Robotic Navigation and Exploration

Week 12: Reinforcement Learning (I)

Min-Chun Hu anitahu@cs.nthu.edu.tw
CS, NTHU

Outline

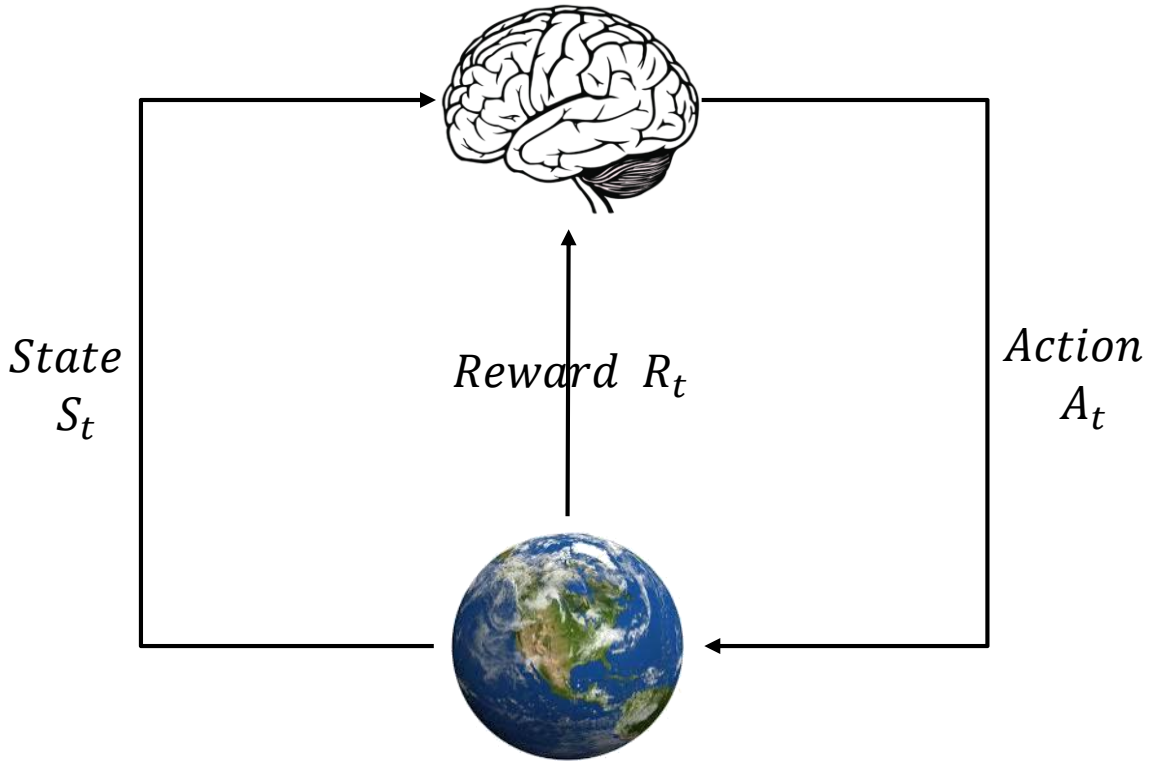
- Introduction to Reinforcement Learning
- Markov Decision Process (MDP)
  - MDP Formulation
  - Value Function and Bellman Equation
  - Dynamic Programming Methods
- Value-based Reinforcement Learning
  - From MDP to RL
  - Temporal Difference Learning (SARSA / Q-Learning)
  - Deep Q-Network (DQN) & Normalized Advantage Function (NAF)

Outline

- Introduction to Reinforcement Learning
- Markov Decision Process (MDP)
  - MDP Formulation
  - Value Function and Bellman Equation
  - Dynamic Programming Methods
- Value-based Reinforcement Learning
  - From MDP to RL
  - Temporal Difference Learning (SARSA / Q-Learning)
  - Deep Q-Network (DQN) & Normalized Advantage Function (NAF)

Reinforcement Learning

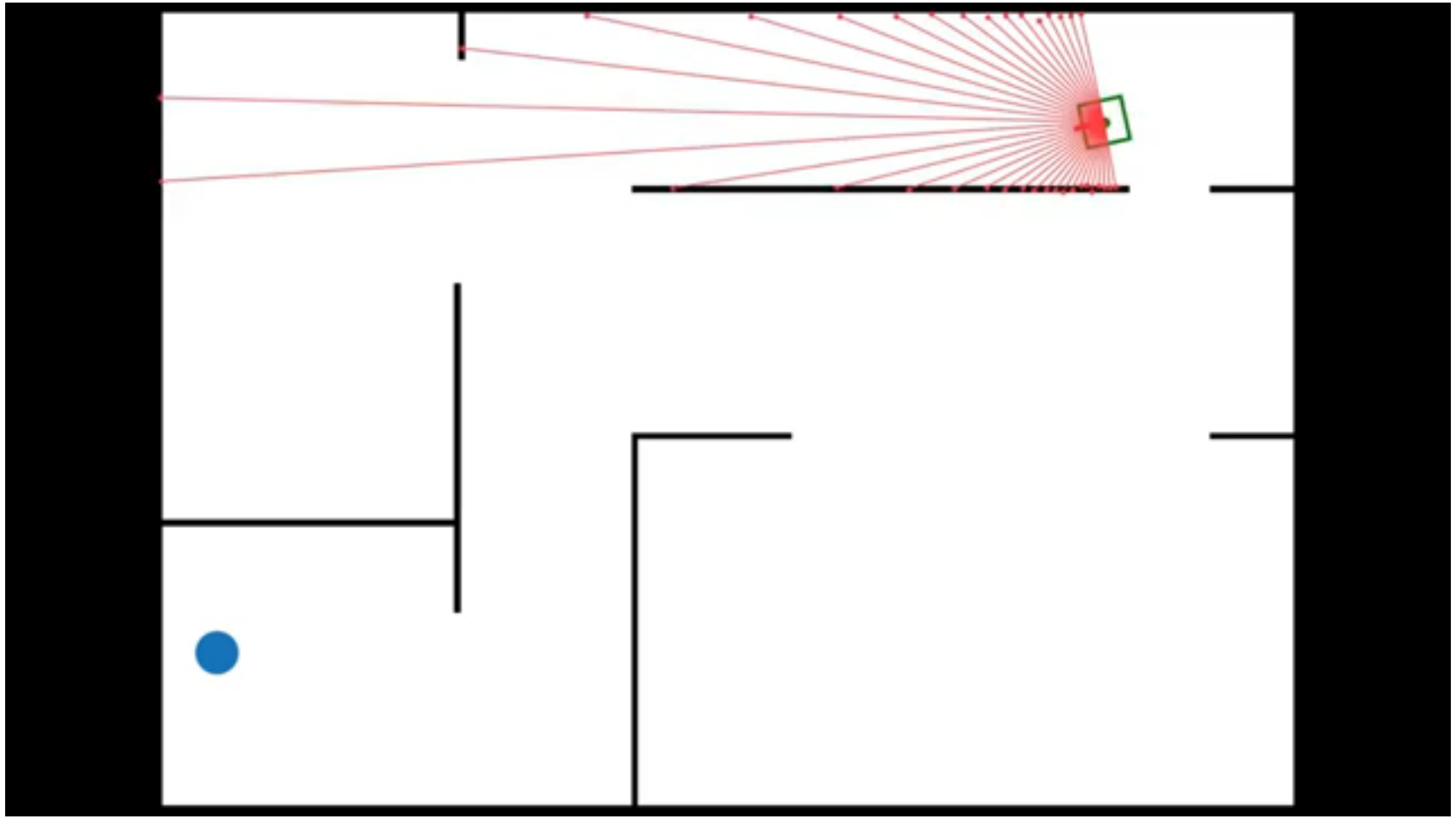
- **R**einforcement **L**earning is learning what to do—how to map situations to actions—so as to maximize a numerical reward signal.



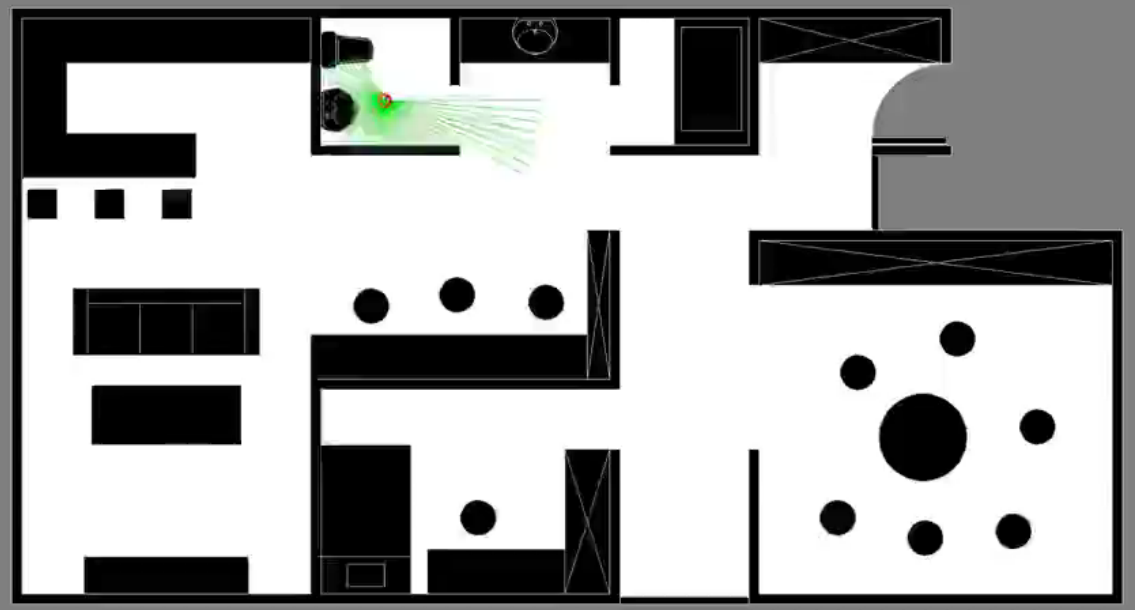
At each step t

- Agent
  - Receive $state\ S_t$
  - Receive $reward\ R_t$
  - Execute $action\ A_t$
- Environment
  - Receive $action\ A_t$
  - Emit $state\ S_{t+1}$
  - Emit $reward\ R_{t+1}$

RL Example – Mapless Navigation



RL Example – Environment Exploration



Reinforcement Learning

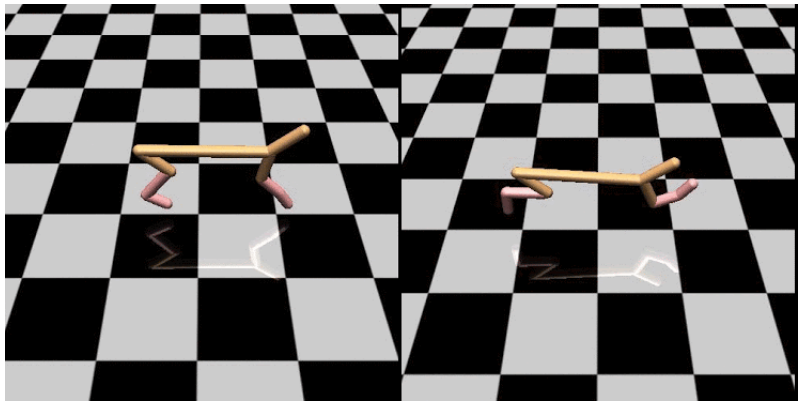
- Conceptually, we can treat the "reward" as the "label" in supervised learning, both provide a goal for the model to optimize.
  - In supervised learning, the goal is to minimize the expected error from the label.
  - In reinforcement learning, the goal is to maximize the expected sum of reward.

- As for "unsupervised learning" and "meta learning", there are corresponding tasks in reinforcement learning field, which are "No explicit reward RL" and "meta RL" respectively.

Reinforcement Learning without Reward

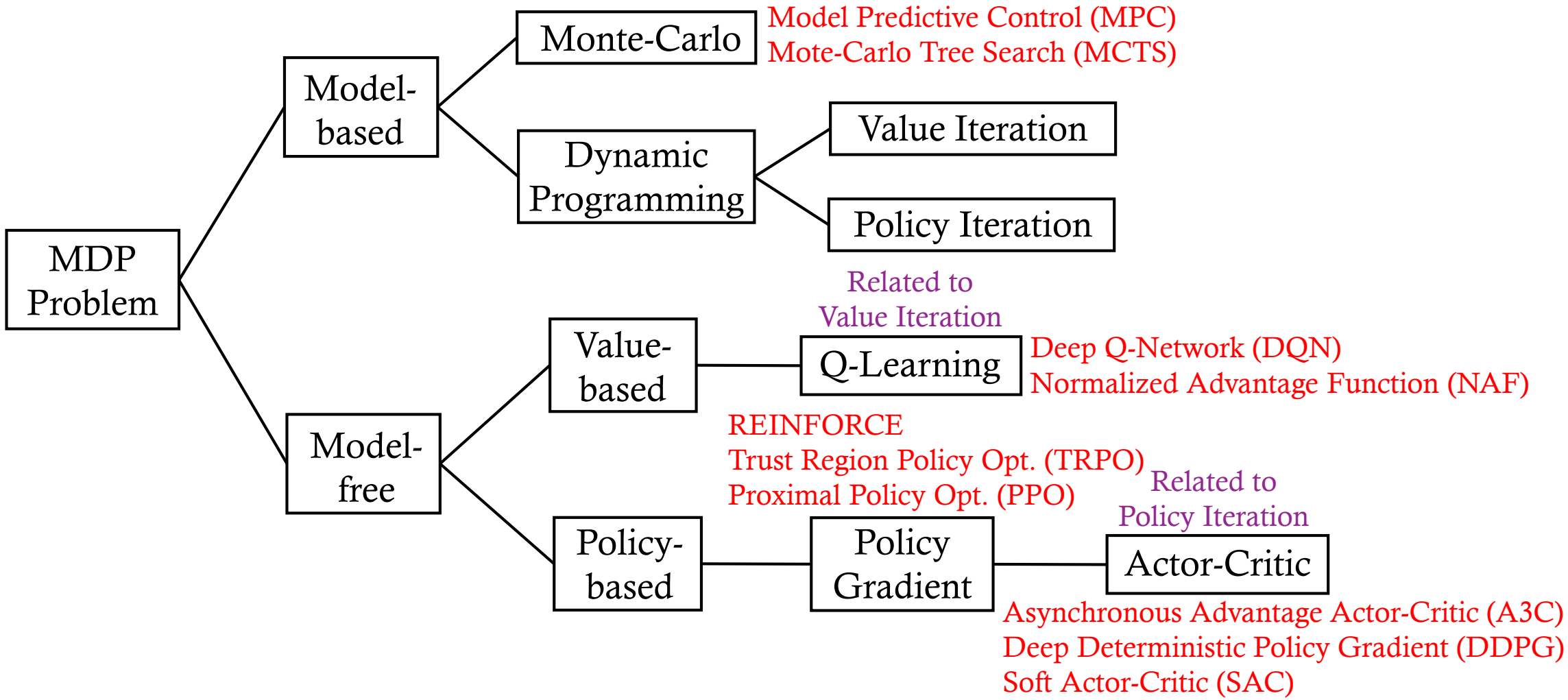
- Unsupervised learning aims to find the potential structure of the data without label.
- No reward RL aims to find the potential strategy of the environment without explicit reward.
- For example, given the control of legs, we can first learn the skills that can critically change the state, such as raising, moving forward, turning, etc., and then use these potential skills to quickly adapt to new tasks.

Curiosity-Driven



Skill-Driven

Reinforcement Learning Algorithms



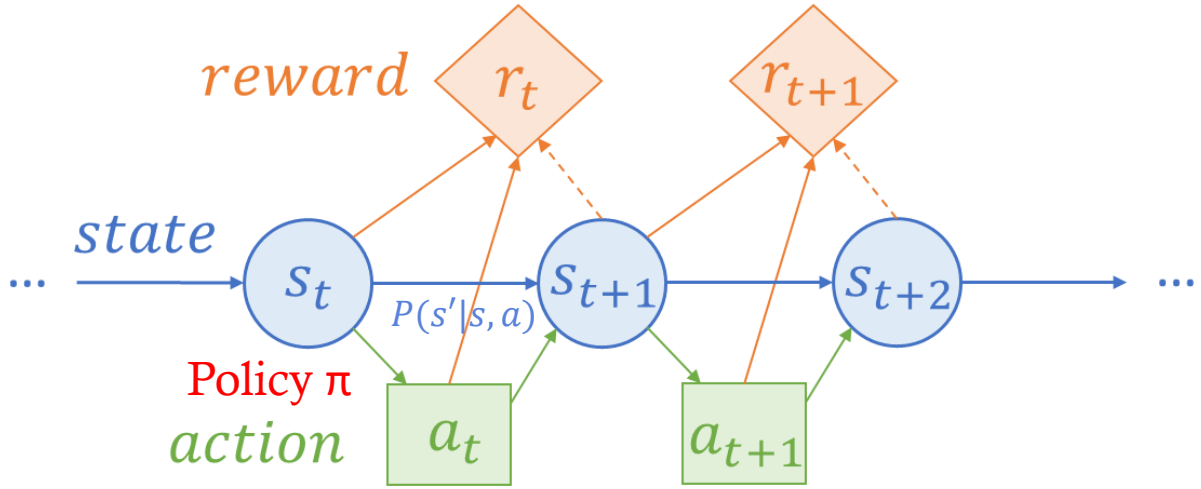
Outline

- Introduction to Reinforcement Learning
- Markov Decision Process (MDP)
  - MDP Formulation
  - Value Function and Bellman Equation
  - Dynamic Programming Methods
- Value-based Reinforcement Learning
  - From MDP to RL
  - Temporal Difference Learning (SARSA / Q-Learning)
  - Deep Q-Network (DQN) & Normalized Advantage Function (NAF)

Markov Decision Process (MDP)

- A mathematical framework for modeling the decision-making process, where the transition probability is assumed to have Markov property.
- The goal of MDP is to find the optimal policy π^* to maximize the expected total reward.

$$\pi^* = \underset{\pi}{\operatorname{argmax}}\, \mathbb{E}_{(s_t,a_t)\sim\pi,P}[r_0 + r_1 + \cdots + r_T]$$
$$= \underset{\pi}{\operatorname{argmax}} \sum_{t=0}^{T} \mathbb{E}_{(s_t,a_t)\sim\pi,P}[r(s_t, a_t)]$$

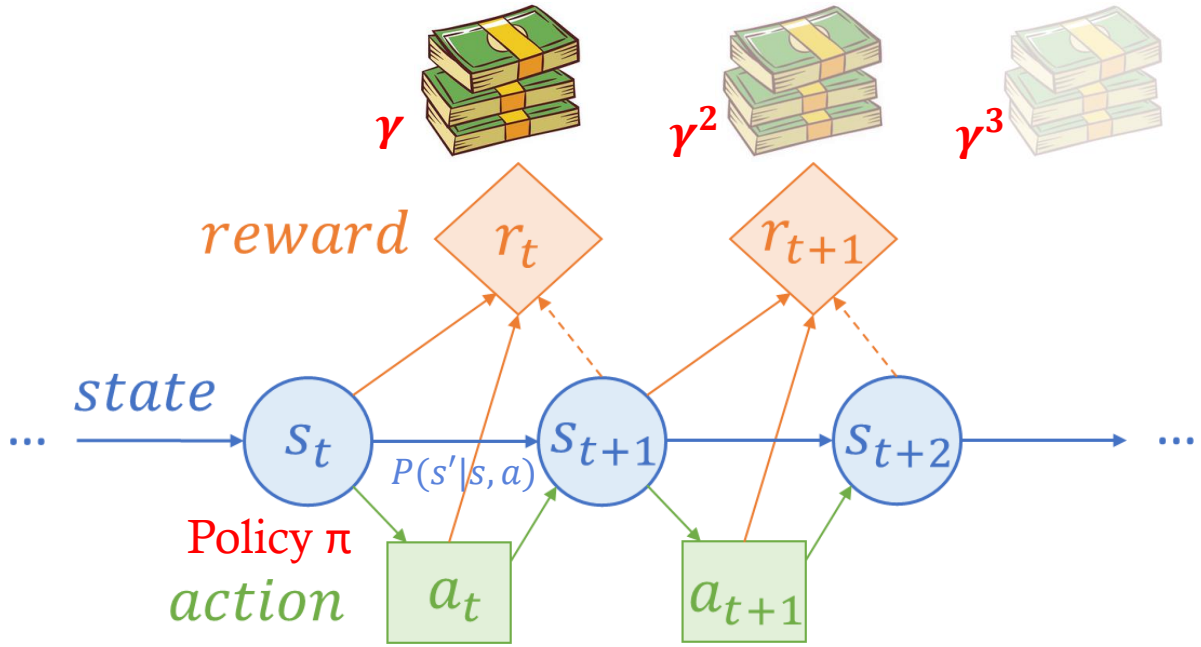


Discount Factor

- The expected total reward may not converge to finite number.
- The current reward are more important than the future reward.

$$\pi^* = \underset{\pi}{\operatorname{argmax}}\, \mathbb{E}_{(s_t,a_t)\sim\pi,P}[r_0 + \boldsymbol{\gamma^1} r_1 + \cdots + \boldsymbol{\gamma^T} r_T]$$

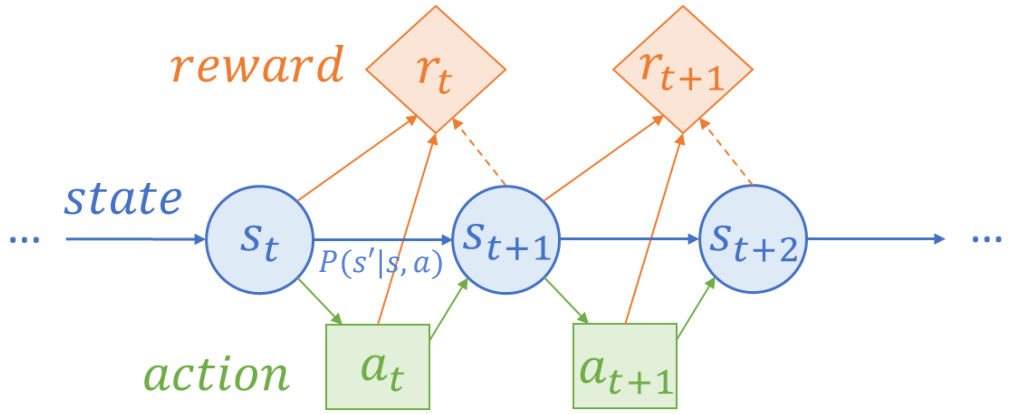
$$= \underset{\pi}{\operatorname{argmax}} \sum_{t=0}^{T} \mathbb{E}_{(s_t,a_t)\sim\pi,P}[\boldsymbol{\gamma^t} r(s_t, a_t)]$$



MDP Formulation

- 5-Tuple $< \mathcal{S}, \mathcal{A}, P, r, \gamma >$
  - State: $s \in \mathcal{S}$
  - Action: $a \in \mathcal{A}$
  - Transition Probability: $P(s'|s,a),\ P: \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0,1]$
  - Reward Function: $r(s,a),\ r: \mathcal{S} \times \mathcal{A} \to \mathbb{R}$
  - Discount Factor: $\gamma \in [0,1]$

- Policy Form
  - Deterministic Policy: $\mu(s) \to a$
  - Stochastic Policy: $\pi(s) \to p(a|s)$



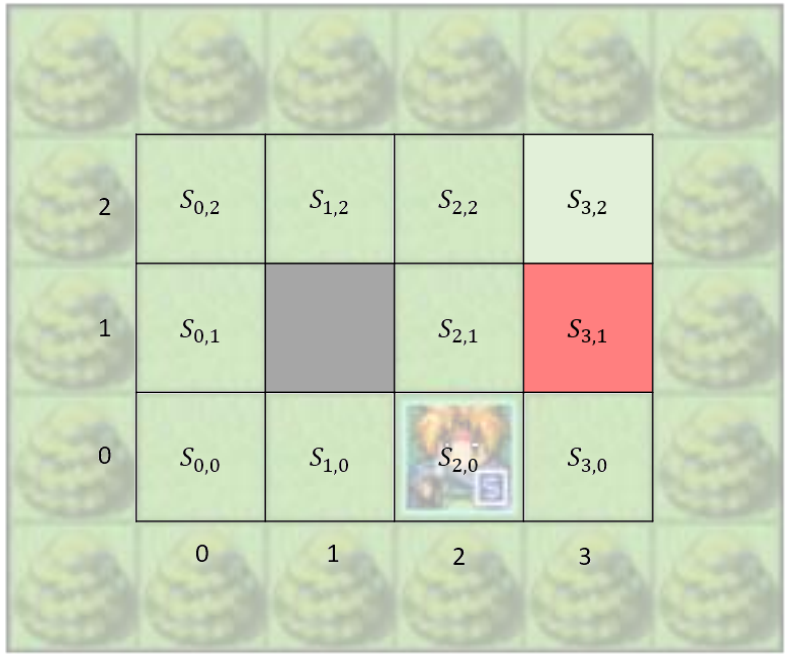
State

- A state describes the current status.
- Sometimes it is partially observed.
- For example, position, velocity, angle, …

$s_{1,2} = (1,2)$

$s_{0,0} = (0,0)$

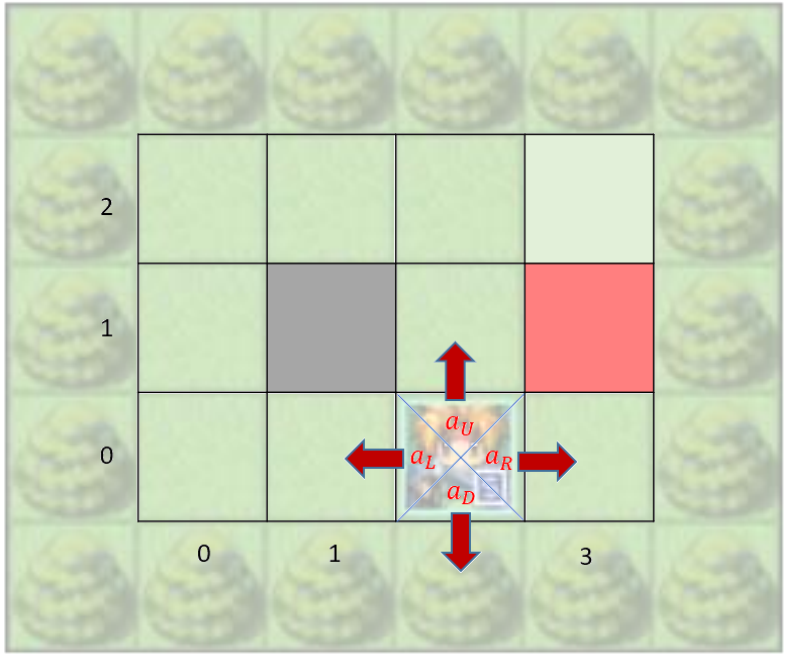
$$s_t = s_{2,1} = (2,1)$$
$$s_{t+1} = s_{3,1} = (3,1)$$
$$\vdots$$



