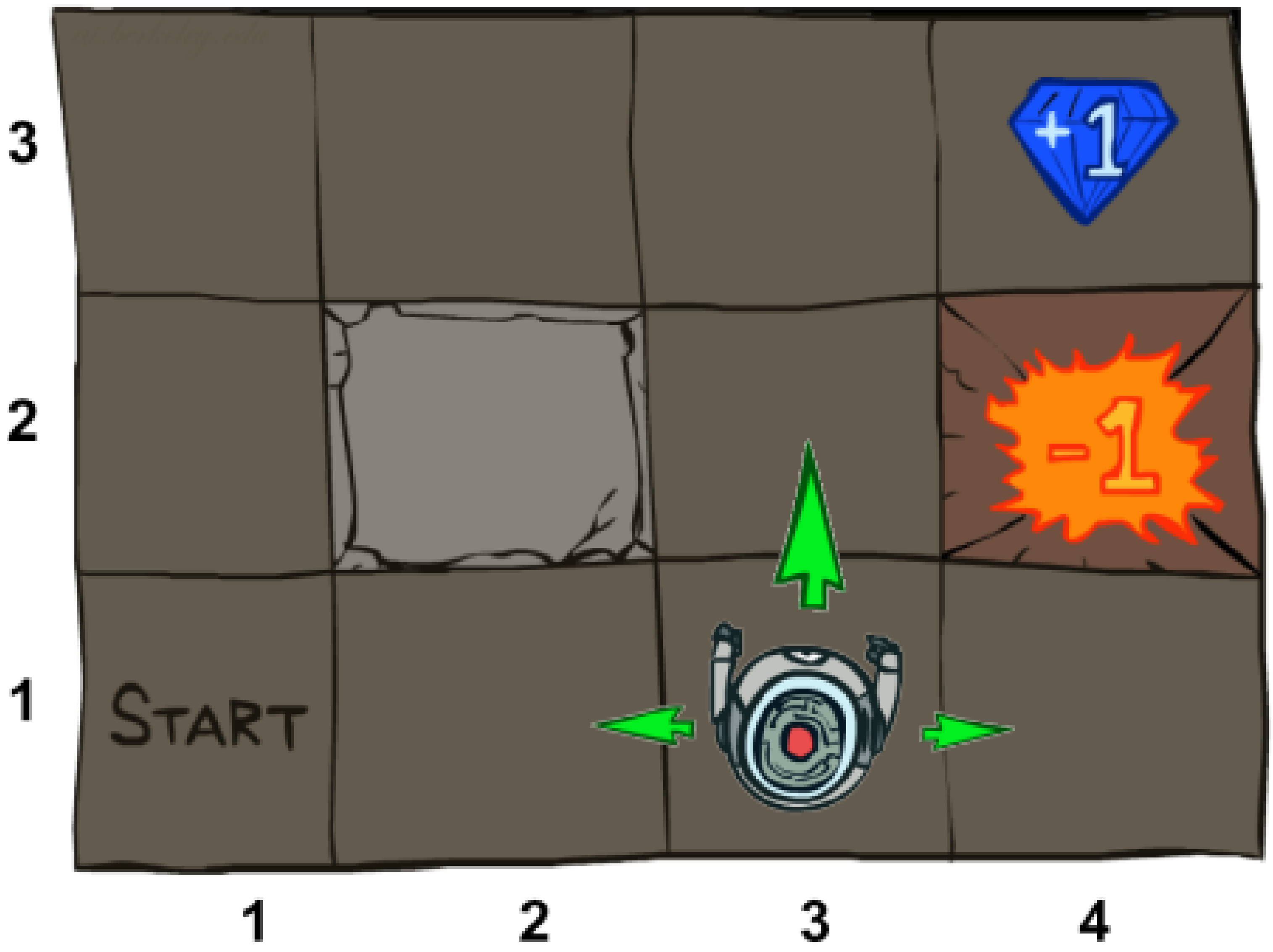
Markov Decision Processes



3
2
1
+1
-1
START
1
2
3
4

Grid world

- Agent lives in a grid, walls block the path
- Transition function $T(s, a, s')$ can be stochastic:
  - Agent chooses "North"
    - 80% chance move north
    - 10% chance move west
    - 10% chance move east
    - if there is a wall in the direction, agent does not move
- Rewards at each time step $R(\cdot)$
  - Typically:
    - Small **living reward** at each time step (often negative i.e. **living cost**)
    - Large rewards at terminal states

What do we think about this setting?

