# ECE 414/517 Reinforcement Learning

LECTURE 9: MONTE CARLO METHODS

SEP. 22 2022

# Policy Iteration vs. Value Iteration

$$V_{k+1}^{\pi}(s) = \sum_{a} \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \alpha V_k^{\pi}(s')]$$

$$V_0 \to V_1 \to V_2 \to \cdots \to V_k \to V_{k+1} \to \cdots \to V^{\pi}$$

$$\pi'(s) = arg\max_{a} \sum_{s',r} p(s', r\,|s, a)[r + \alpha V^{\pi}(s')]$$

$$\pi_0 \overset{E}{\to} V_{\pi_0} \overset{I}{\to} \pi_1 \overset{E}{\to} V_{\pi_1} \overset{I}{\to} \pi_2 \overset{E}{\to} \ldots \overset{I}{\to} \pi_* \overset{E}{\to} V_{\pi_*}$$

$$V_{k+1}(s) = \max_{a} \sum_{s',r} p(s', r|s, a)\big[r + \alpha V_k\ (s')\big]$$

Solving the Bellman Optimality Equation Iteratively

# Value Iteration

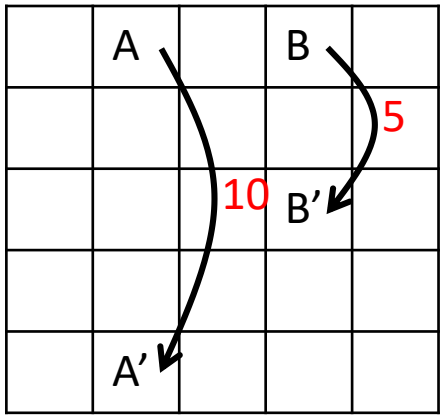
**Value Iteration, for estimating $\pi \approx \pi_*$**

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop:
| $\Delta \leftarrow 0$
| Loop for each $s \in \mathcal{S}$:
| $v \leftarrow V(s)$
| $V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$
| $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Output a deterministic policy, $\pi \approx \pi_*$, such that
$\pi(s) = \text{argmax}_a \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$

# Example: Grid World




$\pi_0(s) = \{0.25, 0.25, 0.25, 0.25\}\ \forall s \in S$

$\gamma = 0.9$

Change $\gamma = 0.8$.
What will happen?

$V_{\pi^*}(1,3)$

| | | | | |
|---|---|---|---|---|
| 21.98 | 24.42 | 21.98 | 19.42 | 17.48 |
| 19.78 | 21.98 | 19.78 | 17.80 | 16.02 |
| 17.80 | 19.78 | 17.80 | 16.02 | 14.42 |
| 16.02 | 17.80 | 16.02 | 14.42 | 12.98 |
| 14.42 | 16.02 | 14.42 | 12.98 | 11.68 |

$V_{\pi^*}$

| | | | | |
|---|---|---|---|---|
| → | ← | ← | ← | ← |
| → | ↑ | ← | ← | ← |
| → | ↑ | ← | ← | ← |
| → | ↑ | ← | ← | ← |
| → | ↑ | ← | ← | ← |

$\pi^*$

| | | | | |
|---|---|---|---|---|
| 11.90 | 14.87 | 11.9 | 10.25 | 8.20 |
| 9.52 | 11.9 | 9.52 | 8.2 | 6.56 |
| 7.62 | 9.52 | 7.62 | 6.56 | 5.25 |
| 6.09 | 7.62 | 6.09 | 5.25 | 4.2 |
| 4.87 | 6.09 | 4.87 | 4.2 | 3.36 |

$V_{\pi^*}$

| | | | | |
|---|---|---|---|---|
| → | ← | ← | ← | ← |
| → | ↑ | ← | ↑ | ← |
| → | ↑ | ← | ↑ | ← |
| → | ↑ | ← | ↑ | ← |
| → | ↑ | ← | ↑ | ← |

$\pi^*$

https://openreview.net/pdf?id=t3BFUDHwEJU

$\gamma = 0.8$

$Q_{\pi^*}(1,3,w) = 0 + 0.8 \cdot 9.52$

$= \underbrace{0} + \underbrace{0.8 \cdot 0} + \underbrace{0.8^2 \cdot 0} + \underbrace{0.8^3 \cdot 14.87}$

