"Where did you go to, if I may ask?" said Thorin to Gandalf as they rode along.

"To look ahead," said he.

"And what brought you back in the nick of time?"

"Looking behind," said he.

J.R.R. Tolkien, *The Hobbit*

CMPUT 655 Introduction to RL

Marlos C. Machado

Class 4/12

Plan

- Wrap up MDPs
- An Example: Working with v_π
- Dynamic programming / Bellman Equations
  - A different solution, albeit limited
- Monte Carlo Methods

Reminder

You **should be enrolled in the private session** we created in Coursera for CMPUT 365.

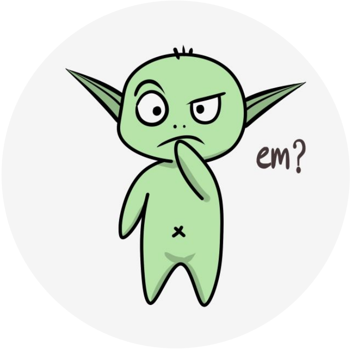
I **cannot** use marks from the public repository for your course marks.

You **need** to **check**, **every time**, if you are in the private session and if you are submitting quizzes and assignments to the private section.

The deadlines in the public session **do not align** with the deadlines in Coursera.

If you have any questions or concerns, **talk with the TAs** or email us `cmput655@ualberta.ca`.

Please, interrupt me at any time!



https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

Optimal Policies and Optimal Value Functions

- Value functions define a partial ordering over policies.
  - $\pi \geq \pi'$ iff $v_\pi(s) \geq v_{\pi'}(s)$ for all $s \in \mathcal{S}$.
  - There is always at least one policy that is better than or equal to all other policies. The *optimal policy*.

$$v_*(s) \doteq \max_\pi v_\pi(s)$$

$$q_*(s,a) \doteq \max_\pi q_\pi(s,a)$$

$$q_*(s,a) = \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a]$$

Optimal Policies and Optimal Value Functions

- Because v$_*$ is the value function for a policy, it must satisfy the self-consistency condition given by the Bellman equation for state values.

$$v_*(s) = \max_{a \in \mathcal{A}(s)} q_{\pi_*}(s, a)$$

Optimal Policies and Optimal Value Functions

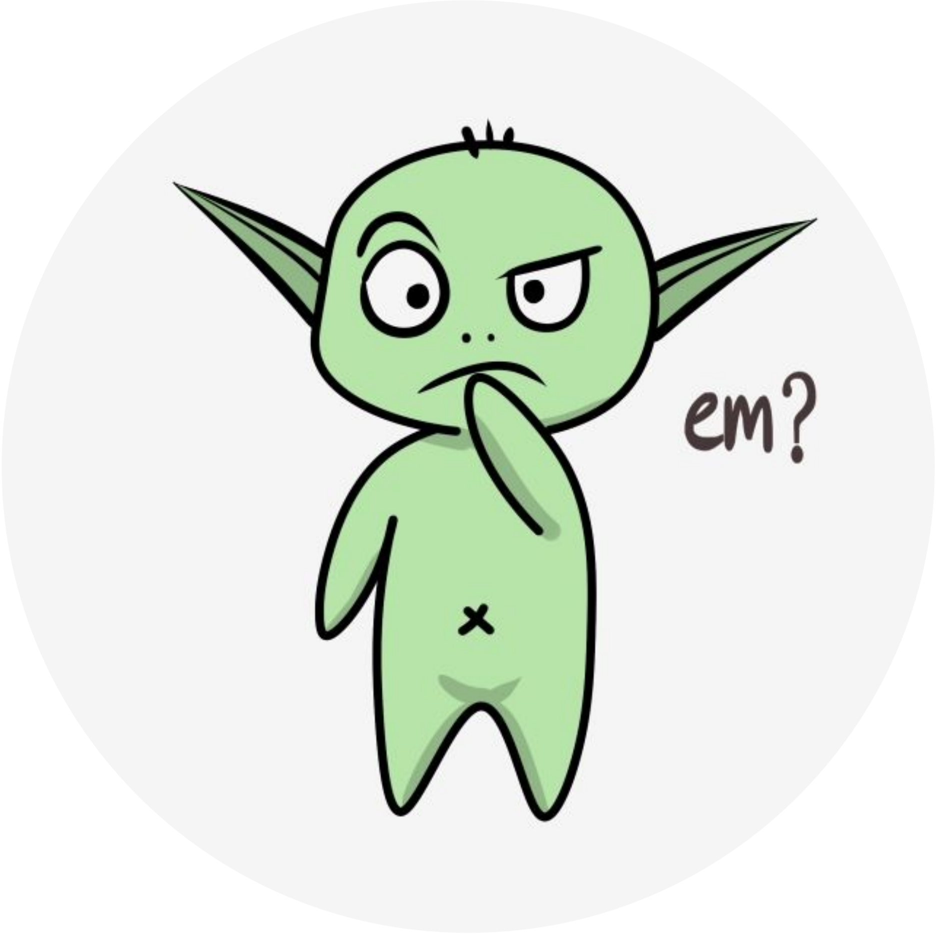
- Because v$_*$ is the value function for a policy, it must satisfy the self-consistency condition given by the Bellman equation for state values.

$$\begin{aligned} v_*(s) &= \max_{a\in\mathcal{A}(s)} q_{\pi_*}(s,a) \\ &= \max_a \mathbb{E}_{\pi_*}[G_t \mid S_t = s, A_t = a] \\ &= \max_a \mathbb{E}_{\pi_*}[R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t = a] \\ &= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \max_a \sum_{s',r} p(s',r|s,a)\big[r + \gamma v_*(s')\big]. \end{aligned}$$

$$\begin{aligned} q_*(s,a) &= \mathbb{E}\Big[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1},a') \;\Big|\; S_t = s, A_t = a\Big] \\ &= \sum_{s',r} p(s',r|s,a)\Big[r + \gamma \max_{a'} q_*(s',a')\Big]. \end{aligned}$$

Also…

I have highlighted a couple of exercises during the class, but there are more. The exercises in Chapter 3 of the book are great. I particularly encourage you to look at Exercises 3.25 —3.29 as well.



https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

Reinforcement learning is very related to search algorithms

“Heuristic search methods can be viewed as expanding the right-hand side of the equation below several times, up to some depth, forming a “tree” of possibilities, and then using a heuristic evaluation function to approximate v_**, at the “leaf” nodes.”*

$$v_*(s) = \max_a \sum_{s',r} p(s',r|s,a)\big[r + \gamma v_*(s')\big].$$

Yay! We solved sequential decision-making problems

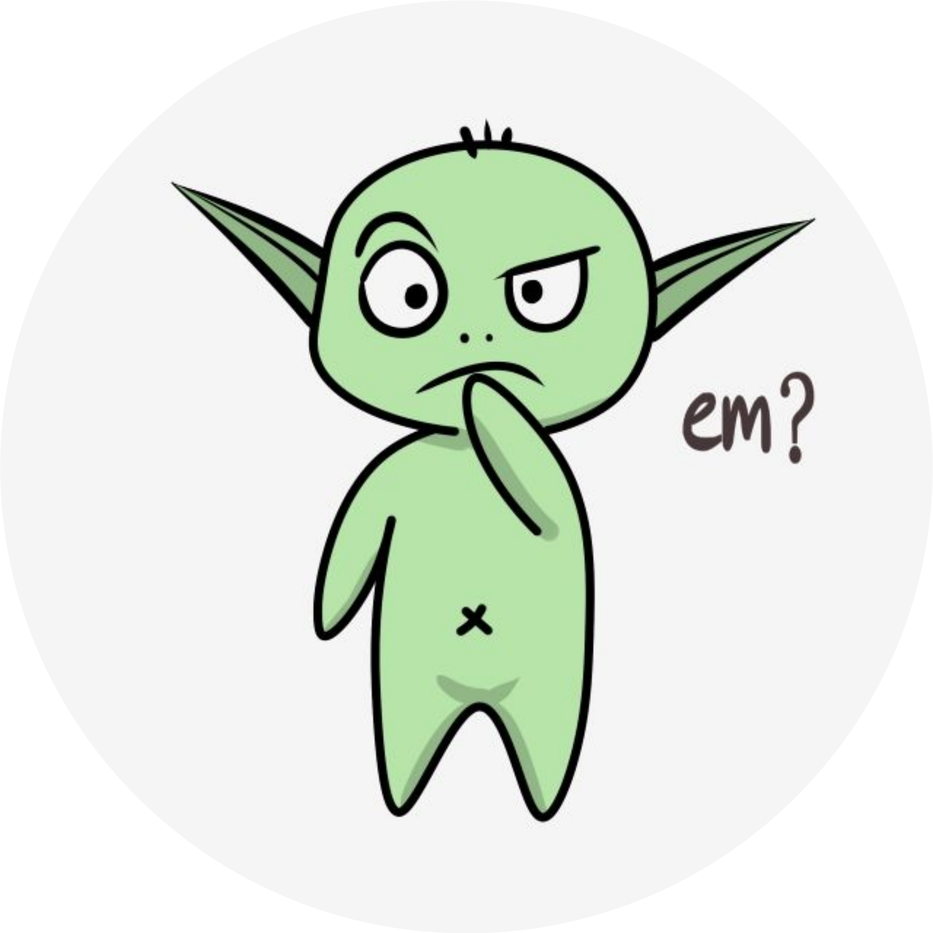
Except…

1.
2.
3.

Yay! We solved sequential decision-making problems

Except…

1. we need to know the dynamics of the environment
2. we have enough computational resources to solve the system of linear eq.
3. the Markov property



https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

An Example: Working with v_π

Whiteboard

Chapter 4

Dynamic Programming

Dynamic Programming – Why?

