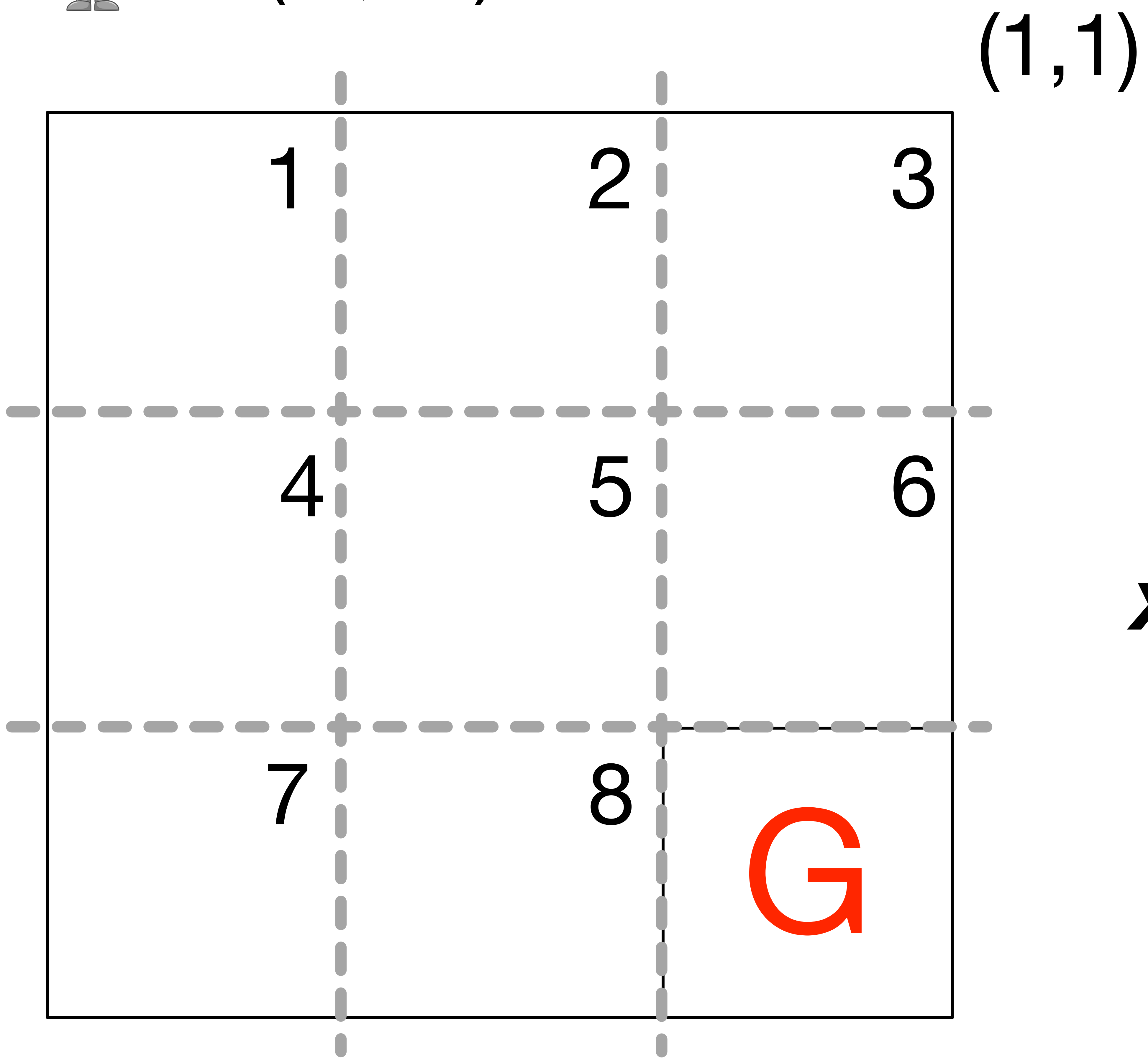
$$\boldsymbol{x}(\mathrm{s}) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

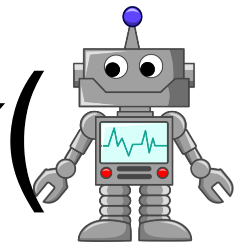
What would the feature vector be if the agent was at this point?



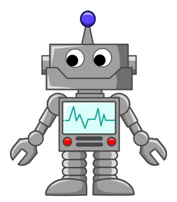
What would the feature vector be if the agent was at this point?

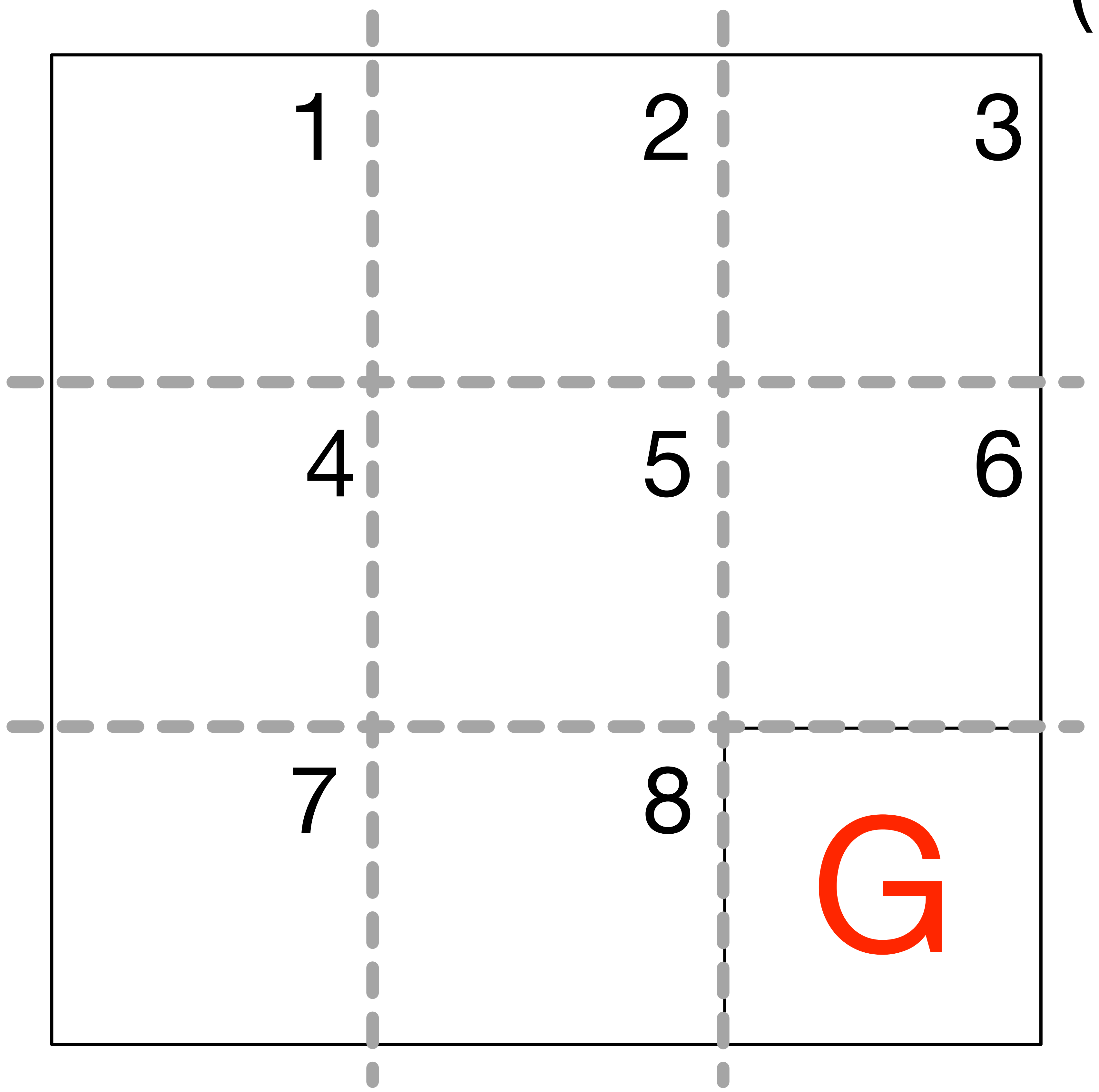
 = (.8, .8)

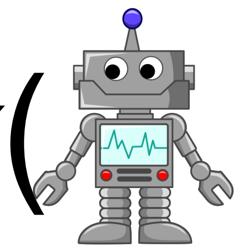


$\boldsymbol{x}$(s) = $\boldsymbol{x}$() = $\boldsymbol{x}(0.8, 0.8)$ =

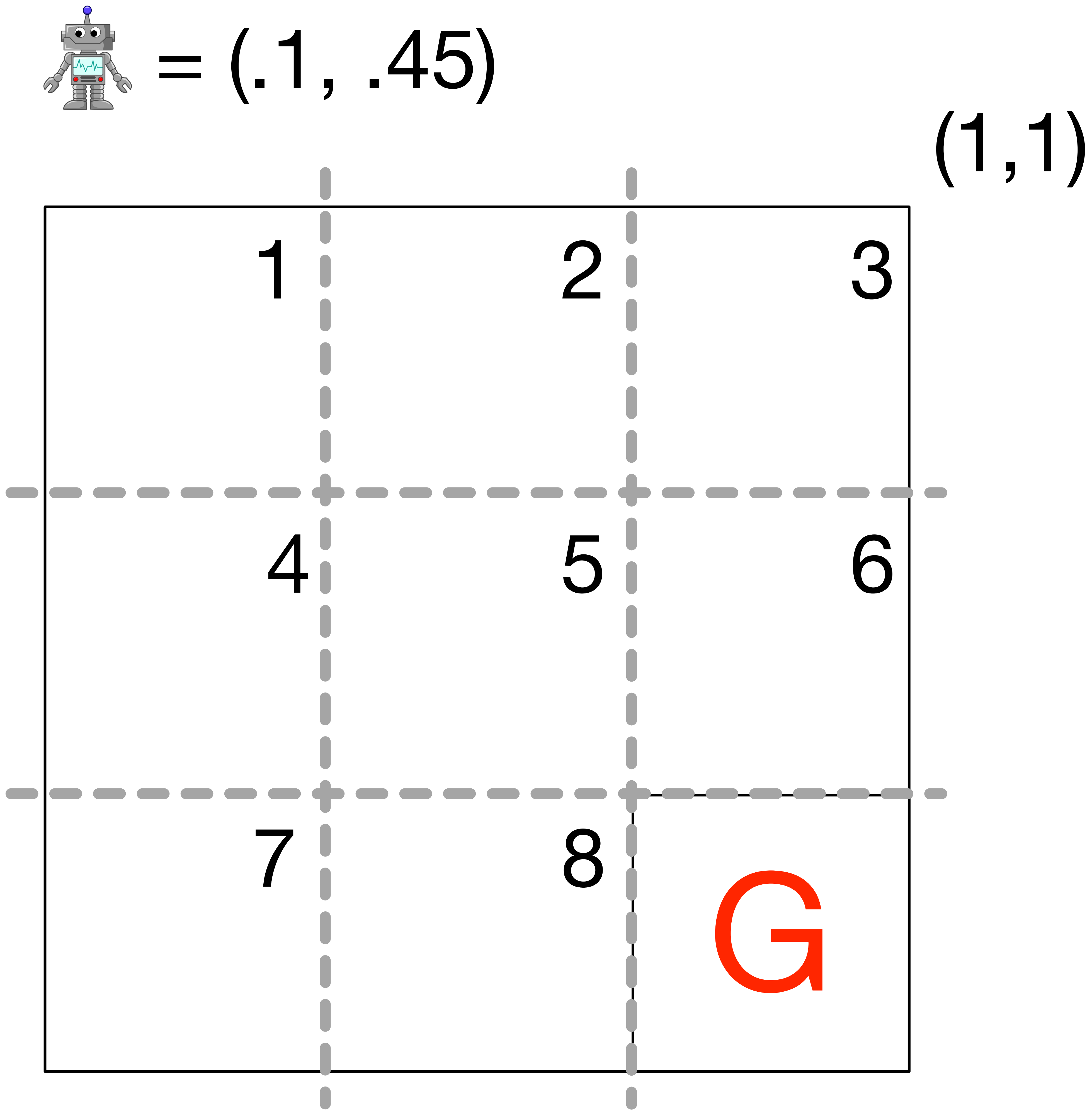
What would the feature vector be if the agent was at this point?

 = (.8, .8)

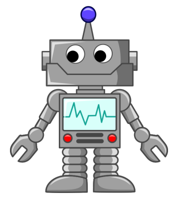


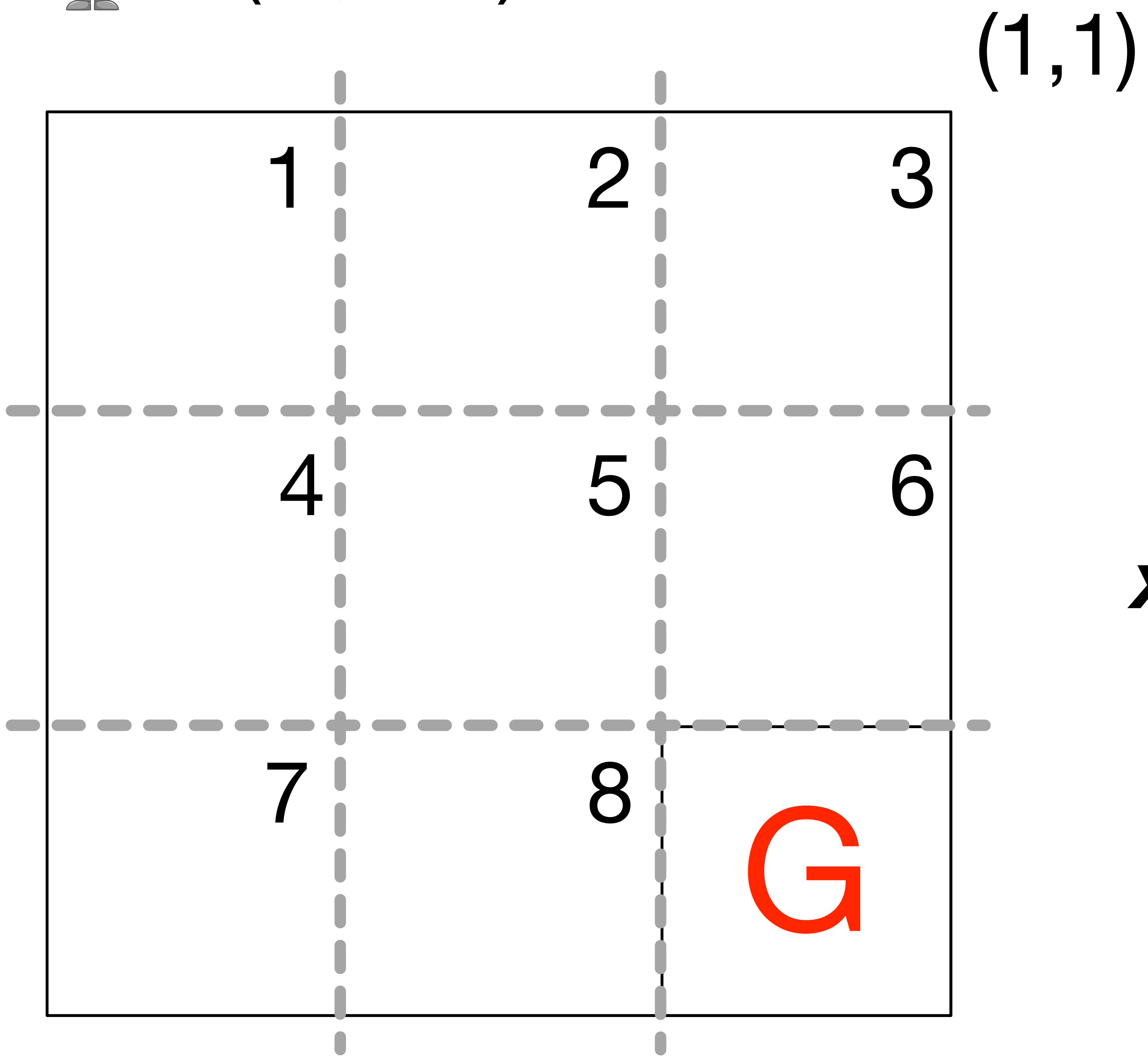
$\boldsymbol{x}(s) =$ $\boldsymbol{x}($$)$ $= \boldsymbol{x}(0.8,0.8) = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$

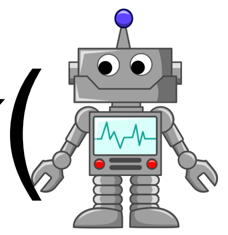
Or here?