- Can be deterministic or stochastic, in two ways:
  - Stochastic transition function
    - *NOTE LB: I prefer to say that the transition function is stochastic, not the actions!*
  - Stochastic reward
- Can we solve it with our existing techniques?
  - If it is deterministic, the T and R known, we can perform planning!
  - If it is stochastic, and T and R known, we can do expectimax!
- The only difference here is that we get some rewards on the way (not very significant)

What will be new here?

- **Markov Decision Processes**
  - If T and R is known, we can find techniques that are much more efficient than expectimax to find a policy.
- **Reinforcement learning**
  - Even if we don't know the T or R, we can find techniques that can find a policy from the reward received after we perform an action.
  - Insight: there is still an MDP there, just now known

Markov decision process

- Set of **states** $s \in S$
- Set of **actions** $a \in A$
- **Transition function** $T(s, a, s') \in [0, 1]$
  - Basically: the probability that taking action a from s lands us in s', i.e. $P(s'|s, a)$ or $P(s_{t+1}|s_t, a_t)$
  - Sometimes called **world model**, or **system dynamics**
- **Reward** function $R(s, a, s')$
  - Sometimes $R(s, a)$ or $R(s)$ or $R(s')$
- **Start state** s_0
- (Sometimes) a set of **terminal states**

Markov?

- Russian mathematician Andrey Markov (1856-1922)
- Whenever we say that a system is Markovian it means something along the lines of **the future does not depend of the past given the present**
- If it would **not** be Markovian, the next state would depend on the past states and actions:

$$P(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, s_{t-2}, a_{t-2} \dots s_0)$$

- As it is Markovian, the state is simply:

$$P(s_{t+1}|s_t, a_t)$$

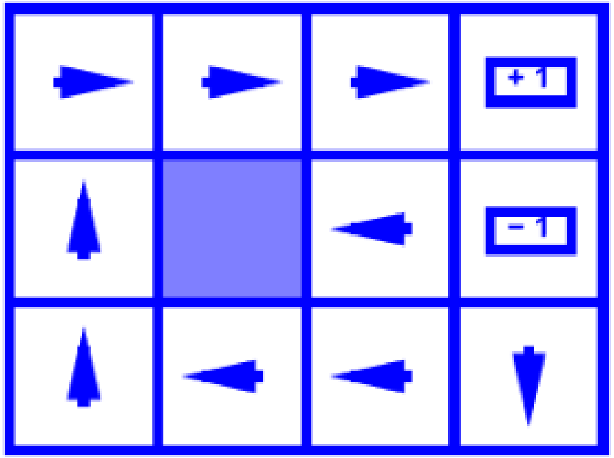
- What this means in practice is that s_t contains all the relevant information about the past.

Policies

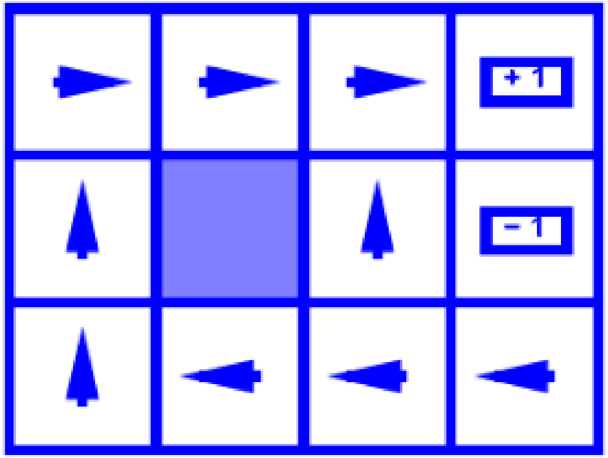
- A **policy** is a function $\pi : S \rightarrow A$, with $\pi(s) \rightarrow a$
  - Optimal policy π^* is one that, if followed, will give you maximum expected utility
- Didn't we calculate policies before?
  - We mention them, but we did not explicitly calculate them.
  - **Planning**: we returned a proposed set of actions $a_1, \ldots a_n$
    - If one of the actions landed you in the wrong state, the plan fails!
  - **Game play**: we created a procedure to calculate the specific value of the policy for one state $\pi(s_{current})$. We won't know the full π. When we land in a new state, we have to run expectimax again.