$=$

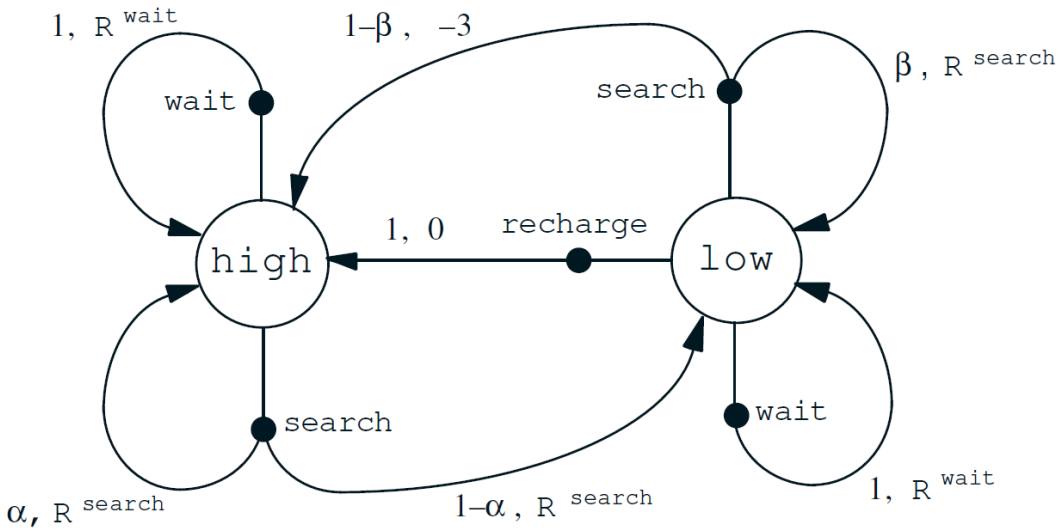
$Q_{\pi^*}(1,3,N) = 0 + 0.8 \cdot 10.25 = 8.2$

# Recycling Robot

h: high
l: low
s: search
w: wait
re: recharge

$$V^*(h) = \max\begin{Bmatrix} R^{search} + \alpha\gamma V^*(h) + (1-\alpha)\gamma V^*(l)] \\ R^{wait} + \gamma V^*(h) \end{Bmatrix}$$

$$V^*(l) = \max\begin{Bmatrix} \beta[R^{search} + \gamma V^*(l)] + (1-\beta)[-3 + \gamma V^*(h)] \\ R^{wait} + \gamma V^*(l) \\ \gamma V^*(h) \end{Bmatrix}$$

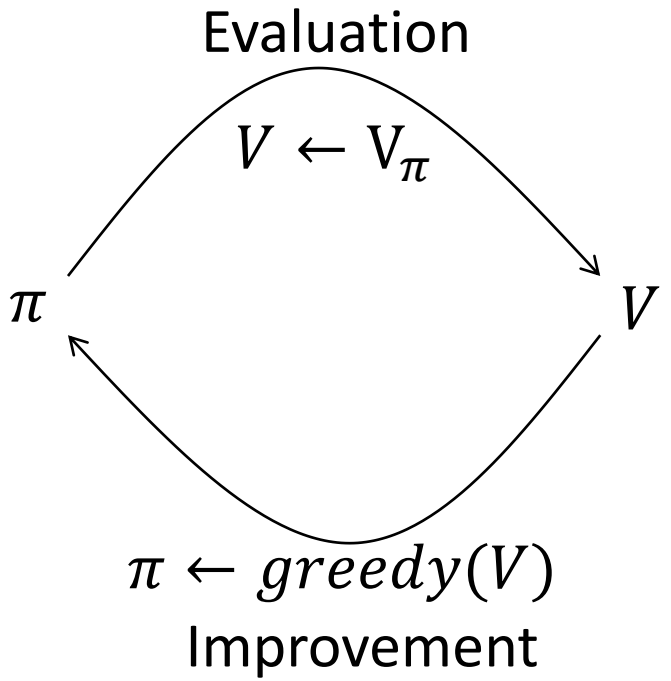


# Efficiency of Dynamic Programming

1. DP may not be practical for large problems (even when environment is known).
2. Still a lot better than exhaustive search.
3. If $|A| = M$ and $|S| = N$ then *DP* is guaranteed to find the optimal policy in less then some polynomial function of $M$ and $N$
4. Exhaustive Search (over deterministic policies):
    1. $M^N$
5. Usually impractical because $|S|$ grows exponentially with the number of state variables (curse of dimensionality)
6. These days DP methods can solve (using PC) MDP's with millions of states.