Action

- The action space defines all possible actions that can be taken for each timestep.
- For example, go forward, turn right, add force, …

$$a_t = a_U = \uparrow$$
$$a_{t+1} = a_R = \rightarrow$$
$$\vdots$$



Reward Function

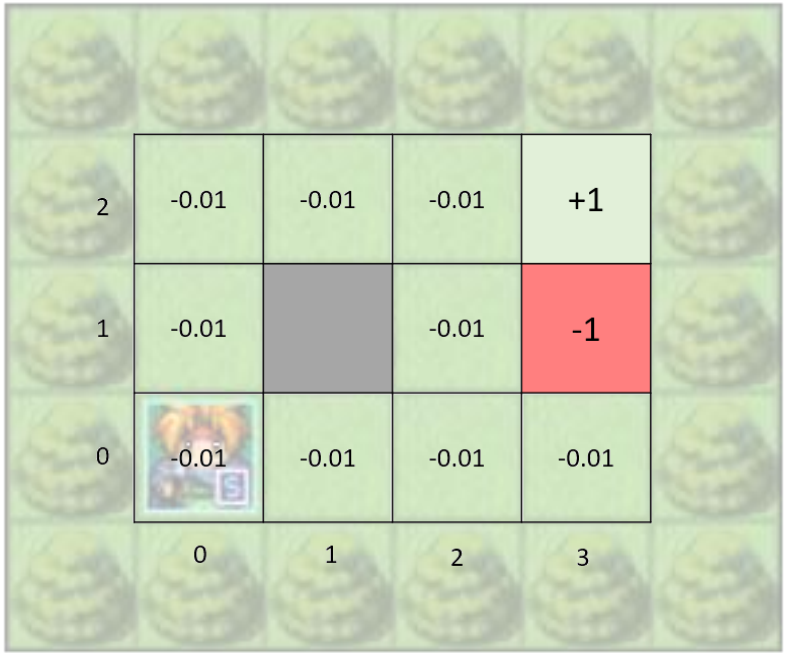
- For each timestep, it gives a reward according to the state-action pair (s_t, a_t) .
- It can be thought of as a signal to guide the agent to take the right actions.

$$r_{t+1} = r(s_t, a_t)$$

$$r_t = r(s_{2,1}, a_U) = r((2,1), \uparrow) = -0.01$$

$$r_{t+1} = r(s_{2,2}, a_R) = r((2,2), \rightarrow) = +1$$

$$\vdots$$



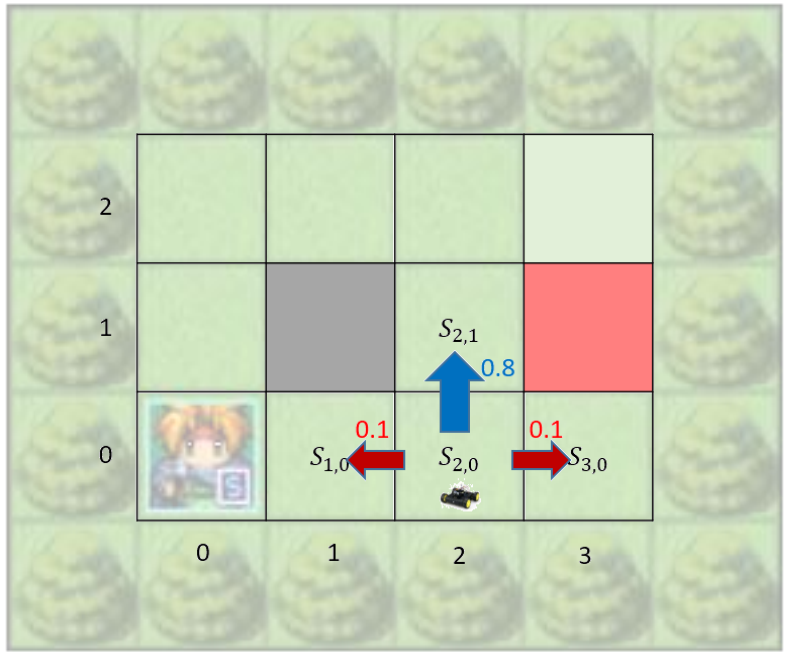
Transition Probability

- It defines the (stochastic) model of the MDP.
- This probability is used to encode all the factors of the environment dynamics.

$$P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$$

$$s' \sim P(s'|s, a)$$

$$P(s_{2,1}|s_{2,0}, a_U) = 0.8$$
$$P(s_{1,0}|s_{2,0}, a_U) = 0.1$$
$$P(s_{3,0}|s_{2,0}, a_U) = 0.1$$
$$P(s_{0,0}|s_{2,0}, a_U) = 0$$



Policy

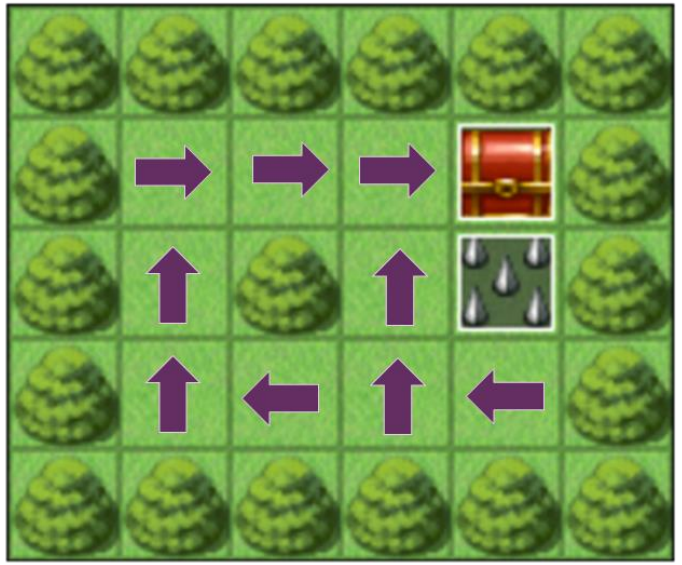
- We need a policy to decide the action to be taken at each timestep.
- The policy can be deterministic or stochastic.

$$a \sim \pi(a|s)$$

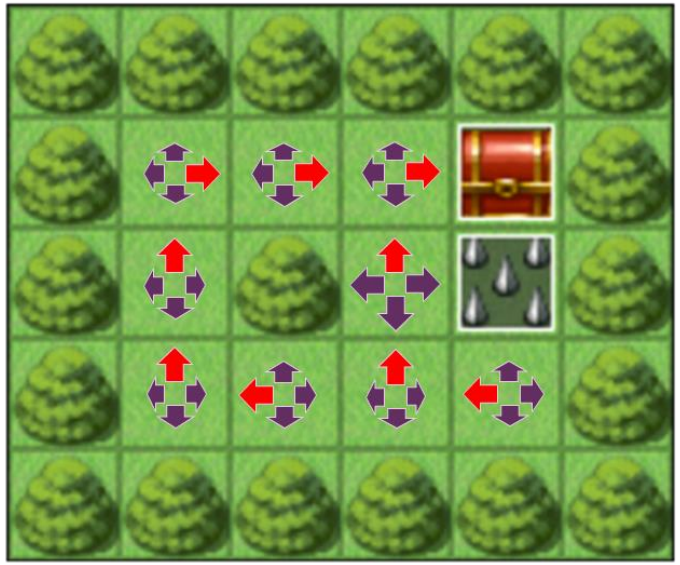
$$\pi(a_U|s_{2,1}) = 1$$
$$\pi(a_D|s_{2,1}) = 0$$
$$\pi(a_L|s_{2,1}) = 0$$
$$\pi(a_R|s_{2,1}) = 0$$

$$\implies \pi(a|s_{2,1}) = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Deterministic



Stochastic



Outline

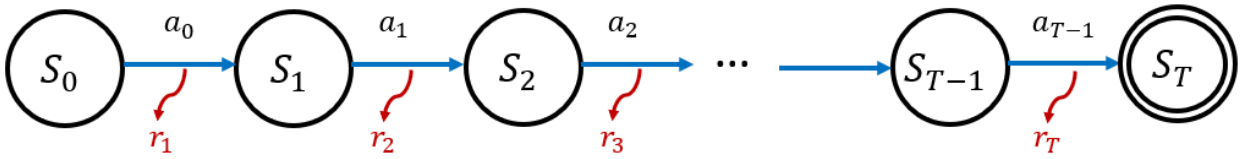
- Introduction to Reinforcement Learning
- Markov Decision Process (MDP)
  - MDP Formulation
  - Value Function and Bellman Equation
  - Dynamic Programming Methods
- Value-based Reinforcement Learning
  - From MDP to RL
  - Temporal Difference Learning (SARSA / Q-Learning)
  - Deep Q-Network (DQN) & Normalized Advantage Function (NAF)

Return

- The goal of the MDP is to take a sequence of actions to maximize the expected total reward.
- Given a sequence of actions, we can define the "Return" as the sum of reward from current state to the terminal state.

$$G_0 = r_1 + r_2 + r_3 + \cdots + r_T = \sum_{t=1}^{T} r_t$$ Undiscounted

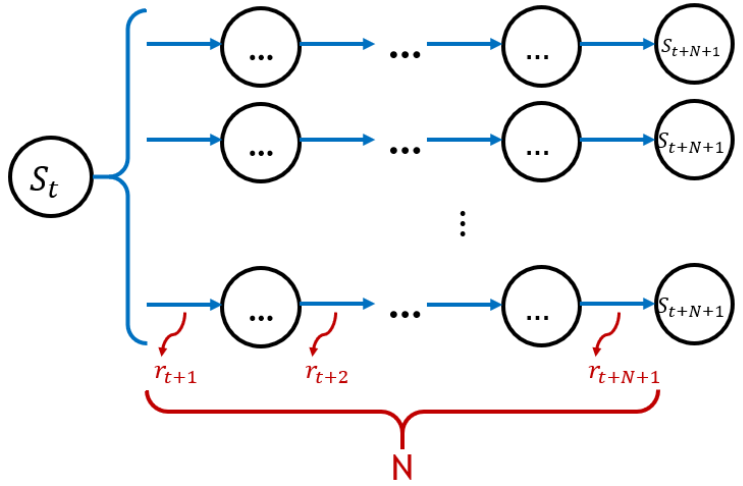
$$G_0 = r_1 + \gamma r_2 + \gamma^2 r_3 + \cdots + \gamma^{T-1} r_T = \sum_{t=1}^{T} \gamma^{t-1} r_t$$ Discounted



Return

- Consider the return from timestep t
- Define the return recursively:

$$G_t = r_{t+1} + \gamma G_{t+1}$$



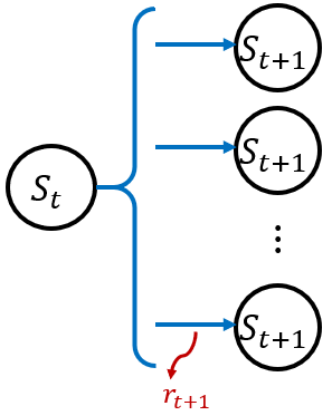
1-step

2-step

⋮

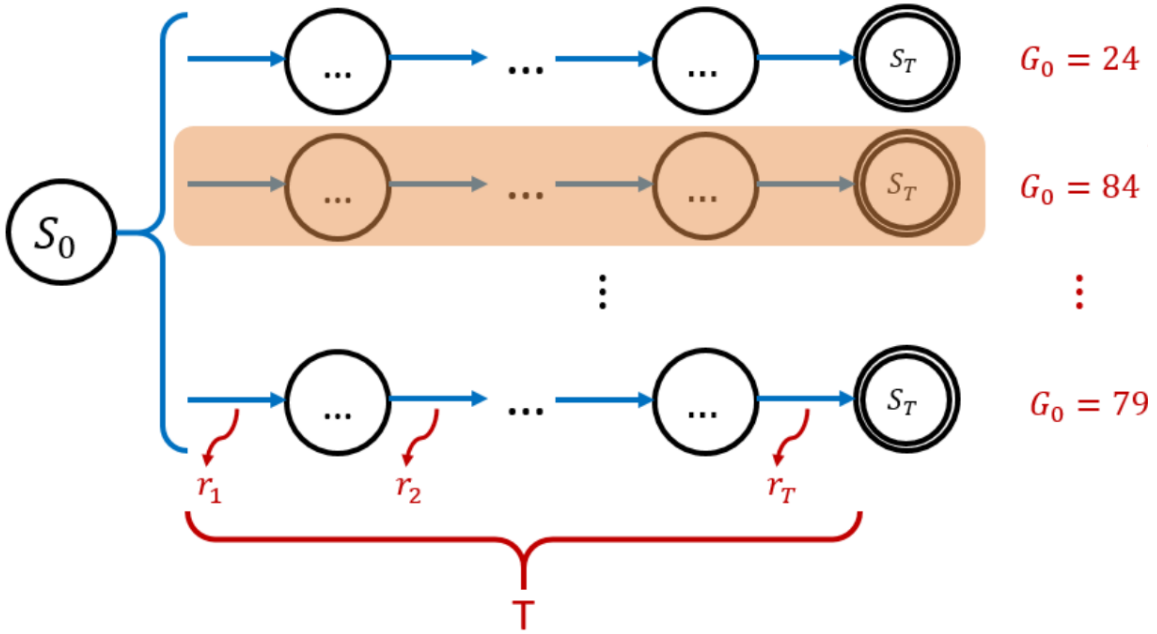
N-step

$$\begin{aligned}
G_t &= r_{t+1} + \gamma(r_{t+2} + \gamma r_{t+3} + \gamma^2 r_{t+4} + \ldots + \gamma^{T-2} r_T) \\
&= r_{t+1} + \gamma G_{t+1} \\
&= r_{t+1} + \gamma r_{t+2} + \gamma^2 G_{t+2} \\
&= \ldots \\
&= r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots + \gamma^N r_{t+N+1} + \gamma^{N+1} G_{t+N+1} \\
&= \sum_{k=0}^{N} \gamma^k r_{t+k+1} + \gamma^{N+1} G_{t+N+1}
\end{aligned}$$



Return

- However, we CANNOT directly find the path that maximizes the total return because we have to consider the stochasticity of the environment.
- Even though the same action sequence is taken for each episode, we probably won't have the same state sequence (won't have the same return value).



State-Value Function

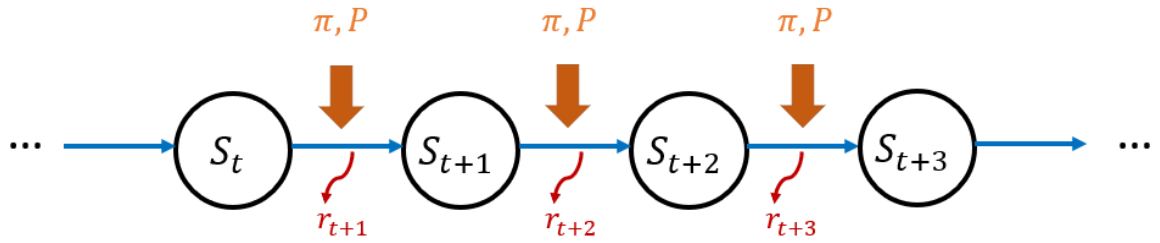
$V(S_{0,2})$	$V(S_{1,2})$	$V(S_{2,2})$	+1
$V(S_{0,1})$		$V(S_{2,1})$	-1
$V(S_{0,0})$	$V(S_{1,0})$	$V(S_{2,0})$	$V(S_{3,0})$

- Consider the expected return from current state

$$V^{\pi}(s_t) = \mathbb{E}_{\pi}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^T r_T | s_t = s]$$
$$= E_{\pi}[G_t | s_t = s]$$

- Recursive Definition

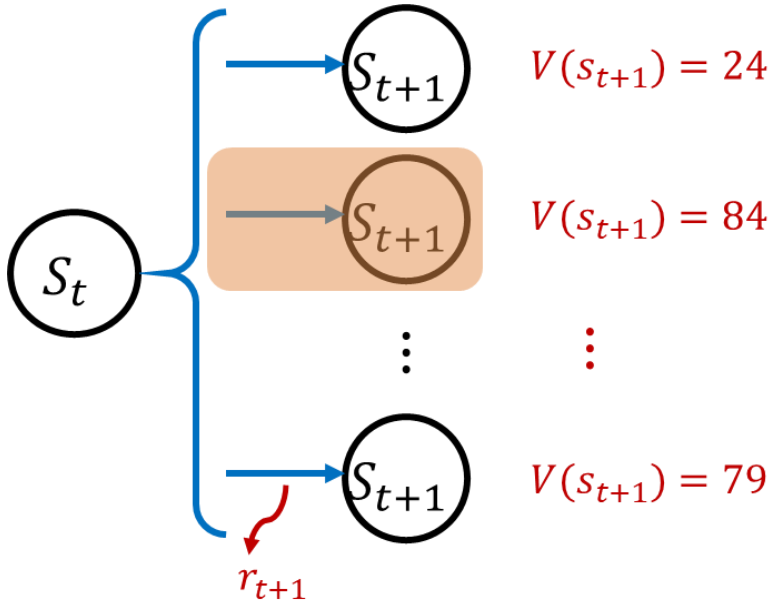
$$V(s) = E_{\pi}[G_t | s_t = s]$$
$$= E_{\pi}[r_{t+1} + \gamma G_{t+1} | s_t = s]$$ Apply recursive form of return
$$= E_{a\sim\pi(a|s)} E_{s'\sim P(s'|s,a)}[r_{t+1} + \gamma E_{\pi}[G_{t+1} | s_{t+1} = s'] | s_t = s]$$
$$= E_{a\sim\pi(a|s)} E_{s'\sim P(s'|s,a)}[r_{t+1} + \gamma V(s') | s_t = s]$$

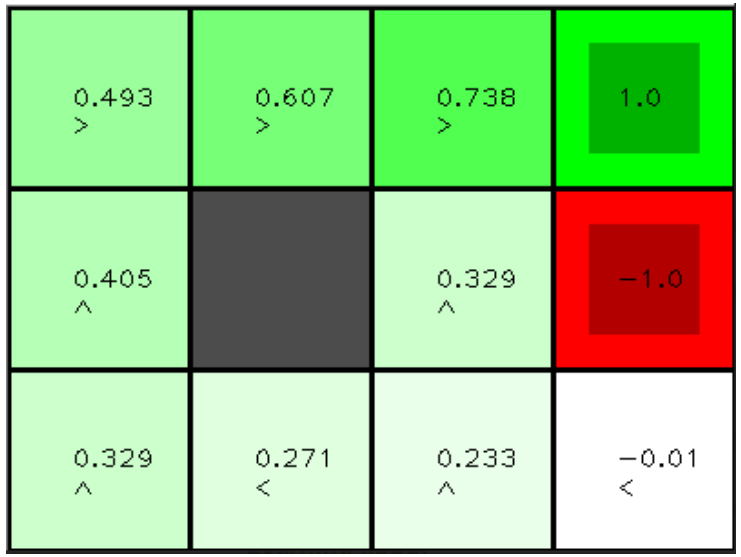


State-Value Function

- If we have the state-value function, we can design a policy to choose an action according to the maximum state-value of the next state s_{t+1}.

$$\pi(s) = \max_a \{\mathbb{E}_{a\sim P}[r(s,a) + \gamma V(s')]\}$$





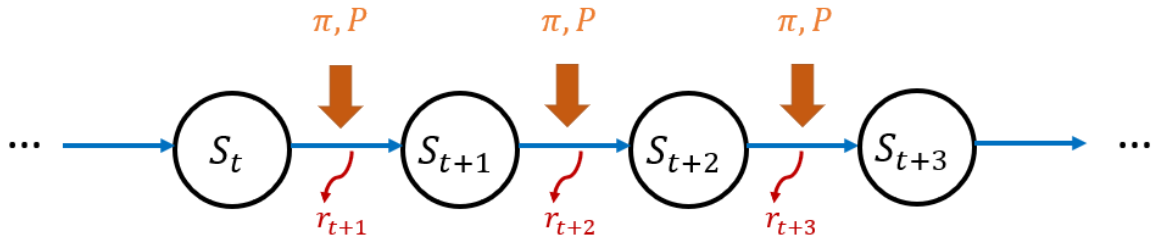
Action-Value Function (Q-function)

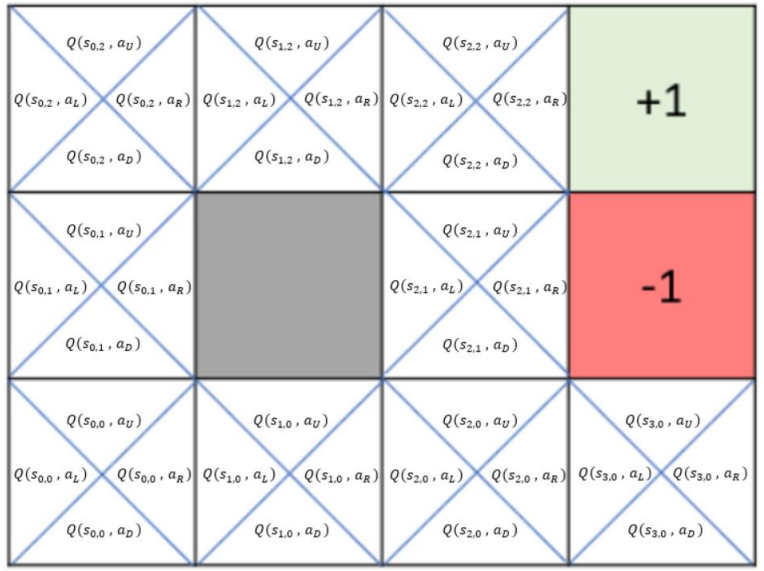
- Consider the expected return from current state s given the action a.

$$Q^{\pi}(s_t) = \mathbb{E}_{\pi}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^T r_T | s_t = s, a_t = a]$$
$$= E_{\pi}[G_t | s_t = s, a_t = a]$$

- Recursive Definition

$$\begin{aligned} Q(s,a) &= E_{\pi}[G_t | s_t = s, a_t = a] \\ &= E_{\pi}[r_{t+1} + \gamma G_{t+1} | s_t = s, a_t = a] \\ &= E_{s' \sim P(s'|s,a)}[r_{t+1} + \gamma E_{\pi}[G_{t+1} | s_{t+1} = s'] | s_t = s, a_t = a] \\ &= E_{s' \sim P(s'|s,a)}[r_{t+1} + \gamma V(s') | s_t = s, a_t = a] \\ &= \sum_{s'} P(s'|s,a)[r(s,a) + \gamma V(s')] \end{aligned}$$

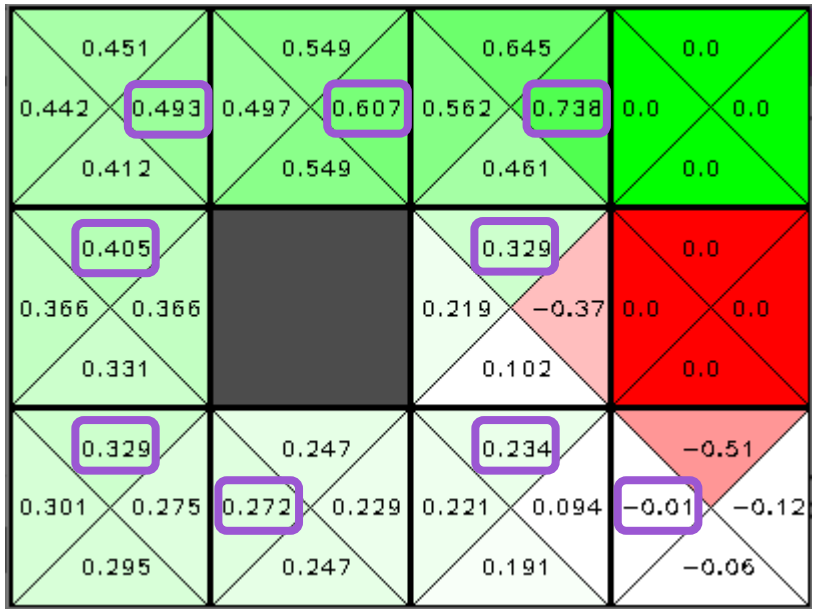




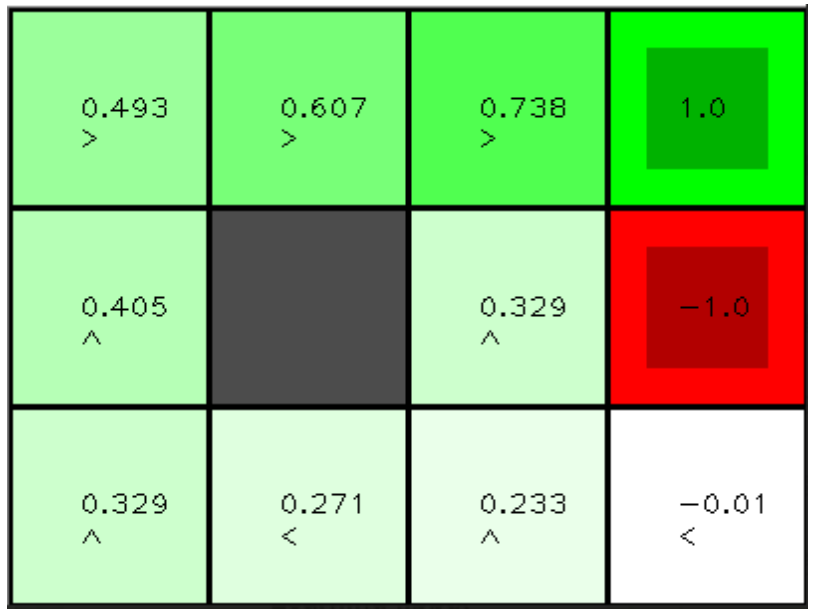
Action-Value Function (Q-function)