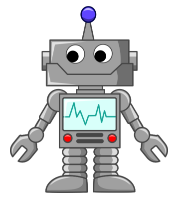
Or here?

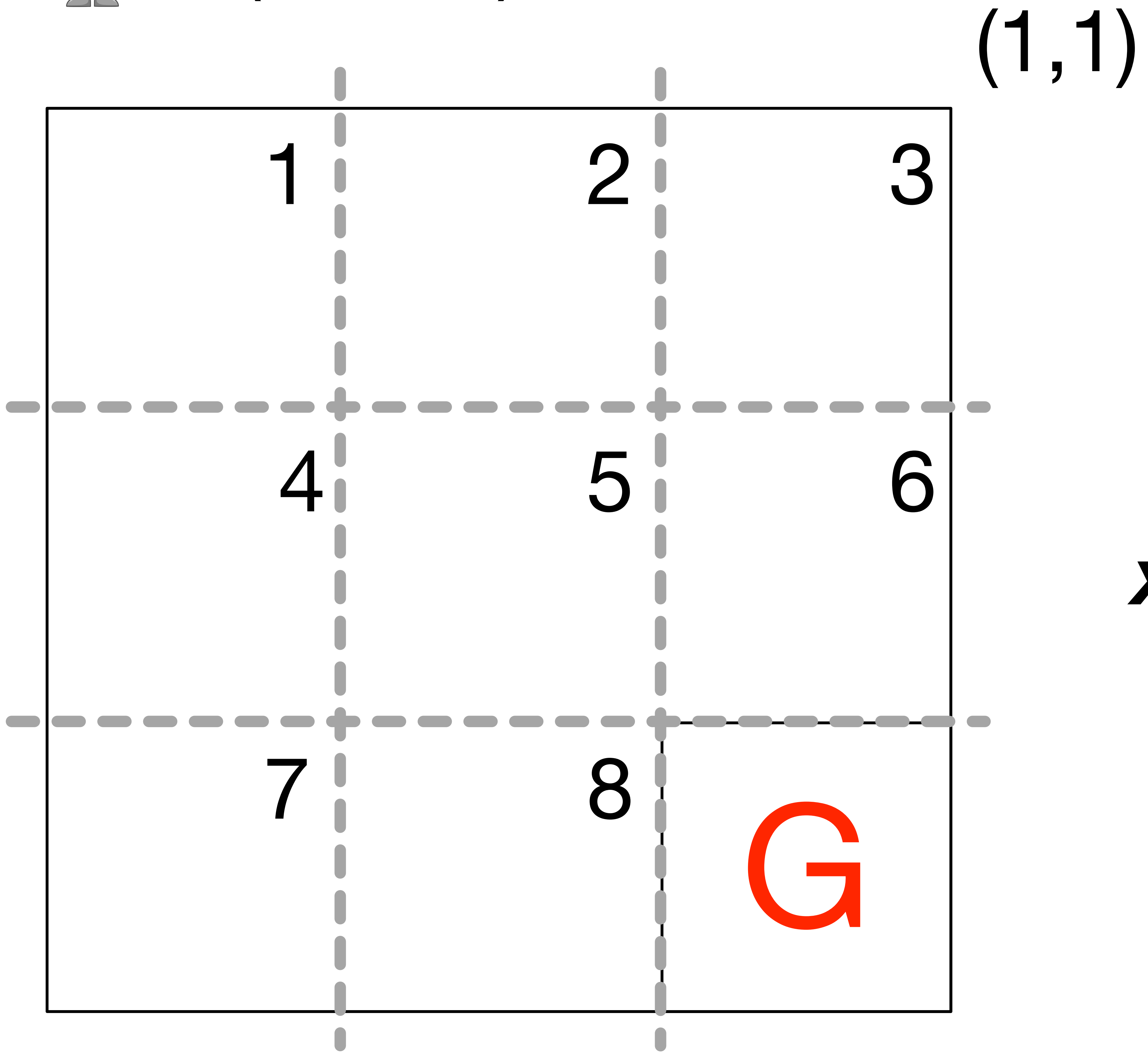
 = (.1, .45)

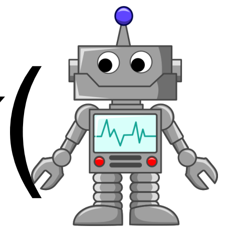


$\boldsymbol{x}$(s) = $\boldsymbol{x}$() = $\boldsymbol{x}$(0.1,0.45) =

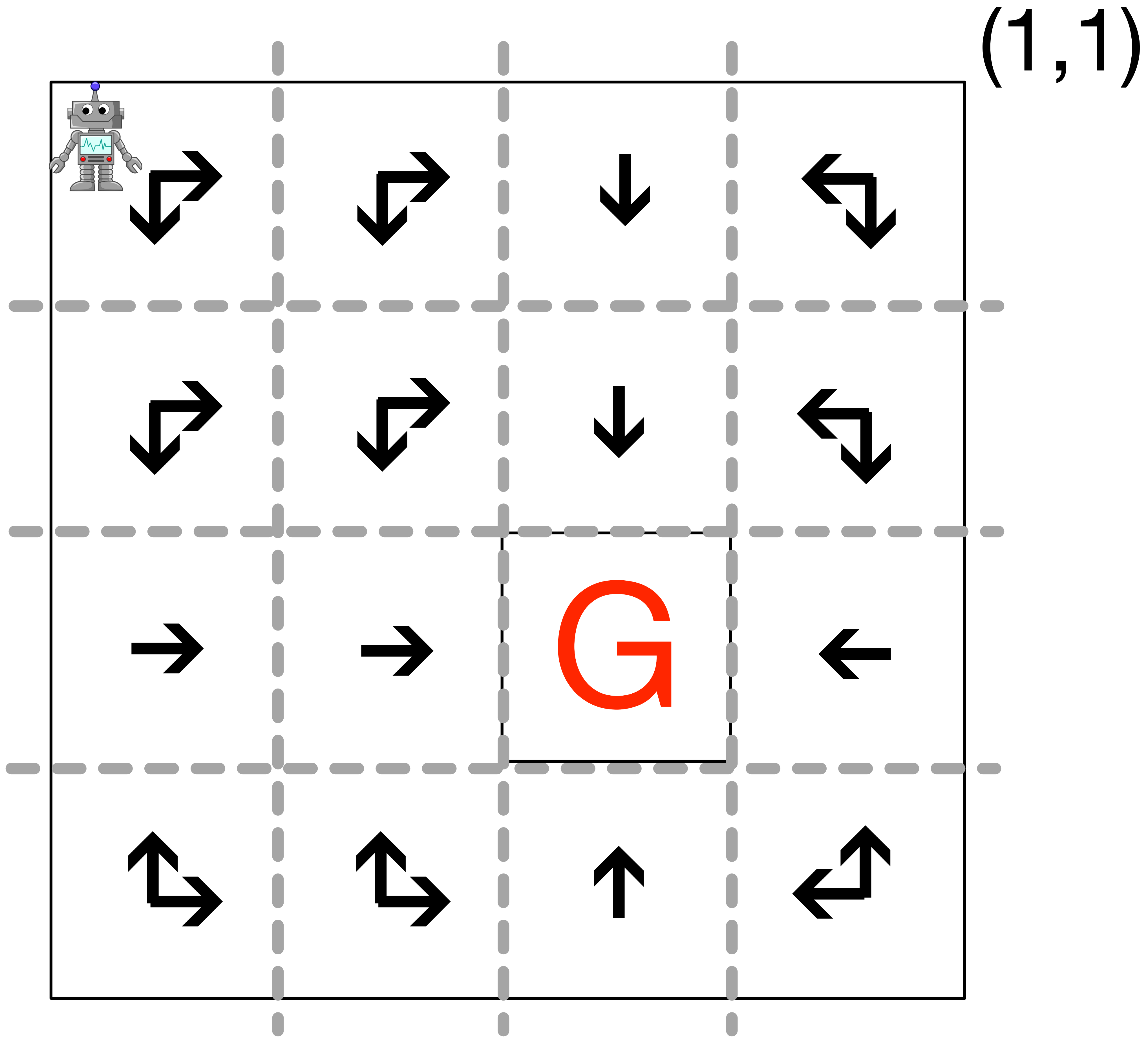
Or here?

 = (.1, .45)



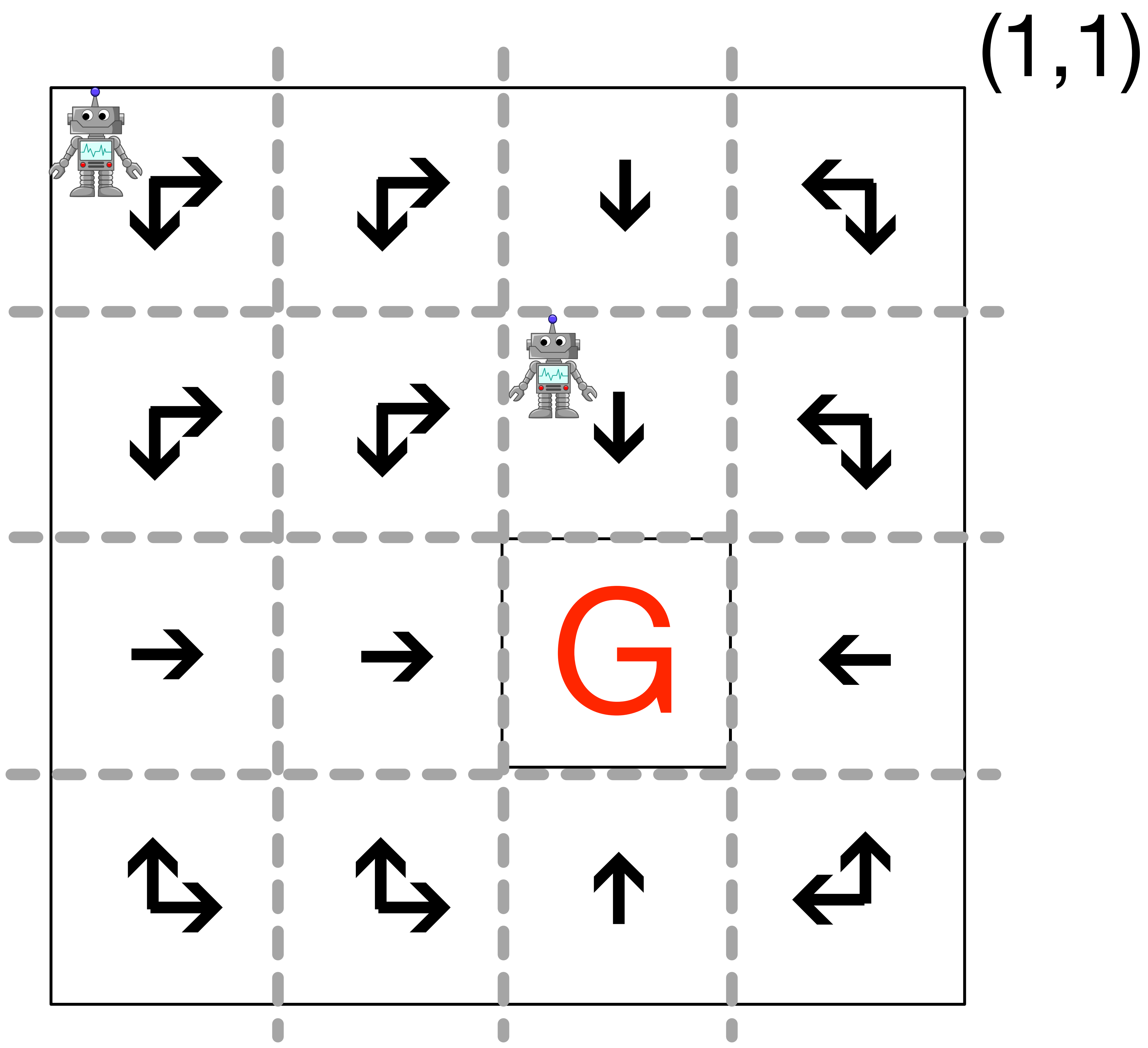
$\boldsymbol{x}(s) = \boldsymbol{x}($  $) = \boldsymbol{x}(0.1, 0.45) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$