# Generalized Policy Iteration

1. Policy iteration –
   - Full policy evaluation (up until convergence)
   - Policy improvement
2. Value Iteration –
   - One iteration of policy evaluations
   - Policy Improvement
3. Asynchronous Dynamic Programming
   - Policy evaluation/policy improvement even further interleaved.
   - For example, updating only a single step before performing policy improvement
   - Can be done stochastically, as long as all states are visited.



# Dynamic Programming

Dynamic programming is the collection of algorithms that can be used to compute optimal policies given perfect model of the environment.

1. DP constitutes a theoretically optimal methodology
2. In reality it is often limited since DP is computationally expensive.
3. As related to other methods:
   - Hope to do “just as well” as DP
   - Hopefully can do it with less computation
   - Hopefully can do it with less memory
   - Hopefully without model
4. Key Idea (In general): Use value function to derive optimal policy
5. Need to find a way to calculate optimal state value function

# Monte Carlo Methods

1. Monte Carlo Methods?
2. The advantage of Monte-Carlo methods in reinforcement learning?
   - Do not require a model of the environment
   - Instead  only requires experience.
3. Experience can be gained in two general forms:
   - Online interaction with an environment –
     - Actually interacting with the environment and observing the transitions
     - No prior knowledge needed.
   - Simulated interactions –
     - Require model of environment but not probability distributions
     - Many times it is easy to generate samples, but not to obtain the probability distributions of state transitions.

# Monte Carlo Methods

1. For using MC we will consider strictly episodic tasks.
   - Experience is divided into episodes that terminate regardless of actions taken.
   - MC methods are incremental in an episode by episode manner.
   - Different than the step by step scheme employed by DP (and later on Temporal Difference).
2. Will have a lot of similarities with DP:
   - Policy Evaluation
   - Policy Improvement
   - Generalized Policy iteration
3. How to perform policy evaluation when model is unknown?
   - computation of a value function given a policy.

# Monte Carlo Policy Evaluation

1. We still need to estimate $V_\pi(s)$. Estimate from experience:
   - Average all returns observed after visiting a given state.
   - Each occurrence of state $s$ in an episode is called a visit to $s$
2. Generate an episode following $\pi$:

$$s_0, a_0, r_1, s_1, a_1, r_2, \dots, s_{T-1}, a_{T-1}, r_T,$$

3. First visit MC:
   - Method averages just the returns following first visits to $s$
   - Easier to show convergence. Why?
4. Every Visit MC:
   - Method averages the returns following all the visits to $s$

# Monte Carlo Policy Evaluation

**First-visit MC prediction, for estimating $V \approx v_\pi$**

Input: a policy $\pi$ to be evaluated

Initialize:

$V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$

$Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Loop forever (for each episode):

Generate an episode following $\pi$: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:

$G \leftarrow \gamma G + R_{t+1}$

Unless $S_t$ appears in $S_0, S_1, \ldots, S_{t-1}$:

Append $G$ to $Returns(S_t)$

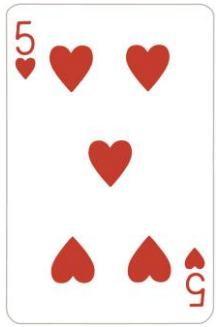
$V(S_t) \leftarrow$ average$(Returns(S_t))$

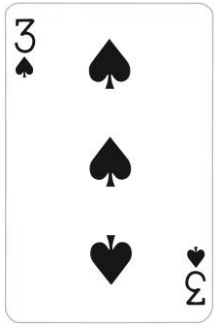
# Blackjack Example

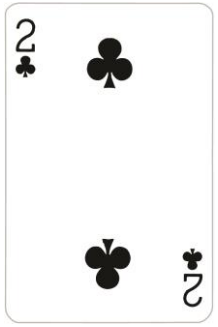
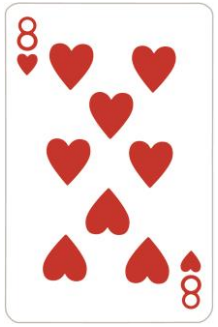
1. The object is to obtain cards where the sum is as great as possible without exceeding 21
   - All face cards (J,Q,K) count as 10
   - Ace can count as either 1 or 11
   - Player competes independently against the dealer.
2. Begins where two cards are dealt to both dealer an agent
3. Player sees his cards, and one of the dealer's.
4. Player can either hit (request another card) or stand (hold his cards)
   - Player can hit multiple times until deciding to stand.
5. After that dealer can do the same (stick or hit)