- “DP provides an essential foundation for the understanding of the methods presented in the rest of this book”.
- … but “classical DP algorithms are of limited utility in reinforcement learning both because of their assumption of a perfect model and because of their great computational expense”.
- “all of these [RL] methods can be viewed as attempts to achieve much the same effect as DP, only with less computation and without assuming a perfect model of the environment”.

Key Idea Behind Dynamic Programming

“To use value functions to organize and structure the search for good policies.”

We use the same equations as before, but we replace an = by a ←, that’s it (we turn Bellman equations into assignments).

Policy Evaluation (Prediction)

Given a policy and an MDP, what's the corresponding value function?

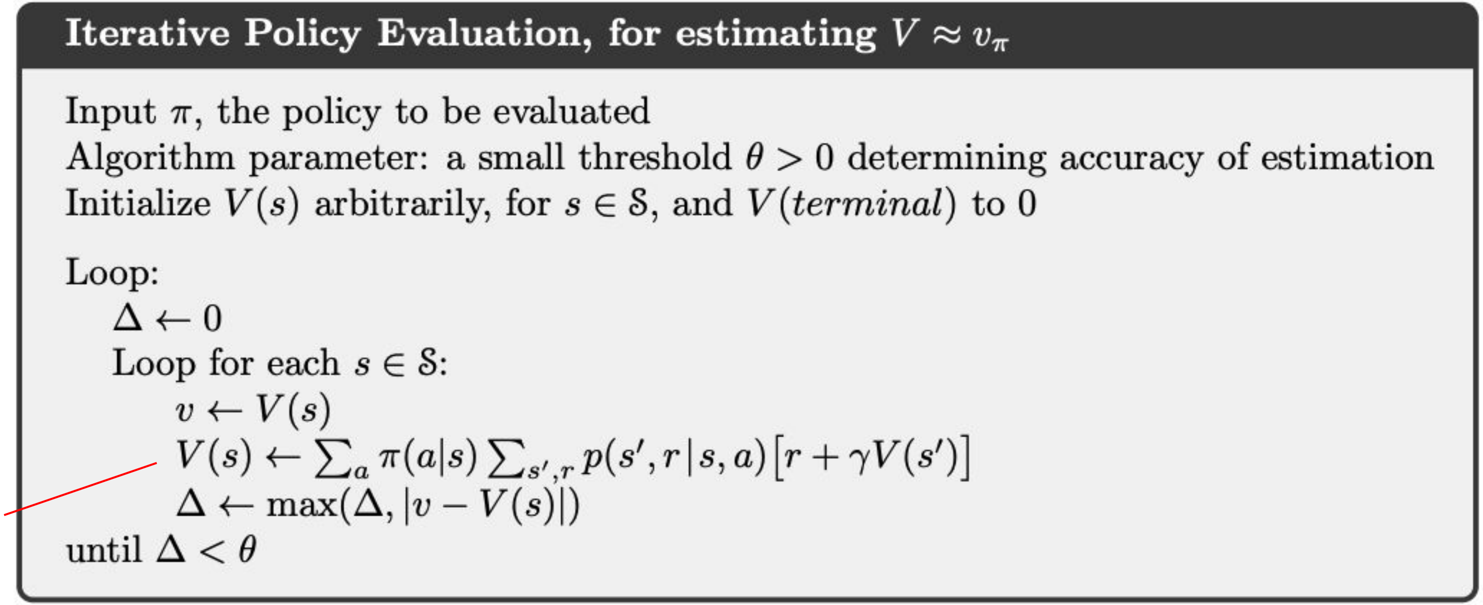
$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\Big[r + \gamma v_\pi(s')\Big] \end{aligned}$$

$$\downarrow$$

$$\begin{aligned} v_{k+1}(s) &\doteq \mathbb{E}_\pi[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\Big[r + \gamma v_k(s')\Big] \end{aligned}$$

expected update

Policy Evaluation (Prediction)