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



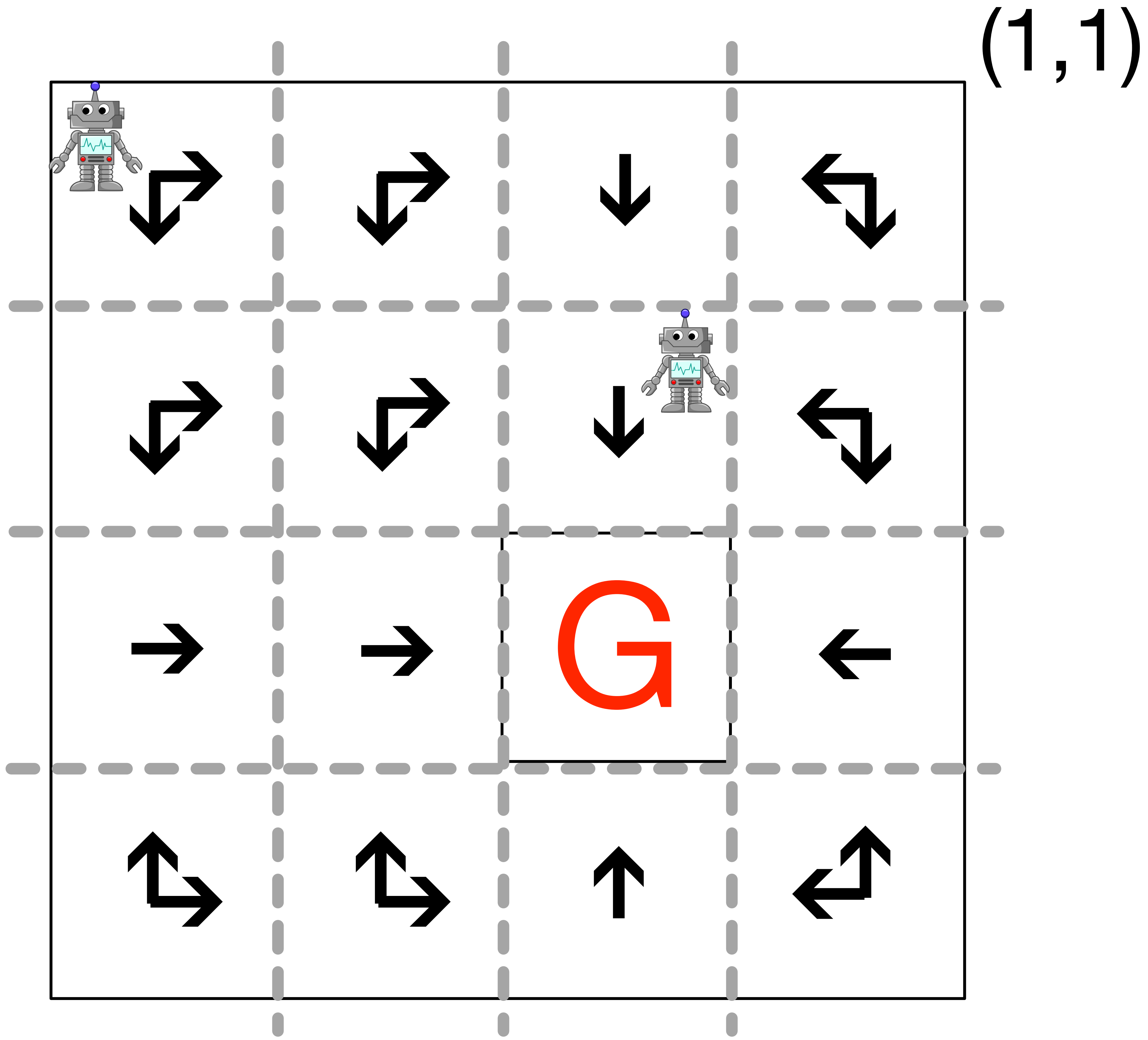
- R = +1 per step
- episodic, gamma = 1
- agent starts in the top left corner
- π = shortest path policy
- what should $\hat{v}(s, \mathbf{w})$ look like?

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



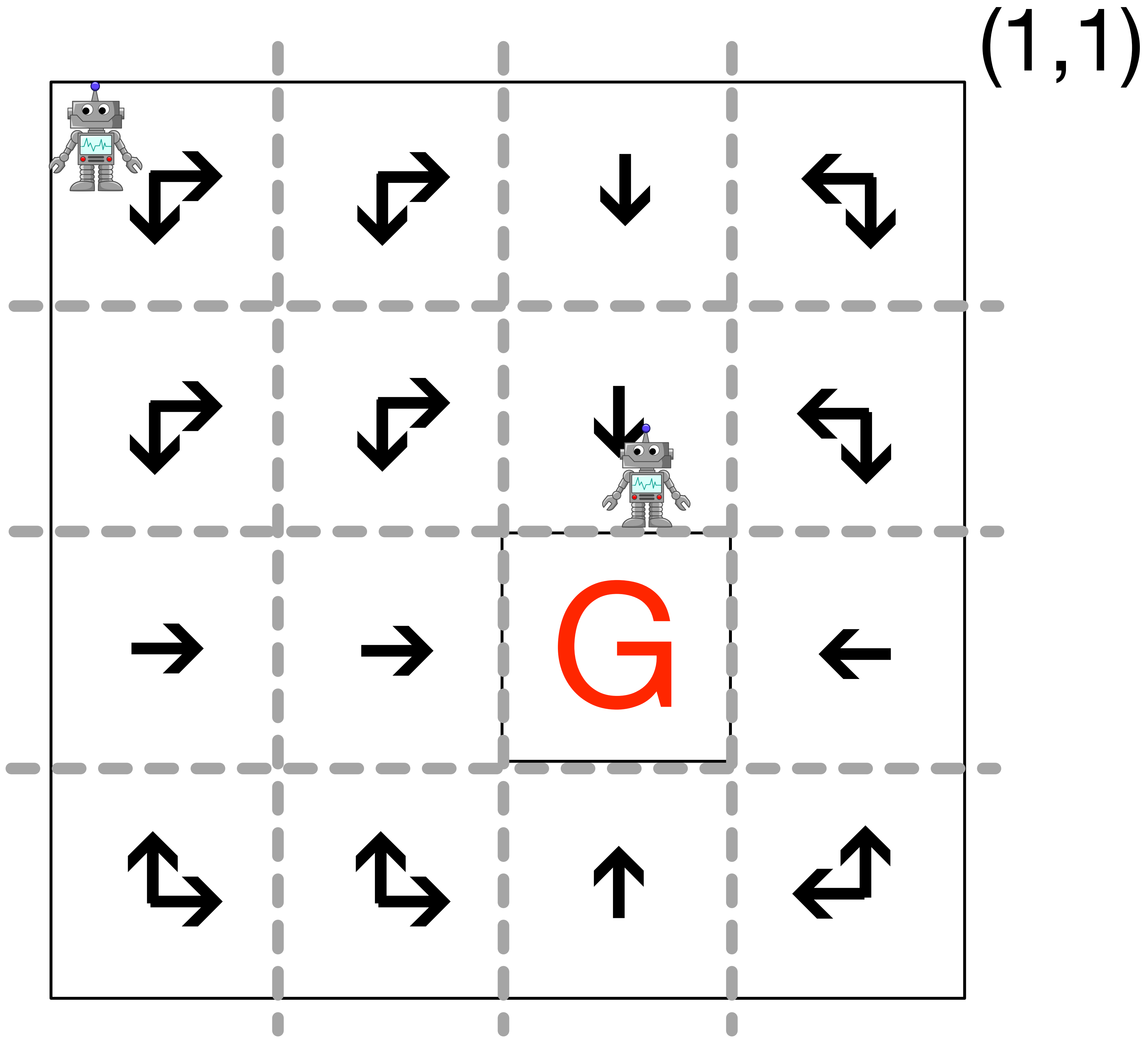
- R = +1 per step
- episodic, gamma = 1
- agent starts in the top left corner
- π = shortest path policy
- what should $\hat{v}(s, \mathbf{w})$ look like?

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



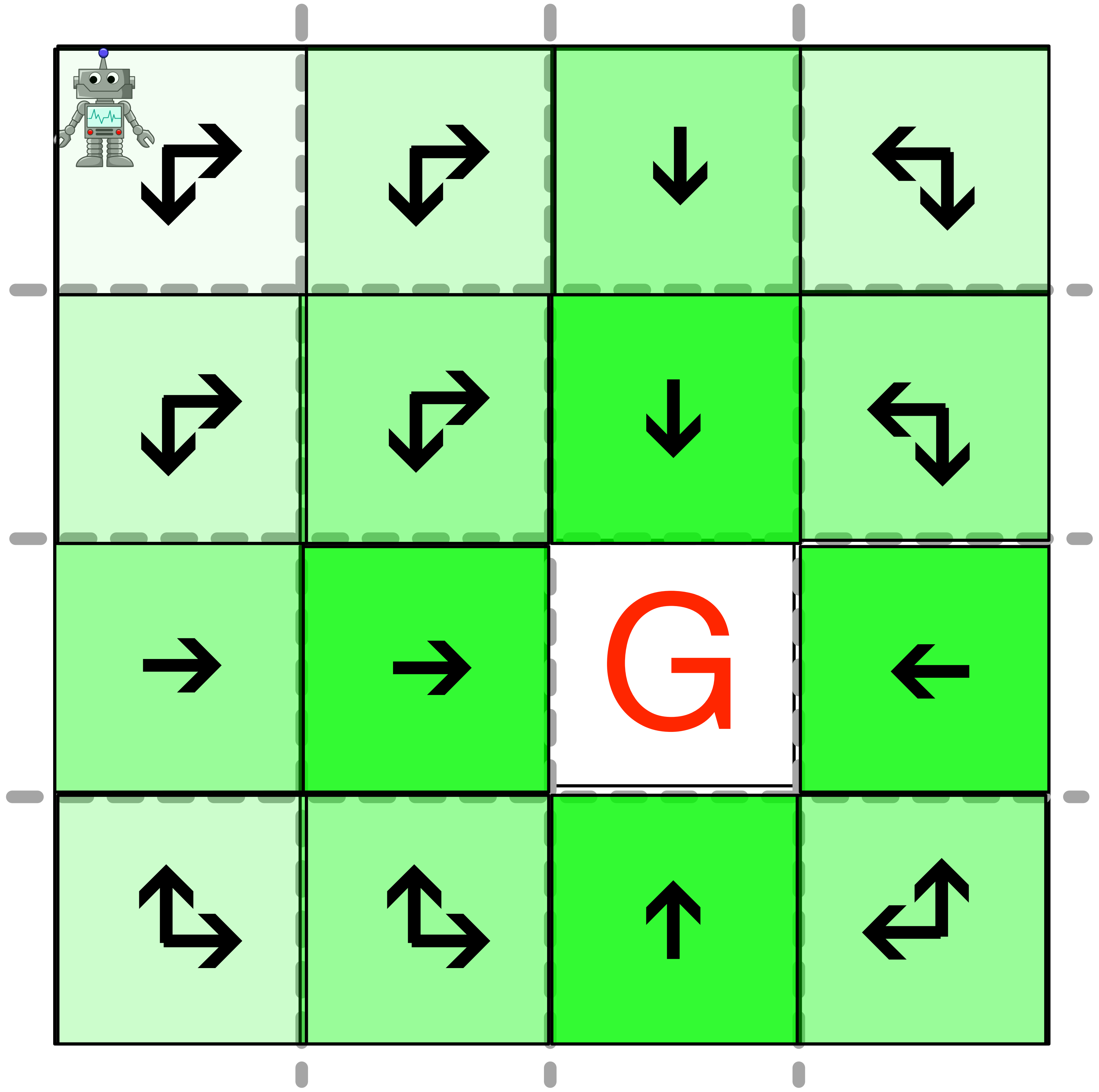
- R = +1 per step
- episodic, gamma = 1
- agent starts in the top left corner
- π = shortest path policy
- what should $\hat{v}(s, \mathbf{w})$ look like?

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
- π = shortest path policy
- what should $\hat{v}(s, \mathbf{w})$ look like?

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

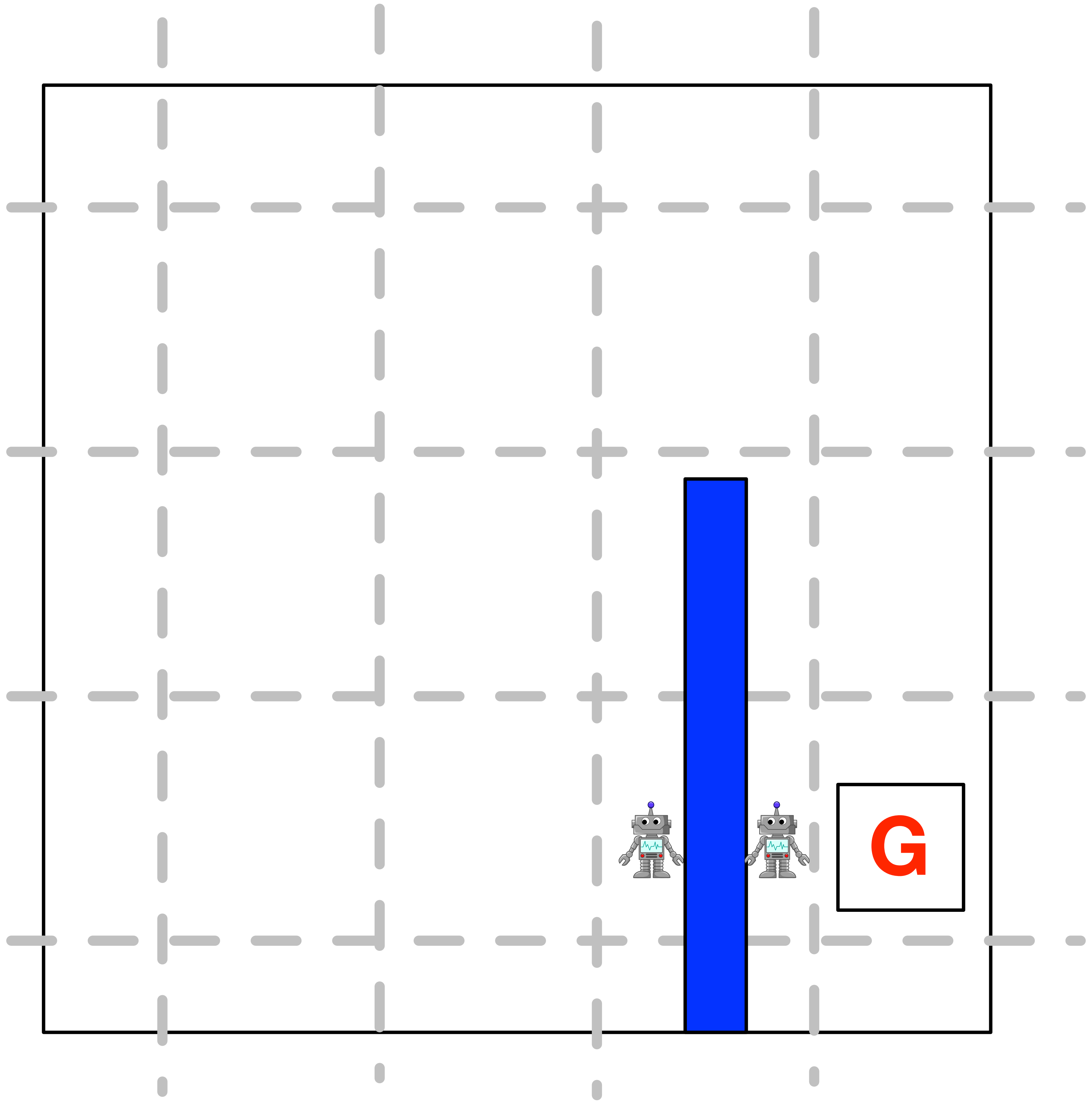


- R = +1 per step
- episodic, gamma = 1
- agent starts in the top left corner
- π = shortest path policy
- what should $\hat{v}(s, \mathbf{w})$ look like?