# Game Example

Dealer Sum: 18



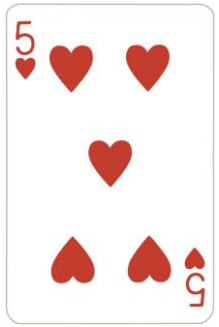


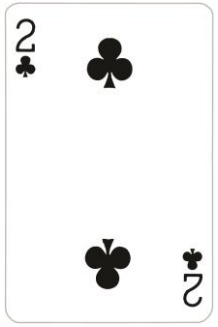





Player Sum: 20

# Game Example

Dealer

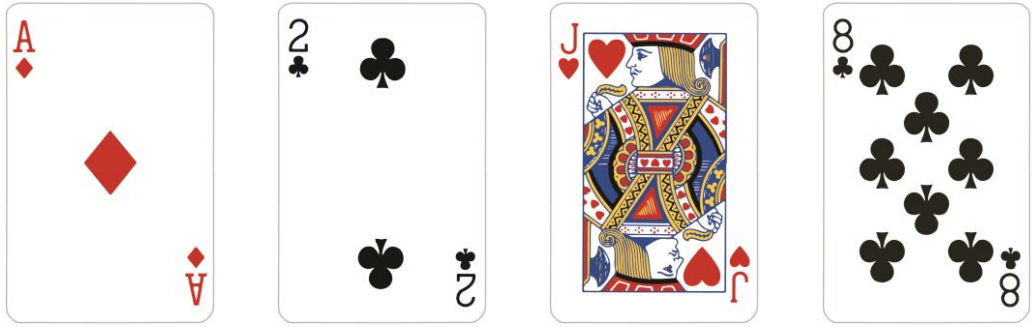


Player Sum: 22 - bust

# Game Example

Dealer Sum: 25 - bust



Player Sum: 21 or **13**

# Blackjack

1. Winning rules:
   - If player is dealt 21 right away, and dealer does not have 21, player wins immediately.
   - If player is dealt 21 right away, and dealer has 21 as well, it is a draw
   - If player exceeds 21 player loses immediately
   - If dealer exceeds 21 player wins immediately
   - If non of the above is true, whoever has the higher number wins.
2. Assumption: Dealer always hits unless sum is 17 or greater, then sticks.
3. Assumption: If a player has a usable ace (i.e. can use it as 11 without exceeding 21), he uses it.
4. Assumption: Cards are dealt from an infinite deck (no advantage of keeping track of cards dealt.
5. Assumption: If sum is less than 12, player always hits.

# Blackjack - MDP

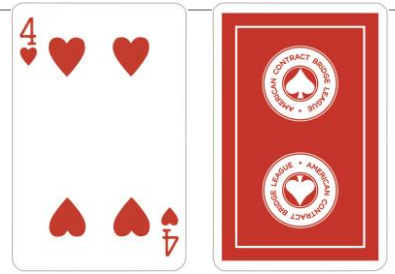
1. Episodic/Nonepisodic?
2. States?
3. Actions?
4. Reward?
5. State value function: array shape?

2. - current sum (12-21)
   - usable ace (0,1)
   - dealer's card (1-10)
3. hit/stand

4. win +1
   lose −1
   tie 0
5. V(s) shape: 10 × 10 × 2

# Policy Evaluation Example

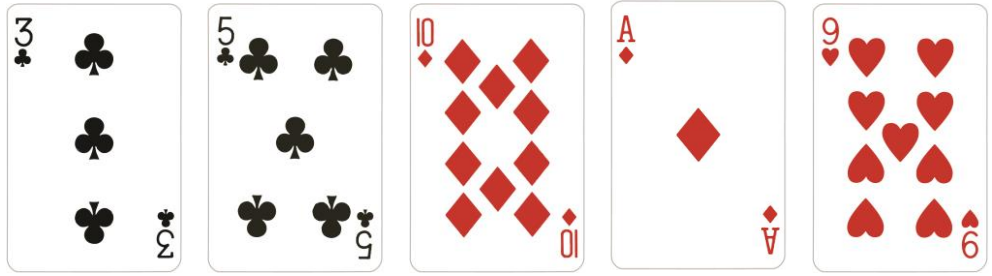
Consider a policy that only stands if the player's sum is 20 or 21.