- If we have the action-value function, we can design a policy to choose an action with maximum action-value.

$$V(s) = \max_a Q(s,a)$$



Q-Value



Value

Bellman Expectation Equation

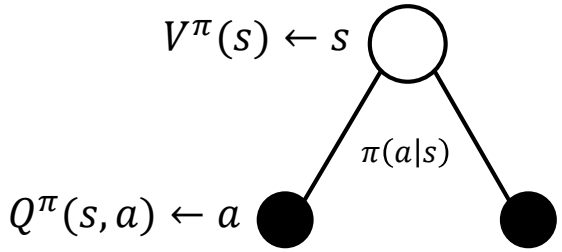
- Bellman Expectation Equation is the recursive form of value functions.
- State-Value

$$V^{\pi}(s) = \mathbb{E}_{\pi}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^T r_T | s_t = s]$$
$$= \sum_a \pi(a|s) Q^{\pi}(s,a) = \sum_a \pi(a|s) \sum_{s'} P(s'|s,a)[r(s,a) + \gamma V^{\pi}(s')]$$

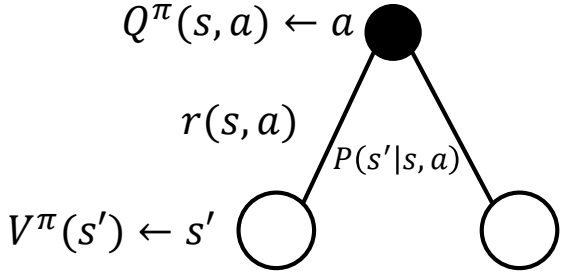
- Action-Value

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^T r_T | s_t = s, a_t = a]$$
$$= \sum_{s'} P(s'|s,a)[r(s,a) + \gamma V^{\pi}(s')] = \sum_{s'} P(s'|s,a) \left[r(s,a) + \gamma \sum_{a'} \pi(a'|s')\, Q(s',a') \right]$$

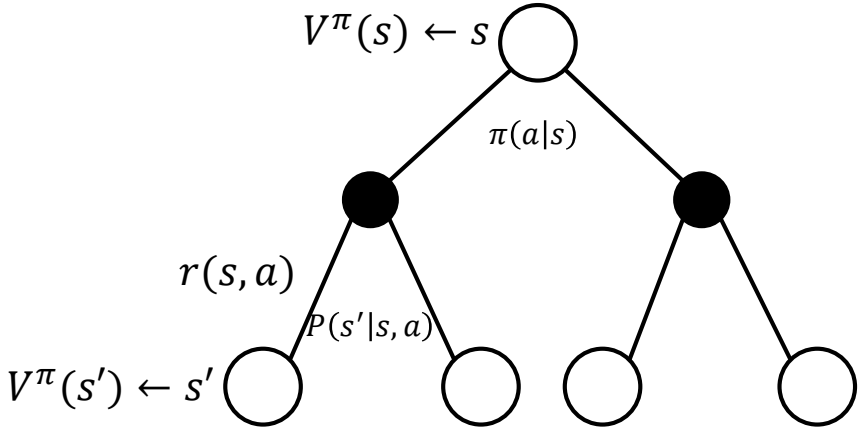
Recursive Relation of Value Functions



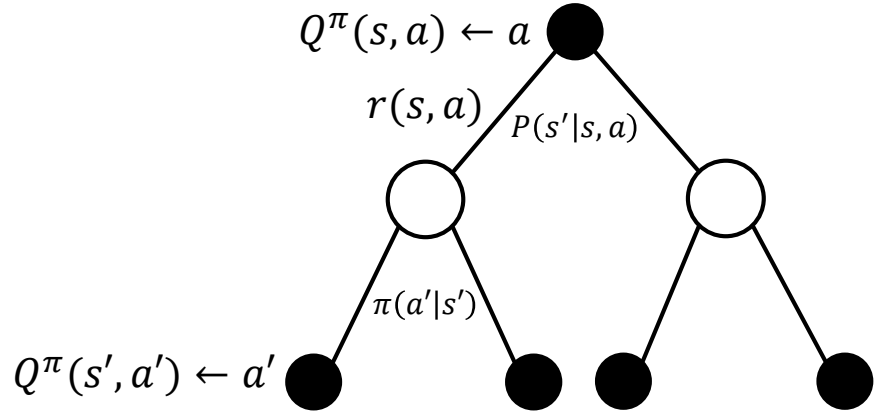
$$V^\pi(s) = \sum_a \pi(a|s) Q^\pi(s,a)$$



$$Q^\pi(s,a) = r(s,a) + \gamma \sum_{s'} P(s'|s,a) V^\pi(s')$$



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



$$Q^\pi(s,a) = r(s,a) + \gamma \sum_{s'} P(s'|s,a) \sum_{a'} \pi(a'|s') Q^\pi(s',a')$$

Bellman Optimality Equation

- Bellman Expectation Equation is the recursive form of optimal value functions.
- State-Value

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

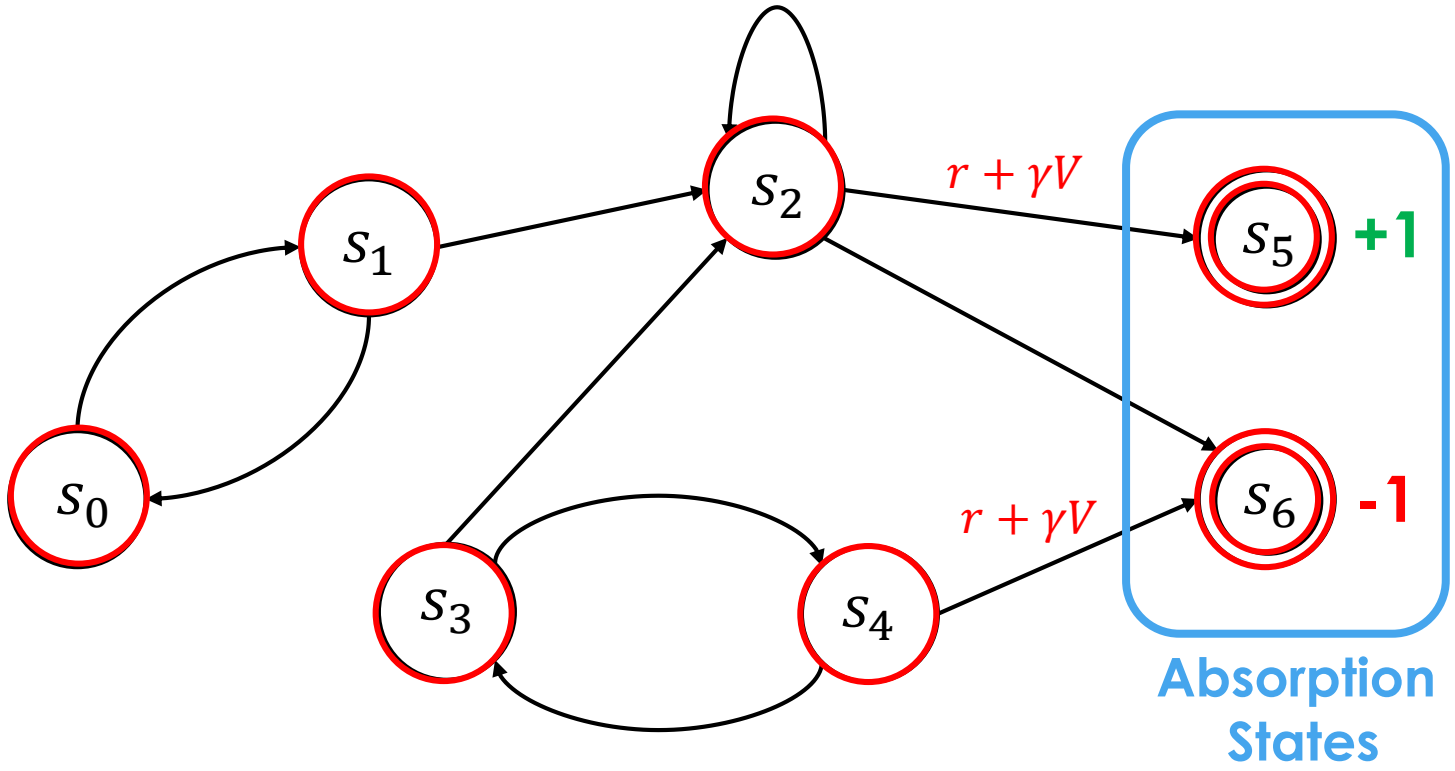
- Action-Value

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

Outline

- Introduction to Reinforcement Learning
- Markov Decision Process (MDP)
  - MDP Formulation
  - Value Function and Bellman Equation
  - Dynamic Programming Methods
- Value-based Reinforcement Learning
  - From MDP to RL
  - Temporal Difference Learning (SARSA / Q-Learning)
  - Deep Q-Network (DQN) & Normalized Advantage Function (NAF)

Dynamic Programming



Bellman Expectation Backup

- Define Bellman Expectation Backup Operator: $\mathcal{T}^{\pi}$

$$\mathcal{T}^{\pi}V(s_t) \triangleq \mathbb{E}_{s_{t+1}\sim P}[r(s_t, a_t) + \gamma V(s_{t+1})]$$

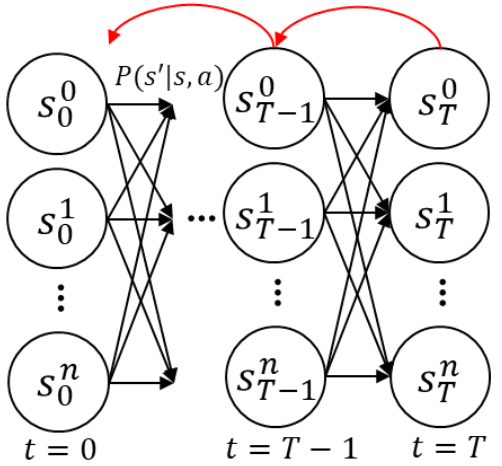
$$\mathcal{T}^{\pi}Q(s_t, a_t) \triangleq \mathbb{E}_{s_{t+1}\sim P}\left[r(s_t, a_t) + \gamma \mathbb{E}_{a_{t+1}\sim \pi}[Q(s_{t+1}, a_{t+1})]\right]$$

- Convergence of Bellman Expectation Backup

$$V \to \mathcal{T}^{\pi}V \to (\mathcal{T}^{\pi})^2V \to (\mathcal{T}^{\pi})^3V \to \cdots \to V^{\pi}$$

$$Q \to \mathcal{T}^{\pi}Q \to (\mathcal{T}^{\pi})^2Q \to (\mathcal{T}^{\pi})^3Q \to \cdots \to Q^{\pi}$$

$$V(s_0) = r_0 + \gamma(r_1 + \gamma(\ldots(r_{T-1} + \gamma V(s_T))))$$



Bellman Optimality Backup

- If we operate the Bellman Optimality Backups to V^* and Q^*

$$\mathcal{T}^* V^*(s_t) \triangleq \max_{\mathrm{a}} \mathbb{E}_{s_{t+1}\sim P}[r(s_t, a_t) + \gamma V(s_{t+1})]$$

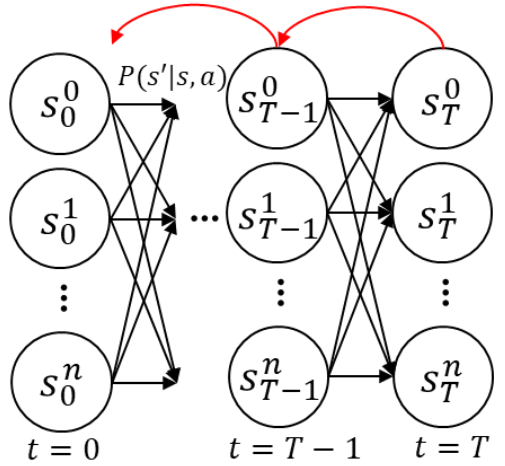
$$\mathcal{T}^* Q^*(s_t, a_t) \triangleq \mathbb{E}_{s_{t+1}\sim P}\left[r(s_t, a_t) + \gamma \max_{a} Q(s_{t+1}, a_{t+1})\right]$$

- Convergence of Bellman Optimality Backup

$$V \to \mathcal{T}^* V \to (\mathcal{T}^*)^2 V \to (\mathcal{T}^*)^3 V \to \cdots \to V^*$$

$$Q \to \mathcal{T}^* Q \to (\mathcal{T}^*)^2 Q \to (\mathcal{T}^*)^3 Q \to \cdots \to Q^*$$

$$V(s_0) = r_0 + \gamma(r_1 + \gamma(\ldots(r_{T-1} + \gamma V(s_T))))$$



Convergence Proof

- Convergence of Bellman Expectation Backup

$$\mathcal{T}^{\pi}V(s_t) \triangleq \mathbb{E}_{s_{t+1}\sim P, a_t\sim\pi}[r(s_t,a_t)+\gamma V(s_{t+1})] = R^{\pi}+\gamma P^{\pi}V$$

maximum element

$$\|\mathcal{T}^{\pi}V - V^{\pi}\|_{\infty} = \|\mathcal{T}^{\pi}V - \mathcal{T}^{\pi}V^{\pi}\|_{\infty} = \|(R^{\pi}+\gamma P^{\pi}V) - (R^{\pi}+\gamma P^{\pi}V^{\pi})\|_{\infty}$$
$$= \|\gamma P^{\pi}(V - V^{\pi})\|_{\infty} \le \gamma \| V - V^{\pi}\|_{\infty}$$

transition probability, <= 1

- Convergence of Bellman Optimal Backup

$$\mathcal{T}^{*}V(s_t) \triangleq \max_{a} \mathbb{E}_{s_{t+1}\sim P}[r(s_t,a_t)+\gamma V(s_{t+1})] = \max_{a}\{R+\gamma P^{a}\}$$

$$\left\|\max_x f(x) - \max_x g(x)\right\| \le \max_x \|f(x) - g(x)\|$$

$$\|\mathcal{T}^{*}V - V^{*}\|_{\infty} = \|\mathcal{T}^{*}V - \mathcal{T}^{*}V^{*}\|_{\infty} = \left\|\max_a\{R+\gamma P^{a}V\} - \max_a\{R+\gamma P^{a}V^{*}\}\right\|_{\infty}$$

$$\le \max_a \|\{R+\gamma P^{a}V\} - \{R+\gamma P^{a}V^{*}\}\|_{\infty} = \gamma \left\|\max_a\{P^{a}\}(V - V^{*})\right\|_{\infty} \le \gamma \| V - V^{*}\|_{\infty}$$

Value Iteration

Initialize array V arbitrarily (e.g., $V(s) = 0$ for all $s \in \mathcal{S}^+$)

Repeat

$\Delta \leftarrow 0$

For each $s \in \mathcal{S}$:

$v \leftarrow V(s)$

$V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a)\left[r + \gamma V(s')\right]$

$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, π, such that

$\pi(s) = \text{argmax}_a \sum_{s',r} p(s', r|s, a)\left[r + \gamma V(s')\right]$

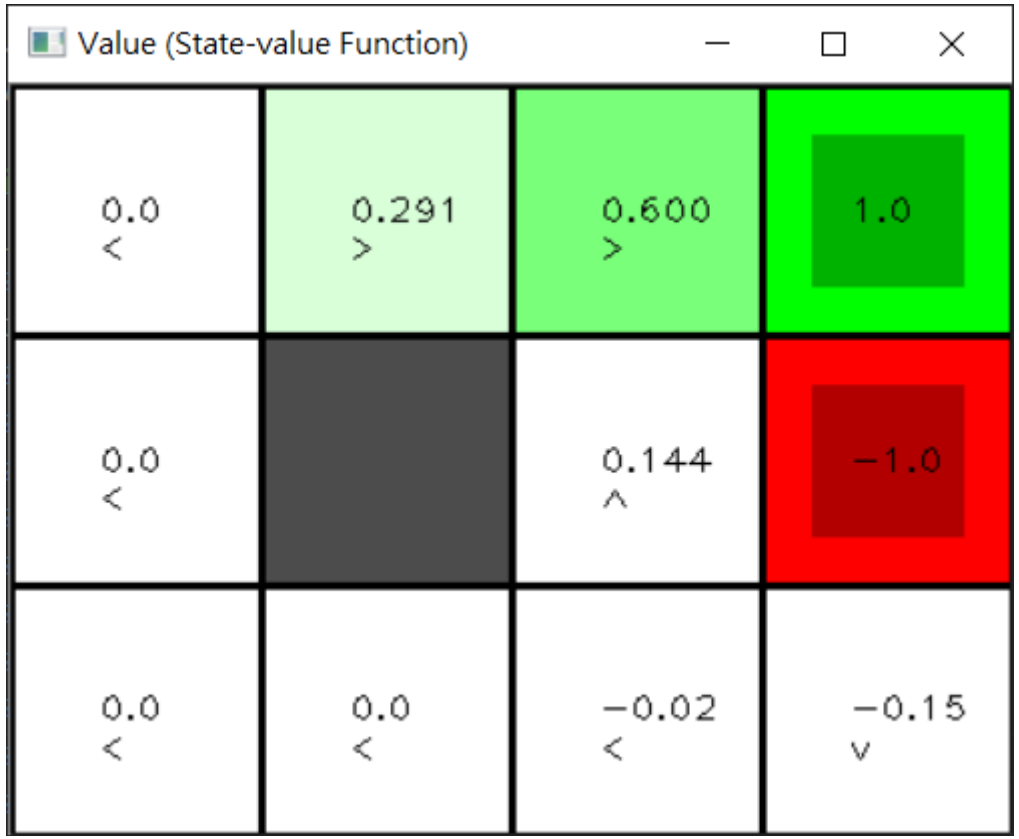
Bellman Optimality Equation:

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

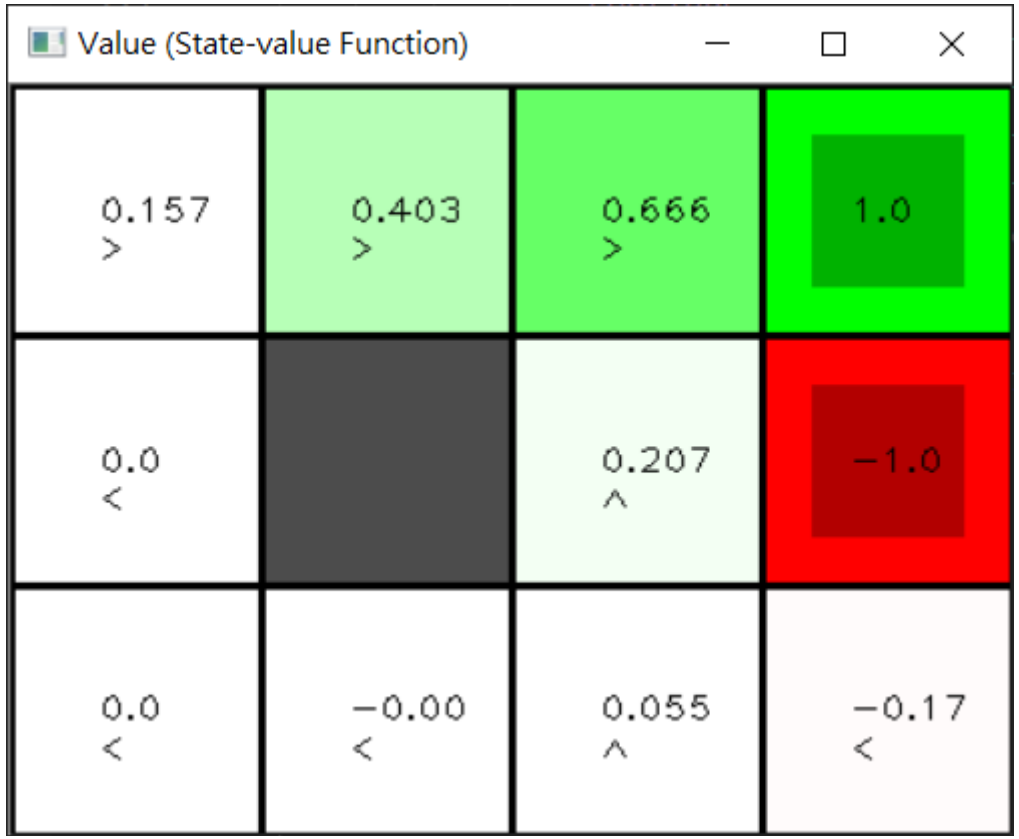
Value Iteration Demo - Initialization

Value Iteration Demo - Iteration 1

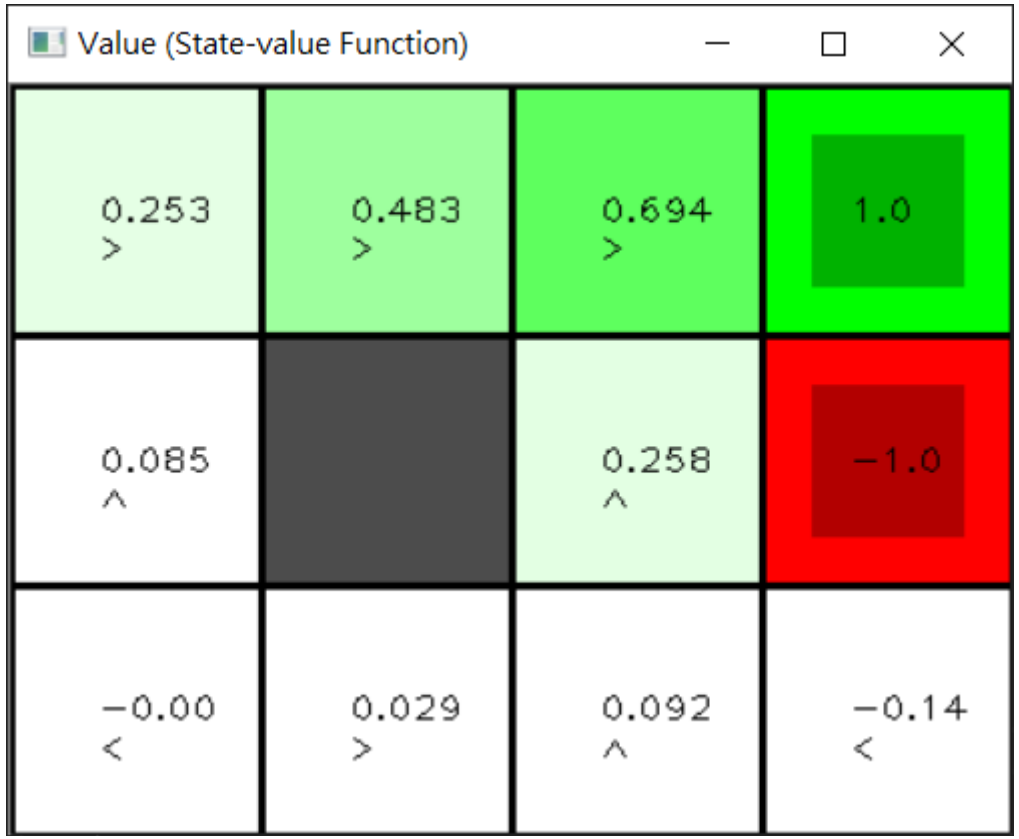
Value Iteration Demo - Iteration 2



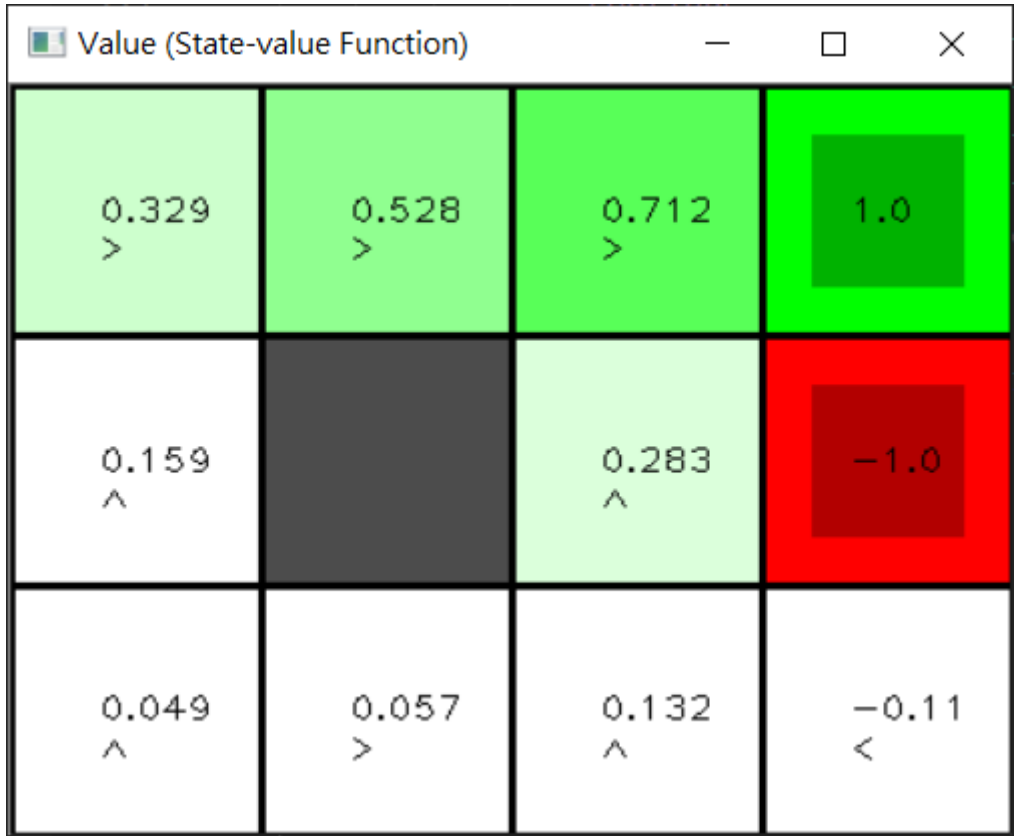
Value Iteration Demo - Iteration 3



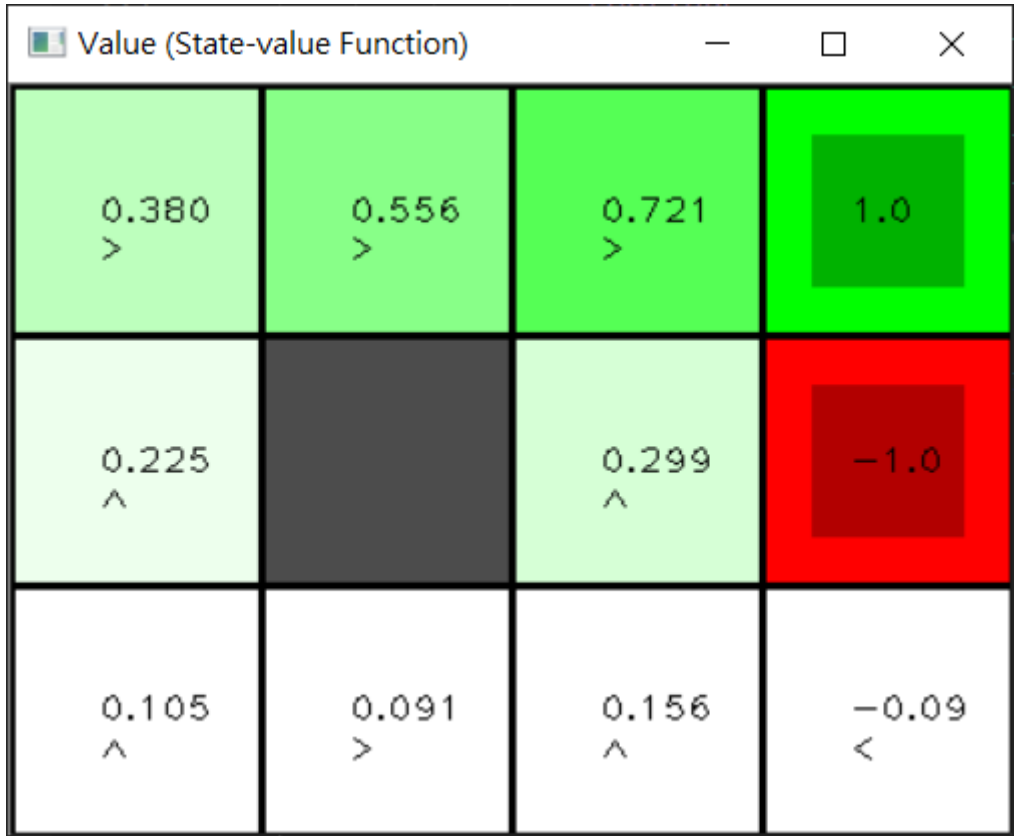
Value Iteration Demo - Iteration 4



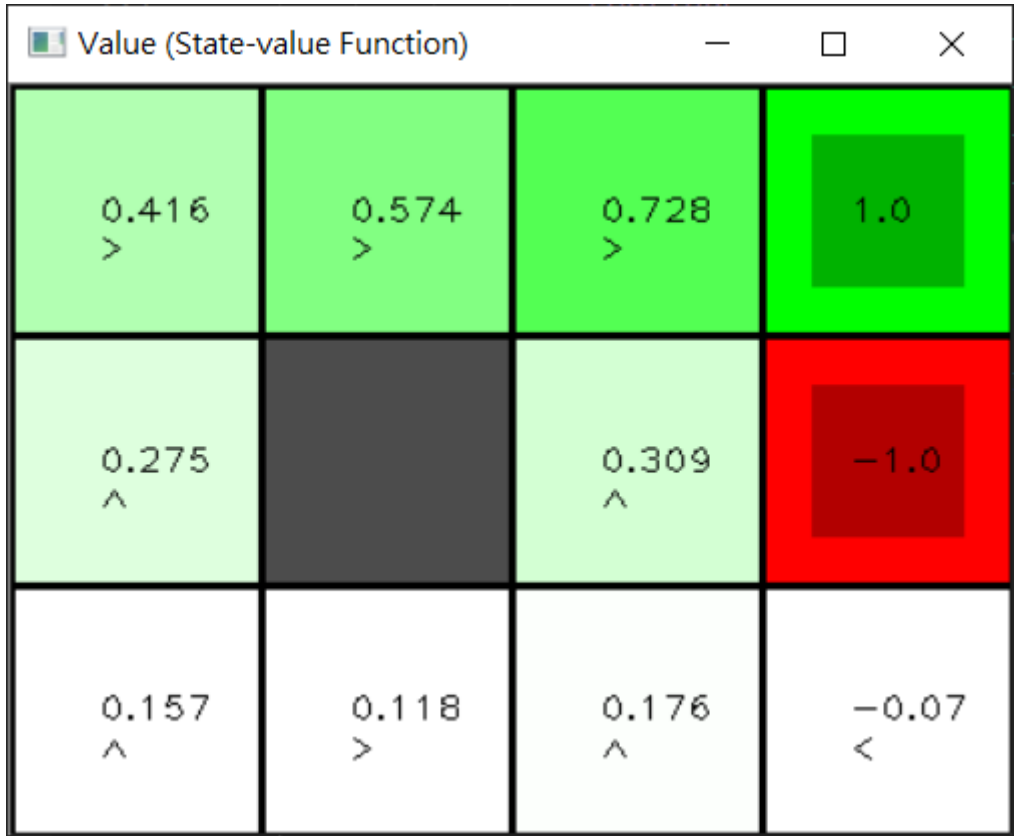
Value Iteration Demo - Iteration 5



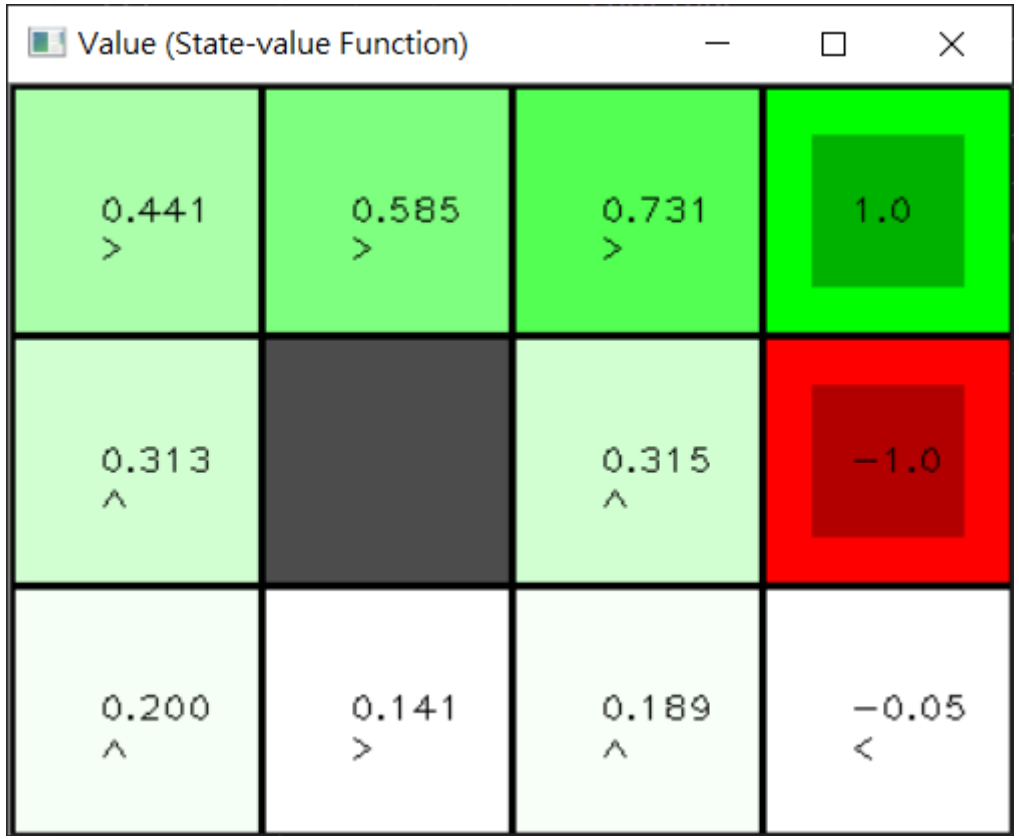
Value Iteration Demo - Iteration 6



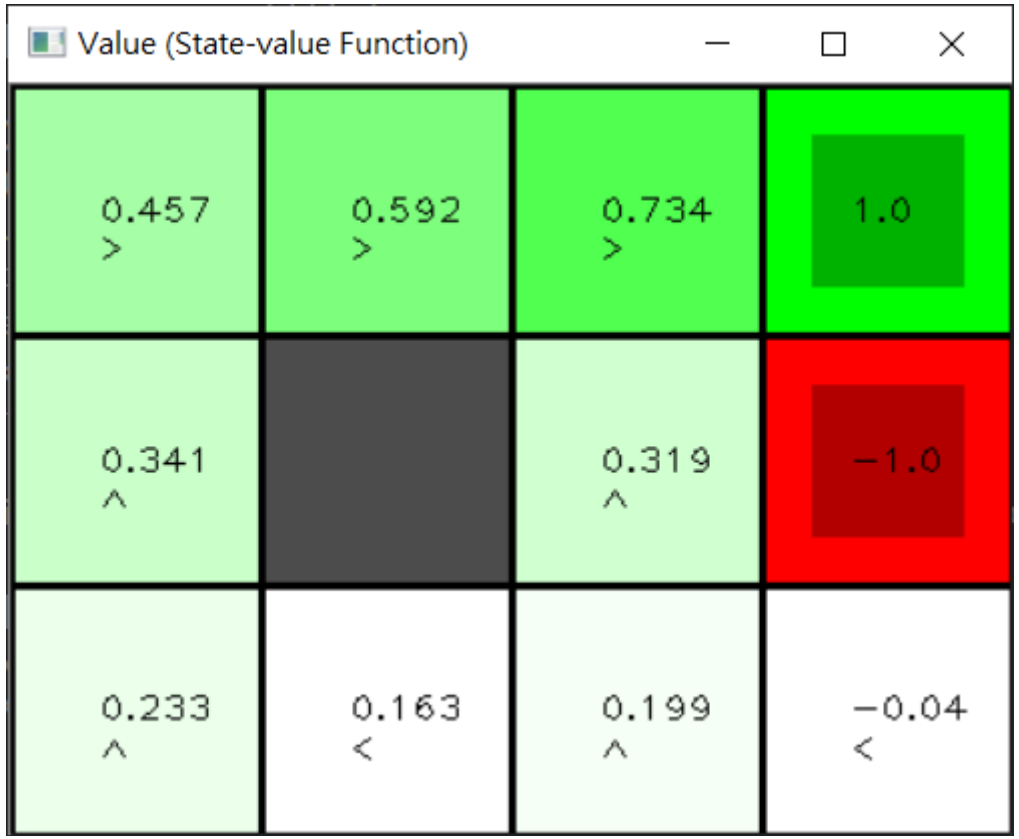
Value Iteration Demo - Iteration 7



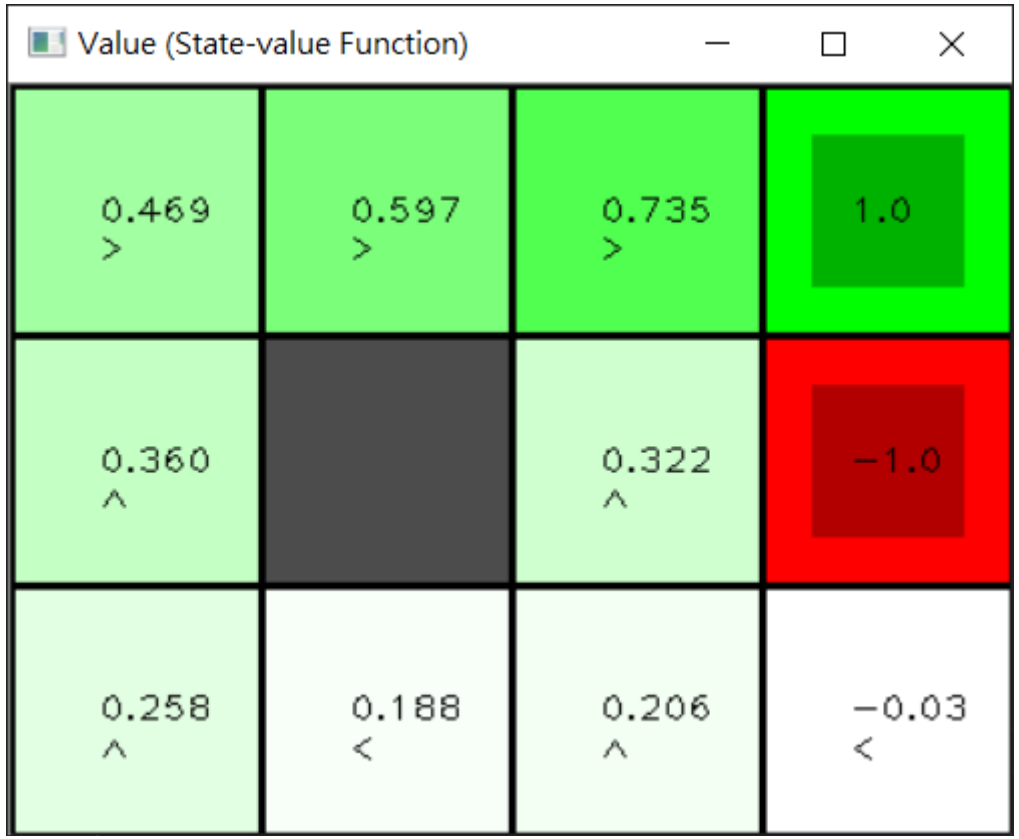
Value Iteration Demo - Iteration 8



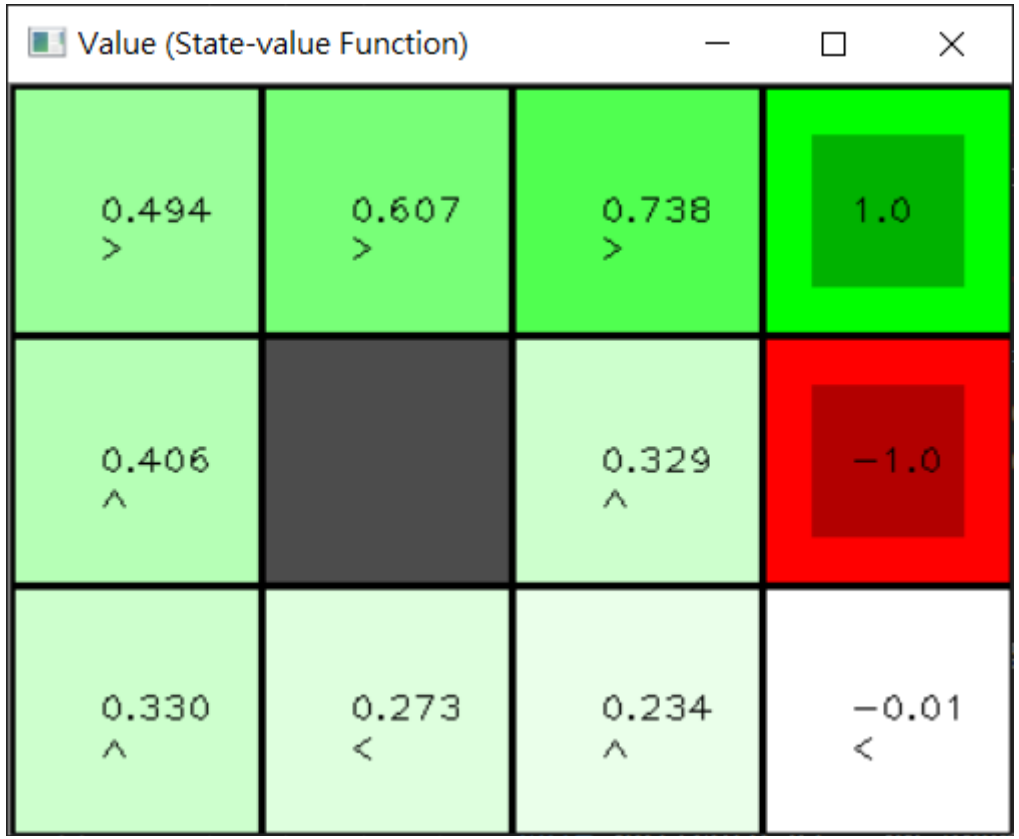
Value Iteration Demo - Iteration 9



Value Iteration Demo - Iteration 10



Value Iteration Demo - Iteration 30



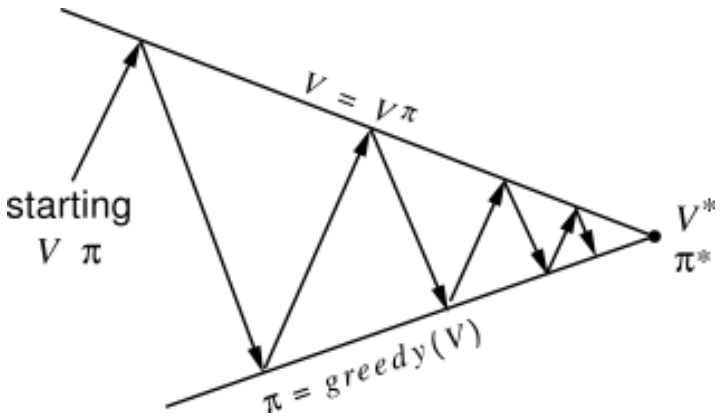
Policy Iteration

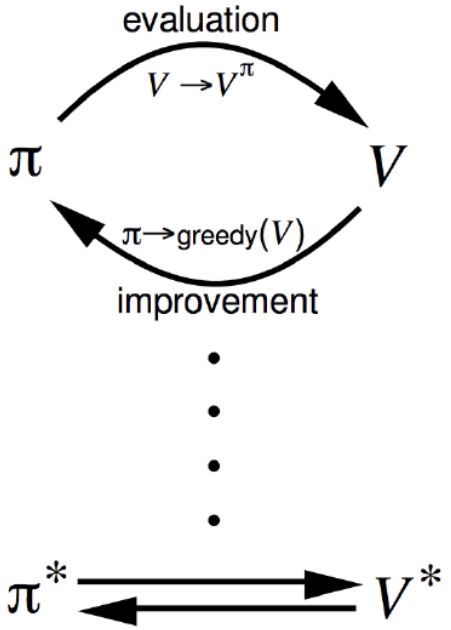
- Given a policy π
  - Evaluate the policy π (Policy Evaluation)

$$V^{\pi}(s_t) = E_{\pi}[G_t | s_t = s]$$

  - Improve the policy by acting greedily with respect to V (Policy Improvement)

$$\pi' = greedy(V^{\pi})$$





Improve the Policy

- Consider a deterministic policy $a = \pi(s)$
- Improve the policy greedily

$$\pi'(s) = \underset{a}{\operatorname{argmax}}\, Q^{\pi}(s, a)$$

- Improves the value from any state s over one step

$$Q^{\pi}\big(s, \pi'(s)\big) = \max_{a} Q^{\pi}(s, a) \geq Q^{\pi}\big(s, \pi(s)\big) = V^{\pi}(s)$$

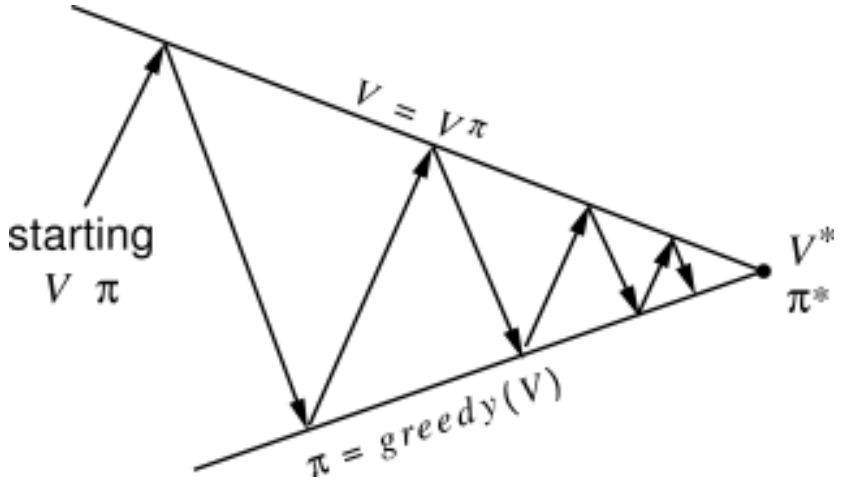
- Proof of improvement

$$\begin{aligned}
&V^{\pi}(s) \leq Q^{\pi}\big(s, \pi'(s)\big) = \mathbb{E}_{\pi'}[r_t + \gamma V^{\pi}(s_{t+1}) | s_t = s] \\
&\leq \mathbb{E}_{\pi'}\big[r_t + \gamma Q^{\pi}\big(s_{t+1}, \pi'(s_{t+1})\big) \big| s_t = s\big] \\
&\leq \mathbb{E}_{\pi'}\big[r_t + \gamma r_{t+1} + \gamma^2 Q^{\pi}\big(s_{t+2}, \pi'(s_{t+2})\big) \big| s_t = s\big] \\
&\leq \mathbb{E}_{\pi'}[r_t + \gamma r_{t+1} + \cdots | s_t = s] \\
&= V^{\pi'}(s)
\end{aligned}$$

$Q^{\pi}\big(s, \pi'(s)\big) \geq V^{\pi}(s)$

$V^{\pi}(s) \leq V^{\pi'}(s)$

Policy Iteration



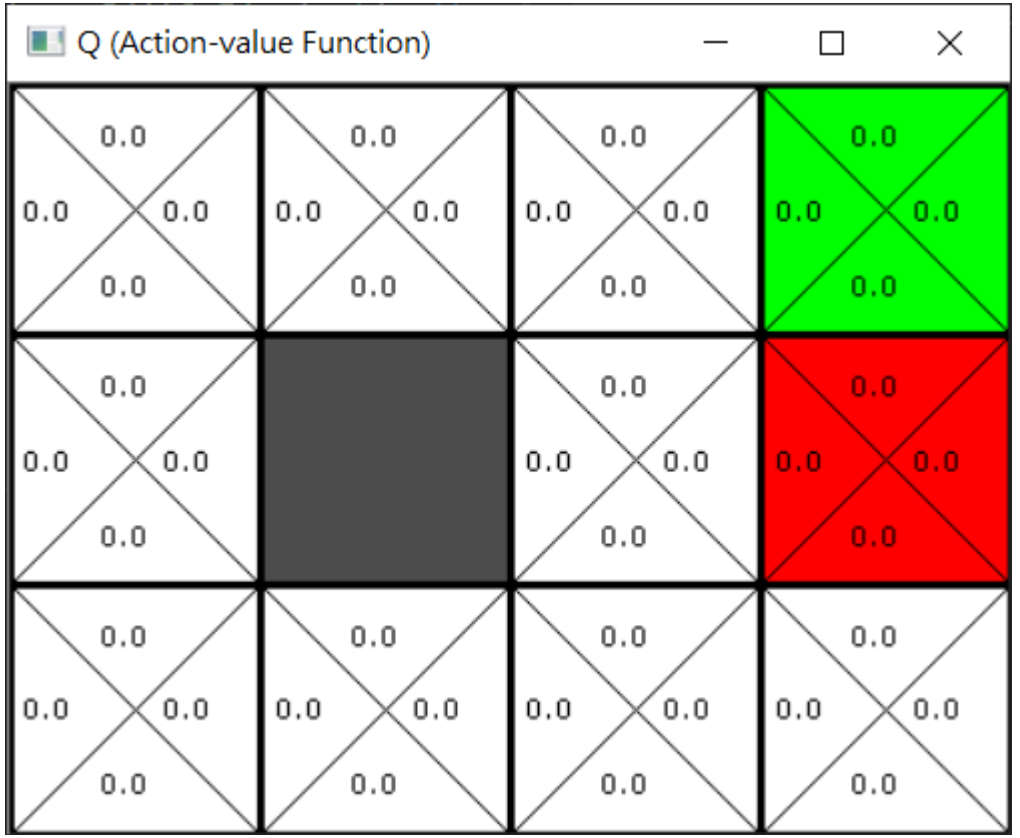
1. Initialization
   $v(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Repeat
   $\Delta \leftarrow 0$
   For each $s \in \mathcal{S}$:
   $temp \leftarrow v(s)$
   $v(s) \leftarrow \sum_{s'} p(s'|s, \pi(s)) \left[r(s, \pi(s), s') + \gamma v(s') \right]$
   $\Delta \leftarrow \max(\Delta, |temp - v(s)|)$
   until $\Delta < \theta$ (a small positive number)

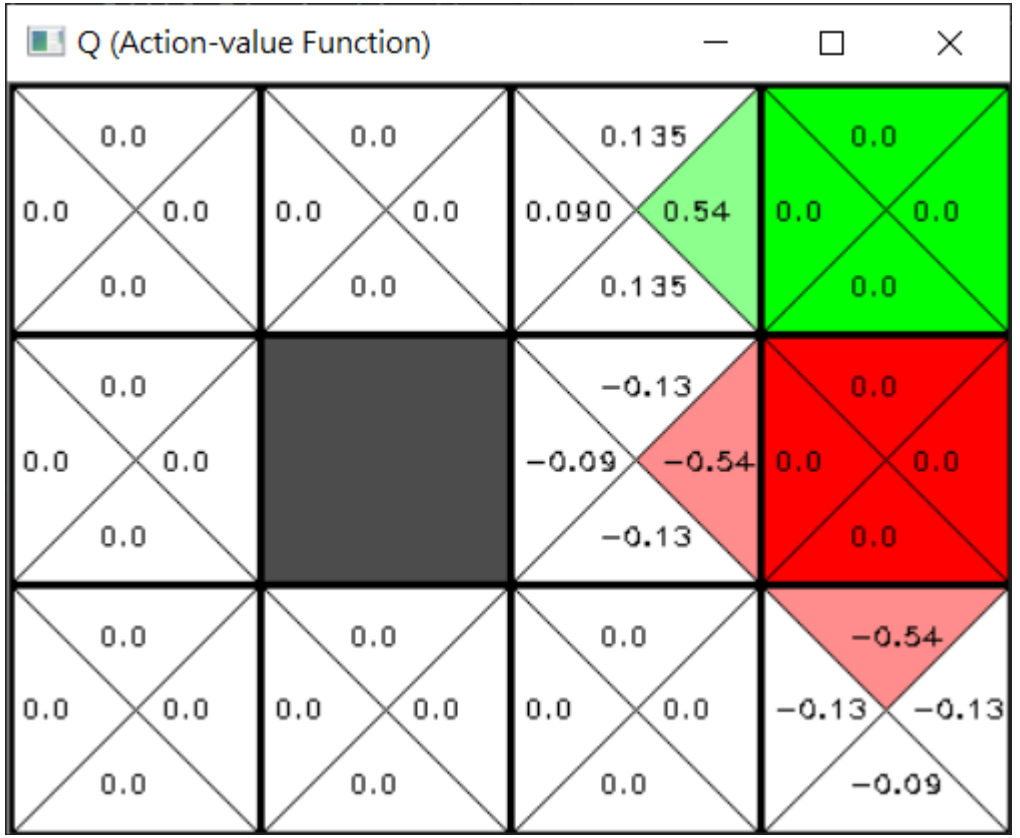
Bellman Equation:
$$V^{\pi}(s) = \sum_{a} \pi(a|s) \sum_{s'} P(s'|s,a)[r(s,a) + \gamma V^{\pi}(s')]$$