Video 2: Generalization and Discrimination

- A key concept in machine learning. We cannot learn all the values separately (in fact we wouldn't want to), so we have to make choices.
- Goals:
  - Understand what is meant by generalization and discrimination
  - Understand how generalization can be beneficial
  - Explain why we want both generalization and discrimination from our function approximation

Exercise: Is there any issue with this state aggregation? Can we represent the optimal action-value function?



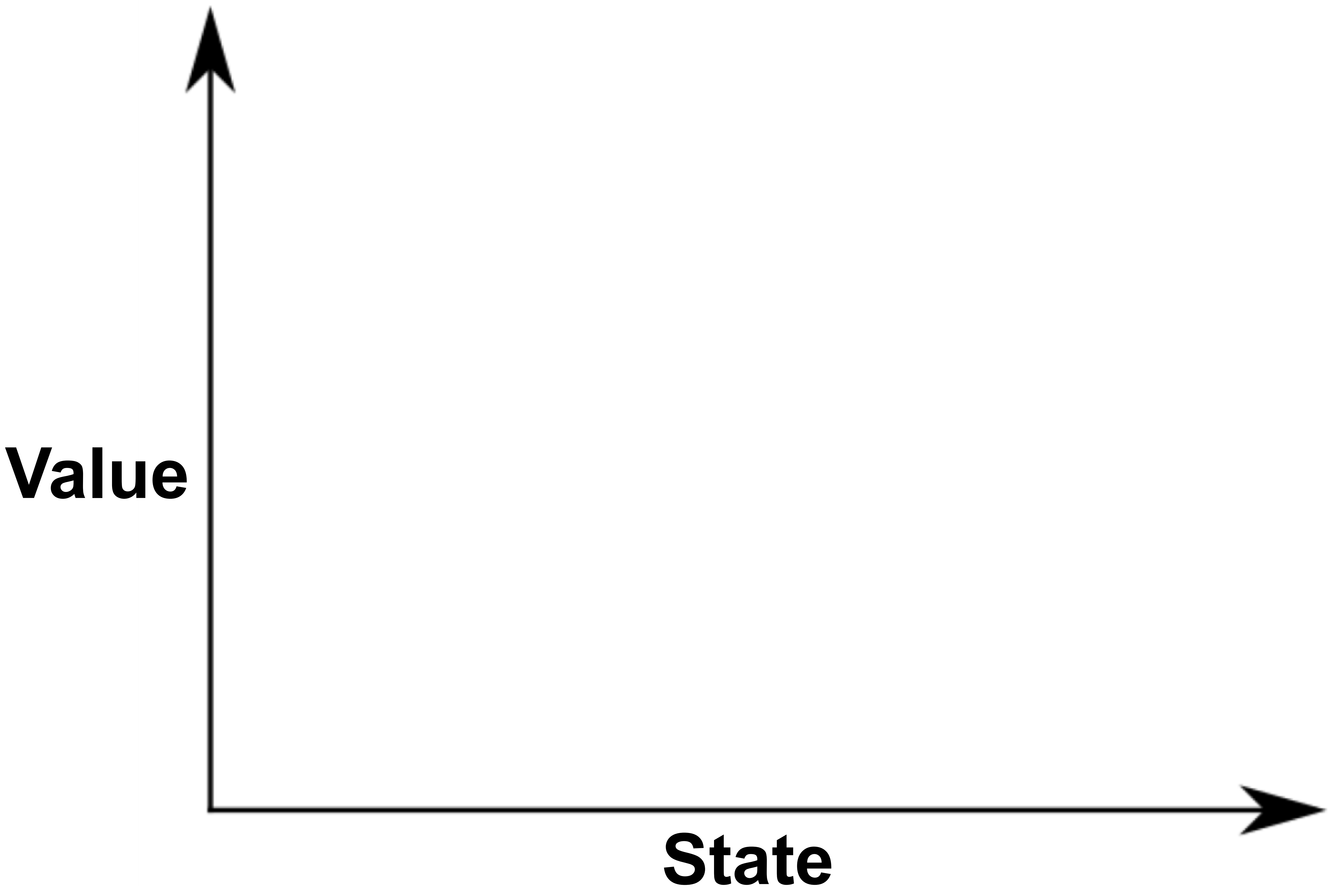
Video 3: Framing Value Estimation as Supervised Learning

- If we can setup the problem of learning a value function (policy evaluation) as a supervised learning problem, then we can borrow methods from supervised learning to do reinforcement learning with function approximation.
- Goals:
  - Understand how value estimation can be framed as a supervised learning problem
  - Recognize that not all function approximation methods are well suited for reinforcement learning.

Video 4: Value Error

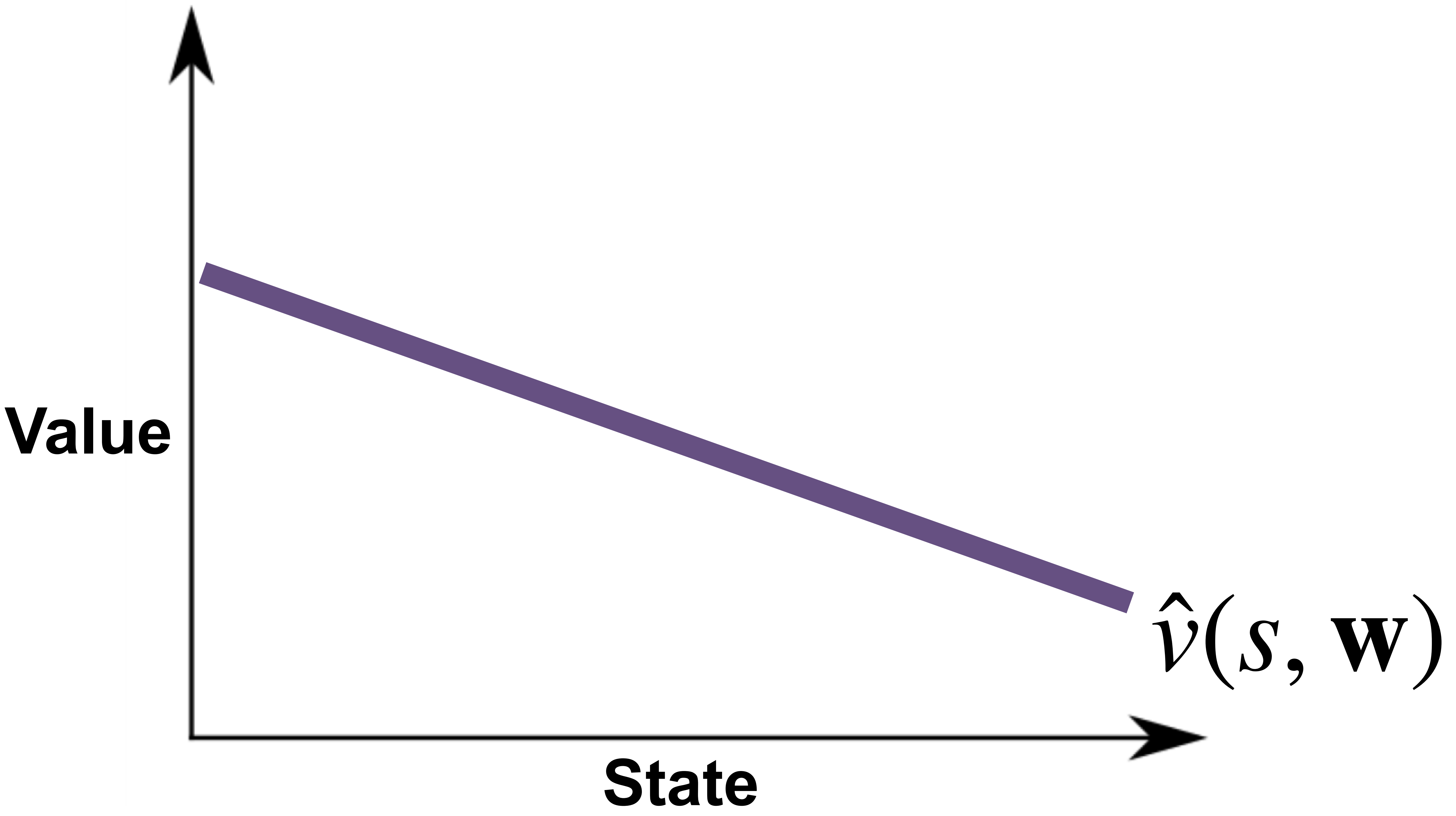
- We want to change the parameters of our function to estimate the value. We need an objective function!
- Goals:
  - Understand the mean-squared value error objective for policy evaluation
  - Explain the role of the state distribution in the objective

The Mean Squared Value Error Objective



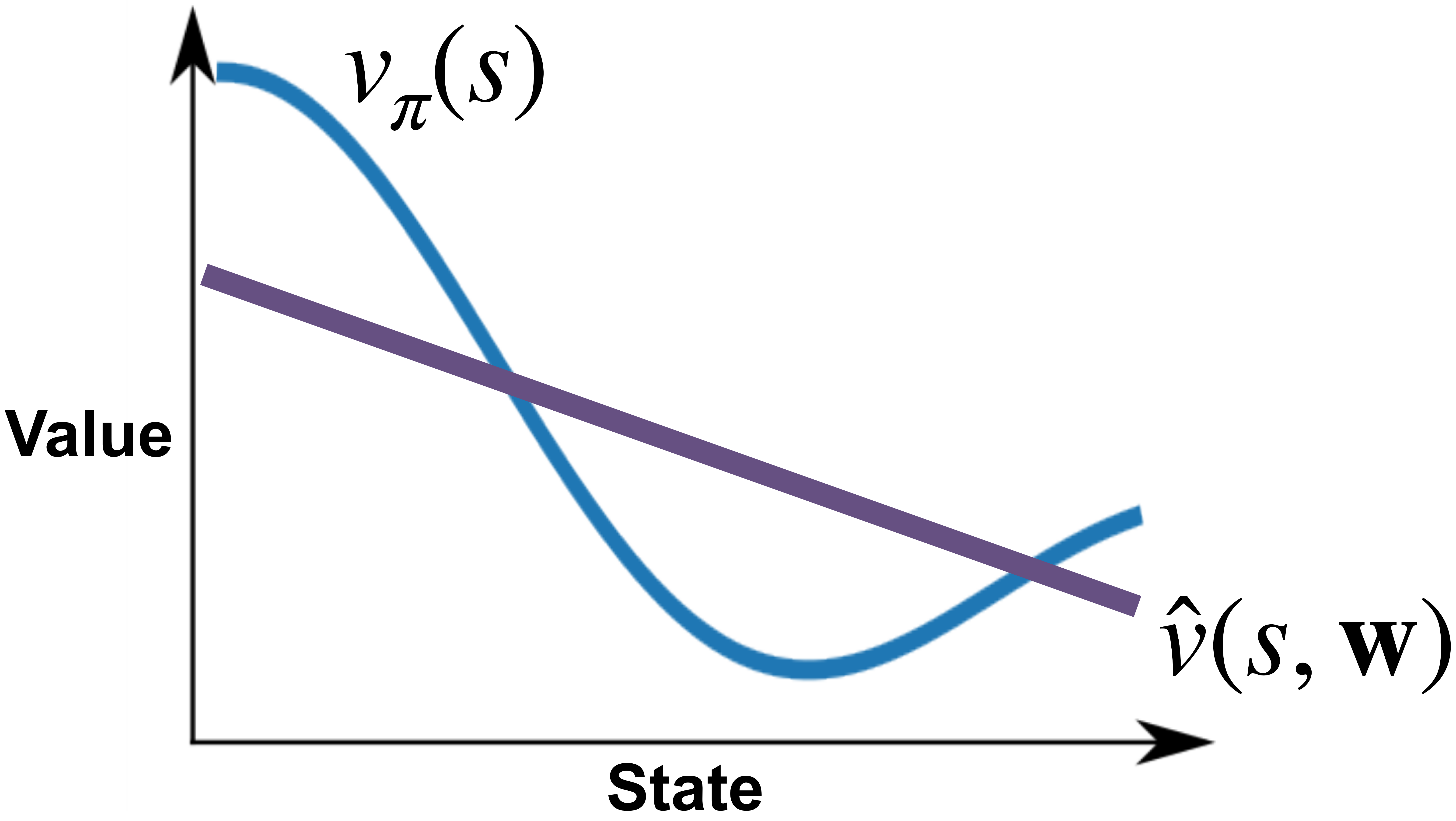
Question: Why didn't we use the Value Error in the tabular setting?