Use first visit or every visit?

Consider the following cases:

player cards:[5, 3] dealer shows: 4
player hits:10
player hits:1
player hits:9
bust!



What values will change and to what?

$S_0$ $a_0$ $R_0$ $S_1$ $a_1$ $R_1$

$(18, 4, 0) \rightarrow \text{hit} \rightarrow 0 \rightarrow (19, 4, 0) \rightarrow \text{hit} \rightarrow -1 \rightarrow T$

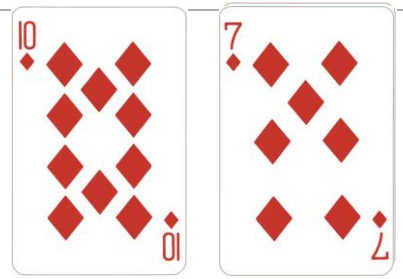
Evaluating $V_\pi(s)$

$$V_\pi(18, 4, 0) = -1$$

$$V_\pi(19, 4, 0) = -1/1$$

# Policy Evaluation Example

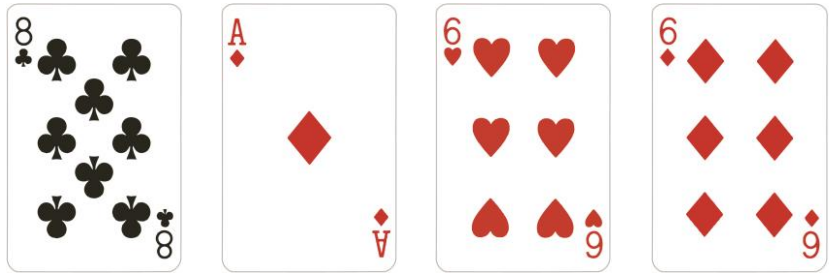
Consider a policy that only stands if the player's sum is 20 or 21.



Use first visit or every visit?

Consider the following cases:

player cards:[11, 8] dealer shows:
10
player hits:6
player hits:6
player holds
dealer sum: 17
reward: 1



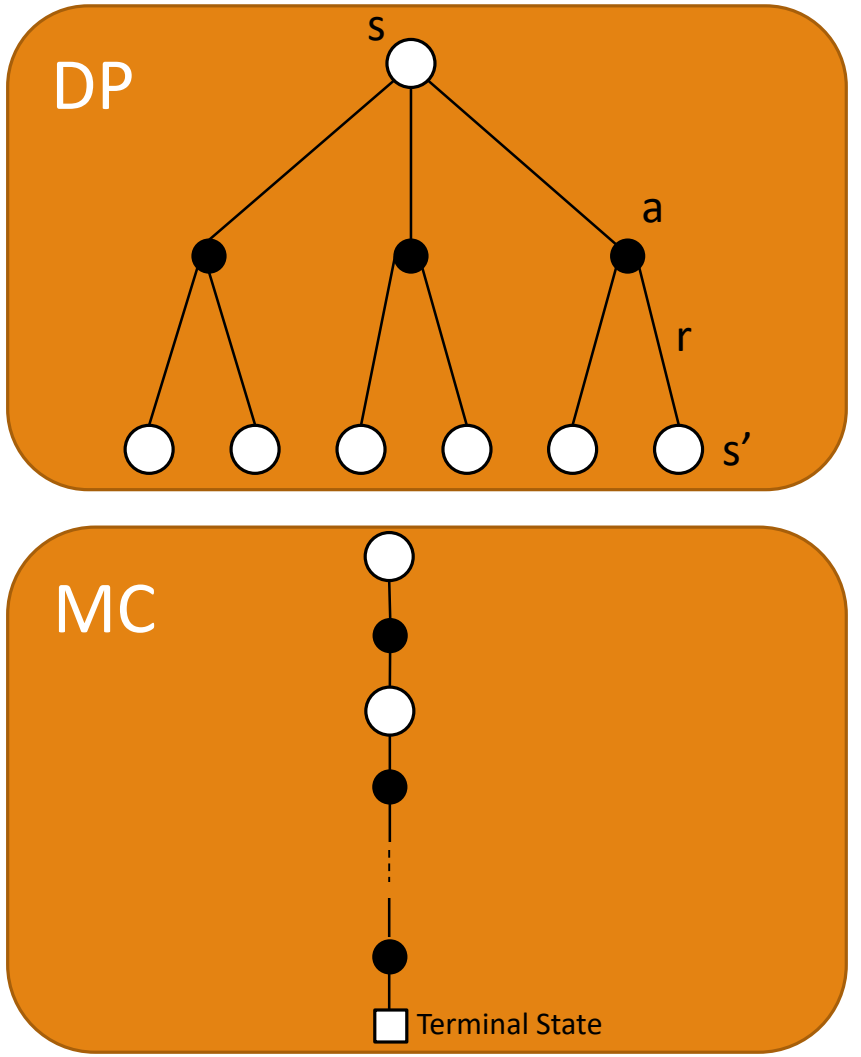
What values will change and to what?