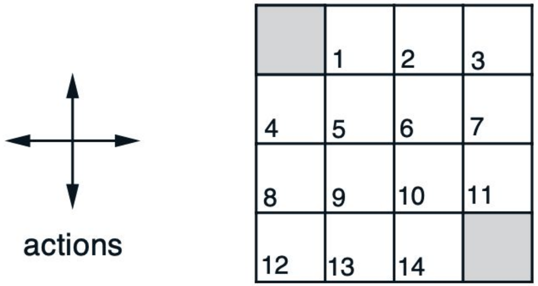


Iterative Policy Evaluation, for estimating $V \approx v_\pi$

```
Input $\pi$, the policy to be evaluated
Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$ arbitrarily, for $s \in \mathcal{S}$, and $V(terminal)$ to 0

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

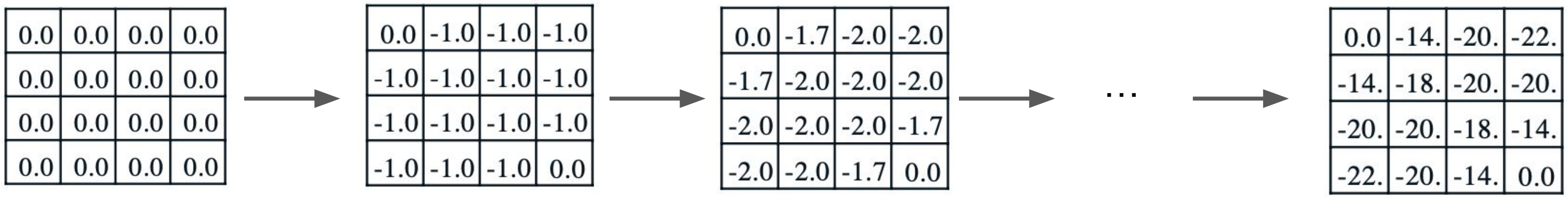
“in-place” update

Policy Evaluation – Example



$R_t = -1$
on all transitions

v_k for the random policy



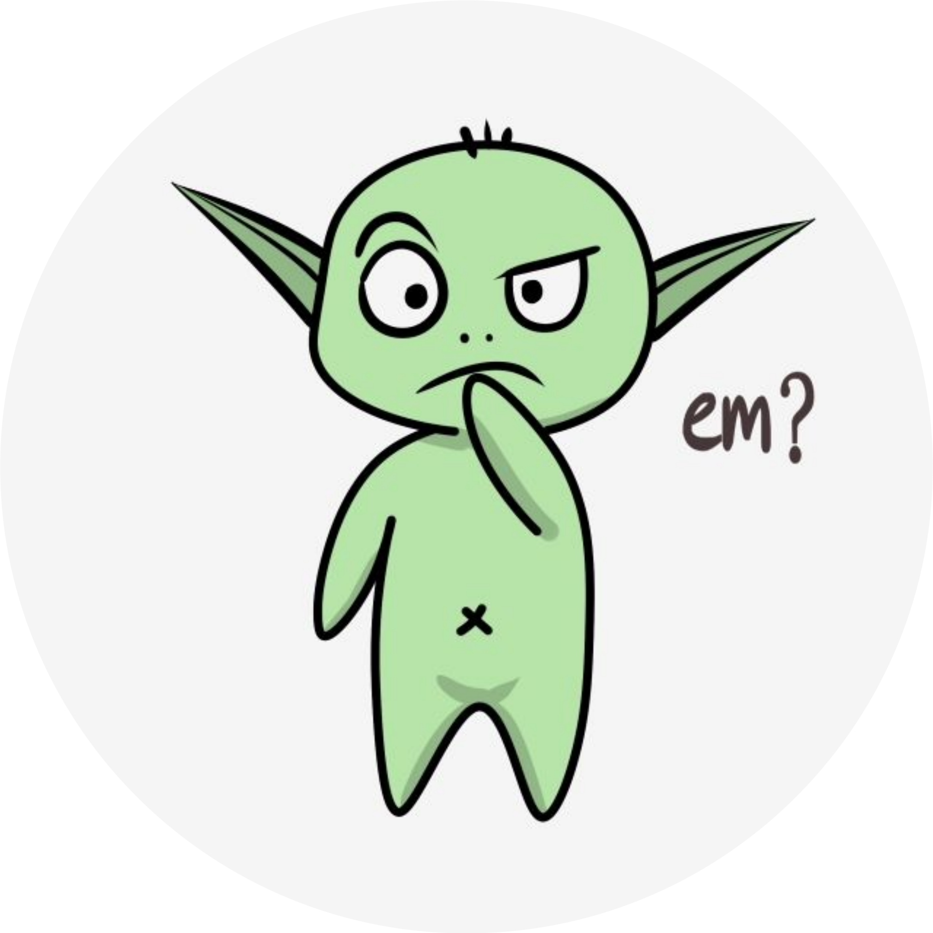
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0

…

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0



https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

Policy Improvement

Given a value function for a policy π, how can we get a better policy π'?

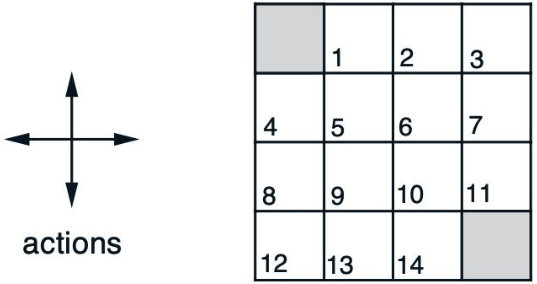
We already know how good policy π is, what if we acted differently now? What if instead of selecting action π(s) we selected action a ≠ π(s), but then we followed π?

We know the value of doing that!

$$\begin{aligned} q_\pi(s,a) &\doteq \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \sum_{s',r} p(s',r|s,a)\Big[r + \gamma v_\pi(s')\Big]. \end{aligned}$$

If this new action is better, in general this new policy is better overall

Policy Improvement – Intuition

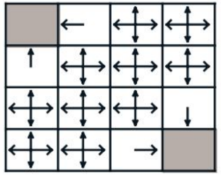


$R_t = -1$
on all transitions

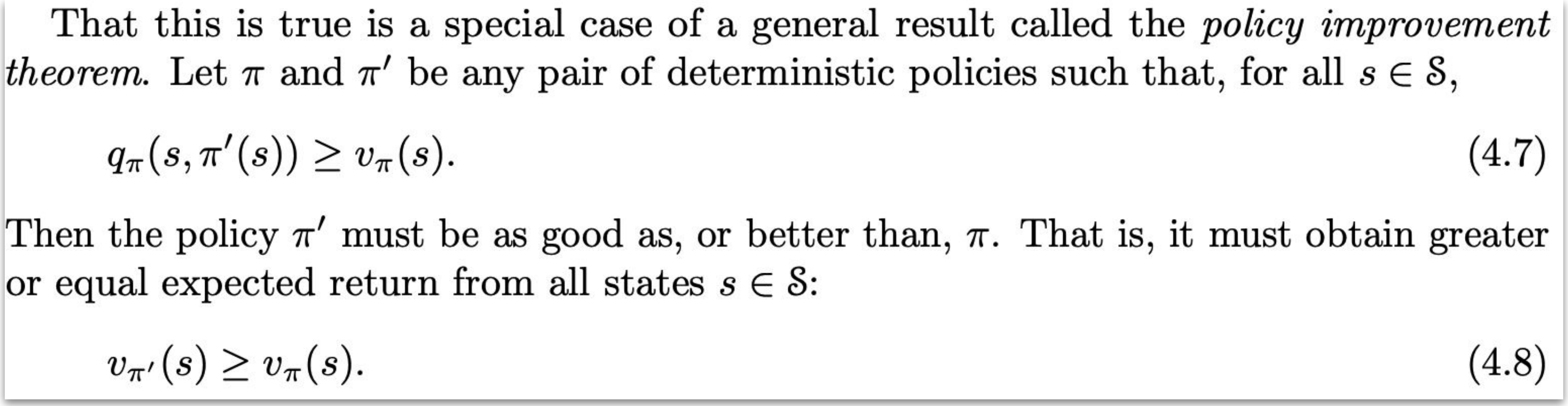
v_k for the random policy

$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0



Policy Improvement Theorem



That this is true is a special case of a general result called the *policy improvement theorem*. Let π and π' be any pair of deterministic policies such that, for all $s \in \mathcal{S}$,

$$q_\pi(s, \pi'(s)) \geq v_\pi(s). \tag{4.7}$$

Then the policy π' must be as good as, or better than, π. That is, it must obtain greater or equal expected return from all states $s \in \mathcal{S}$:

$$v_{\pi'}(s) \geq v_\pi(s). \tag{4.8}$$

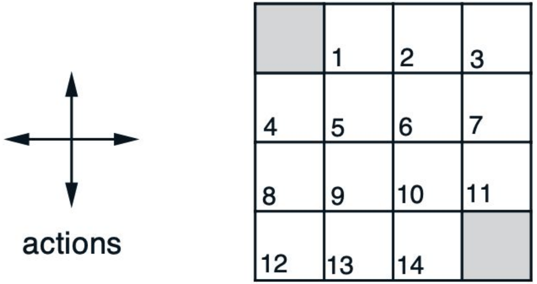
A more aggressive update

Instead of doing it for a particular action in a single state, we can consider changes at *all* states and to *all* possible actions.

$$\begin{aligned}
\pi'(s) &\doteq \operatorname*{arg\,max}_a q_\pi(s,a) \\
&= \operatorname*{arg\,max}_a \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a] \\
&= \operatorname*{arg\,max}_a \sum_{s',r} p(s',r \mid s,a)\Big[r + \gamma v_\pi(s')\Big],
\end{aligned}$$

This is called *policy improvement*. And eventually it converges to the optimal policy.

Policy Improvement – Intuition

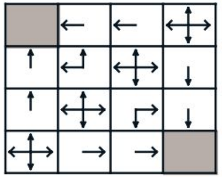


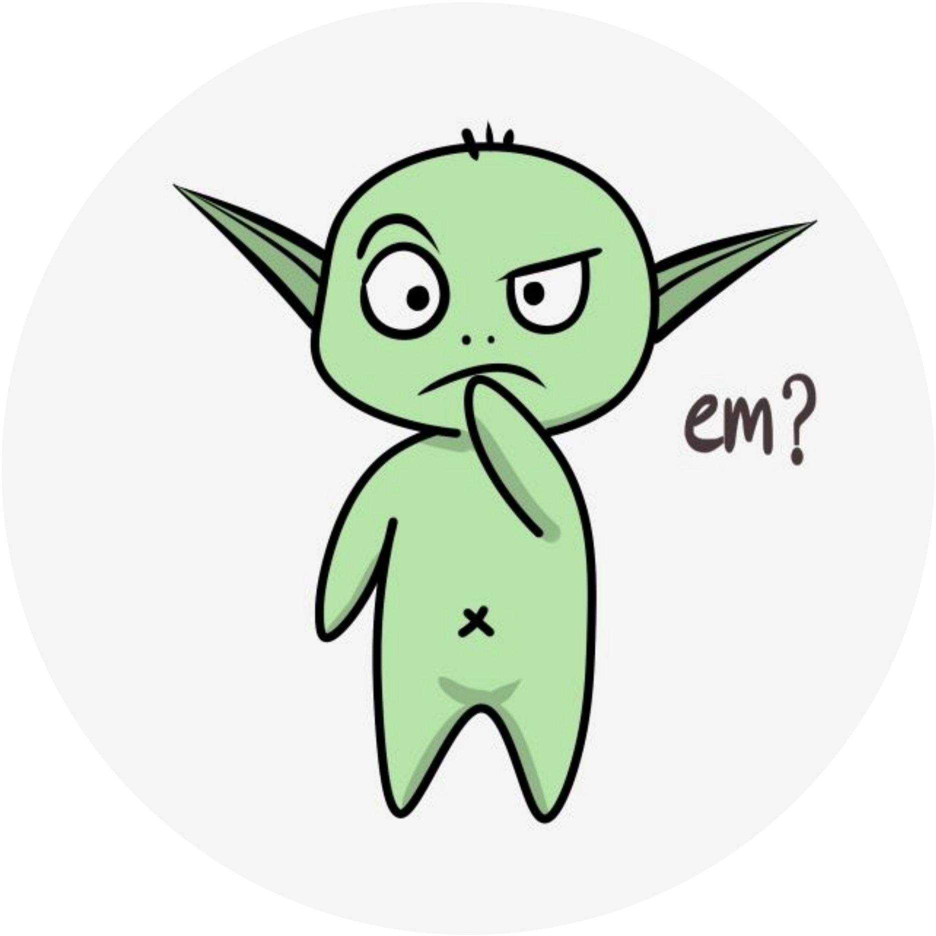
$R_t = -1$
on all transitions

v_k for the random policy

$k = 2$

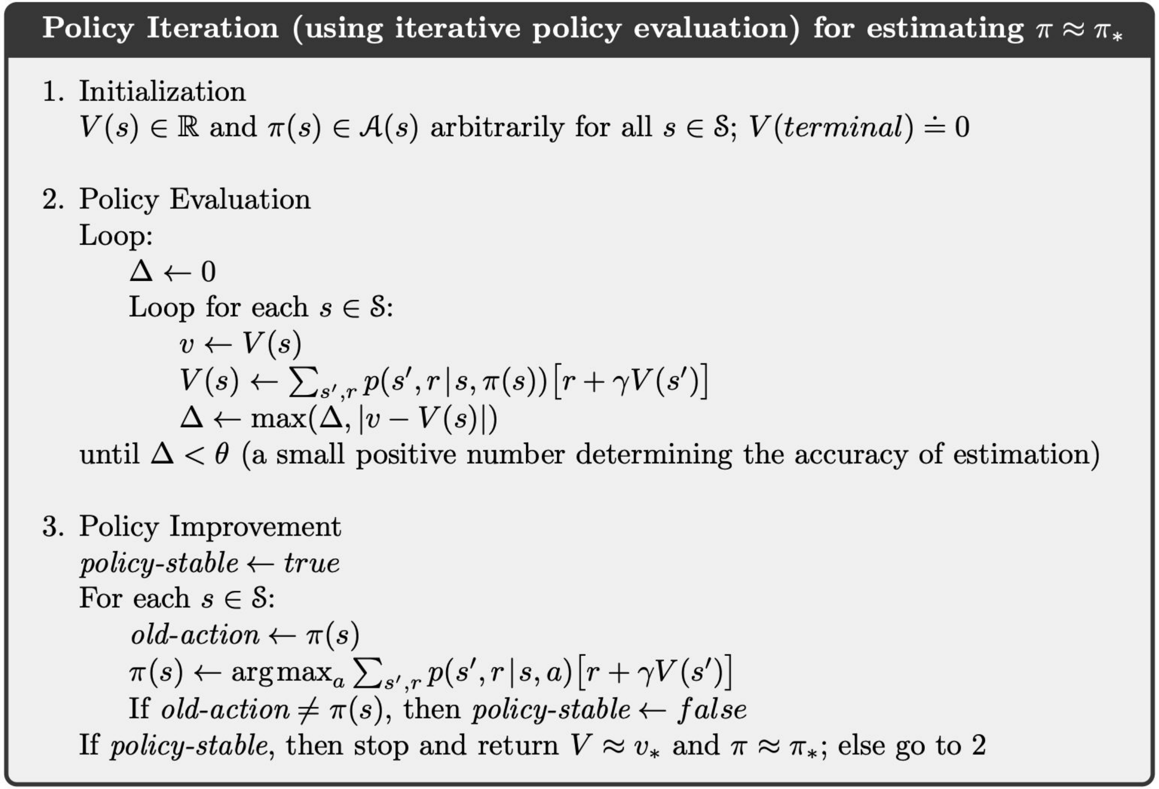
0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0





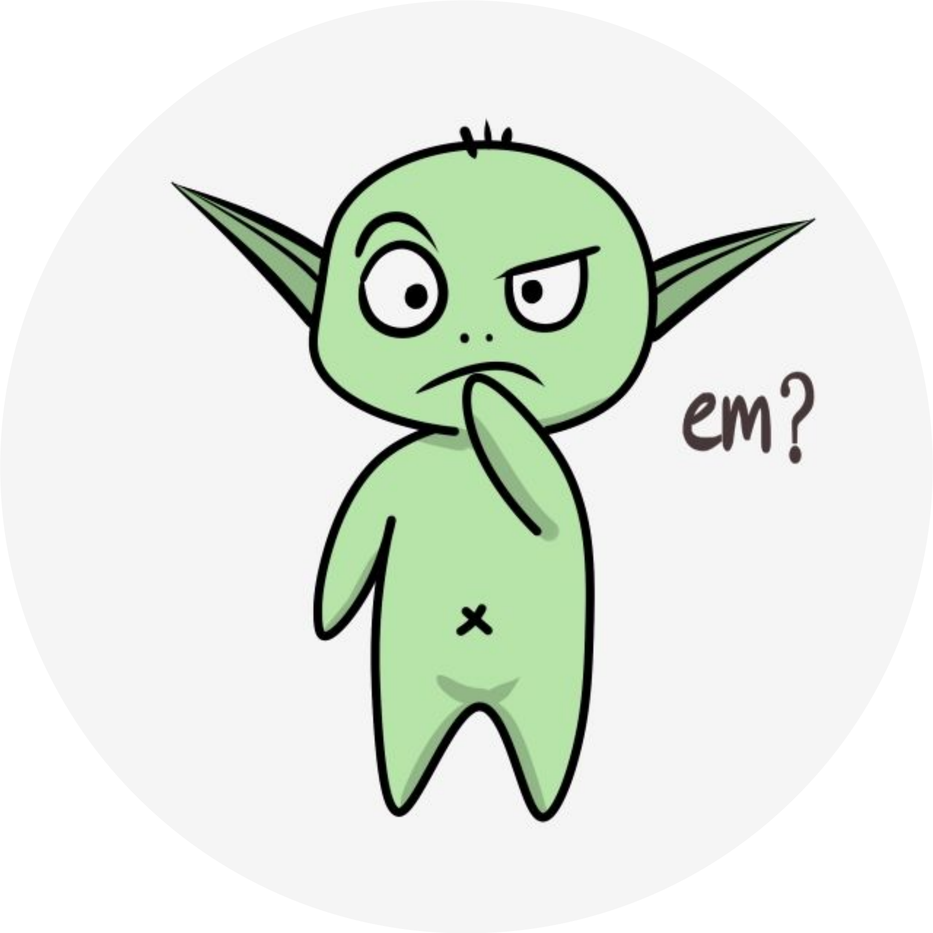
https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

Policy Iteration: Interleaving Policy Eval. and Improvement



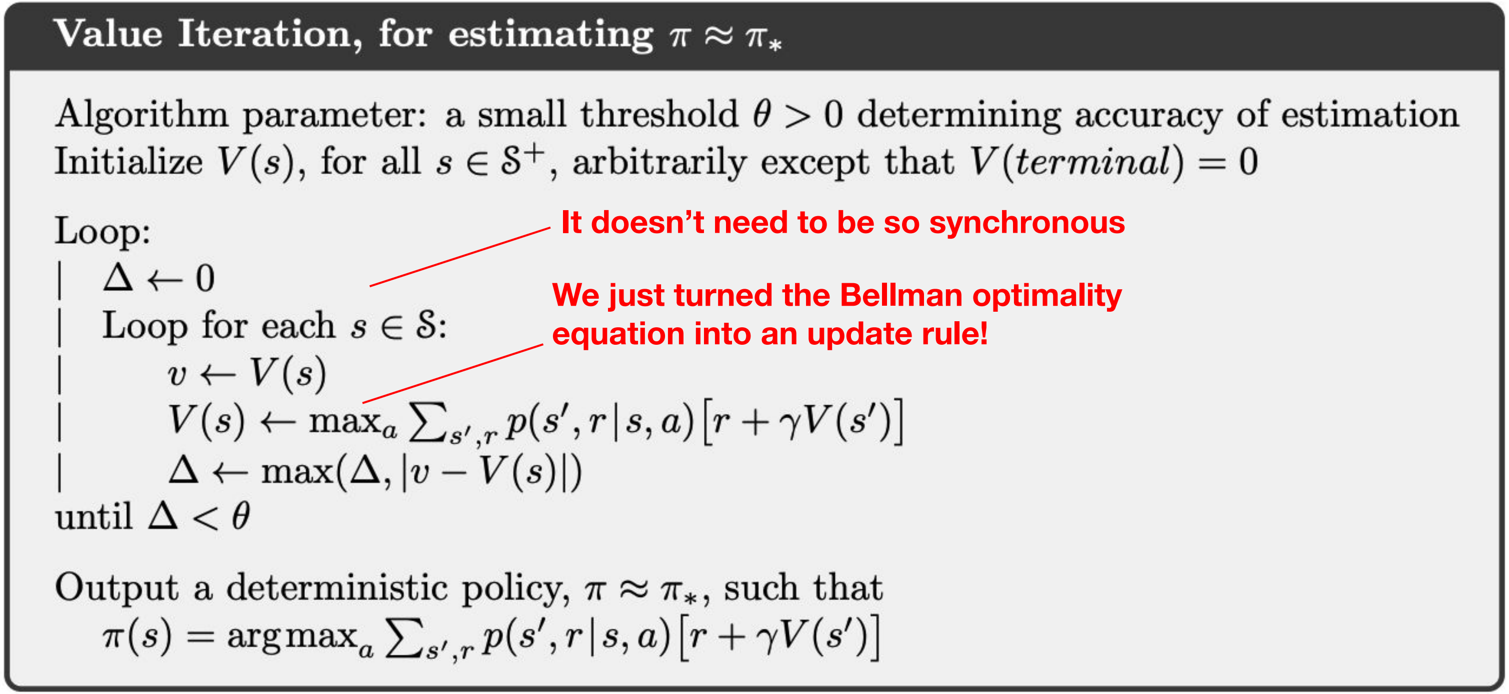
Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