The Mean Squared Value Error Objective



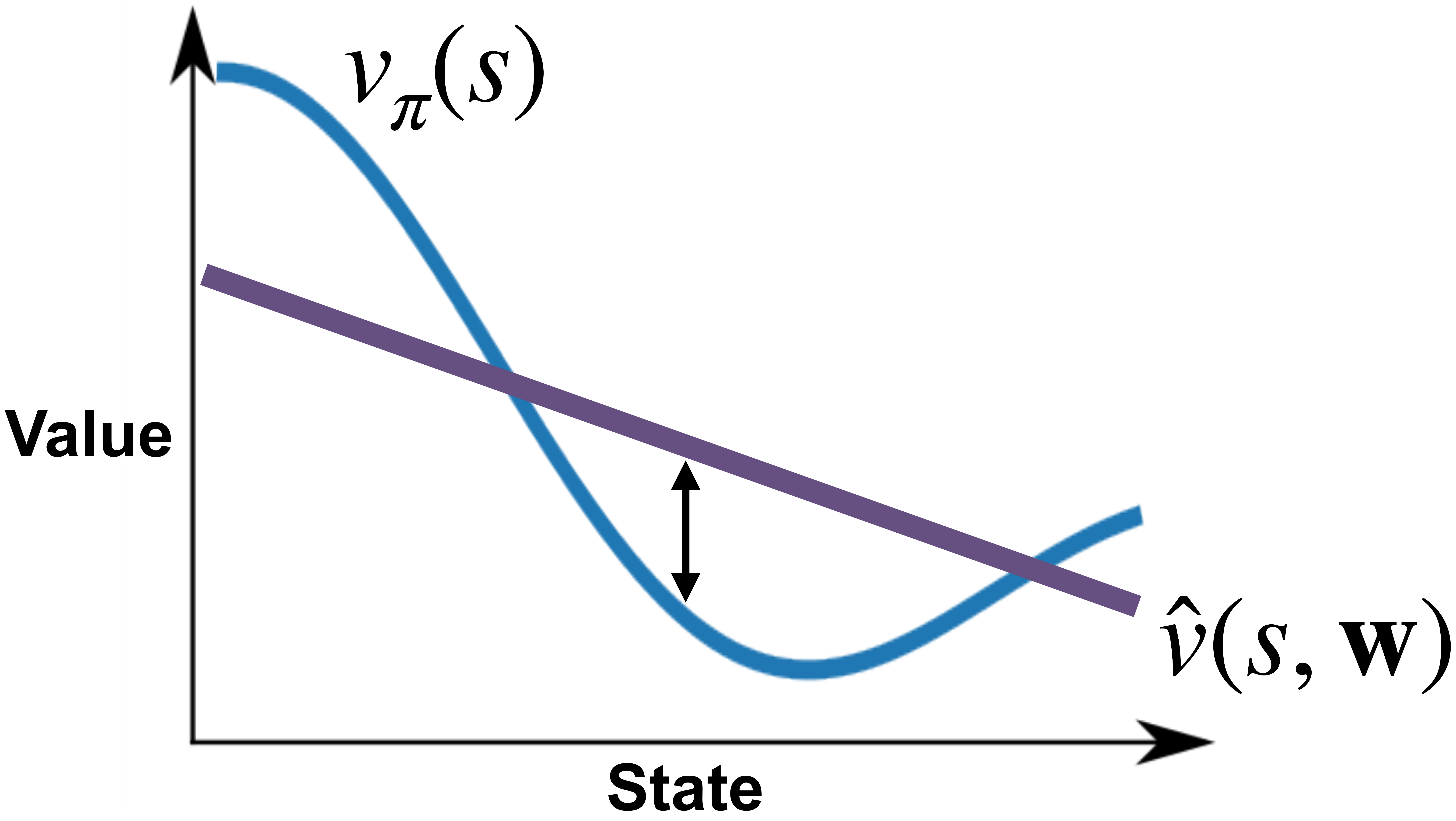
Question: Why didn't we use the Value Error in the tabular setting?

The Mean Squared Value Error Objective



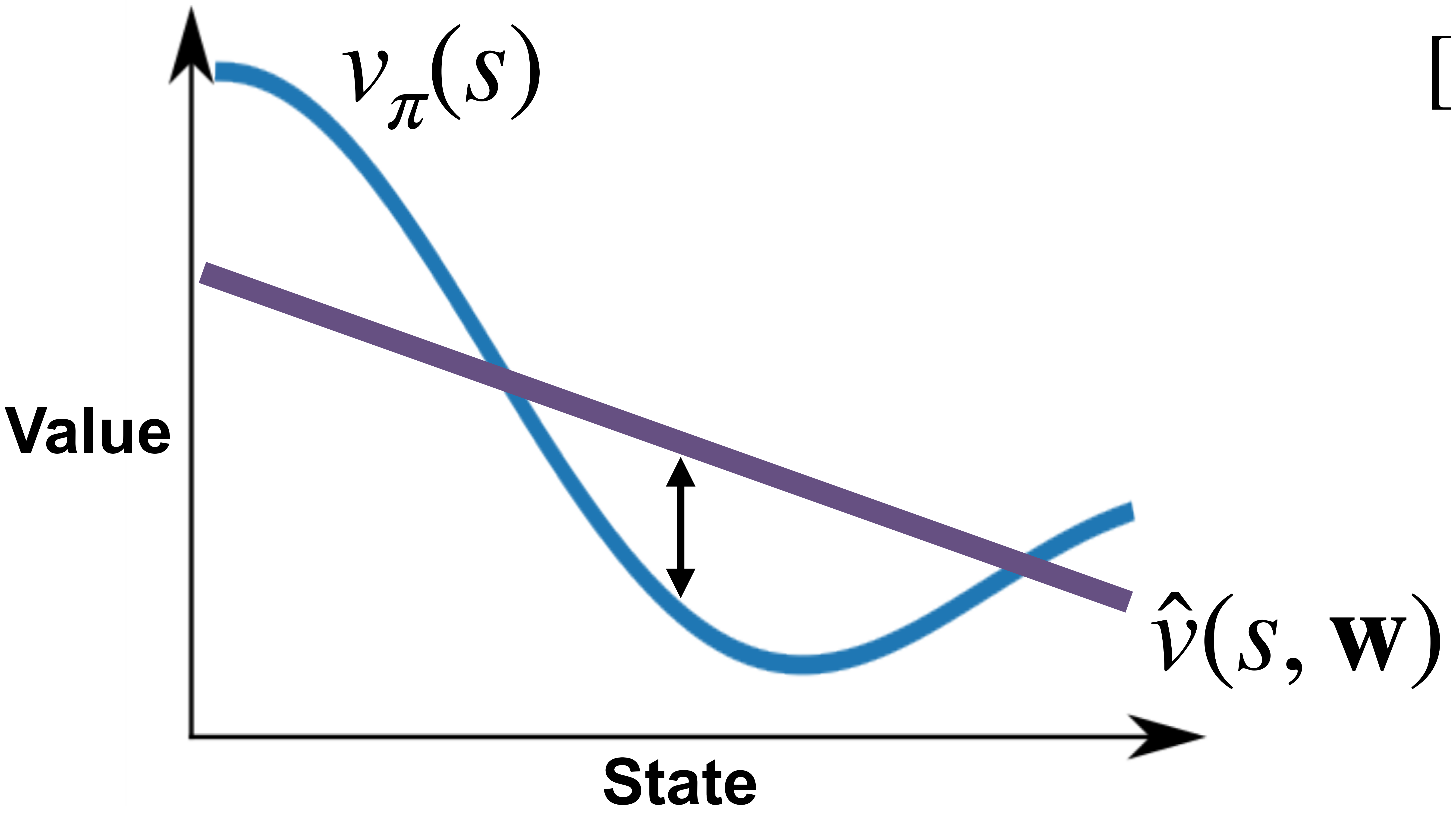
Question: Why didn't we use the Value Error in the tabular setting?