Optimal policies in an MDP

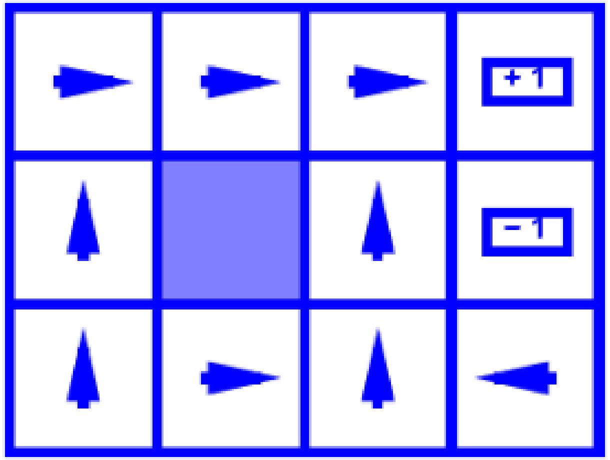
- Solving an MDP means finding an **optimal policy** which maximizes **expected utility** when followed
  - Expected? Transition function is random, so you cannot be sure, only in expectation.
- An explicit policy defines a **reflect agent**
  - You can compute the whole policy, and then during execution time $\pi(s) \to a$ is just a lookup table
  - This is not what we did with minmax, expectimax etc. - whenever you reached a state, you started the calculation from there...



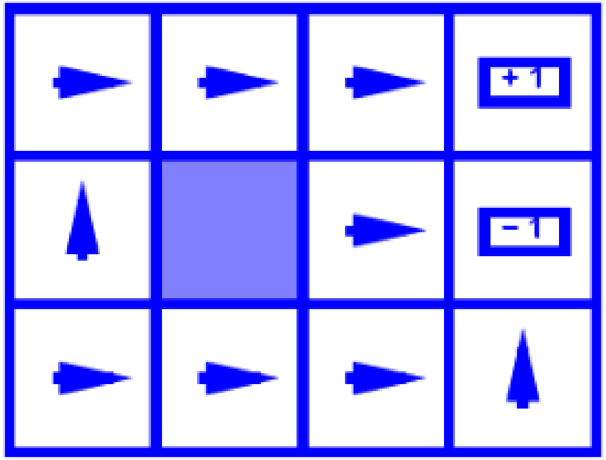
R(s) = -0.01



R(s) = -0.03



R(s) = -0.4



R(s) = -2.0

Utilities of sequences

- In games, we usually have to express preferences over final outcomes
  - We had shown that we can assign a utility value to the final outcome to express our preferences.
- Here we have **rewards** that are given after every action
  - We need to discuss over preferences over **sequences of rewards**
  - Not quite that simple as add them up!

Sequences of rewards

- What do we prefer?
  - [1, 2, 3, 4] or [4, 3, 2, 1] ?
  - [1, 0, 0, 0] or [0, 0, 0, 1] ?
  - [1] or [0, 0, 0, 0, 1]?
  - [9] or [0, 0, 0, ..., 0, 10]
- Definition: a preference is **stationary** iff

$$[a_1, a_2, \ldots] \succ [b_1, b_2, \ldots] \Leftrightarrow [r, a_1, a_2, \ldots] \succ [r, b_1, b_2, \ldots]$$

- We quite often want our preferences to be stationary.

Discounting

- We often want to express the fact that we prefer rewards earlier.
- To achieve this, we **discount** rewards as they are further in time.
- If we want our utilities to be both **discounted** and **stationary** there is only one way to define them:

$$U([r_0, r_1, \ldots]) = r_0 + \gamma r_1 + \gamma^2 r_2 +$$

with $\gamma \leq 1$. If $\gamma = 1$, we have **additive** utilities.