```
1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$; $V(terminal) \doteq 0$

2. Policy Evaluation
   Loop:
       $\Delta \leftarrow 0$
       Loop for each $s \in \mathcal{S}$:
           $v \leftarrow V(s)$
           $V(s) \leftarrow \sum_{s',r} p(s',r|s,\pi(s))\big[r + \gamma V(s')\big]$
           $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement
   $policy\text{-}stable \leftarrow true$
   For each $s \in \mathcal{S}$:
       $old\text{-}action \leftarrow \pi(s)$
       $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]$
       If $old\text{-}action \neq \pi(s)$, then $policy\text{-}stable \leftarrow false$
   If $policy\text{-}stable$, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2
```



https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

Value Iteration



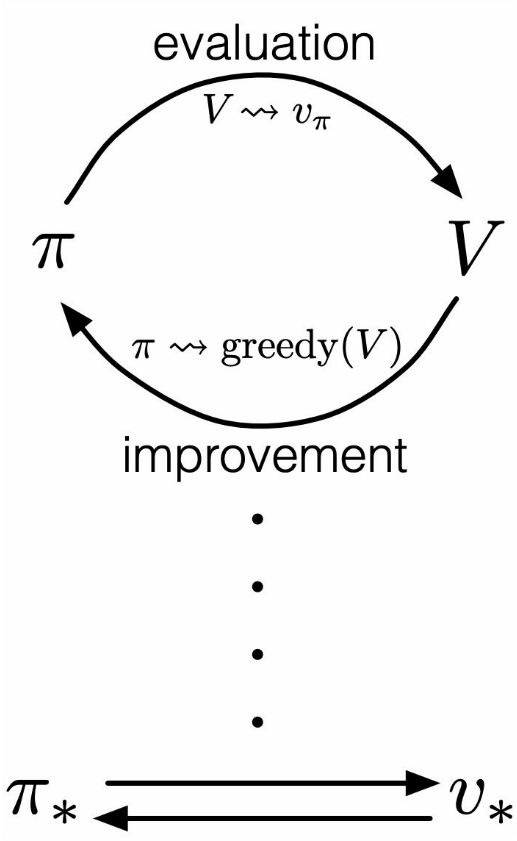
Value Iteration, for estimating $\pi \approx \pi_*$

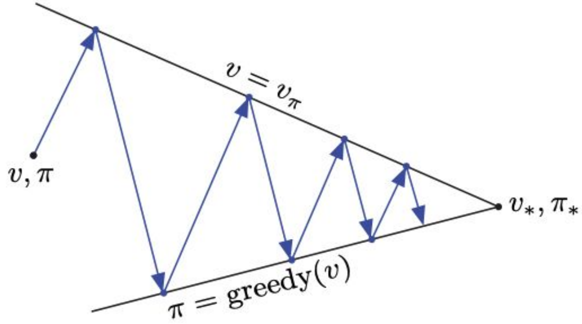
```
Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