The Mean Squared Value Error Objective



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

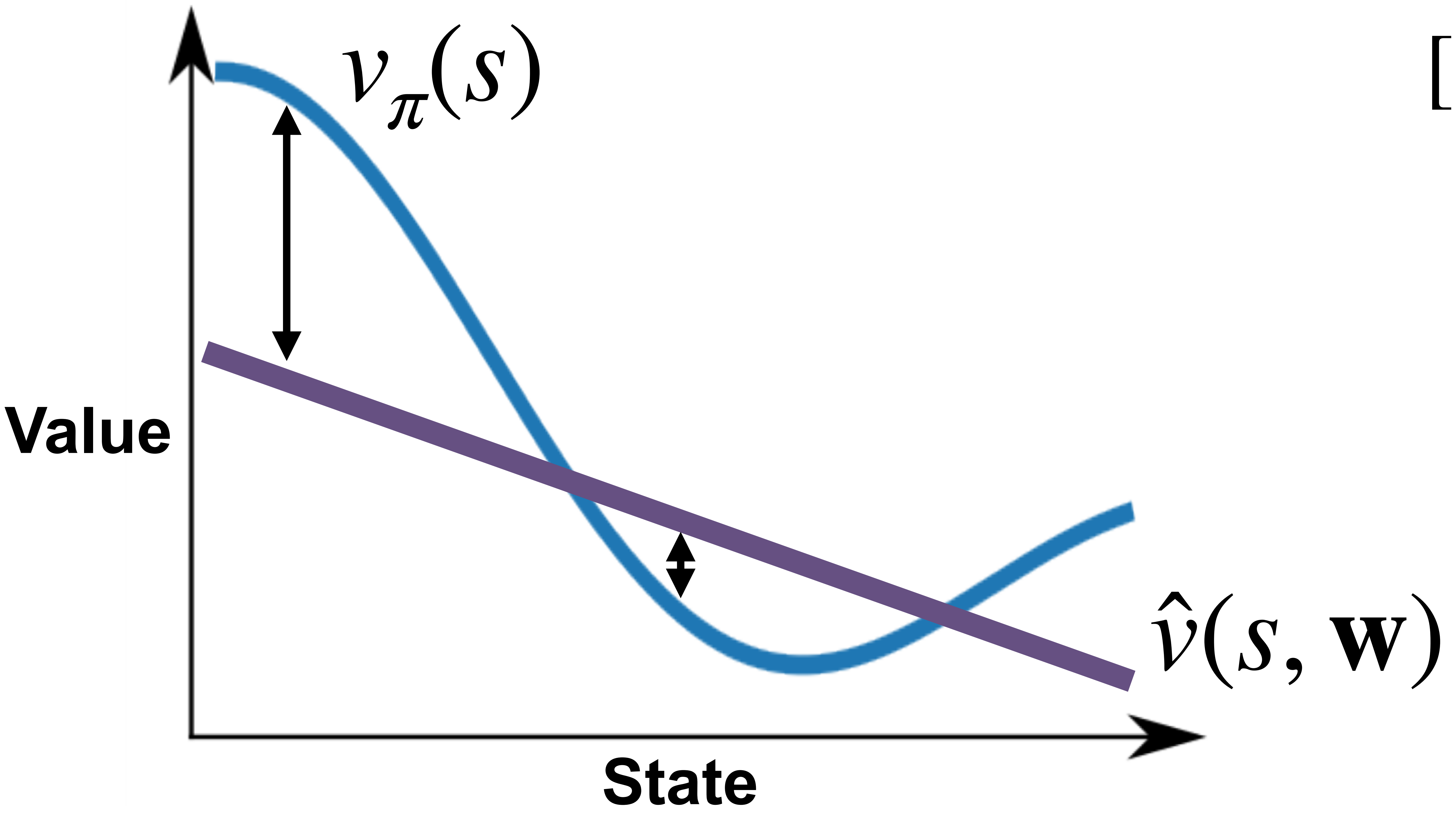
The Mean Squared Value Error Objective



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

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

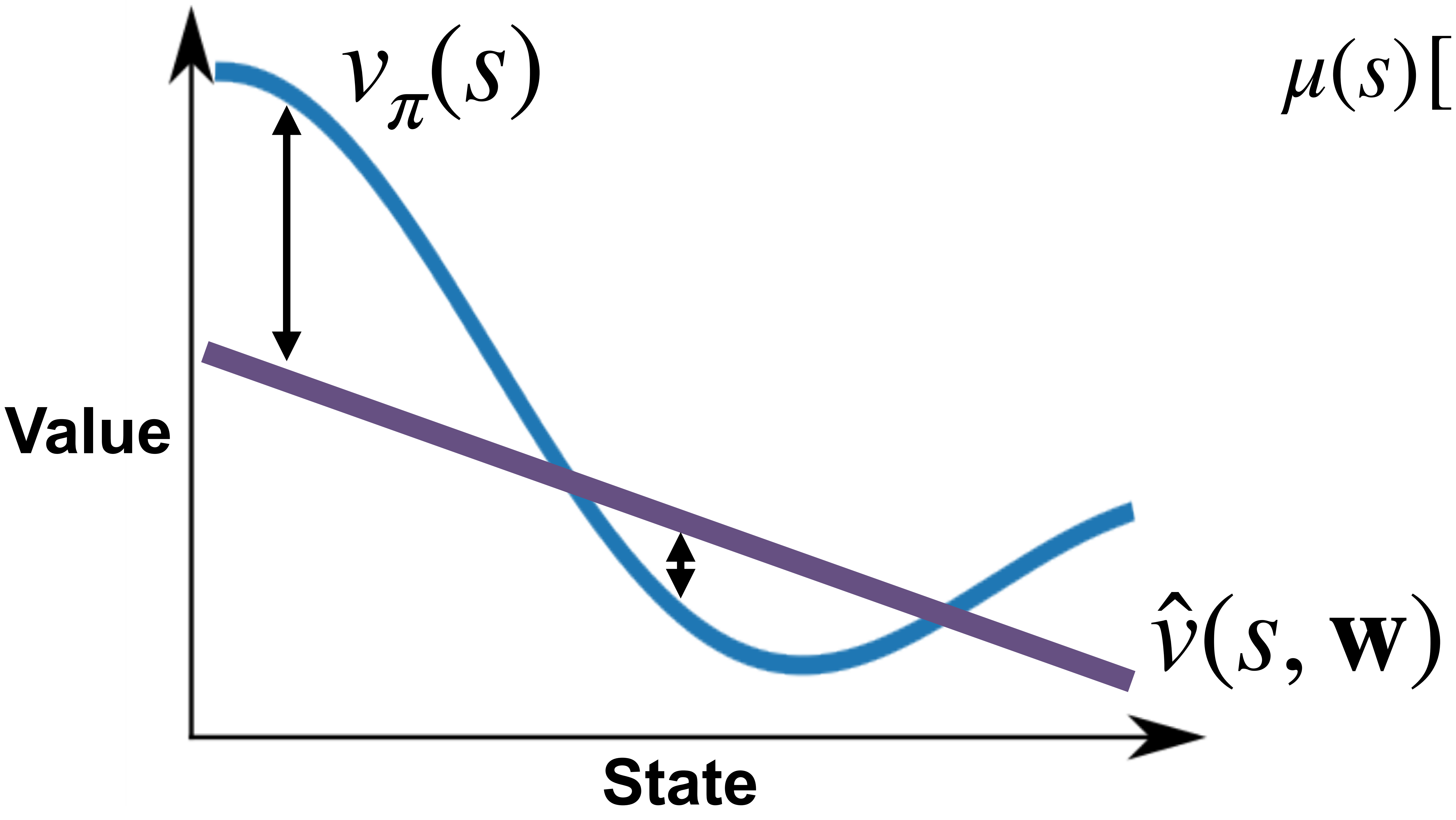
The Mean Squared Value Error Objective



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

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

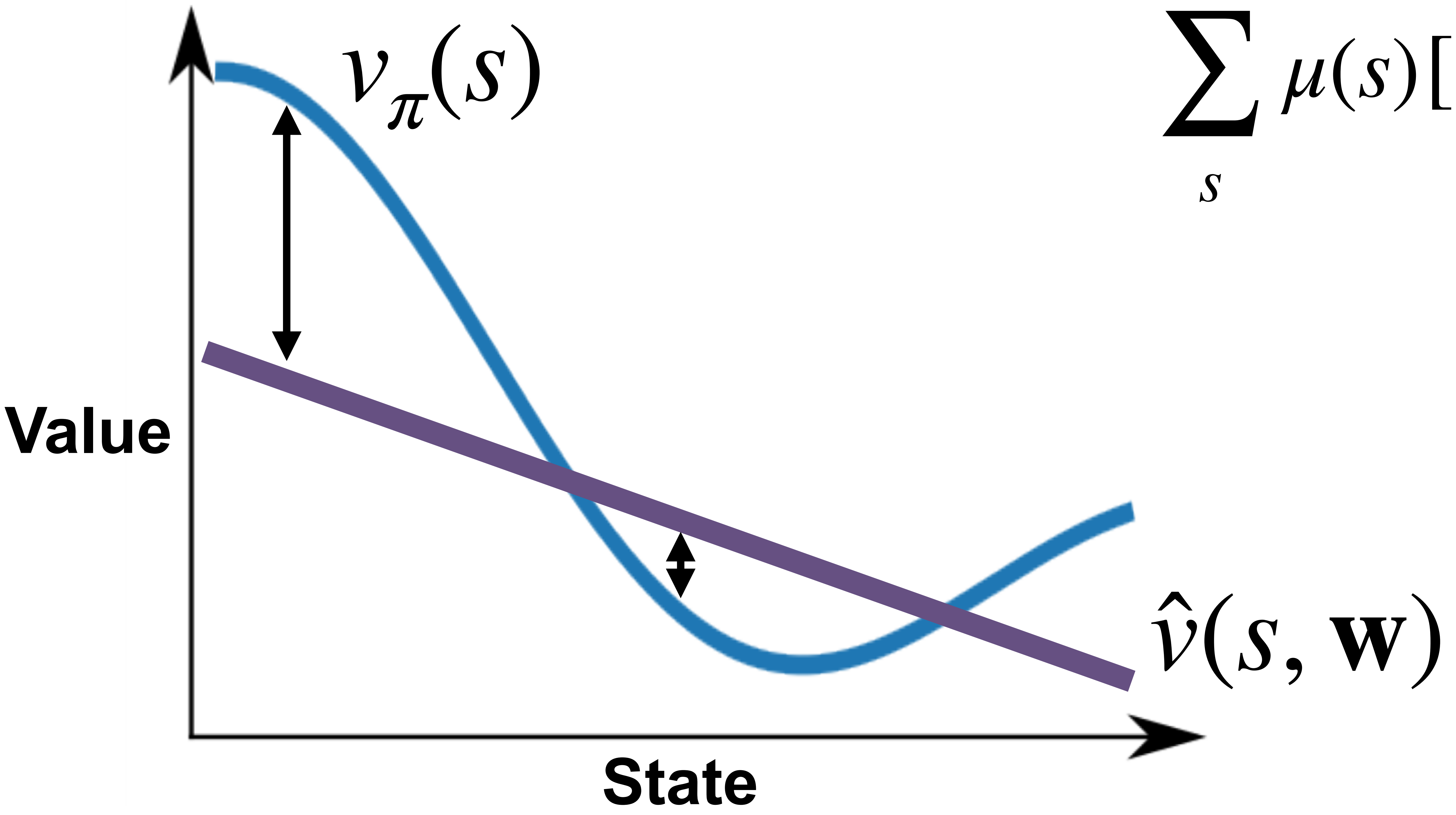
The Mean Squared Value Error Objective



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

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

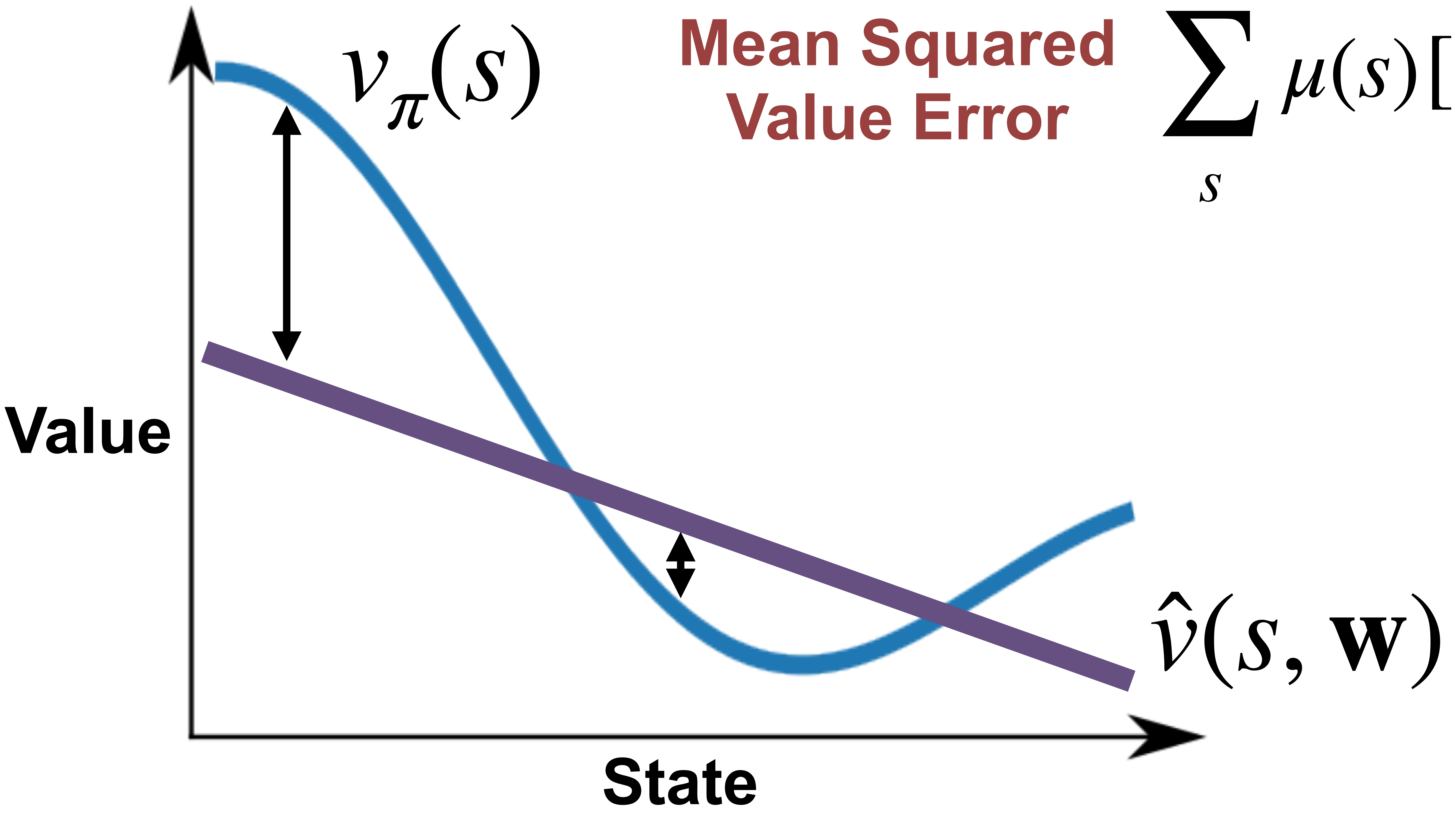
The Mean Squared Value Error Objective



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

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

The Mean Squared Value Error Objective

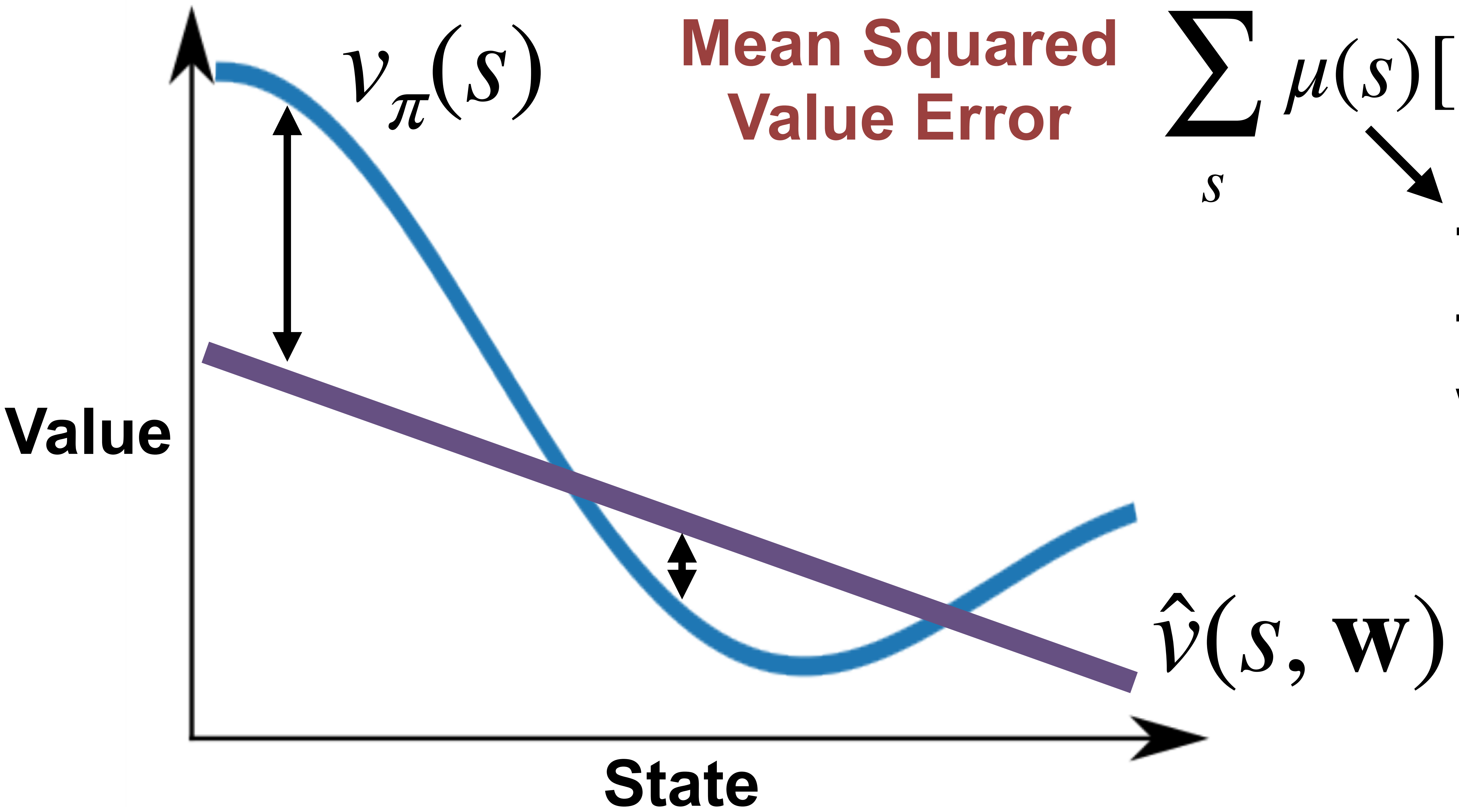


Mean Squared Value Error

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

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

The Mean Squared Value Error Objective



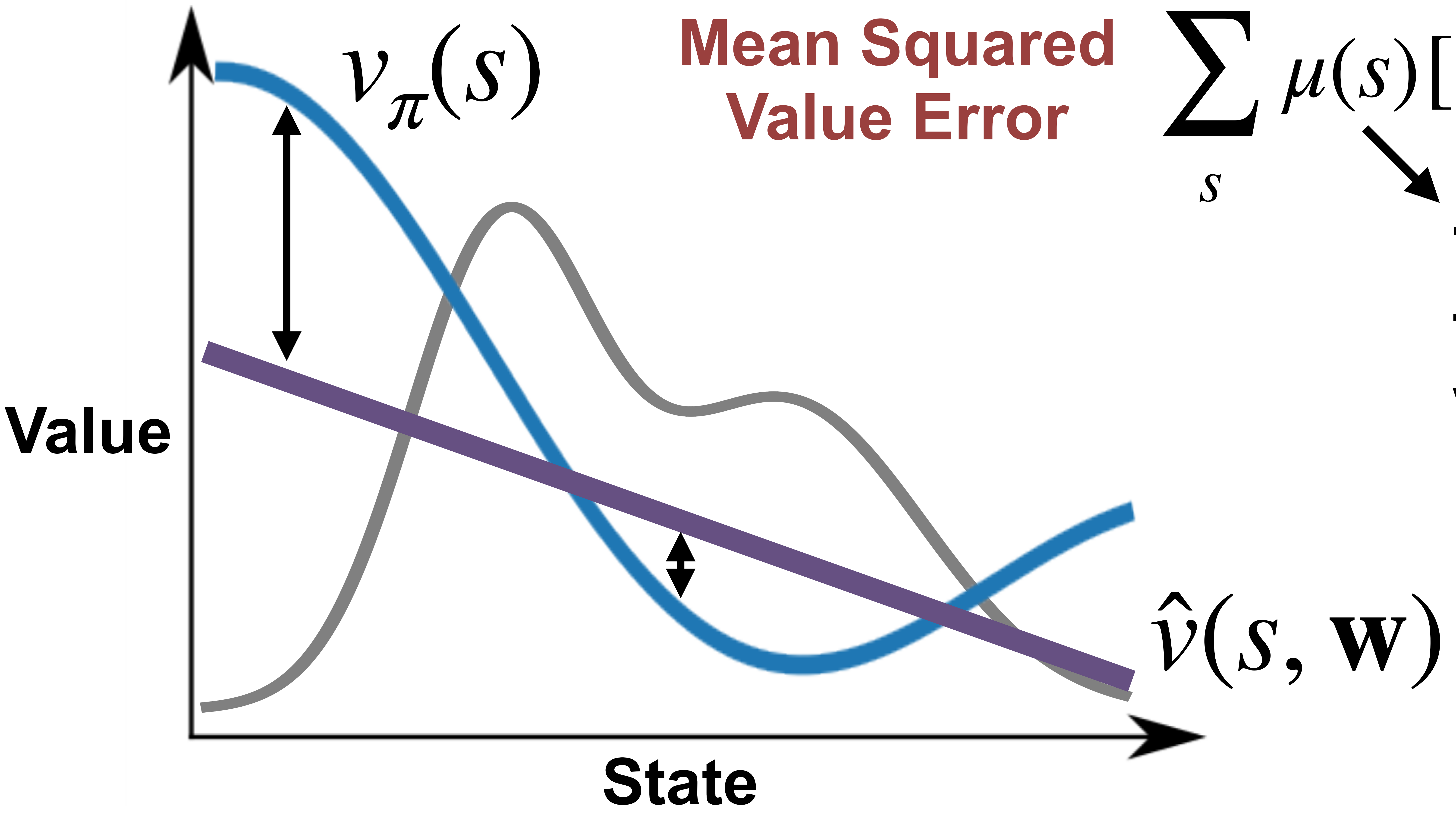
Mean Squared Value Error

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

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

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

The Mean Squared Value Error Objective



Mean Squared Value Error

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

$\mu(s)$: **The fraction of time we spend in S when following policy π**

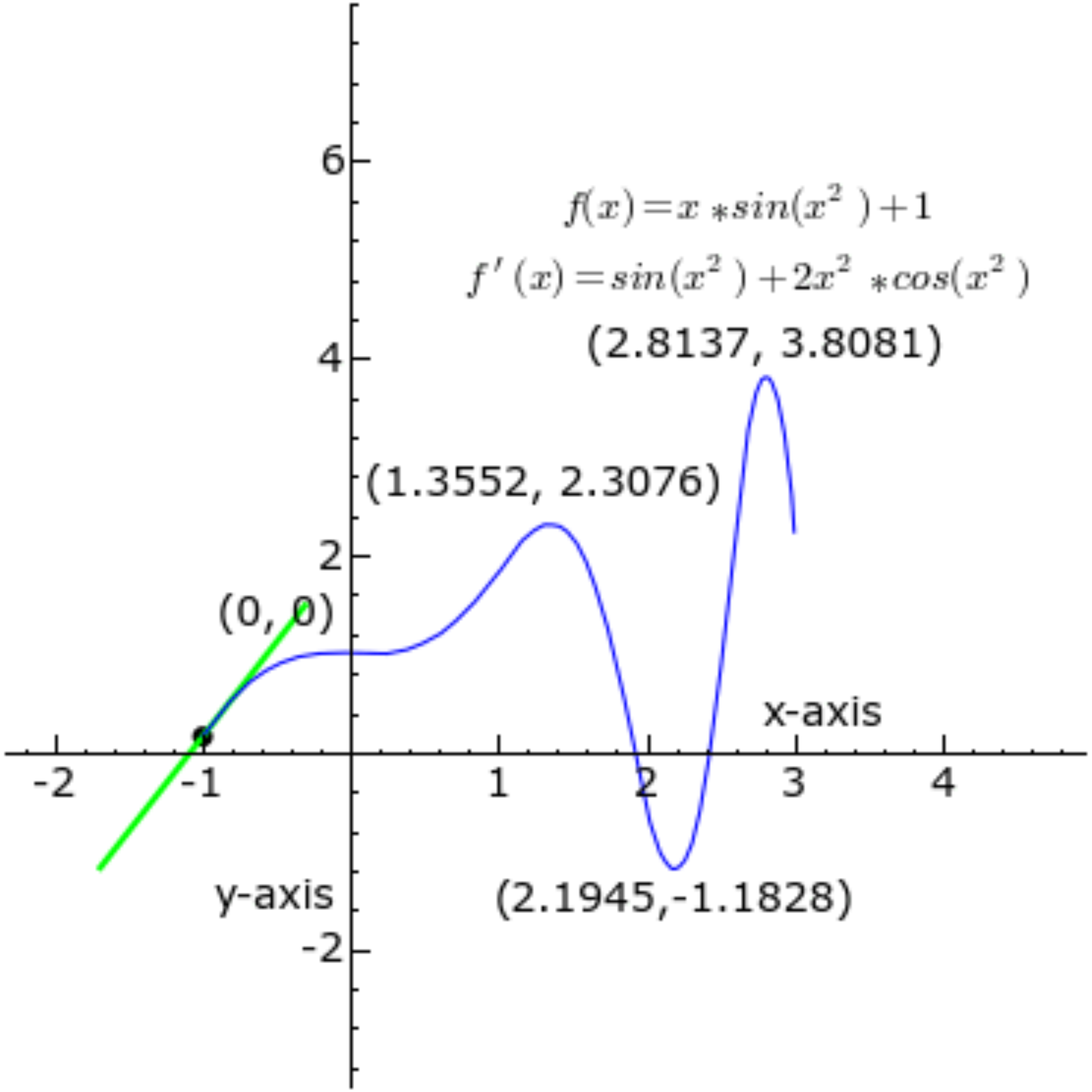
Question: Why didn't we use the Value Error in the tabular setting?

Video 5: Introducing Gradient Descent

- An algorithm for adapting the parameters of our estimate of the value function.
- Goals:
  - Understand the idea of gradient descent
  - Understand that gradient descent converges to stationary points

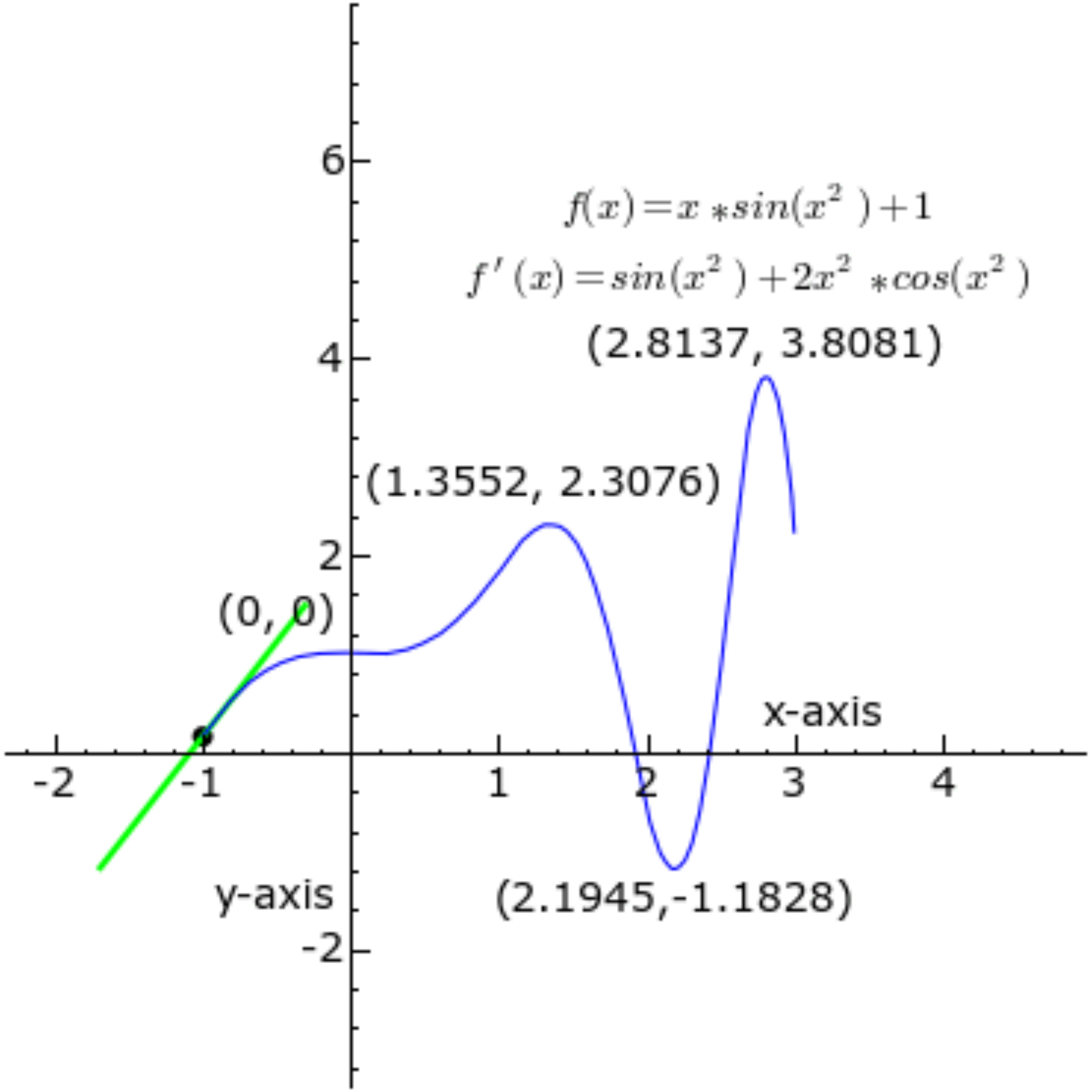
Question

- Why do we care about finding stationary points? i.e., point w where the gradient is zero



Question

- Why do we care about finding stationary points? i.e., point w where the gradient is zero



Video 6: Gradient Monte Carlo for Policy Evaluation

- We use gradient descent idea to get an online algorithm to adjust the parameters of our value function estimate
- Goals:
  - Understand how to use gradient descent and stochastic gradient descent to minimize value error
  - Outline the gradient Monte Carlo algorithm for value estimation