Infinite utilities

What if the game lasts forever? Additive utility can lead to infinite utility, is this ok?

Some solutions:

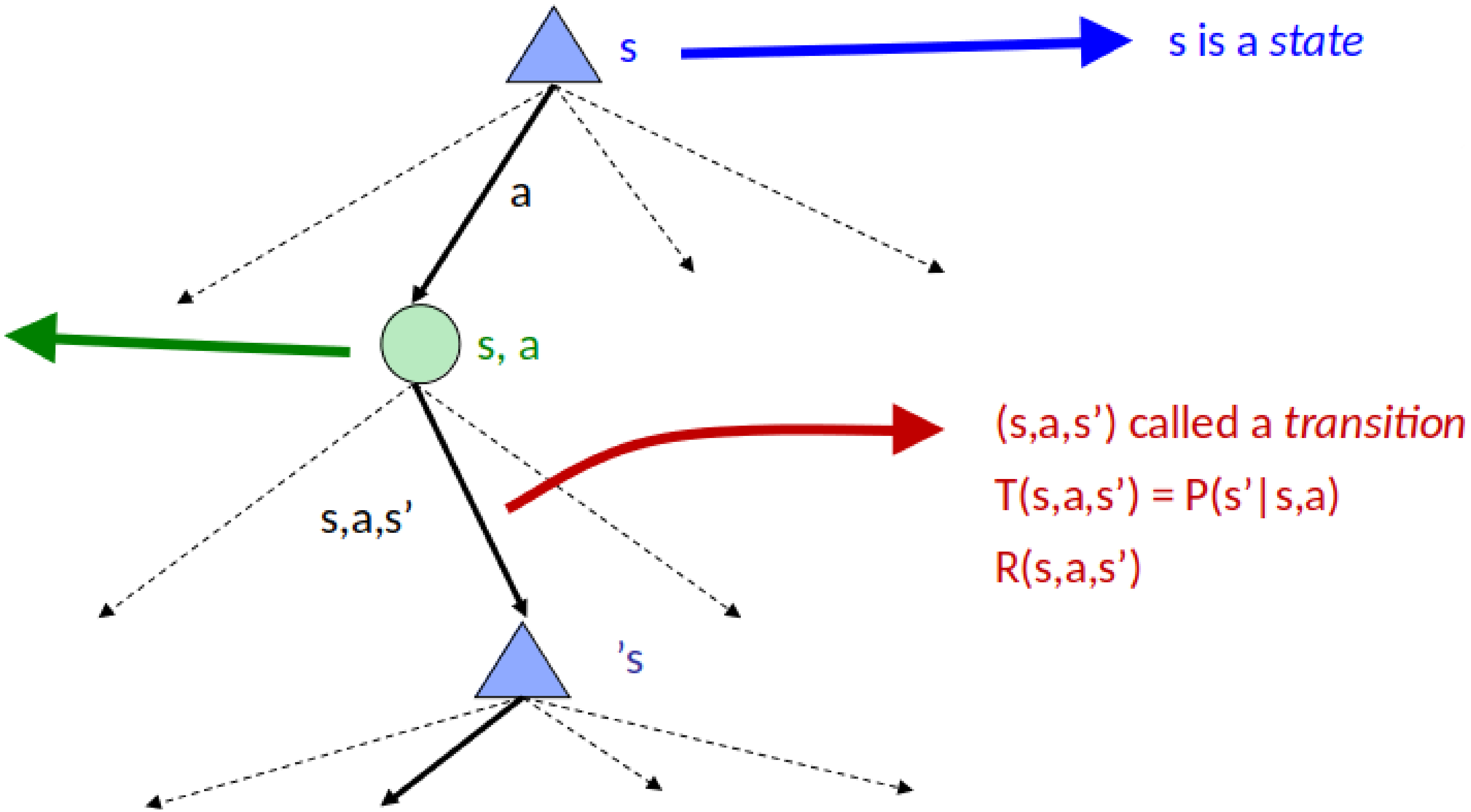
- Absorbing state: set up the game such that for every policy a terminal state will be eventually reached
- Finite horizon search: terminate episodes after fixed T steps
  - Will give non-stationary policies: π will depend on the time left
- Discounting with $\gamma < 1$

$$U([r_0, \ldots r_\infty]) = \sum_{t=0}^{\infty} \gamma^t r_t \leq \frac{r_{max}}{1-\gamma}$$