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
    $\pi(s) = \arg\max_a \sum_{s',r} p(s', r | s, a)\big[r + \gamma V(s')\big]$
```

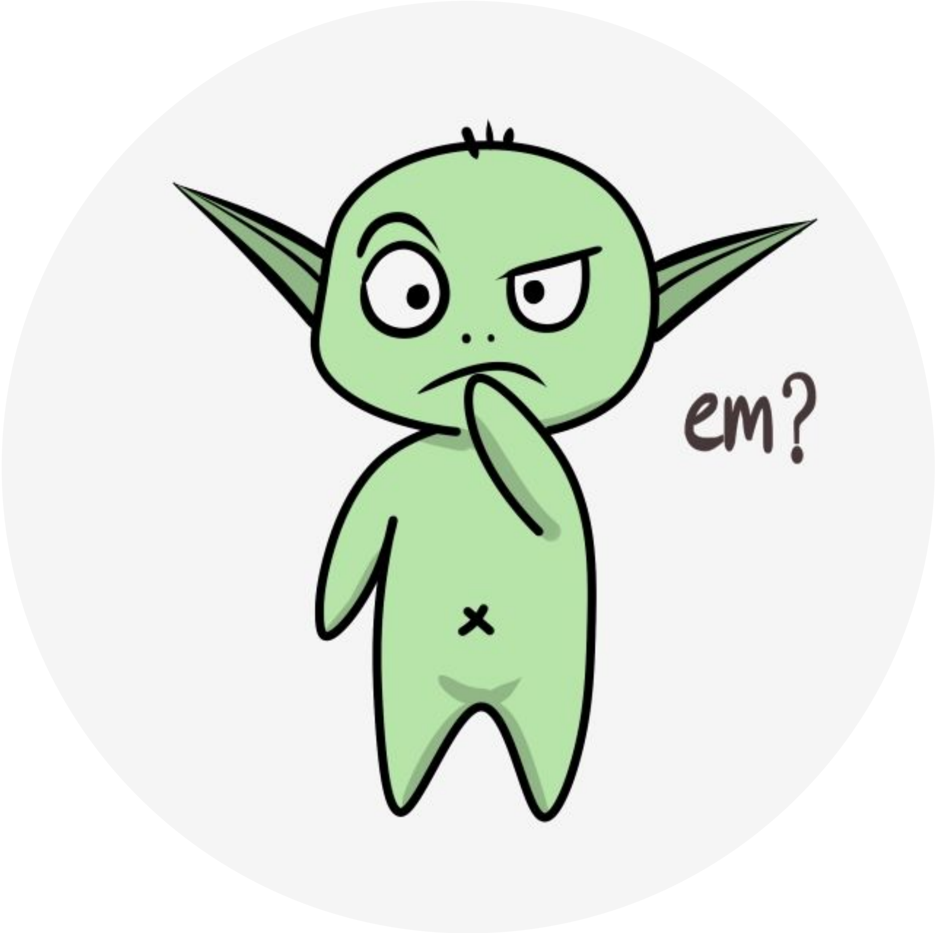
It doesn't need to be so synchronous

We just turned the Bellman optimality equation into an update rule!

Generalized Policy Iteration


evaluation
$V \rightsquigarrow v_\pi$
π
V
$\pi \rightsquigarrow \mathrm{greedy}(V)$
improvement
π_*
v_*





https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

The Bellman Operator

Functions that map elements of a space onto itself, are called *operators*.

$$v(s) = \mathbb{E}_\pi[r + \gamma v(s')|S_t = s]$$

We are transforming a value-function vector into another value-function vector.

The Bellman Operator

Functions that map elements of a space onto itself, are called *operators*.

$$v(s) = \mathbb{E}_\pi[r + \gamma v(s')|S_t = s]$$

We are transforming a value-function vector into another value-function vector.

The Bellman operator is the mapping $T_\pi : \mathbb{R}^{|\mathscr{S}|} \rightarrow \mathbb{R}^{|\mathscr{S}|}$ defined by

$$(T_\pi v)(s) = \mathbb{E}_\pi[r + \gamma v(s')|S_t = s]$$

and we can be quite concise: $v = T_\pi v.$

The Bellman Backup is a Contraction

The operator $T_\pi : \mathbb{R}^{|\mathscr{S}|} \rightarrow \mathbb{R}^{|\mathscr{S}|}$ is a γ– contraction:

$$||T_\pi \mathbf{v} - T_\pi \mathbf{v}'||_\infty =$$

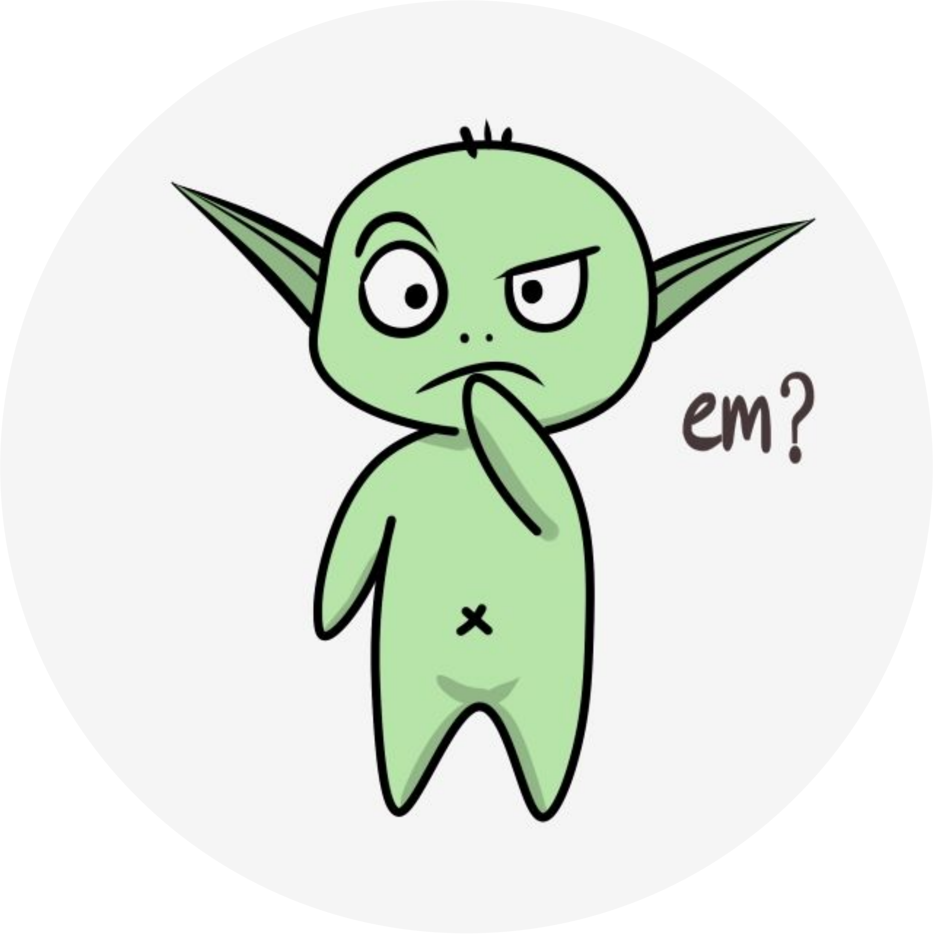
The Bellman Backup is a Contraction

The operator $T_\pi : \mathbb{R}^{|\mathscr{S}|} \rightarrow \mathbb{R}^{|\mathscr{S}|}$ is a γ– contraction:

$$
\begin{aligned}
||T_\pi \mathbf{v} - T_\pi \mathbf{v}'||_\infty &= ||(\mathbf{r}_\pi + \gamma \mathbf{P}_\pi \mathbf{v}) - (\mathbf{r}_\pi + \gamma \mathbf{P}_\pi \mathbf{v}')||_\infty \\
&= ||\gamma \mathbf{P}_\pi \mathbf{v} - \gamma \mathbf{P}_\pi \mathbf{v}'||_\infty \\
&= ||\gamma \mathbf{P}_\pi (\mathbf{v} - \mathbf{v}')||_\infty \\
&\leq \gamma ||\mathbf{v} - \mathbf{v}'||_\infty
\end{aligned}
$$

$\mathbf{P}_\pi$ is linear

Because $(\mathbf{P}_\pi \mathbf{v})(s)$ is a convex combination of elements from **v**, it must be that $||\mathbf{P}_\pi \mathbf{v}||_\infty \leq ||\mathbf{v}||_\infty$



https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

Chapter 5

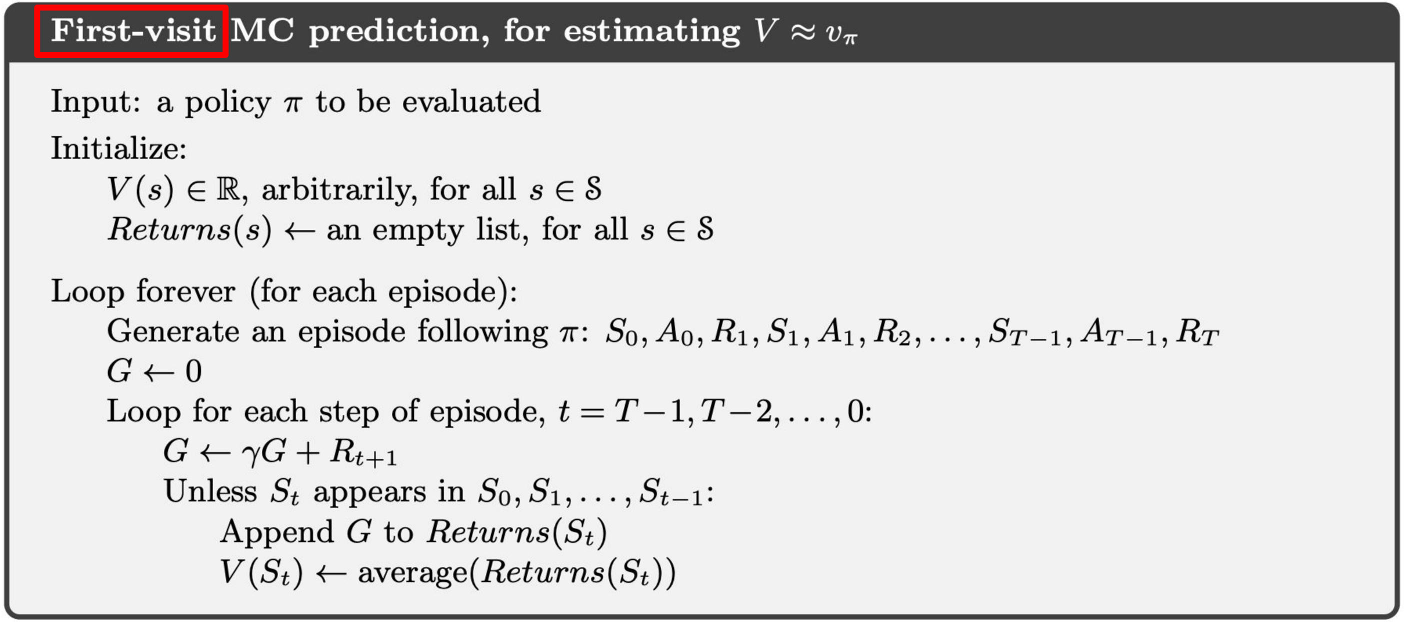
Monte Carlo Methods

Monte Carlo Methods – Why?

- This is our **first learning** method.
- We do not assume complete knowledge of the environment.
- “Monte Carlo methods **require only experience** — sample sequences of states, actions, and rewards from actual or simulated interaction with an environment.” 🤯
- It works! And different variations are used everywhere in the field (n-step returns, TD(λ), MCTS–AlphaGo/AlphaZero–, etc).
- … but we still need a model, albeit only a sample model.

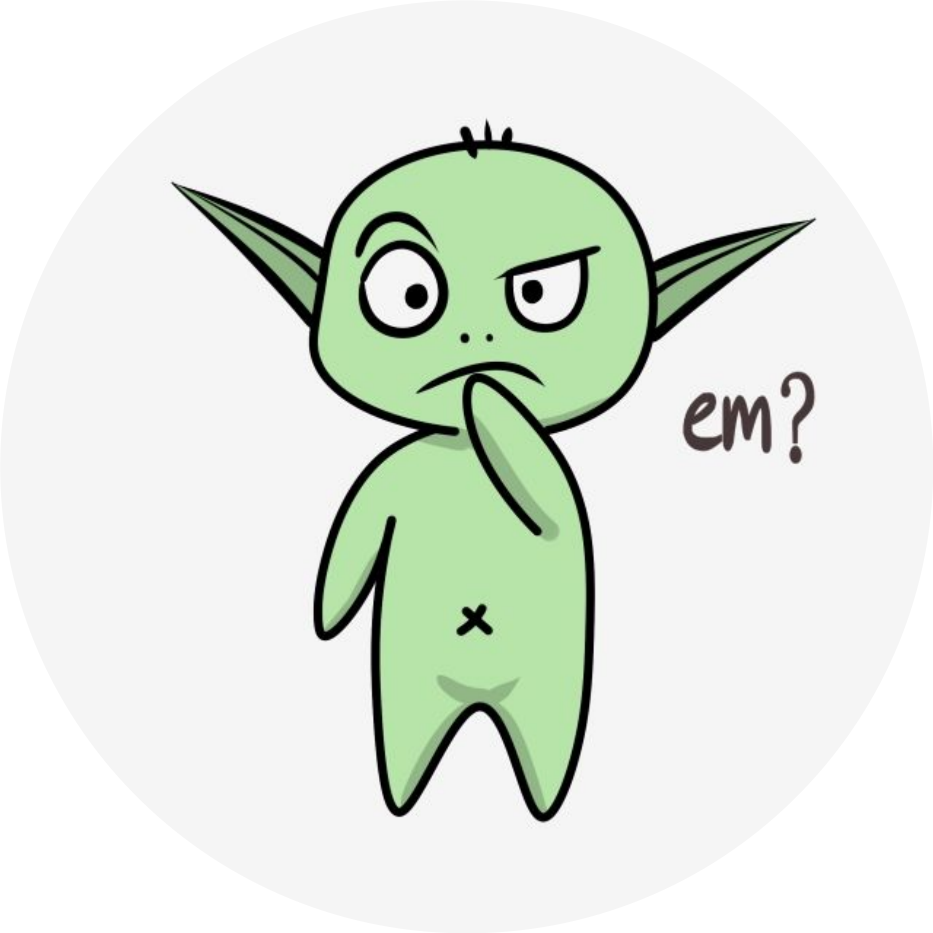
MC Methods are ways of solving the RL problem based on avg. sample returns (similar to bandits, but instead of rewards we are sampling returns).

Monte Carlo Prediction



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

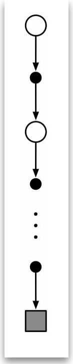
```
Input: a policy $\pi$ to be evaluated
Initialize:
    $V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$
    $Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