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   $temp \leftarrow \pi(s)$
   $\pi(s) \leftarrow \arg\max_a \sum_{s'} p(s'|s,a) \left[r(s,a,s') + \gamma v(s') \right]$
   If $temp \neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return v and π; else go to 2

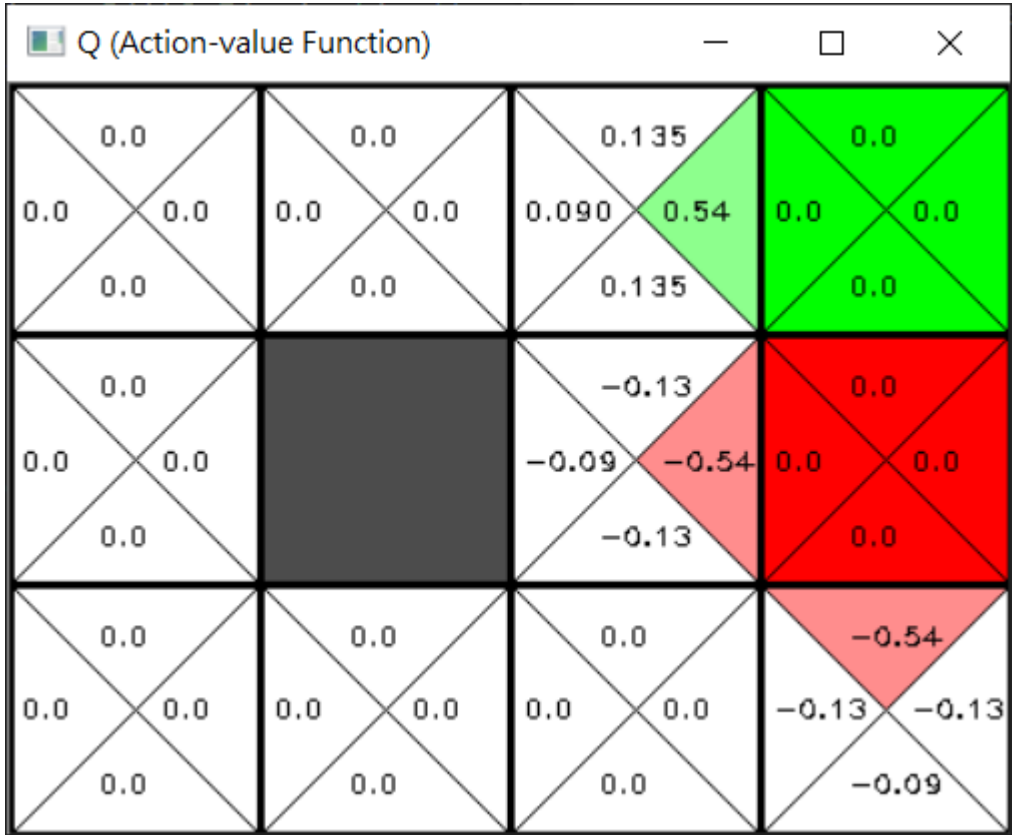
Policy Iteration Demo - Initialization



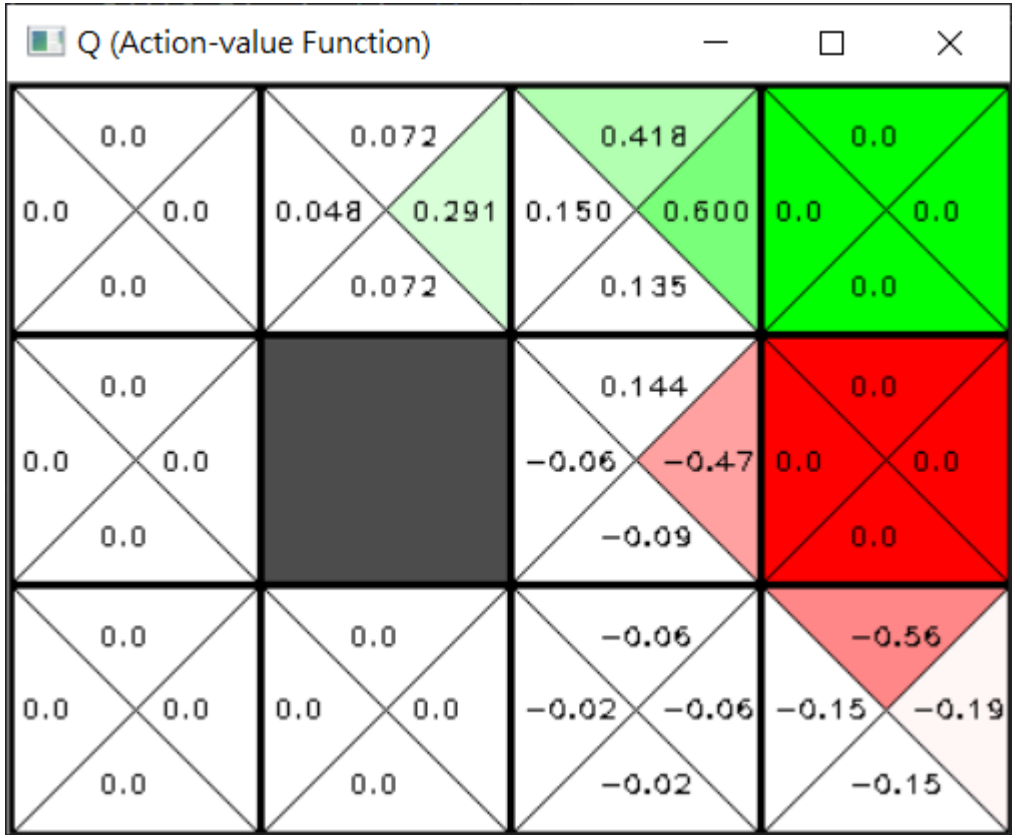
Policy Iteration Demo - Evaluation 1



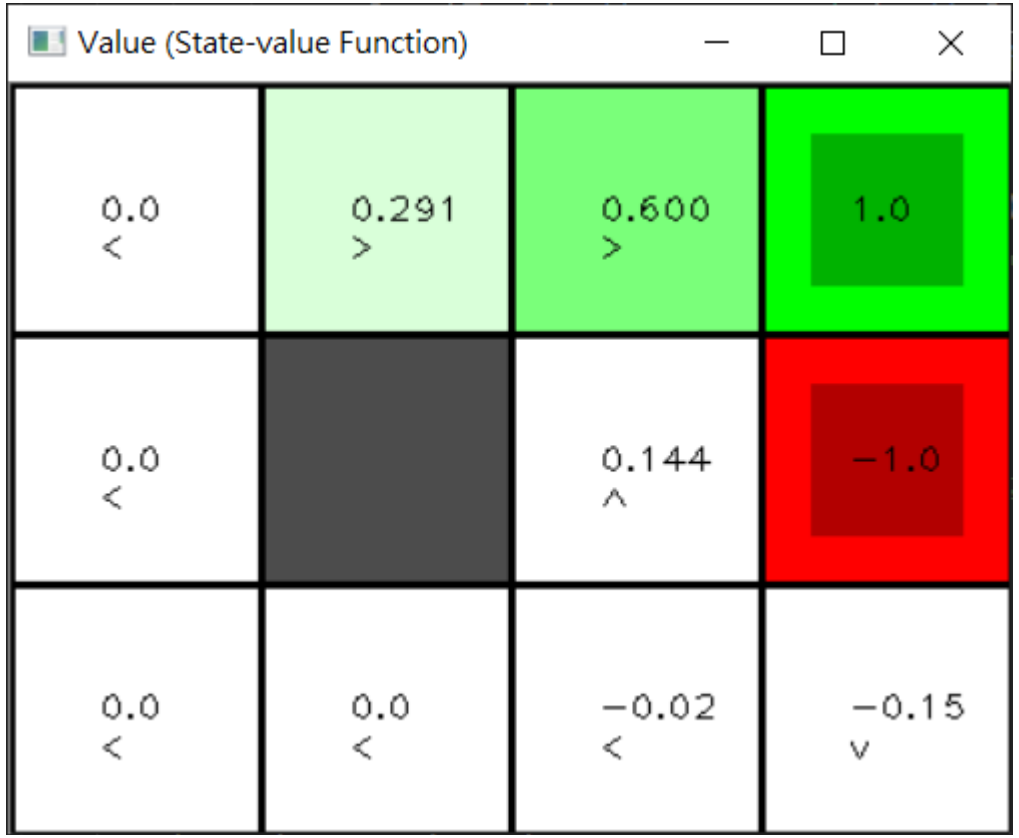
Policy Iteration Demo - Improvement 1

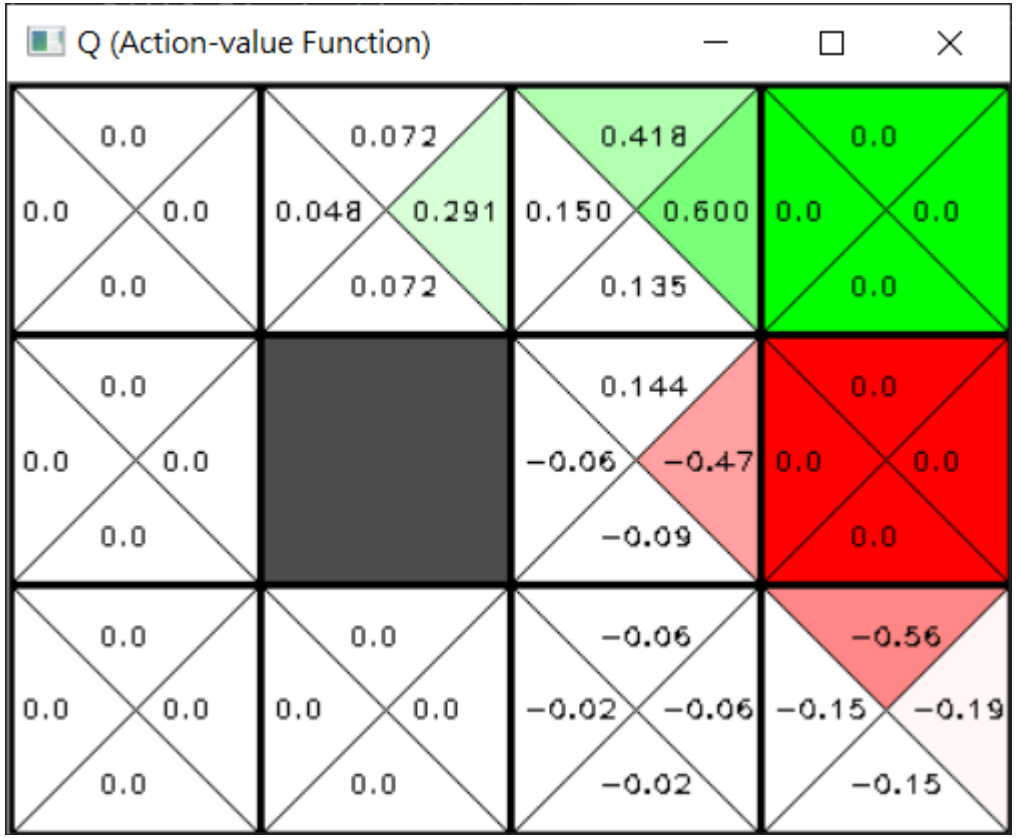


Policy Iteration Demo - Evaluation 2

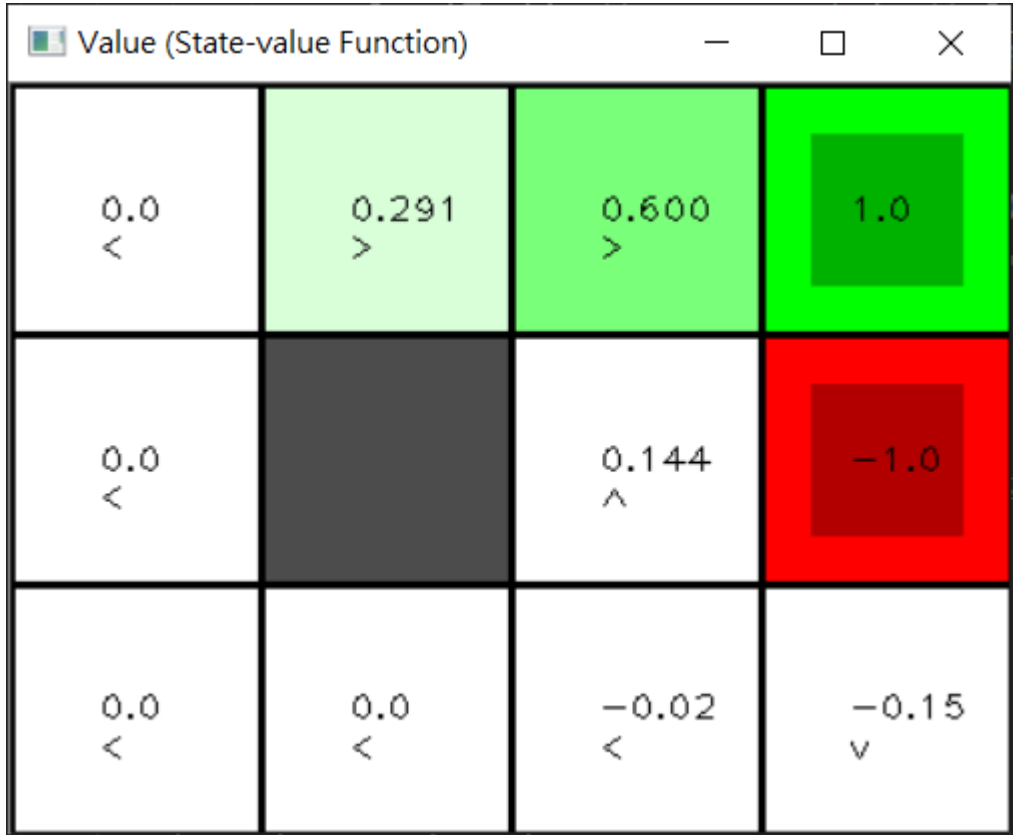


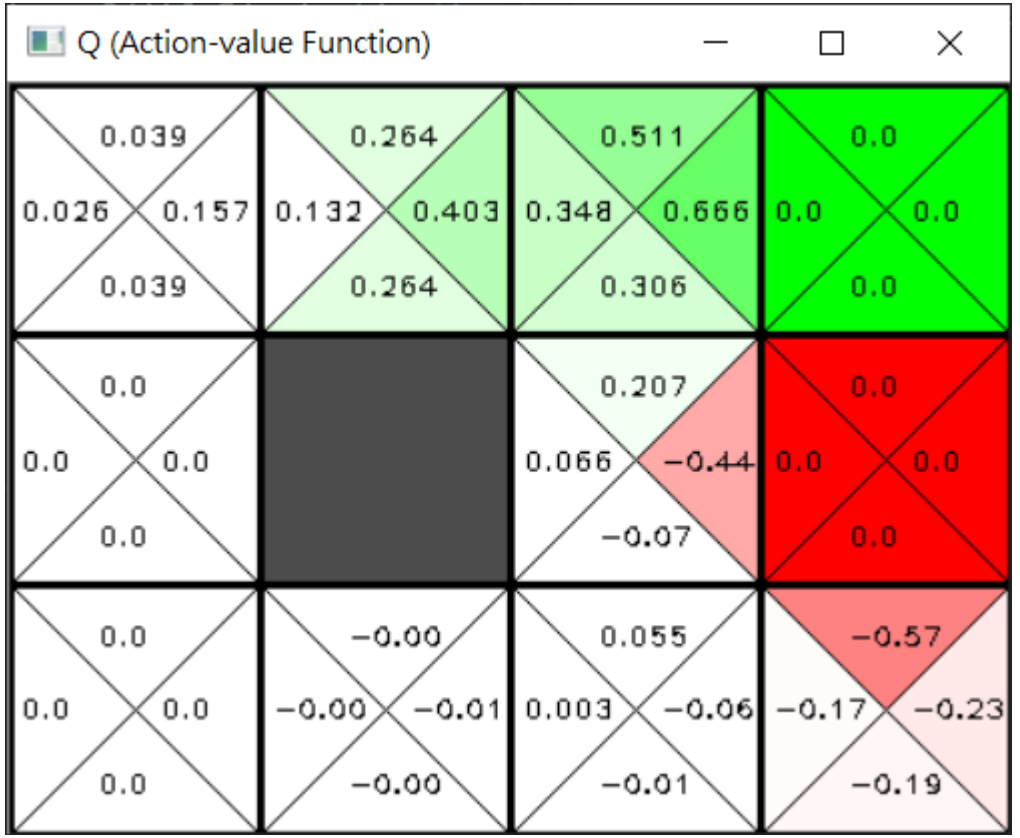
Policy Iteration Demo - Improvement 2



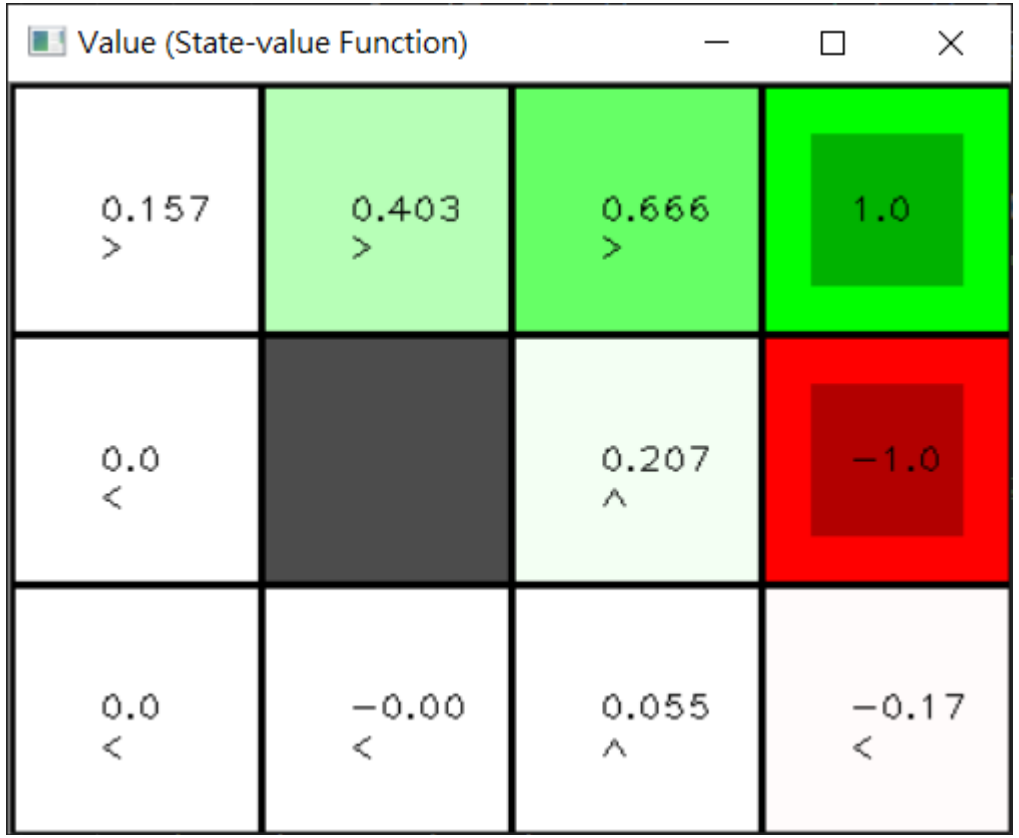


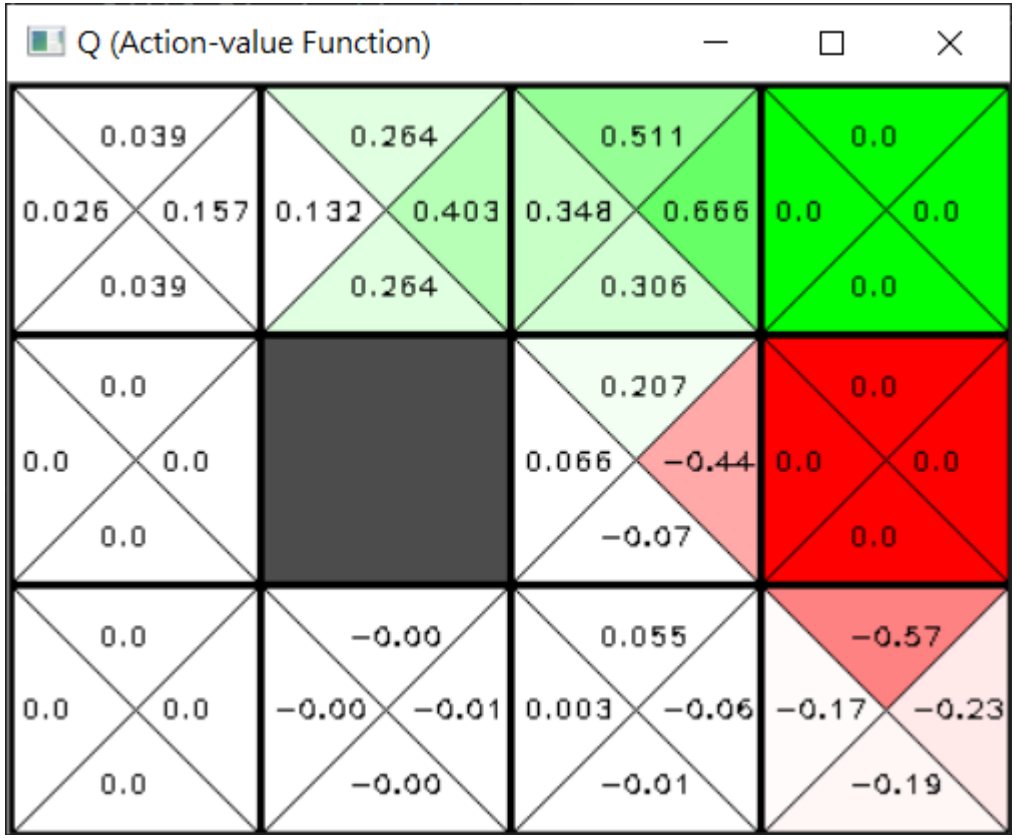
Policy Iteration Demo - Evaluation 3



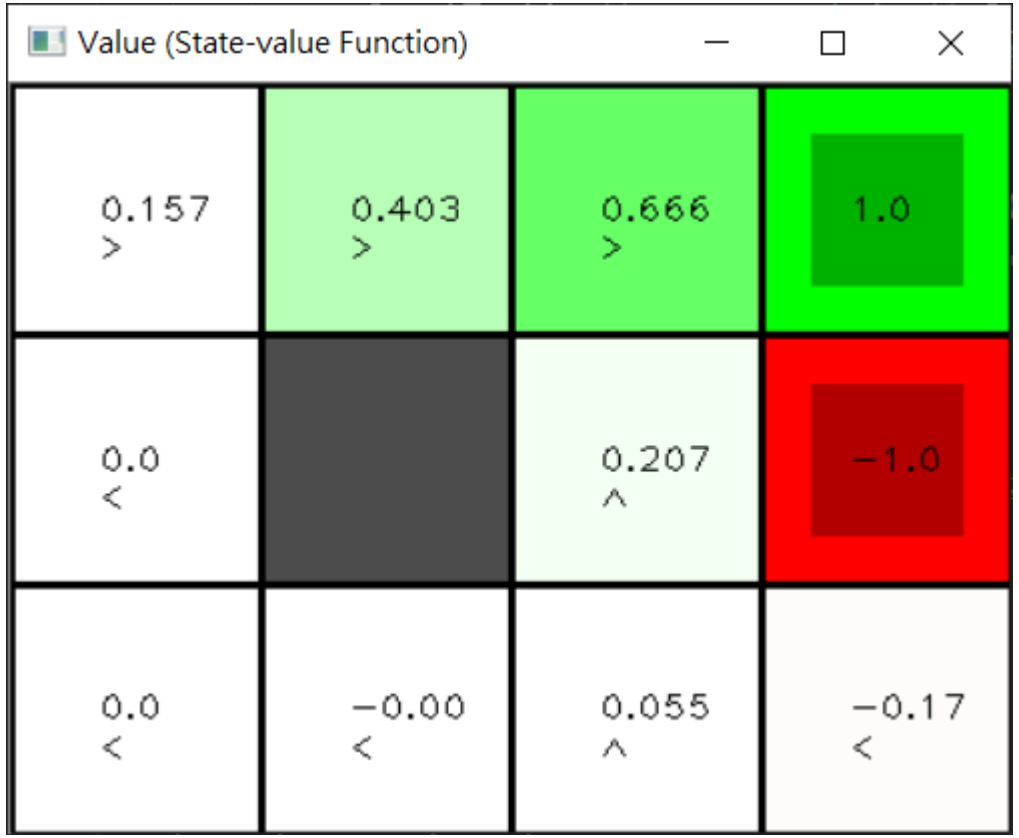


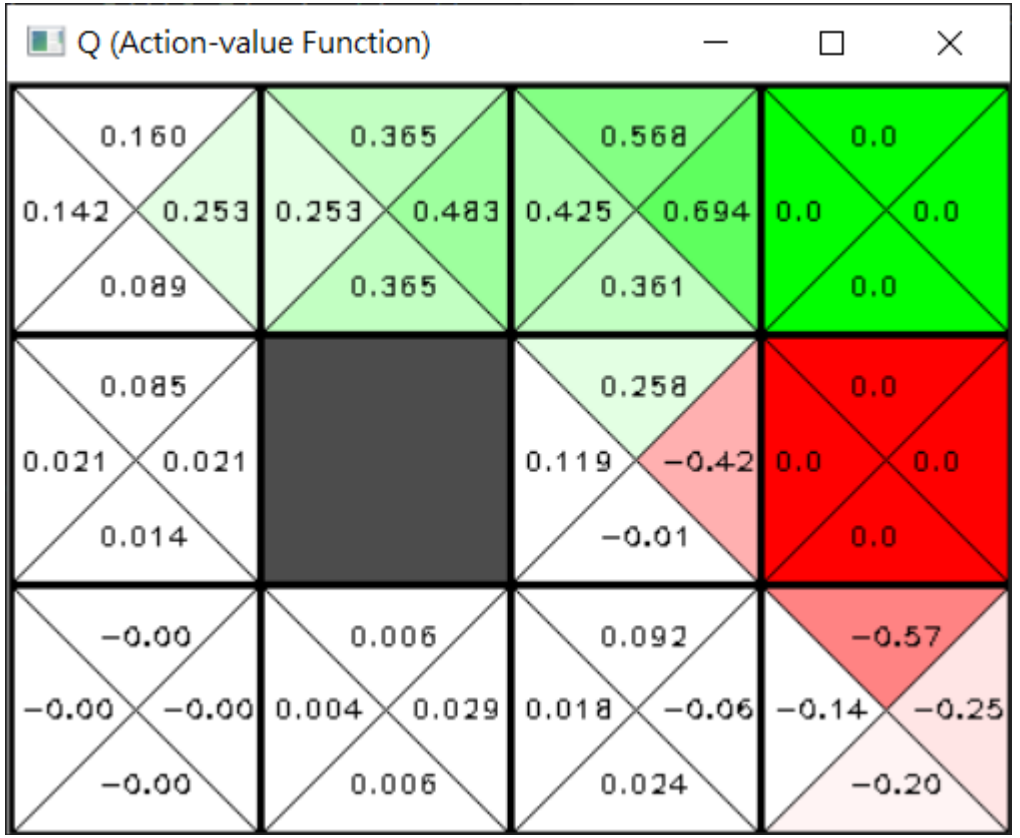
Policy Iteration Demo - Improvement 3



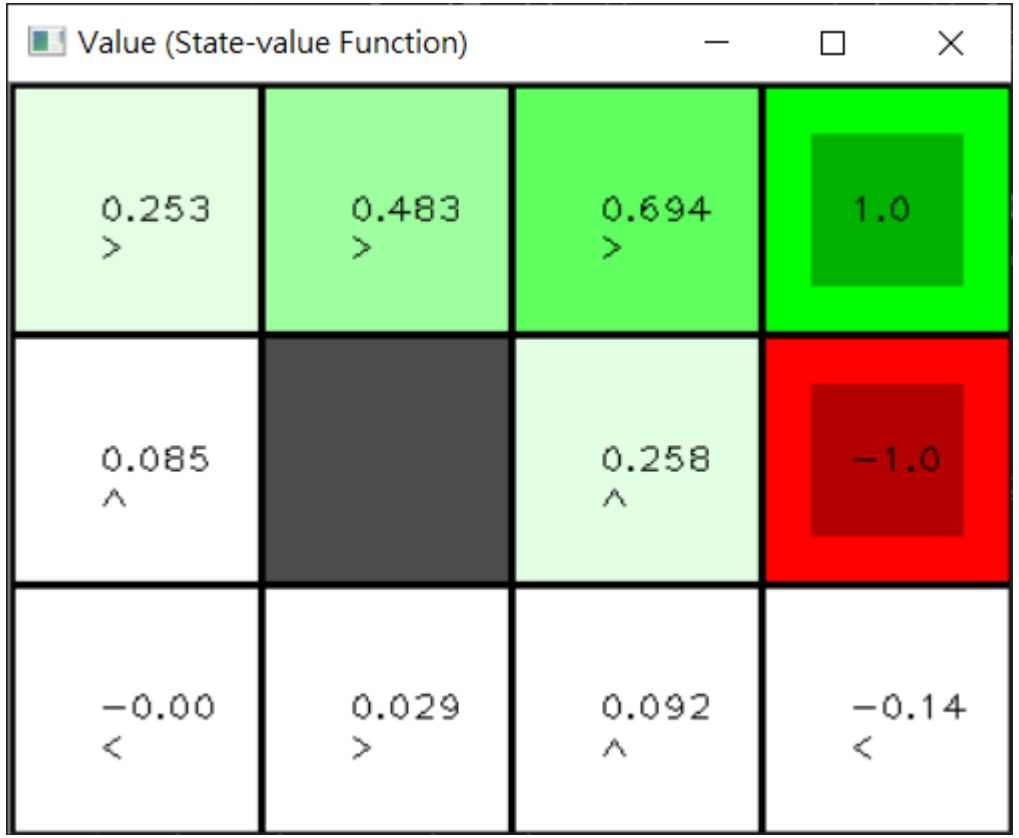


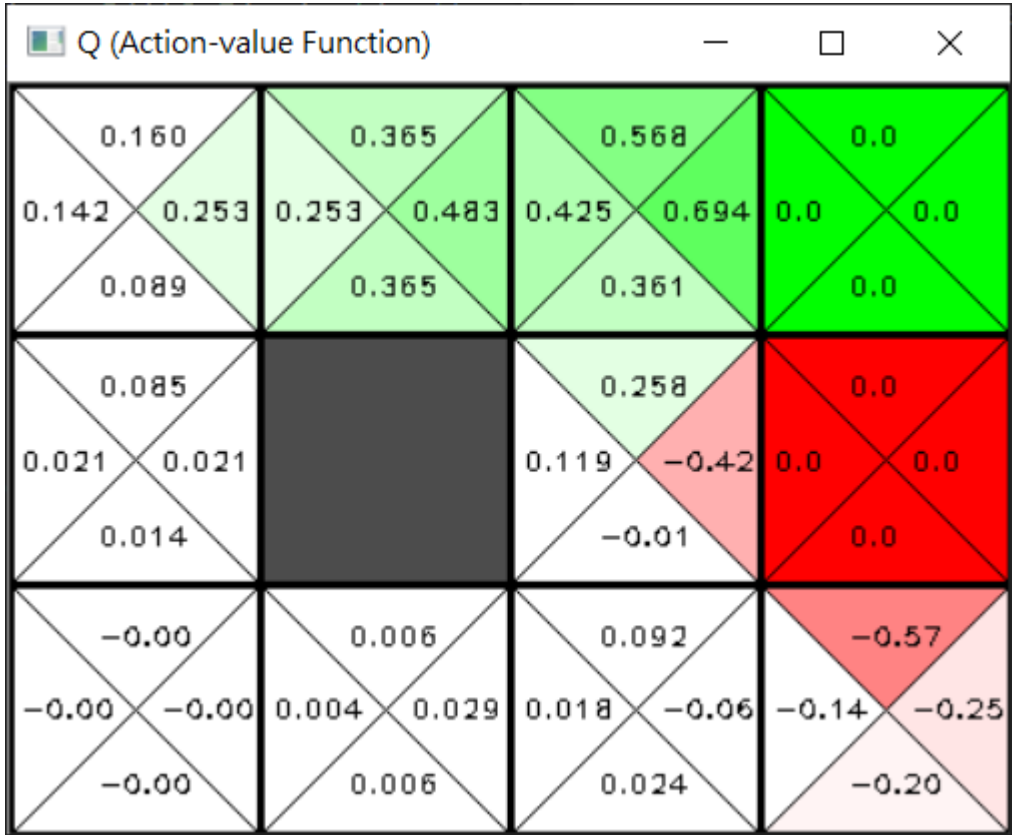
Policy Iteration Demo - Evaluation 4



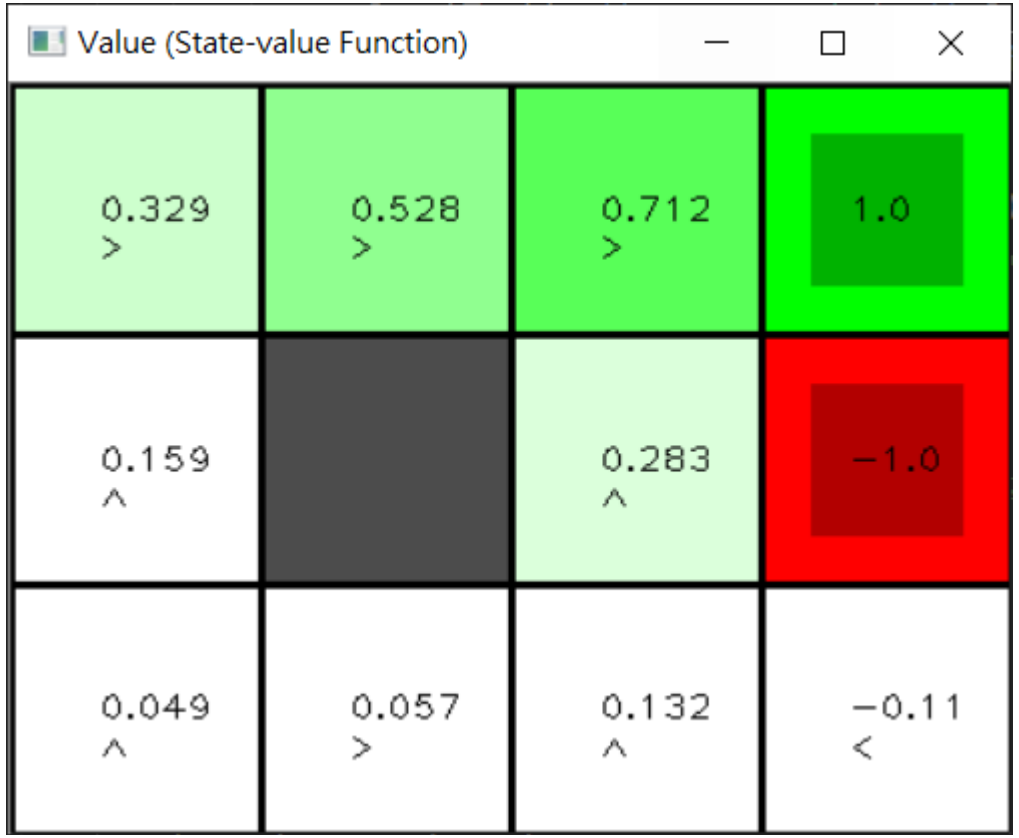


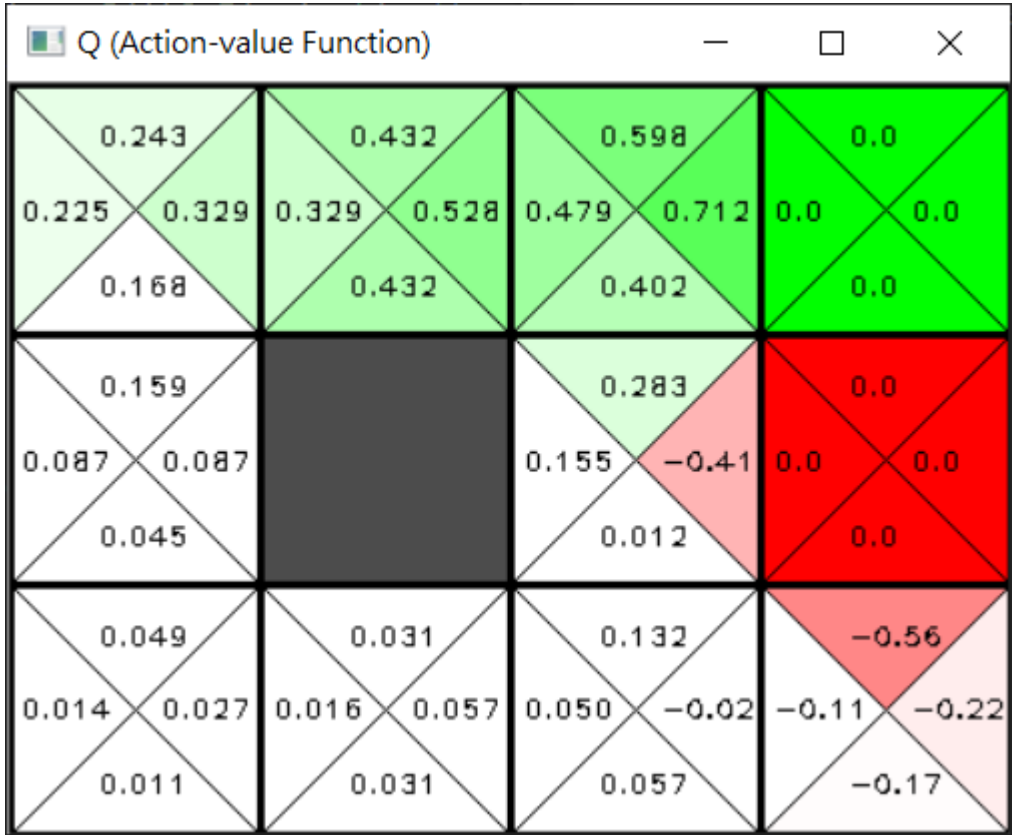
Policy Iteration Demo - Improvement 4



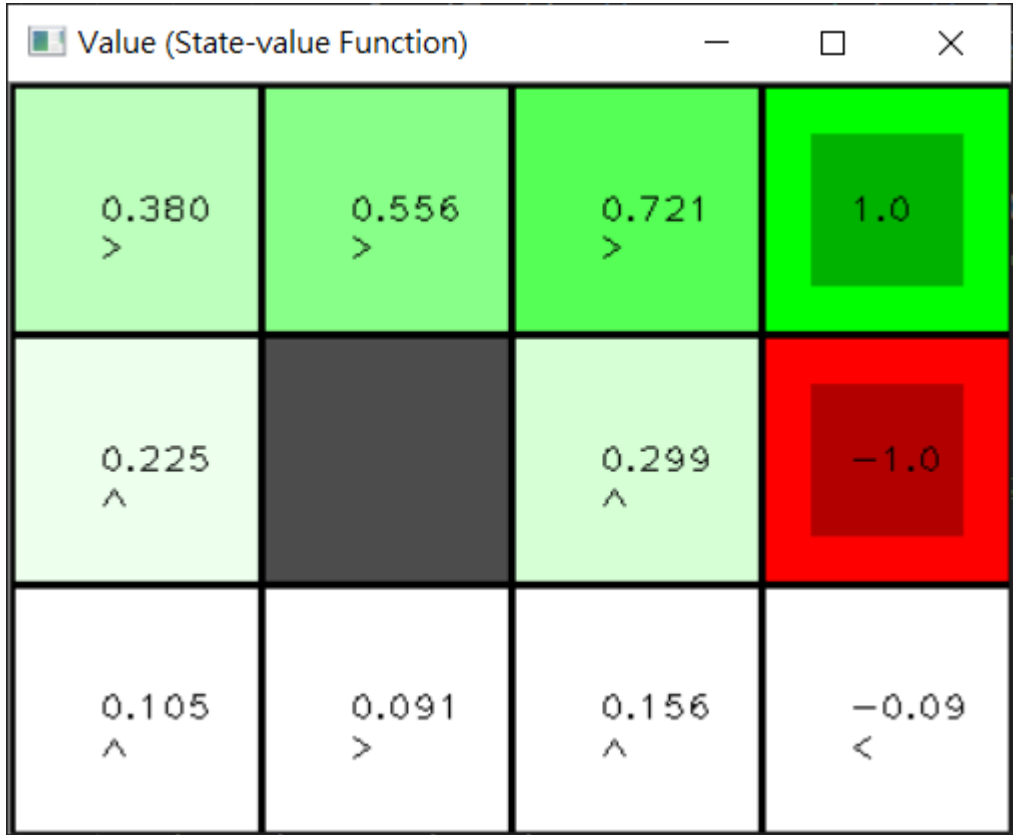


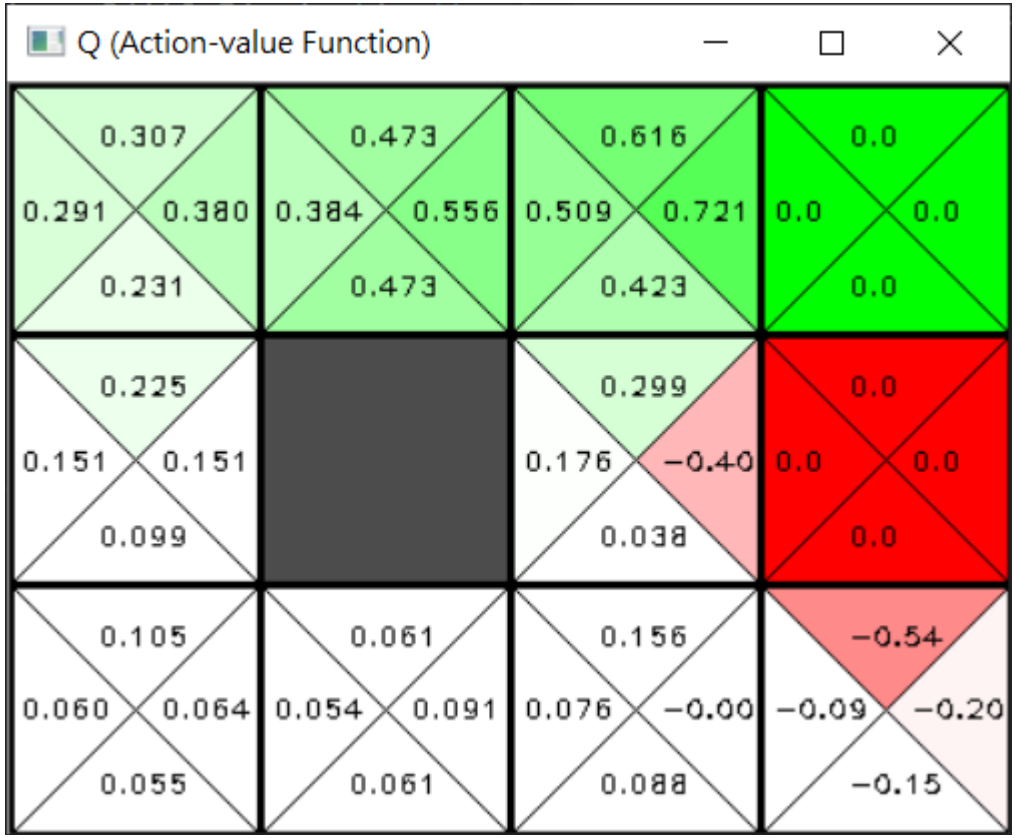
Policy Iteration Demo - Iteration 5



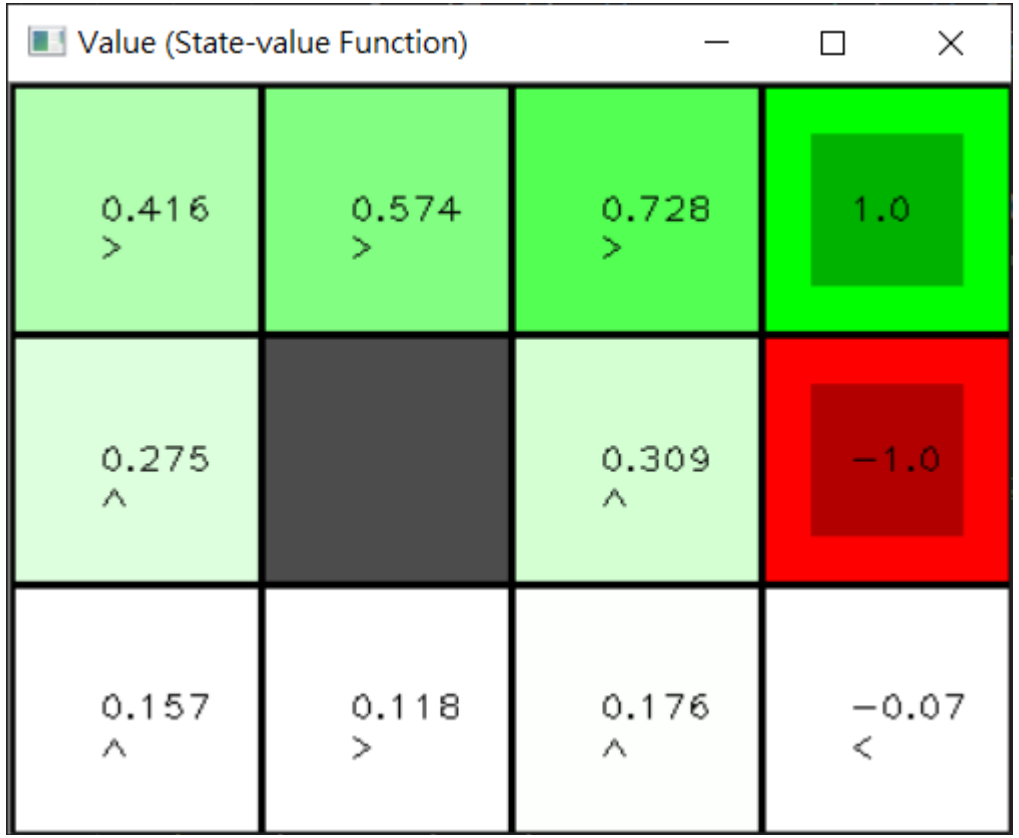


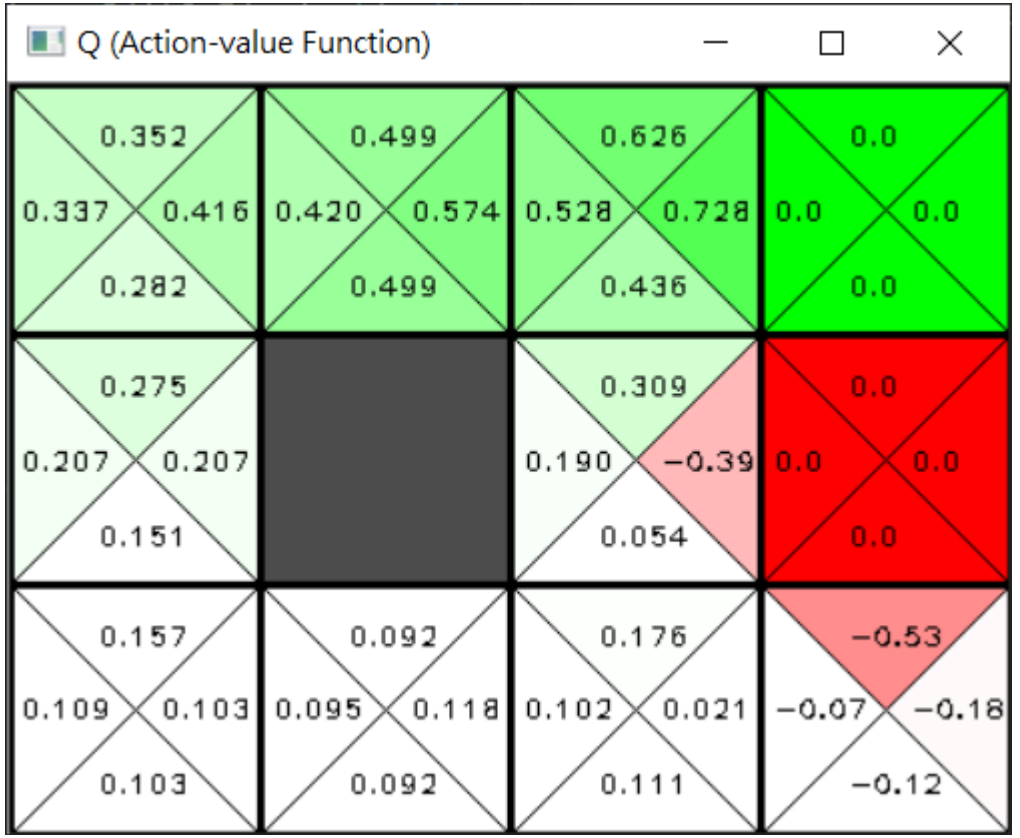
Policy Iteration Demo - Iteration 6



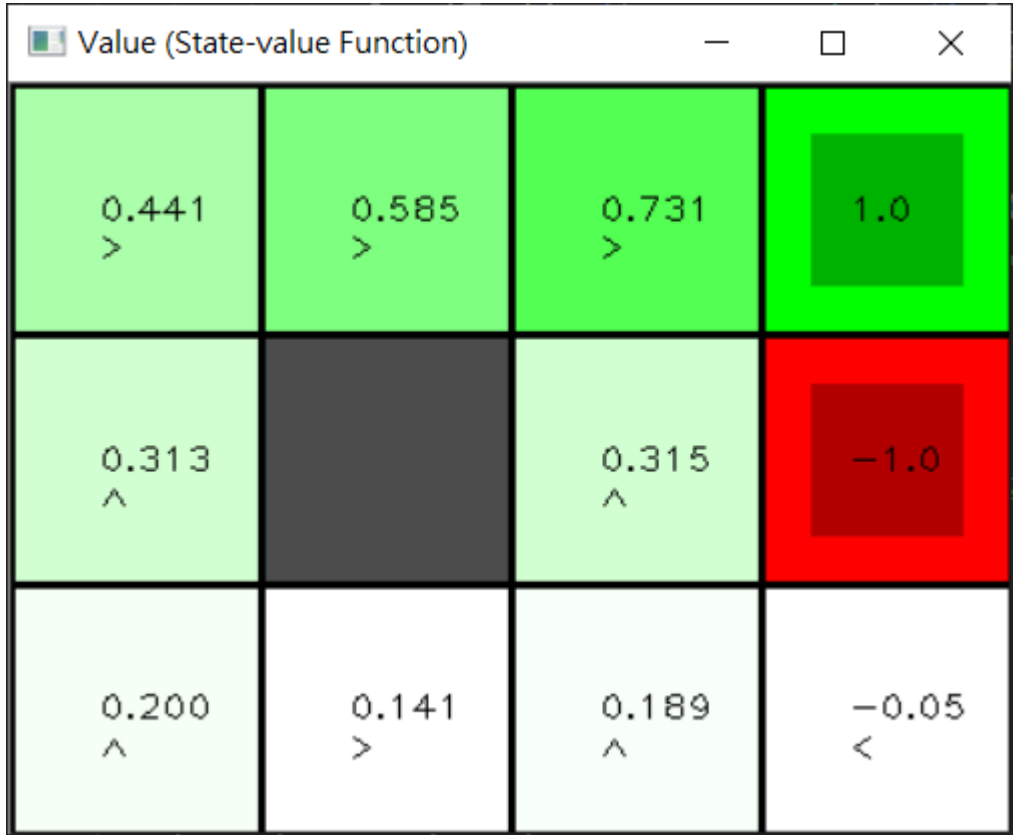


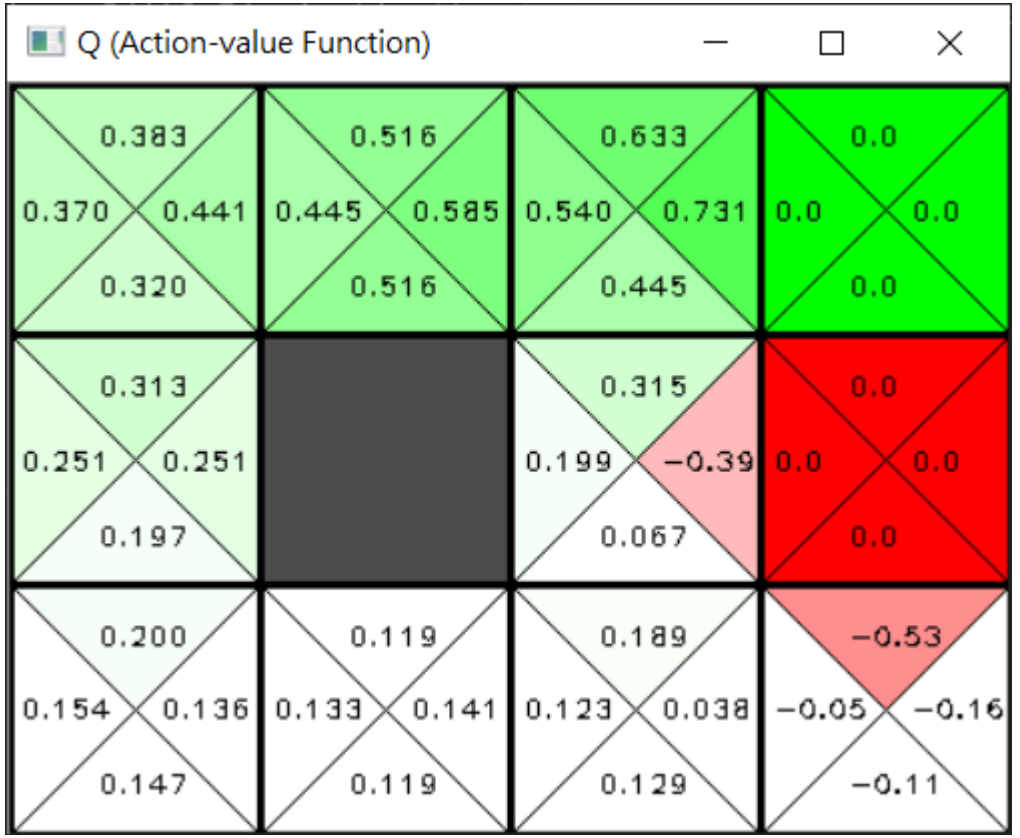
Policy Iteration Demo - Iteration 7