# Blackjack – Use DP?

1. Although we have complete knowledge of the environment, it would not be easy to apply DP policy evaluation.
2. Hard to calculate $p(s', r|s, a)$
   - e.g. player's sum is 14 and dealer's displayed card is jack – what is the expected reward?
3. Since all of these calculations must be done prior to running DP – it is often an impractical approach
4. In contrast, generating sample games (for MC) is much easier to do
5. Surprising insight: even if the environment's dynamics are known, MC is often a more efficient method to apply
   - Estimating values are independent (no "bootstrapping") – why helpful?
   - Optimal policy trajectory corresponds to a small state subset

# Monte Carlo Vs. Dynamic Programming

1. Can the backup diagram be applied to MC methods?
   - Recall that backup diagrams show top node to be updated and below all the transitions and leaf nodes that contribute to the update
2. For MC the root is a state node and below are all the nodes visited until terminal node is reached.
   - Shows only transitions sampled on one episode.
3. DP focuses on one-step transitions, whereas MC goes all the way to the end of the episode.
4. Updating Values for states is independent in MC.
   - computational complexity of updating one node is independent of $|s|$
   - An attractive feature since one can estimate only a subset of the node values (not have to do all)



# Now what?

1. We described a way to find $V^{\pi}(s)$. Now what?
2. Since we do not have model, cannot easily do policy improvement: find the new greedy $\pi$

$$\pi'(s) = arg\max_{a} \sum_{s',r} p(s', r\,|s, a)[r + \alpha V^{\pi}(s')]$$

3. Instead we should estimate $Q^{\pi}(s, a)$.
4. Use the same MC approach. The agent records the rewards received after taking action $a$ at state $s$
   - Problem?
5. Need to force exploration or some actions will never be chosen.
   - How?
6. One way: **exploring starts**. Each state-action pair at the beginning of an episode has a non zero probability.
   - Will consider the general stochastic approach later.

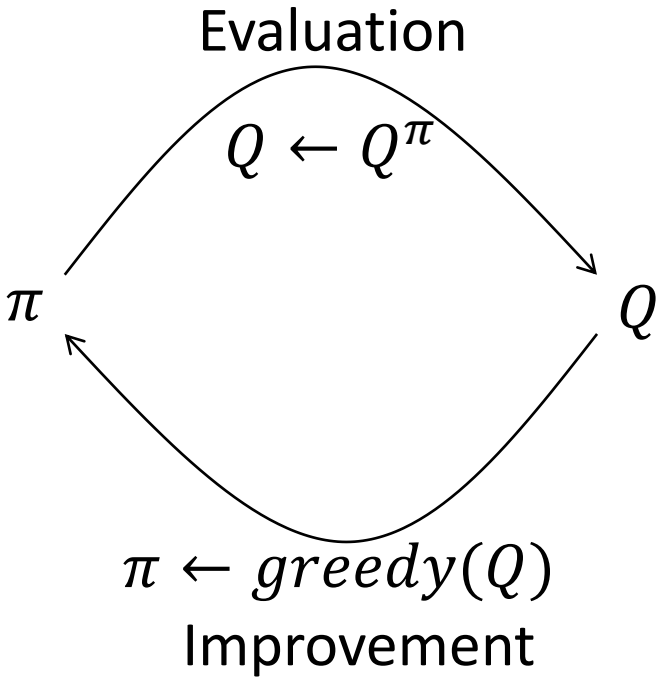
# Monte Carlo Control

How do we use the policy evaluation step to find the optimal policy?