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 $Returns(S_t)$
            $V(S_t) \leftarrow \text{average}(Returns(S_t))$
```



https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

Some useful information / reminders about MC Methods

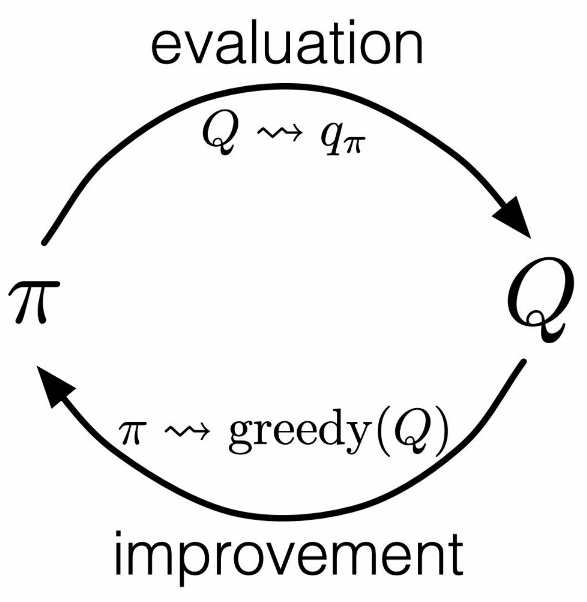
- Often it is much easier to get samples than to get the distribution of next events. Recall the Blackjack example in the textbook.
- Monte Carlo methods do not *bootstrap* (the estimate for one state does not build upon the estimate of any other state).
- First/every-visit MC converge to $v_\pi(s)$ as the number of visits to s goes to infinity. In first-visit MC, each return is i.i.d. and has finite variance ¯_(ツ)_/¯
- The computational cost of estimating the value of a single state is independent of the number of states.



Monte Carlo Estimation of Action Values

- If we don’t have access to a model, we need to estimate *action* values.
- Same as before, but now we visit state-action pairs ¯_(ツ)_/¯
  But to estimate q_* we need to estimate the value of *all* actions from each state.
  Solution? Exploration! … or exploring starts 😒

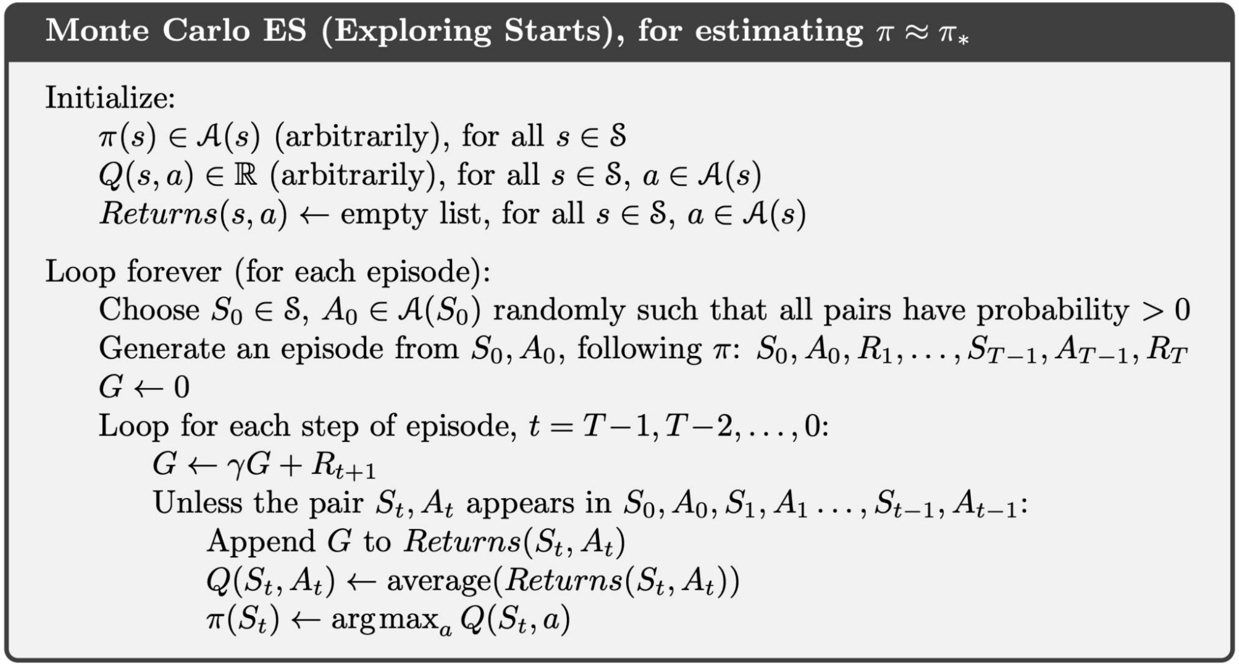
Monte Carlo Control


evaluation
$Q \rightsquigarrow q_\pi$
π
Q
$\pi \rightsquigarrow \text{greedy}(Q)$
improvement

$$\pi_0 \xrightarrow{\text{E}} q_{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} q_{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \cdots \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} q_*$$

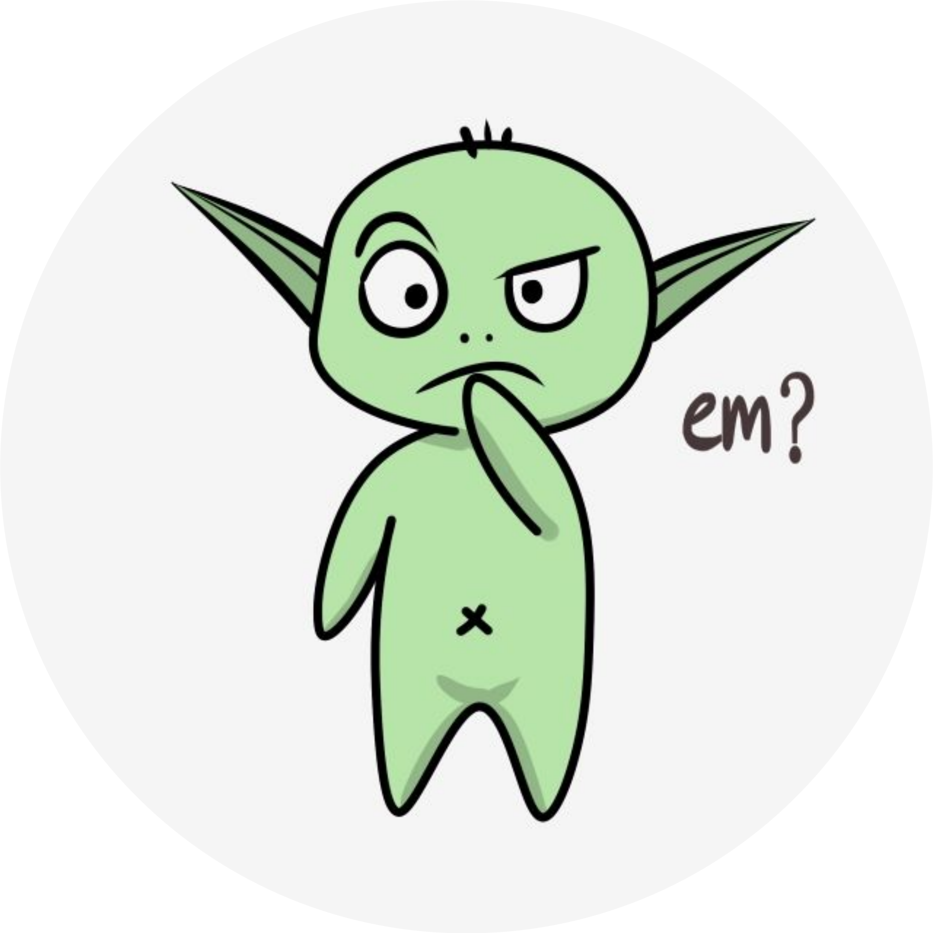
$$\pi(s) \doteq \arg\max_a q(s, a)$$

Monte Carlo ES



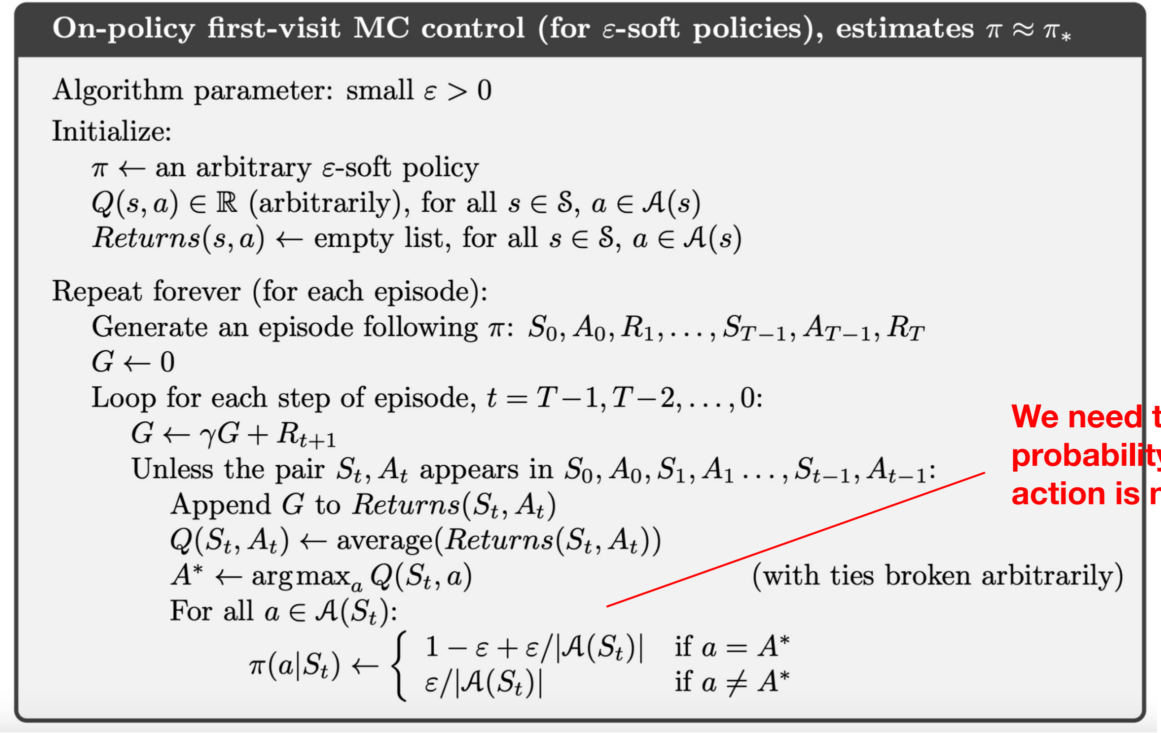
Monte Carlo ES (Exploring Starts), for estimating $\pi \approx \pi_*$