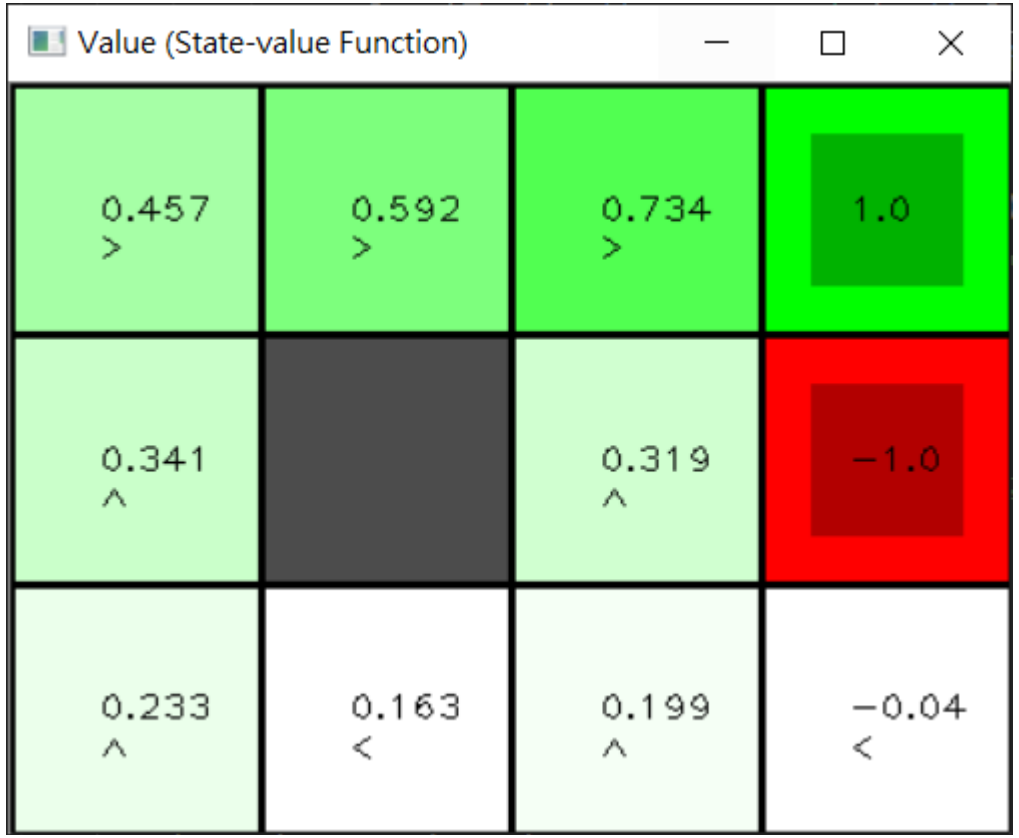


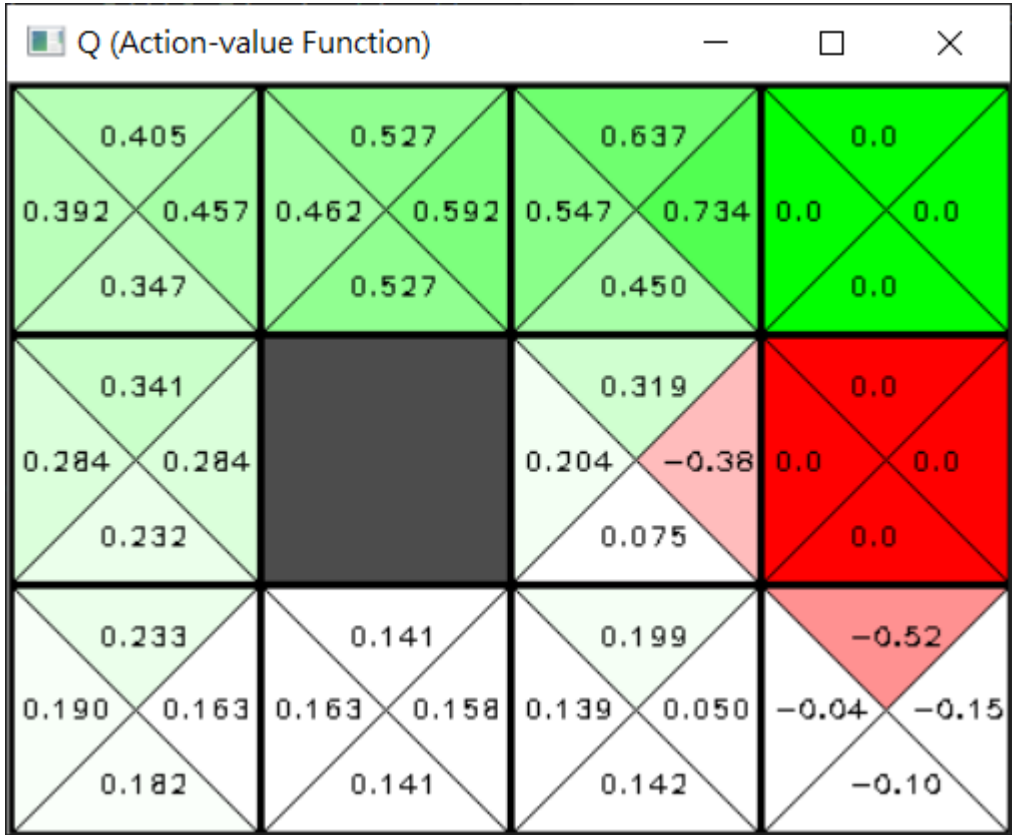
Policy Iteration Demo - Iteration 8



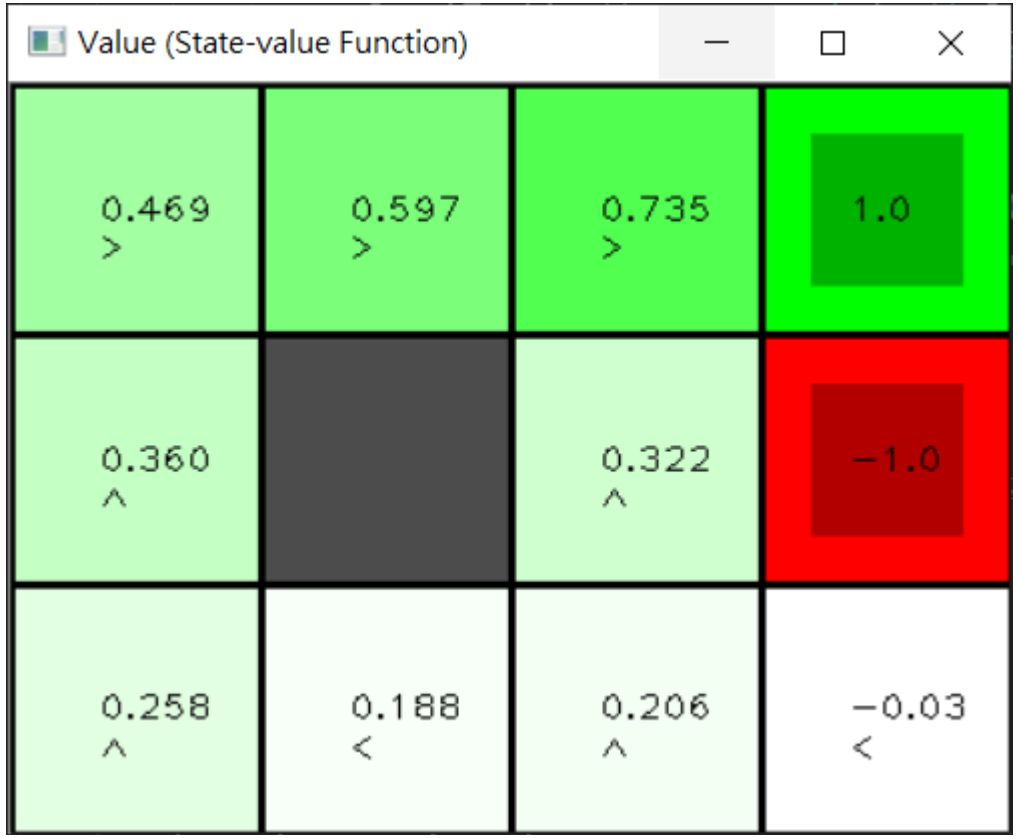


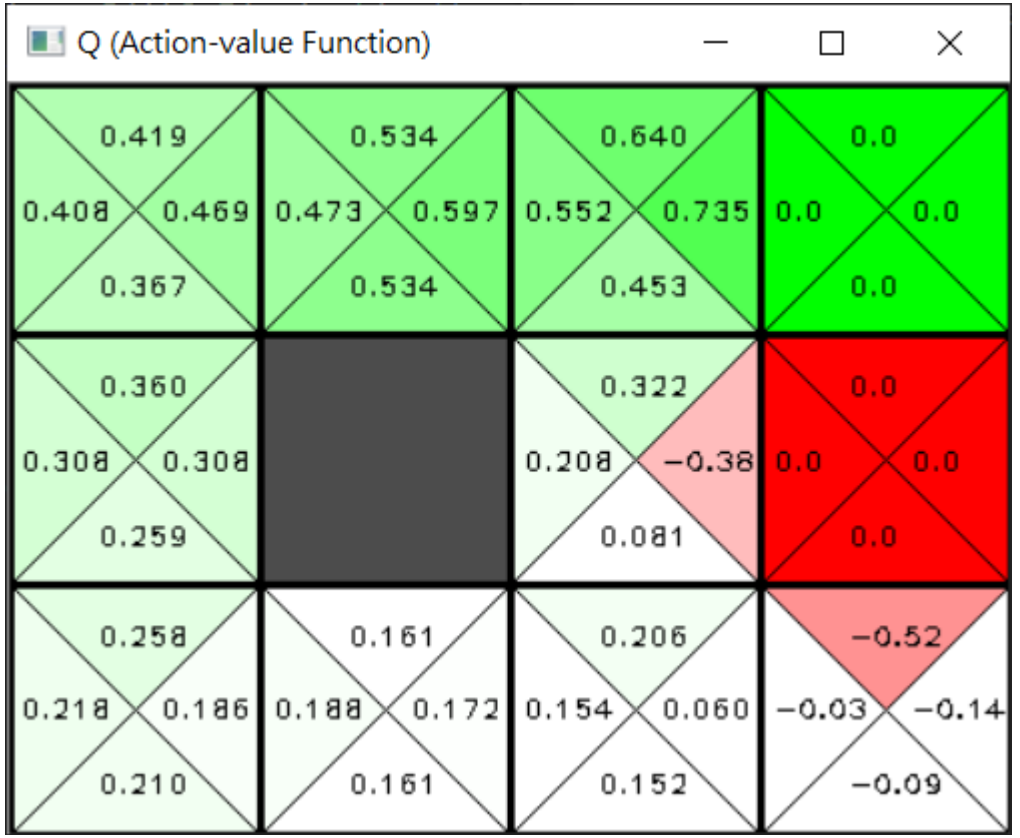
Policy Iteration Demo - Iteration 9



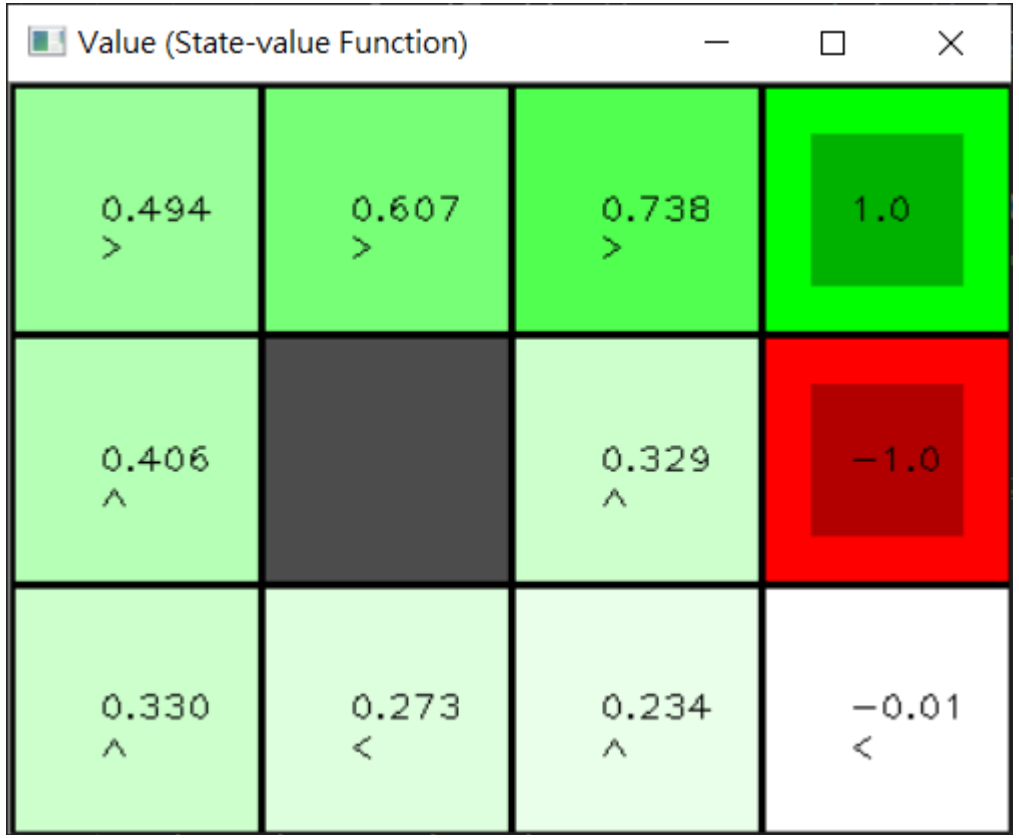


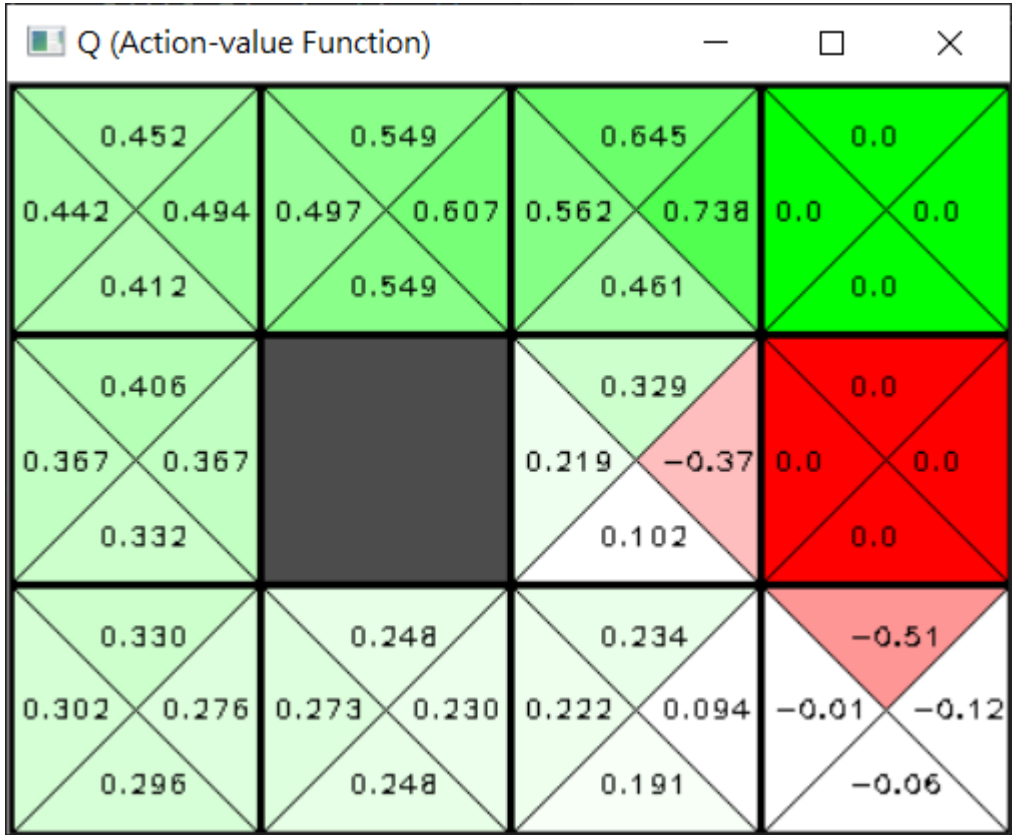
Policy Iteration Demo - Iteration 10





Policy Iteration Demo - Iteration 30





Outline

- Introduction to Reinforcement Learning
- Markov Decision Process (MDP)
  - MDP Formulation
  - Value Function and Bellman Equation
  - Dynamic Programming Methods
- Value-based Reinforcement Learning
  - From MDP to RL
  - Temporal Difference Learning (SARSA / Q-Learning)
  - Deep Q-Network (DQN) & Normalized Advantage Function (NAF)

From MDP to RL

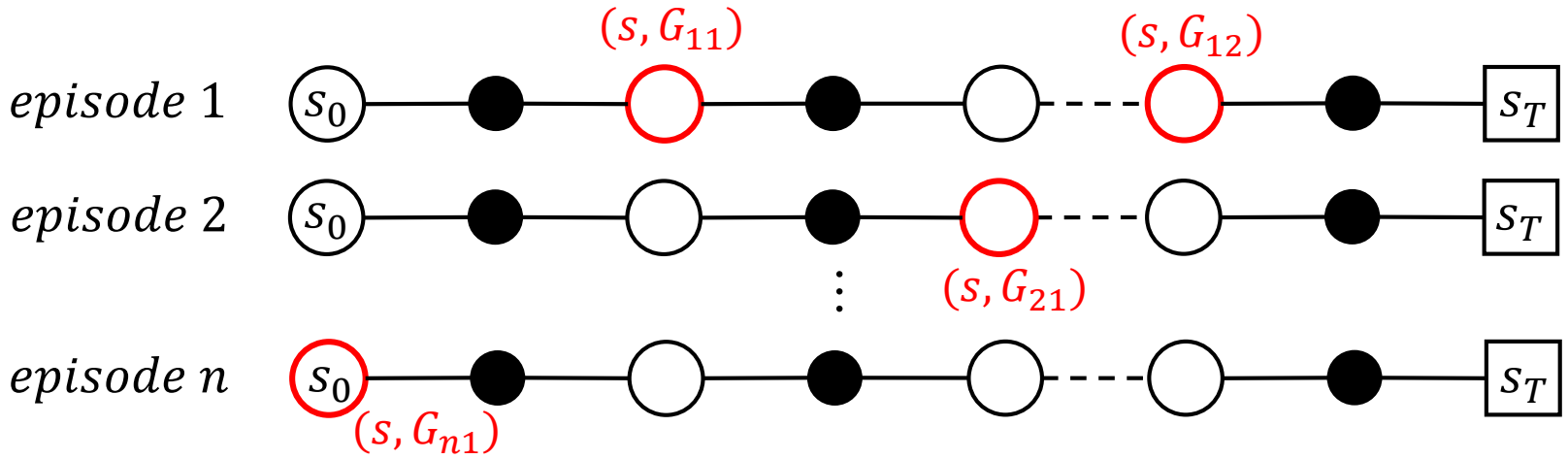
- In MDP problem, we can utilize dynamic programming and bellman backup operator to get the converge value function.
- However, the environment dynamic $P(s'|s,a)$ (transition probability) is unknown in practical, thus we cannot calculate the expectation term.

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

$$Q^{\pi}(s,a) = r(s,a) + \gamma \sum_{s'} \underset{??}{P(s'|s,a)} \sum_{a} \pi(a'|s') Q^{\pi}(s',a')$$

- In reinforcement learning, we utilize sampling method to estimate the value function $V(s), Q(s,a)$.
  - Monte-Carlo Method
  - Temporal-Difference Method

Monte-Carlo Method (MC)



$G_t = r_t + \gamma r_{t+1} + \cdots + \gamma^T r_T \quad (Return)$

$V^\pi(s_t) = \mathbb{E}_\pi[G_t]$

$N(s_t) \leftarrow N(s_t) + 1$ *(visit times)*