Video 7: State Aggregation with Monte Carlo

- So far we have said the value function could be any parametric function. Here we use a particular one---state aggregation. Simple and effective. And we run an experiment on a big Random Walk Problem
- Goals:
  - Understand how state aggregation can be used to approximate the value function
  - Apply Gradient Monte-Carlo with state aggregation

Video 8: Semi-gradient TD for Policy Evaluation

- TD with function approximation. Now we can learn value functions, in continuous state spaces AND update the value function parameters on every time-step!!
- Goals:
  - Understand the TD-update for function approximation
  - Outline the Semi-gradient TD algorithm for value estimation.

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

Input: the policy π to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S}^+ \times \mathbb{R}^d \to \mathbb{R}$ 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'
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R + \gamma\hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})\big]\nabla\hat{v}(S,\mathbf{w})$
        $S \leftarrow S'$
    until S is terminal

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

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

Input: the policy π to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S}^+ \times \mathbb{R}^d \to \mathbb{R}$ 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'
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R + \gamma\hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})\big]\nabla\hat{v}(S,\mathbf{w})$
    $S \leftarrow S'$
  until S is terminal

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

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

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

Video 9: Comparing TD and MC with State Aggregation

- An experiment comparing TD and MC with a simple function approximation.
- Goals:
  - Understand that TD converges to biased value estimates
  - Understand that TD can learn faster than Gradient Monte Carlo.

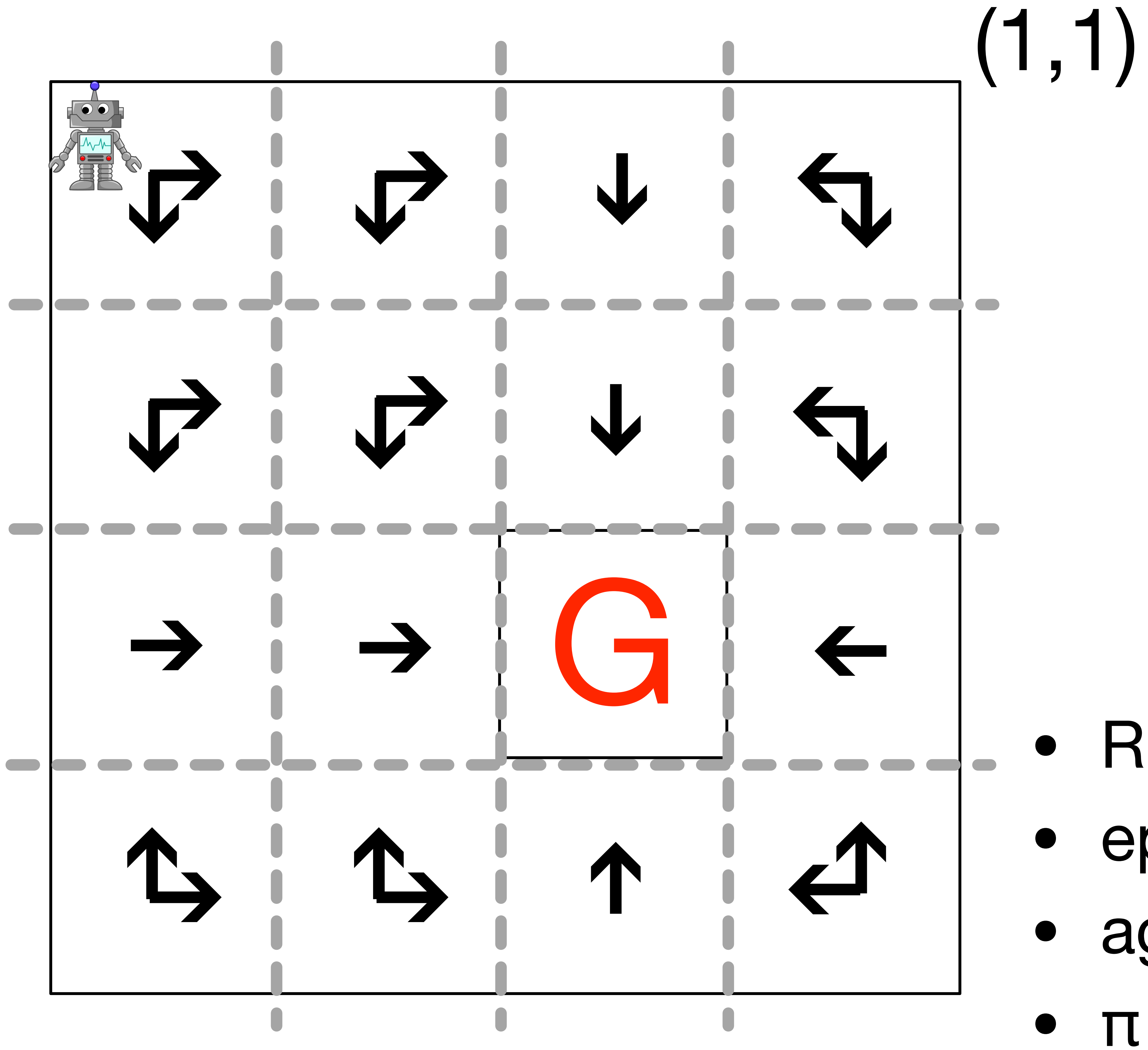
Video 10: The Linear TD Algorithm

- Linear function functions are special. Most of the theory in RL is for the case of linear function approximation. The algorithms can work well, if we have good features.
- Goals:
  - Derive the TD-update with linear function approximation
  - Understand that tabular TD is a special case of linear semi-gradient TD
  - Understand why we care about linear TD as a special case.