```
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 $S_0 \in \mathcal{S}$, $A_0 \in \mathcal{A}(S_0)$ randomly 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 $Returns(S_t, A_t)$
            $Q(S_t, A_t) \leftarrow$ average$(Returns(S_t, A_t))$
            $\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$
```



https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

MC Control without Exploring Starts



On-policy first-visit MC control (for ε-soft policies), estimates $\pi \approx \pi_*$

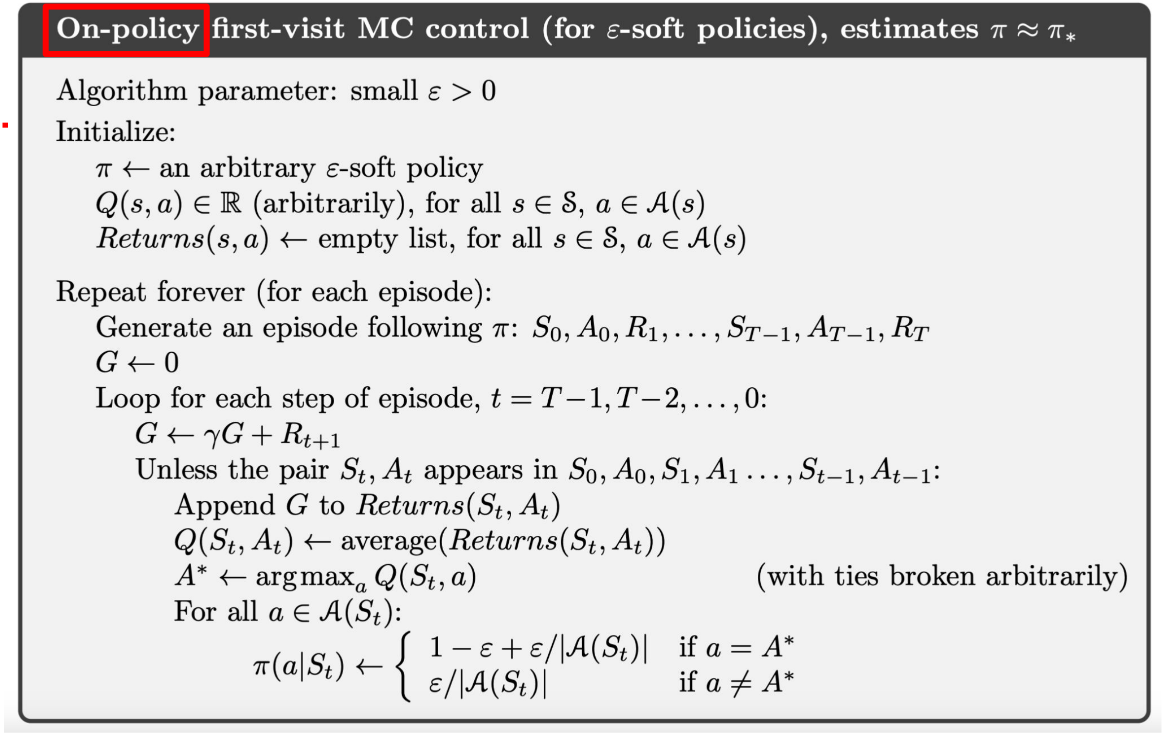
```
Algorithm parameter: small $\varepsilon > 0$
Initialize:
    $\pi \leftarrow$ an arbitrary $\varepsilon$-soft policy
    $Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$
    $Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Repeat forever (for each episode):
    Generate an episode 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 $Returns(S_t, A_t)$
            $Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$
            $A^* \leftarrow \arg\max_a Q(S_t, a)$                    (with ties broken arbitrarily)
            For all $a \in \mathcal{A}(S_t)$:
                $\pi(a|S_t) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(S_t)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(S_t)| & \text{if } a \neq A^* \end{cases}$
```

We need to ensure that the probability we select each action is not zero.

MC Control without Exploring Starts

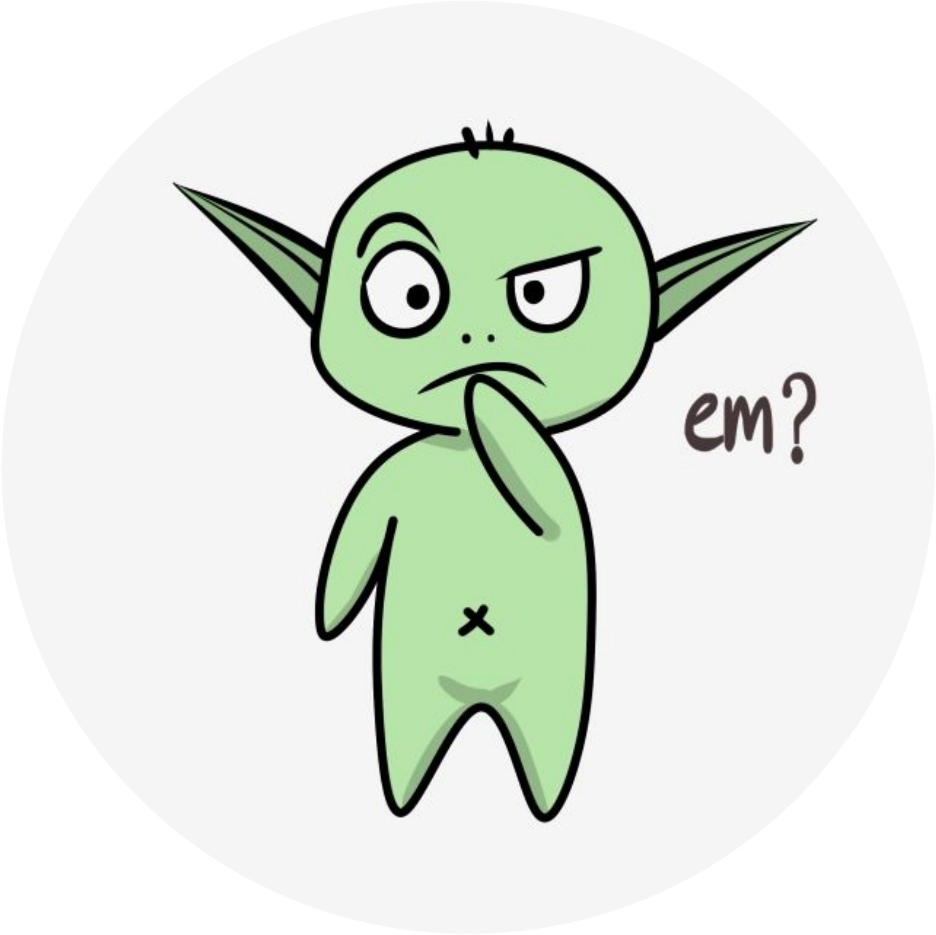
On-policy: You learn about the policy you used to make decisions.

Off-policy: You learn about a policy that is different from the one you used to make decisions.



On-policy first-visit MC control (for ε-soft policies), estimates $\pi \approx \pi_*$