1. Policy Evaluation
   - Achieved by averaging returns over many outcomes
2. Policy improvement
   - Policy improvement is done by selecting greedy actions, i.e.:

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

$$\pi_0 \xrightarrow{E} Q_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} Q_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \dots \xrightarrow{I} \pi_* \xrightarrow{E} Q_{\pi_*}$$



# Monte Carlo Control

1. The policy improvement theorem holds since for all $s$:

$$\begin{aligned} Q^{\pi_k}\left(s, \pi_{k+1}(s)\right) &= Q^{\pi_k}(s, \underset{a}{arg\max}\, Q^{\pi_k}(s,a)) \\ &= \max_a Q^{\pi_k}(s,a) \\ &\geq Q^{\pi_k}\left(s, \pi_k(s)\right) \\ &= V^{\pi_k}(s) \end{aligned}$$

2. If the two policies are equal, they are both optimal
   - This way MC can lead to optimal policies with no model
3. Assumes exploring starts and infinite number of episodes for MC policy evaluation
4. Latter not really needed (similar to DP):
   - Update only to a given level of performance
   - Don't solve for $Q^{\pi_k}$ but move towards it
   - Alternate between evaluation and improvement per episode.

# Monte Carlo (Exploring Starts)

**Monte Carlo ES (Exploring Starts), for estimating $\pi \approx \pi_*$**

Initialize:

$\pi(s) \in \mathcal{A}(s)$ (arbitrarily), for all $s \in \mathcal{S}$

$Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

$Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Loop forever (for each episode):

Choose $S_0 \in \mathcal{S}$ and $A_0 \in \mathcal{A}(S_0)$ such that all pairs have probability $> 0$

Generate an episode from $S_0, A_0$, following $\pi$: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:

$G \leftarrow \gamma G + R_{t+1}$

Unless the pair $S_t, A_t$ appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:

Append $G$ to $Returns(S_t, A_t)$

$Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$

$\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$

# No Exploring Starts

1. Exploring starts not always possible.
   - Why?
2. On the other hand can't simply follow greedy approach – no exploration.
3. We can employ an $\epsilon$-greedy policy instead, where $\pi(s,a) > 0 \quad \forall \quad s,a$:
   - With probability $1-\epsilon$ choose the greedy action
   - With probability $\epsilon$ – explore uniformly
4. Probabilities of actions:
   - For non max actions: $\frac{\epsilon}{|\mathcal{A}(s)|}$
   - For max action: $1-\epsilon+\frac{\epsilon}{|\mathcal{A}(s)|}$

# Policy Improvement under $\epsilon$-greedy

Show that the policy improvement theory holds.

$$
\begin{aligned}
Q^{\pi_k}\big(s, \pi_{k+1}(s)\big) &= \sum_a \pi_{k+1}(a|s)\, Q^{\pi_k}(s,a) \\
&= \frac{\epsilon}{|\mathcal{A}(s)|} \sum_a Q^{\pi_k}(s,a) + (1-\epsilon) \max_a Q^{\pi_k}(s,a) \\
&\geq \frac{\epsilon}{|\mathcal{A}(s)|} \sum_a Q^{\pi_k}(s,a) + (1-\epsilon) \sum_a \frac{\pi_k(a|s) - \frac{\epsilon}{|\mathcal{A}(s)|}}{1-\epsilon} Q^{\pi_k}(s,a) \\
&= \frac{\epsilon}{|\mathcal{A}(s)|} \sum_a Q^{\pi_k}(s,a) - \frac{\epsilon}{|\mathcal{A}(s)|} \sum_a Q^{\pi_k}(s,a) + \pi_k(a|s) Q^{\pi_k}(s,a) \\
&= V^{\pi_k}(s)
\end{aligned}
$$