Video 11: The True Objective for TD

- A bit of theory about TD with function approximation. What does the algorithm converge to?
- Goals:
  - Understand the fixed point of linear TD
  - Describe a theoretical guarantee on the mean squared value error at the TD fixed point

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



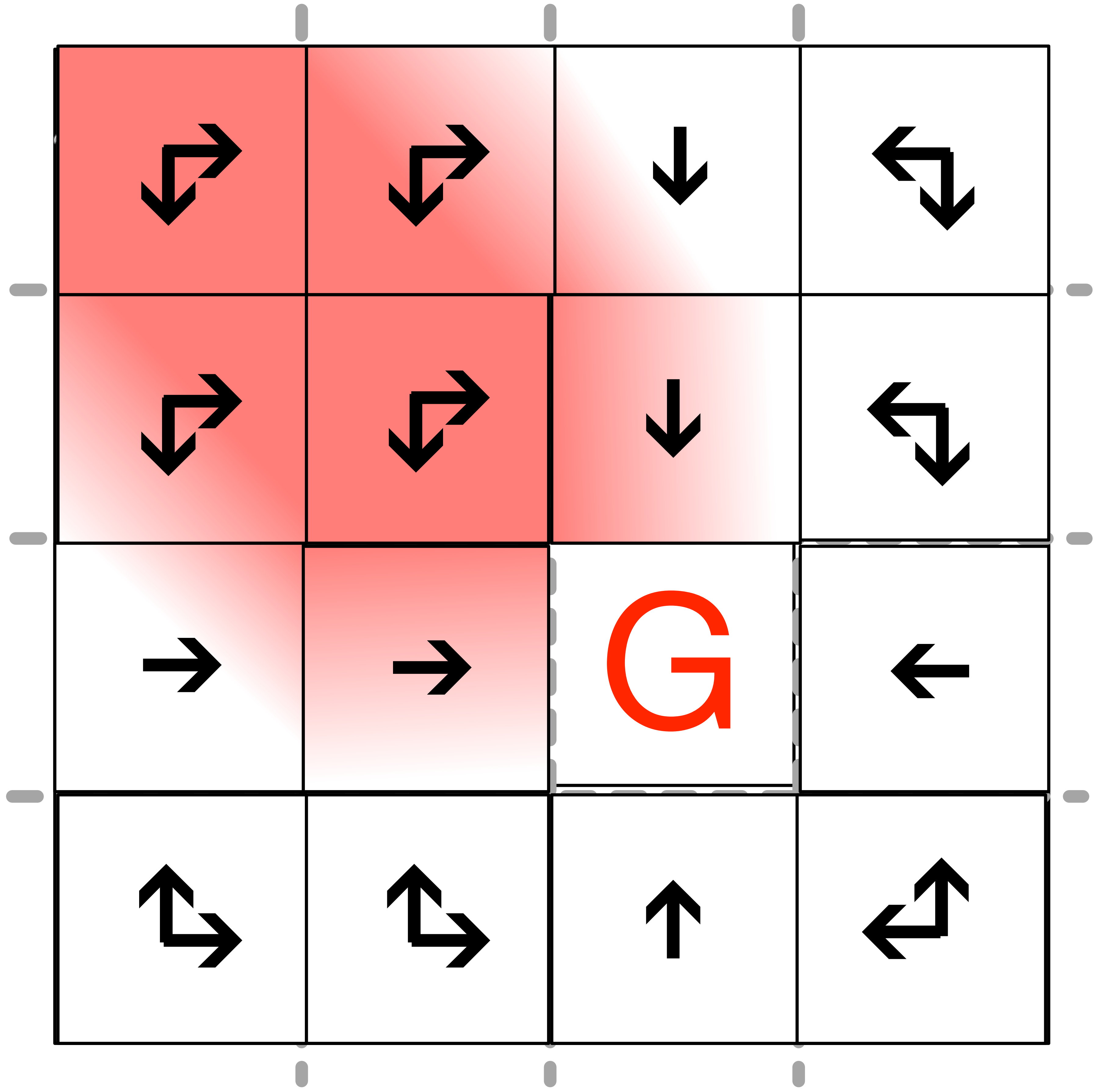
- R = +1 per step
- episodic, gamma = 1
- agent starts in the top left corner
- π = shortest path policy

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

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



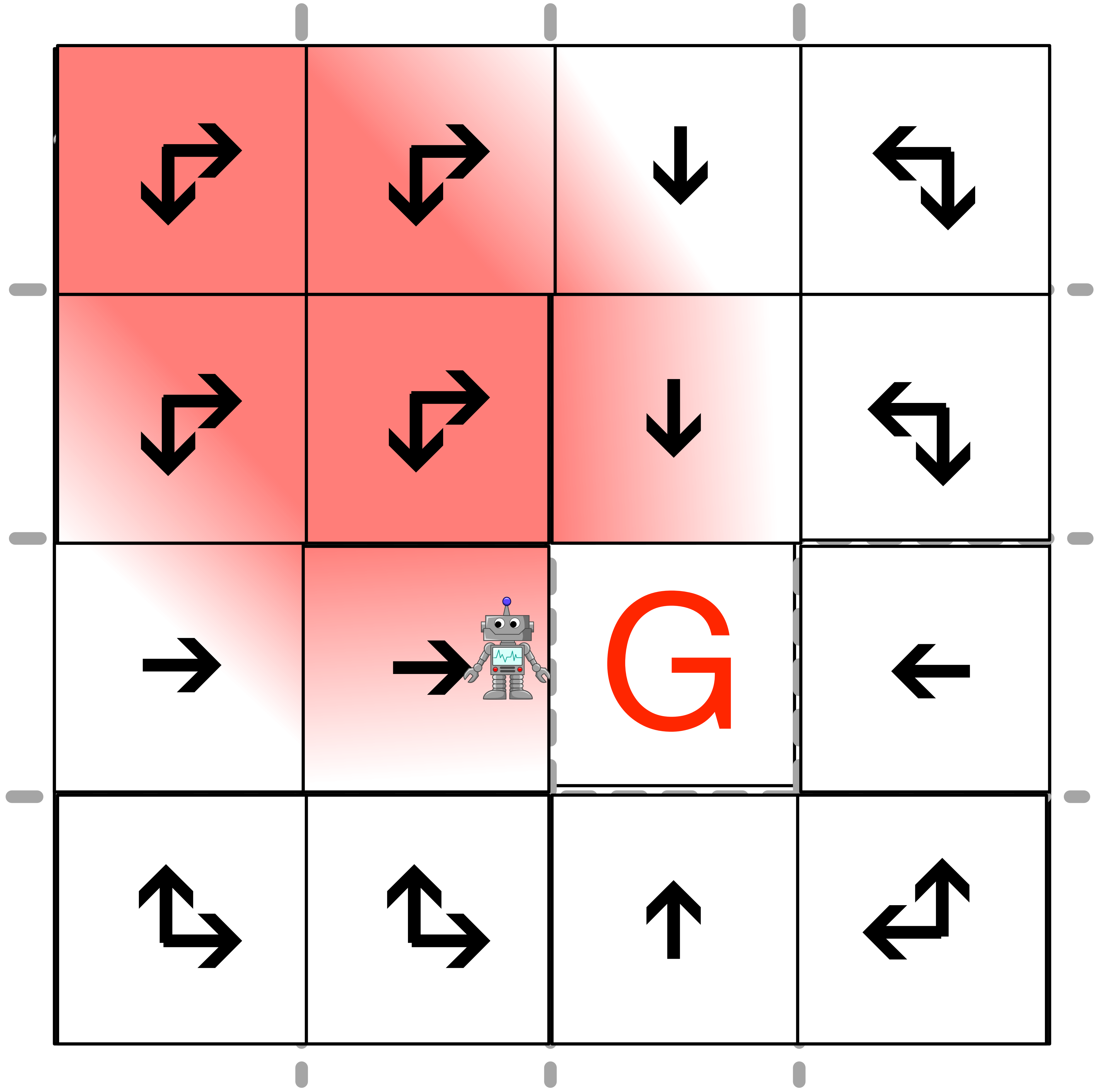
- R = +1 per step
- episodic, gamma = 1
- agent starts in the top left corner
- π = shortest path policy

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

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

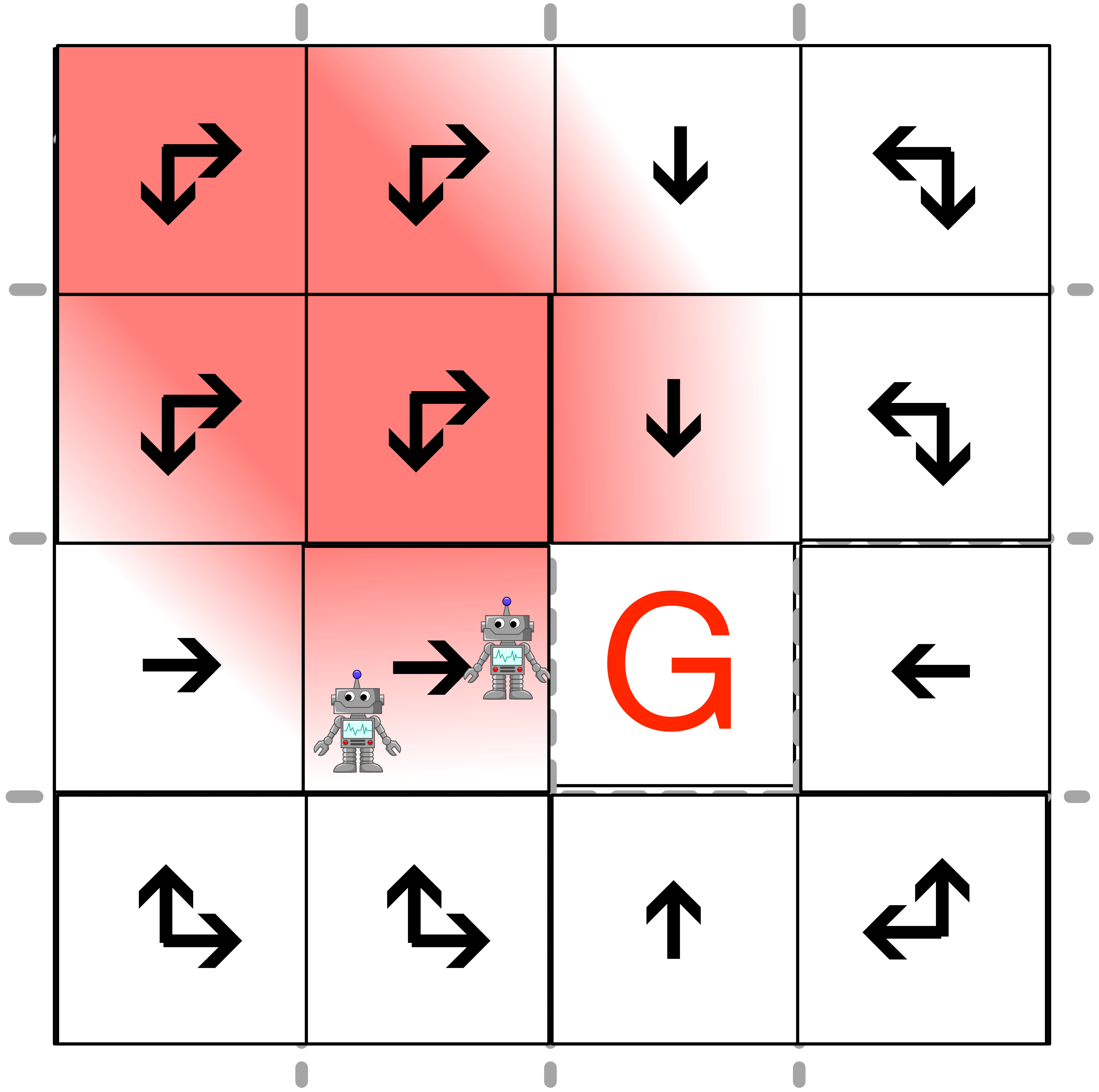


Mean Squared Value Error

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

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

μ(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:

    - **w** = **w -** α gradient

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:

   - **w** = **w** - α gradient

gradient descent

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:

    - **w** = **w** - α gradient

gradient descent

lets try out the recipe

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

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

- We will use **State Aggregation**

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

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**

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. 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

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

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

2. Write down your objective function

- We will use the value error

2. Write down your objective function

- We will use the value error

$$\overline{VE}(\mathbf{w}) \doteq \sum_{s \in \mathcal{S}} \mu(s) \left[v_\pi(s) - \hat{v}(s, \mathbf{w}) \right]^2$$

2. Write down your objective function

- We will use the value error

$$\overline{VE}(\mathbf{w}) \doteq \sum_{s \in \mathcal{S}} \mu(s)[v_\pi(s) - \hat{v}(s, \mathbf{w})]^2$$

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

2. Write down your objective function

- We will use the value error

$$\overline{VE}(\mathbf{w}) \doteq \sum_{s \in \mathcal{S}} \mu(s) [v_\pi(s) - \hat{v}(s, \mathbf{w})]^2$$

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

state aggregation

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

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

$$\nabla \overline{VE}(\mathbf{w}) =$$

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

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

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

$$\nabla \overline{VE}(\mathbf{w}) =$$

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

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

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

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

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

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

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

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

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

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 **x**

4. Simplify general gradient ...

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

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

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

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

4. Simplify general gradient ...

gradient of linear value function approximation (state agg.)

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

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

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

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

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

5. Make weight update rule: **w** = **w** - α gradient

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

5. Make weight update rule: **w** = **w** - α gradient

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

5. Make weight update rule: **w** = **w** - α gradient

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

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

5. Make weight update rule: **w** = **w** - α 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

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

Wait, Wait, Wait!! We don't have v_π

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

Wait, Wait, Wait!! We don't have v_π

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

Let's replace it with something we do have!

Let's replace v_π with something we do have!

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

Let's replace v_π with something we do have!

Let's call it's replacement Ut

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

Let's replace v_π with something we do have!

Let's call it's replacement Ut

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

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

Let's replace v_π with something we do have!

Let's call it's replacement Ut

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

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

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

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

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

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!

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!

$$U_t \doteq G_t$$

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!

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

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

Exercise Question

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

- Why can't we directly optimize the MSVE? We know the stochastic gradient descent update would be the following

$$\mathbf{w}_t + \alpha[v_\pi(S_t) - \hat{v}(S_t, \mathbf{w})] \nabla \hat{v}(S_t, \mathbf{w})$$

- Further, why doesn't the TD fixed point minimize the MSVE?