```
Algorithm parameter: small $\varepsilon > 0$
Initialize:
    $\pi \leftarrow$ an arbitrary $\varepsilon$-soft policy
    $Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$
    $Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Repeat forever (for each episode):
    Generate an episode 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 $Returns(S_t, A_t)$
            $Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$
            $A^* \leftarrow \arg\max_a Q(S_t, a)$                (with ties broken arbitrarily)
            For all $a \in \mathcal{A}(S_t)$:
                $\pi(a|S_t) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(S_t)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(S_t)| & \text{if } a \neq A^* \end{cases}$
```

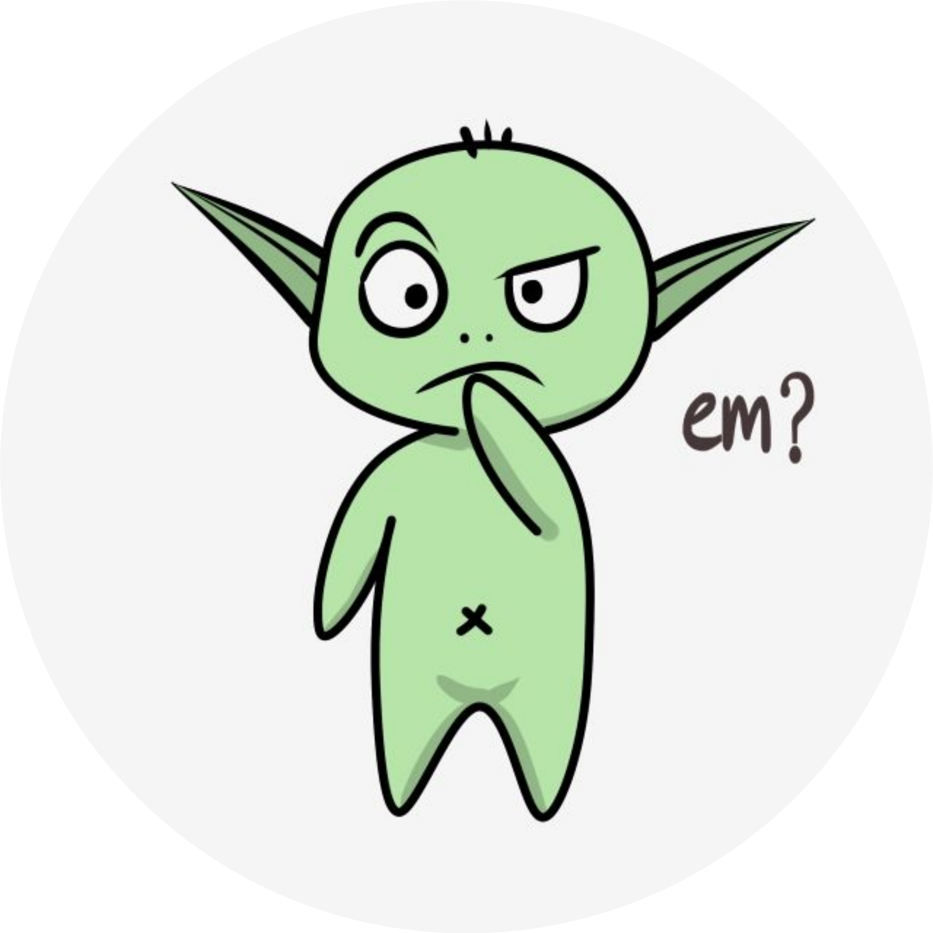


https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

Policy iteration works for ε-soft policies

Why an ε-greedy policy w.r.t. q_π is an improvement over any ε-soft policy π?

$$\begin{aligned}
q_\pi(s, \pi'(s)) &= \sum_a \pi'(a|s) q_\pi(s,a) \\
&= \frac{\varepsilon}{|\mathcal{A}(s)|} \sum_a q_\pi(s,a) \ + \ (1-\varepsilon) \max_a q_\pi(s,a) \\
&\geq \frac{\varepsilon}{|\mathcal{A}(s)|} \sum_a q_\pi(s,a) \ + \ (1-\varepsilon) \sum_a \frac{\pi(a|s) - \frac{\varepsilon}{|\mathcal{A}(s)|}}{1-\varepsilon} q_\pi(s,a) \\
&= \frac{\varepsilon}{|\mathcal{A}(s)|} \sum_a q_\pi(s,a) \ - \ \frac{\varepsilon}{|\mathcal{A}(s)|} \sum_a q_\pi(s,a) \ + \ \sum_a \pi(a|s) q_\pi(s,a) \\
&= v_\pi(s).
\end{aligned}$$



https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

Next class

- What **I** plan to do:
  - Finish overview of Monte Carlo Methods (Chapter 5 of the textbook)
  - Overview of Temporal-Difference Learning (Chapter 6 of the textbook)
  - Overview of n-step Bootstrapping (Chapter 7 of the textbook)

- What I recommend **YOU** to do for next class:
  - Read Chapters 6 and 7 of the textbook.
  - Submit Practice Quiz and Programming Assignment for Sample-based Learning Methods: TD Learning Methods for Prediction (Week 2).
  - Submit Practice Quiz and Programming Assignment for Sample-based Learning Methods: TD Learning Methods for Control (Week 3).