$V(s_t) \leftarrow V(s_t) + \frac{1}{N(s_t)}(G_t - V(s_t))$

$N(s_t, a_t) \leftarrow N(s_t, a_t) + 1$ *(visit times)*

$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \frac{1}{N(s_t, a_t)}(G_t - Q(s_t, a_t))$

$$\mu_k = \frac{1}{k}\sum_{i=1}^{k} x_i = \frac{1}{k}\left(x_k + \sum_{i=1}^{k-1} x_i\right)$$
$$= \frac{1}{k}(x_k + (k-1)\mu_{k-1})$$
$$= \mu_{k-1} + \frac{1}{k}(x_k - \mu_{k-1})$$

Temporal-Difference Method (TD)

- Learns from incomplete episodes, by bootstrapping.

Monte-Carlo Estimation

$$V(s_t) \leftarrow V(s_t) + \alpha(G_t - V(s_t))$$

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(G_t - Q(s_t, a_t))$$

Temporal-Difference

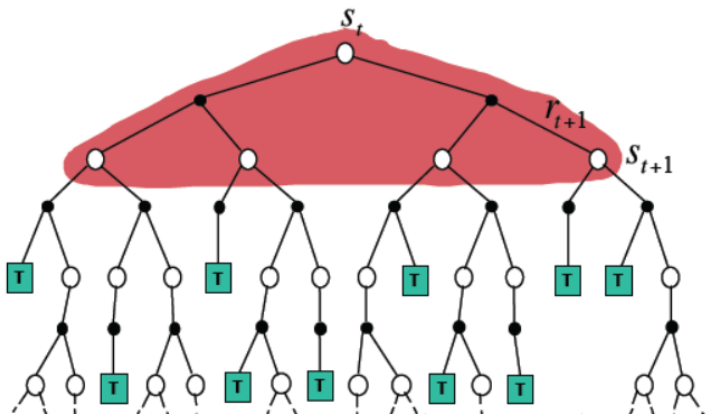
$$V(s_t) \leftarrow V(s_t) + \alpha(\underbrace{r_t + \gamma V(s_{t+1})}_{\text{TD-Target}} - V(s_t))$$
TD-Error

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(\underbrace{r_t + \gamma Q(s_{t+1}, a_{t+1})}_{\text{TD-Target}} - Q(s_t, a_t))$$
TD-Error

Comparison of DP/MC/TD

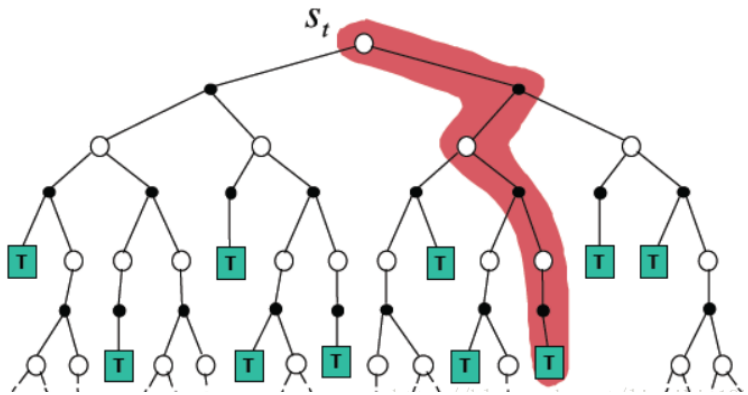
Dynamic Programming

$$V(S_t) \leftarrow \mathbb{E}_\pi \left[R_{t+1} + \gamma V(S_{t+1}) \right]$$



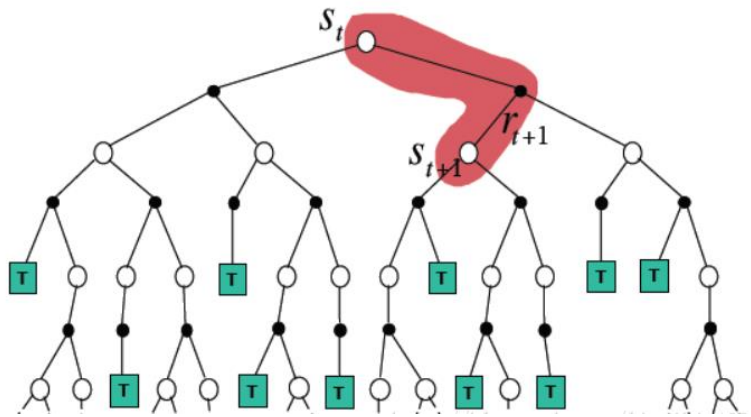
Monte-Carlo

$$V(S_t) \leftarrow V(S_t) + \alpha \left(G_t - V(S_t) \right)$$



Temporal-Difference

$$V(S_t) \leftarrow V(S_t) + \alpha \left(R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \right)$$



Exploration and Exploitation

- By using Monte-Carlo or temporal-difference method, we can estimate the value functions. However, different from the common estimation problem, the decision we make will affect the data we sample.

- The estimation of the value function is inaccurate at first. If we always use the estimated value function to select the action, we might trap in the local maxima and lose the chance to find the better policy.

- Thus, we need to select different action to explore the environment and utilize the learned knowledge to exploit the good policy.

How do we decide when to explore and when to exploit ?

Multi-armed Bandit Problem

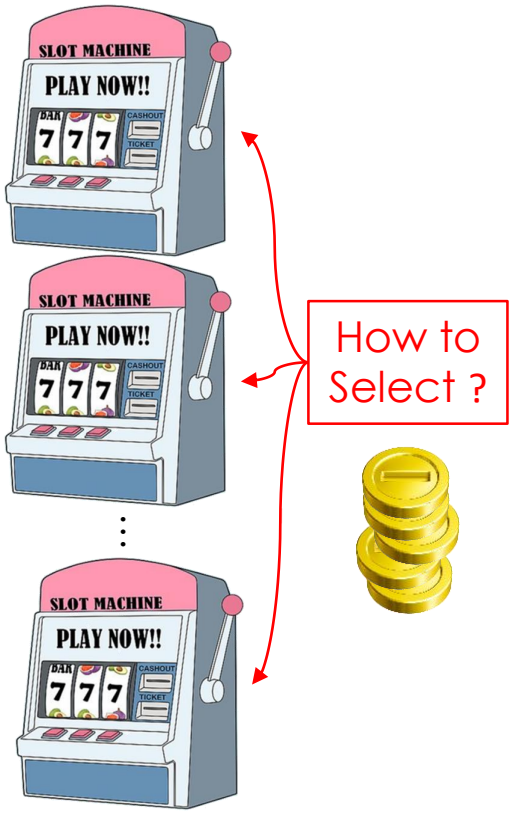
- The multi-armed bandit problem is a classic problem that well demonstrates the exploration-exploitation dilemma.

- Here we have a one-armed bandit machine. Once we pull the arms will receive a reward. We can estimate the expected reward after several tries.

One-armed Bandit

Multi-armed Bandit Problem

- The multi-armed bandit problem is a classic problem that well demonstrates the exploration-exploitation dilemma.
- Here we have a one-armed bandit machine. Once we pull the arms will receive a reward. We can estimate the expected reward after several tries.
- Then, suppose we have multiple one-armed bandit machines, each time we can choose one to pull the arms. Now the problem is, how can we get the most total reward after limit attempts ?



Multi-armed Bandit

ε – first

- Exploration:
  - We do not know the expectation of each bandit at first, so we randomly select the actions at the first **ε** percent attempts.
- Exploitation:
  - After several attempts, we can calculate the expected reward. In the remaining **(1-ε)** percent attempts, we can always select the machine with maximum expected reward.

ε – greedy

- Is the first ε percent attempts enough for estimate the expectation correctly ? Can we use each attempt more efficiently to earn rewards ?
- ε – greedy allow us to simultaneously learn to estimate and earn reward.
- In each round of selection, there will be a probability of **ε** to randomly select a slot machine (exploration), and a probability of **(1-ε)** will select the machine with the largest average return in the past (exploitation).

Outline

- Introduction to Reinforcement Learning
- Markov Decision Process (MDP)
  - MDP Formulation
  - Value Function and Bellman Equation
  - Dynamic Programming Methods
- Value-based Reinforcement Learning
  - From MDP to RL
  - Temporal Difference Learning (Q-Learning / SARSA)
  - Deep Q-Network (DQN) & Normalized Advantage Function (NAF)

Off-Policy vs. On-Policy

- On-Policy:
  - The agent learned and the agent interacting with environment is the same.
- Off-Policy:
  - The agent learned and the agent interacting with environment is different.

On-Policy: 阿光下棋

Off-Policy: 佐為下棋，阿光在旁邊看

By Hung-yi Lee

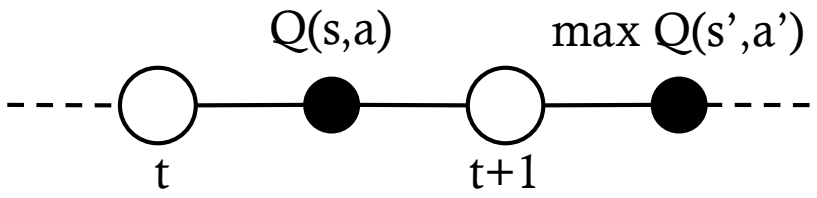
Q-Learning

Bellman Optimality Equation:

$$Q^*(s,a) = \sum_{s'} P(s'|s,a)\left[r(s,a) + \gamma \max_{a'} Q^*(s',a')\right]$$

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    Choose A from S using policy derived from Q (e.g., ϵ-greedy)
    Take action A, observe R, S'
    $Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a Q(S',a) - Q(S,A)\right]$
    $S \leftarrow S'$
  until S is terminal

Off-Policy Method



Q-Learning Demo

SARSA

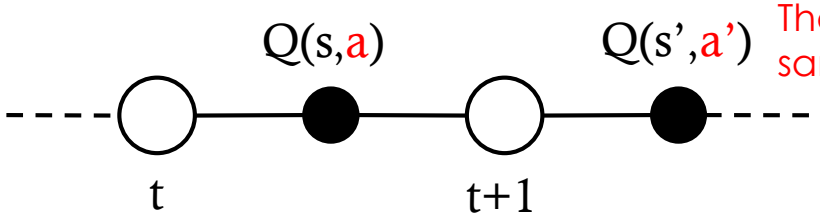
Bellman Expectation Equation:

$$Q^{\pi}(s,a) = r(s,a) + \gamma \sum_{s'} P(s'|s,a) \sum_{a'} \pi(a'|s') Q^{\pi}(s',a')$$

- State-Action-Reward-State-Action (SARSA)

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Choose A from S using policy derived from Q (e.g., ϵ-greedy) — Policy Improvement
  Repeat (for each step of episode):
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ϵ-greedy)
    $Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma Q(S',A') - Q(S,A)\right]$ — Policy Evaluation
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

On-Policy Method



The action have to sample from same policy.

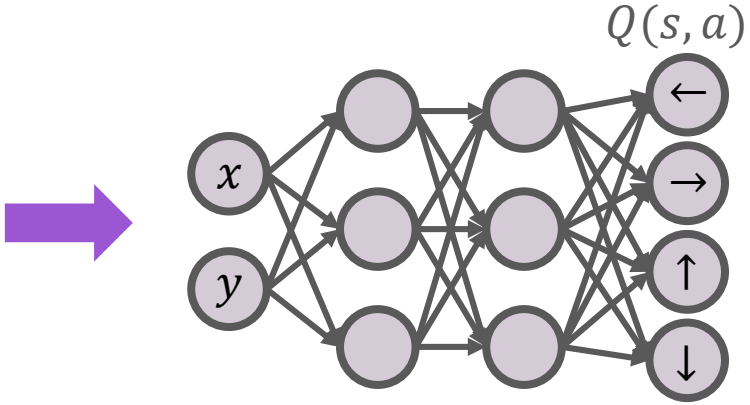
Outline

- Introduction to Reinforcement Learning
- Markov Decision Process (MDP)
  - MDP Formulation
  - Value Function and Bellman Equation
  - Dynamic Programming Methods
- Value-based Reinforcement Learning
  - From MDP to RL
  - Temporal Difference Learning (SARSA / Q-Learning)
  - Deep Q-Network (DQN) & Normalized Advantage Function (NAF)

Deep Q-Network

- Large State Dimension Problem
  - Utilize function approximator to replace the Q table

Q(s,a)	←	→	↑	↓
(0,0)				
(0,1)				
(1,0)				
(1,1)				



  - Minimize the objective of td-error

Mean-Square Error TD-Learning :

$$L(\theta) = \mathbb{E}_{\{s,a,r,s\prime\}}[\left(r + \gamma \max_{a'} Q\left(s', a'; \theta^{old}\right) - Q(s, a; \theta)\right)^2]$$

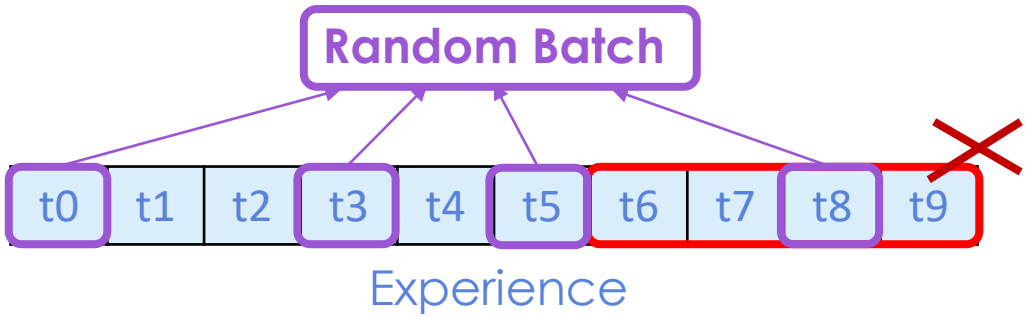
Deep Q-Network

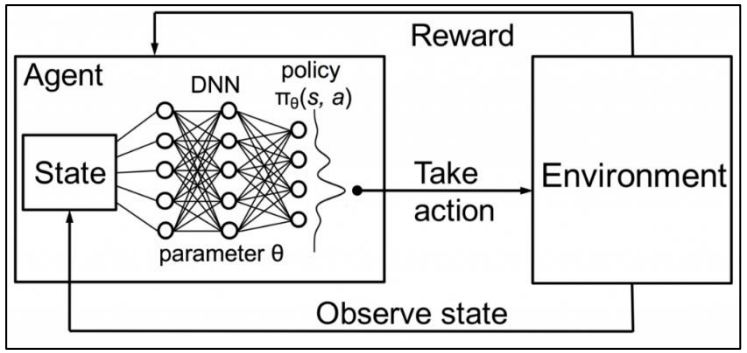
- Some design details:
  - **Two Q Networks (Evaluation/Target)** :

    Stable the optimization process.
  - **Clip Reward** :

    Stable the optimization process.
  - **Experience Replay**:

    Break the time relation of sample.





Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$

For episode = 1, M **do**

  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

  For t = 1,T **do**

    With probability ε select a random action a_t

    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$

    Execute action a_t in emulator and observe reward r_t and image x_{t+1}

    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D

    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D

    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$

    Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ

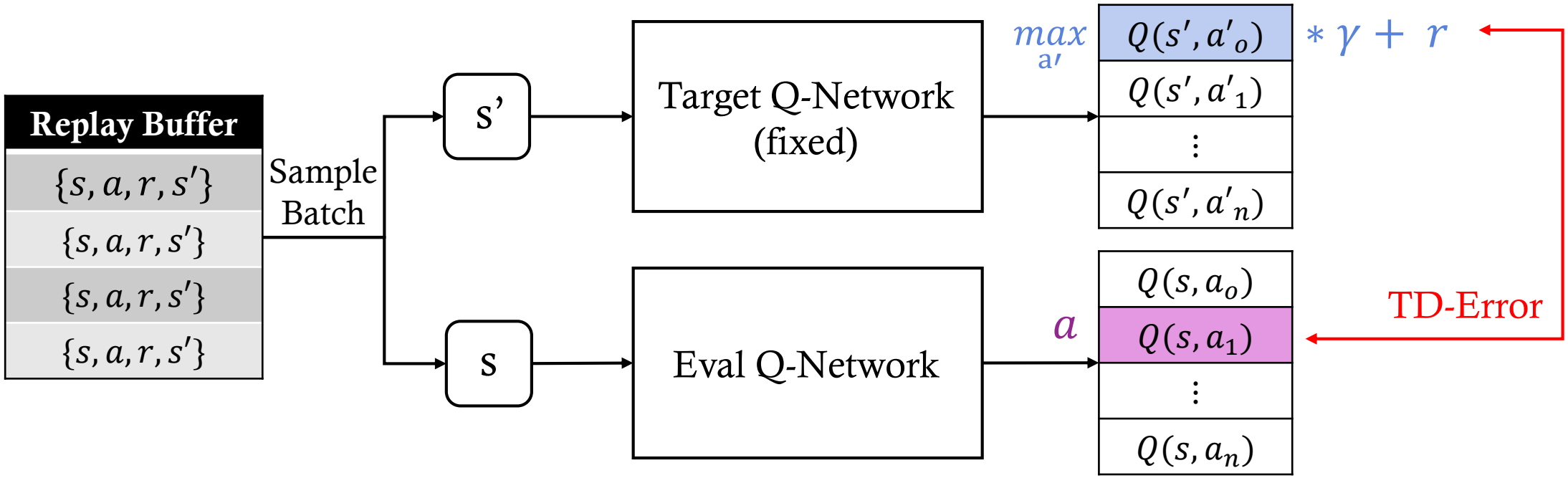
    Every C steps reset $\hat{Q} = Q$

  End For

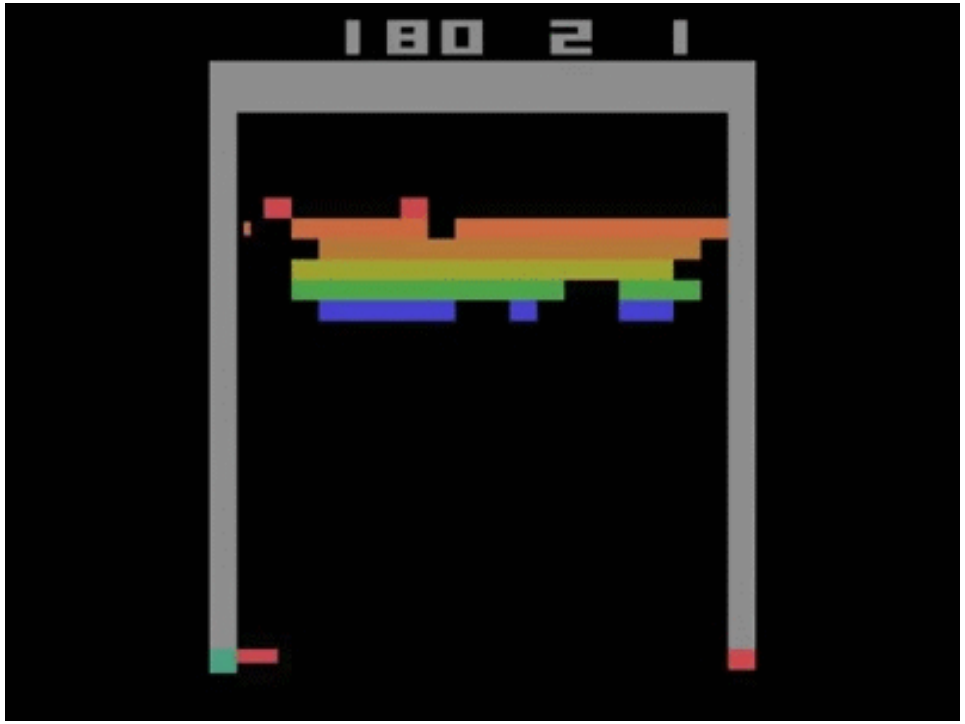
End For

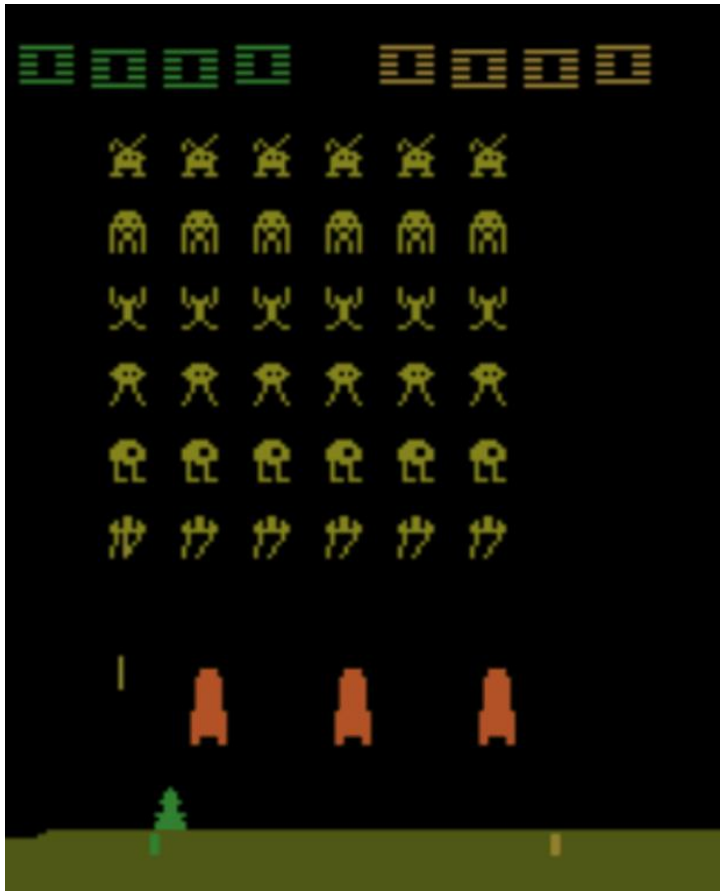
Deep Q-Network

$$L(\theta) = \mathbb{E}_{\{s,a,r,s'\}}\left[\left(r + \gamma \max_{a'} Q_{target}(s', a'; \theta^{old}) - Q_{eval}(s, a; \theta)\right)^2\right]$$



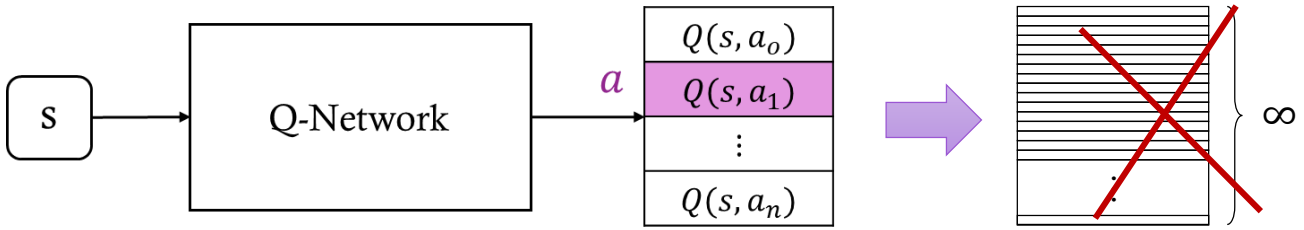
Deep Q-Network





Continuous Deep Q-Learning

- What if the action space is continuous ?



- The network output $V(s), \mu(s), L(s)$, define Normalized Advantage Function $A(s, a)$.

Single Variable Situation:

$$Q(s,a) = -\frac{1}{2}p(a-\mu)^2 + V(s)$$

$$Q(s,a) = A(s,a|\theta^A) + V(s|\theta^V)$$

$$A(s,a|\theta^A) = -\frac{1}{2}(a-\mu(s|\theta^\mu))^T P(s|\theta^P)(a-\mu(s|\theta^\mu))$$

Make sure the maximum Q given the action $a = \mu(s|\theta^\mu)$

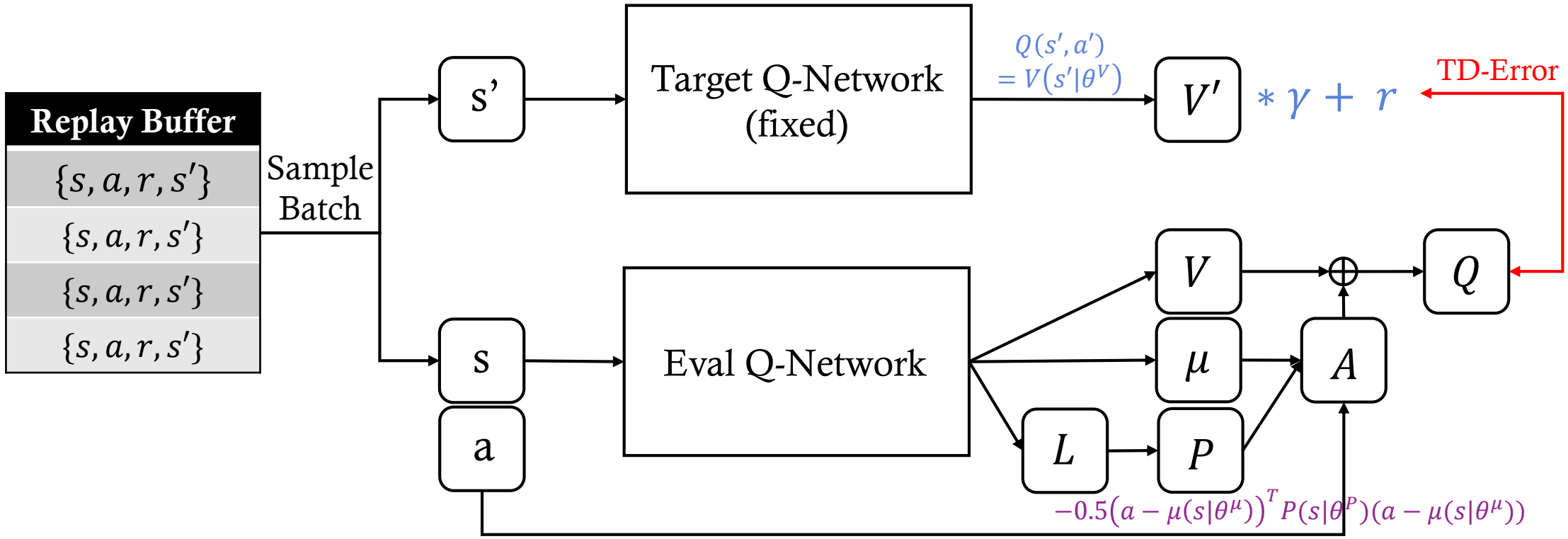
$P(s|\theta^P) = L(s|\theta^P)L(s|\theta^P)^T$

Symmetric Matrix — Upper-Triangle Matrix

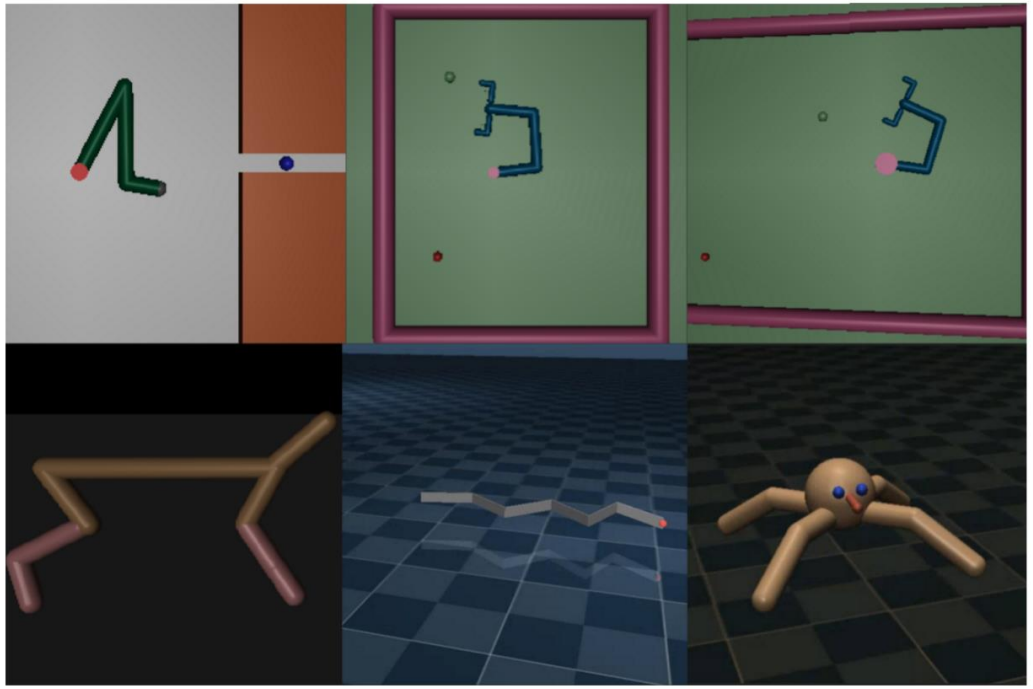
- We can select the action $a = \mu(s|\theta^u)$ as the maximal Q control.

Normalized Advantage Function (NAF)

$$L(\theta) = \mathbb{E}_{\{s,a,r,s'\}}[\left(r + \gamma \max_{a'} Q_{target}(s', a'; \theta^{old}) - Q_{eval}(s, a; \theta)\right)^2]$$



Normalized Advantage Function (NAF)



Algorithm 1 Continuous Q-Learning with NAF

Randomly initialize normalized Q network $Q(\boldsymbol{x}, \boldsymbol{u}|\theta^Q)$.
Initialize target network Q' with weight $\theta^{Q'} \leftarrow \theta^Q$.
Initialize replay buffer $R \leftarrow \emptyset$.
for episode=1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state $\boldsymbol{x}_1 \sim p(\boldsymbol{x}_1)$
  for t=1, T **do**
    Select action $\boldsymbol{u}_t = \mu(\boldsymbol{x}_t|\theta^\mu) + \mathcal{N}_t$
    Execute $\boldsymbol{u}_t$ and observe r_t and $\boldsymbol{x}_{t+1}$
    Store transition $(\boldsymbol{x}_t, \boldsymbol{u}_t, r_t, \boldsymbol{x}_{t+1})$ in R
    for iteration=1, I **do**
      Sample a random minibatch of m transitions from R
      Set $y_i = r_i + \gamma V'(\boldsymbol{x}_{i+1}|\theta^{Q'})$
      Update θ^Q by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(\boldsymbol{x}_i, \boldsymbol{u}_i|\theta^Q))^2$
      Update the target network: $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$
    end for
  end for
end for