Utility quiz here

Quantities

- state s
- q-state s, a - describes the state of affairs after we committed to an action, but not yet performed it
- $V^*(s)$ **expected utility** if we **started from** s and **performed optimally** in the future
- $Q^*(s, a)$ **expected utility** if we **started from q-state** s, a (that is, we committed to an a) and **performed optimally** in the future
- $\pi^* s$ optimal policy - the optimal action from state s


s
s is a *state*
a
s, a
(s,a,s') called a *transition*
T(s,a,s') = P(s'|s,a)
R(s,a,s')
s,a,s'
's

Relationships between the values

- This looks very much like an expectimax tree:
    - Take the maximum in the states
    - Take the expectation in the q-state

The Bellman equations

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

$$Q^*(s,a) = \sum_{s'} T(s,a,s') \left(R(s,a,s') + \gamma V^*(s')\right)$$

$$V^*(s) = \max_a \sum_{s'} T(s,a,s') \left(R(s,a,s') + \gamma V^*(s')\right)$$

Can we just solve this?

$$V^*(s) = \max_a \sum_{s'} T(s,a,s') \left(R(s,a,s') + \gamma V^*(s')\right)$$

- n states, n equations, can we just solve this?
- unfortunately, this is not a linear system of equations: the problem is the **max**
- We need a different idea.

Time limited values

- $V_k(s)$ the optimal value if we start from s and follow the optimal strategy for k steps.
- This is what depth-k expectimax would give.

Value iteration

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left(R(s,a,s') + \gamma V_k(s') \right)$$

- Let us call this a "Bellman update"
- Repeat until convergence
  - We can prove that it does converge to the optimal values
- Complexity of each iteration $O(S^2 A)$
  - It is not that simple to find out how many iterations we need until no change
- It is a way to solve the Bellman equation with a fixed-point technique

Convergence of value iteration

- If the tree is actually limited in depth eg M, then V_M is the final value.
- If the tree is infinite, $\gamma < 1$, and max reward is R_{max}
  - V_{k+1} and V_k is at most R_{max} in the last step, discounted with γ^k
  - The difference $\gamma^k R_{max} \to 0$ when $k \to \infty$
  - So the values will converge
- However:
  - The max of the state rarely changes!
  - Very often the policy converges before the values!

Policy extraction from the V values

- We usually don't care that much about the $V^*(s)$ value.
- We want to act in the world, we need the policy $\pi^*(s)$.
- Not quite that simple! We need to do one step of expectimax:

$$\pi^*(s) = \operatorname*{argmax}_a \sum_{s'} T(s, a, s') \left(R(s, a, s') + \gamma V^*(s')\right)$$

Policy extraction from the Q values

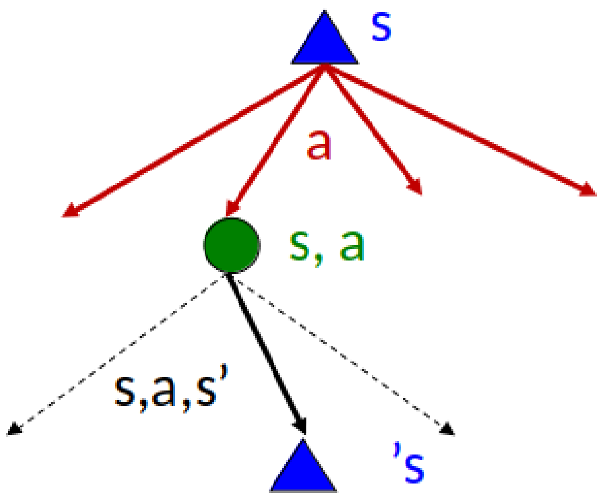
$$\pi^*(s) = \underset{a}{\operatorname{argmax}}\, Q^*(s, a)$$

- Actions are easier to select from q-values

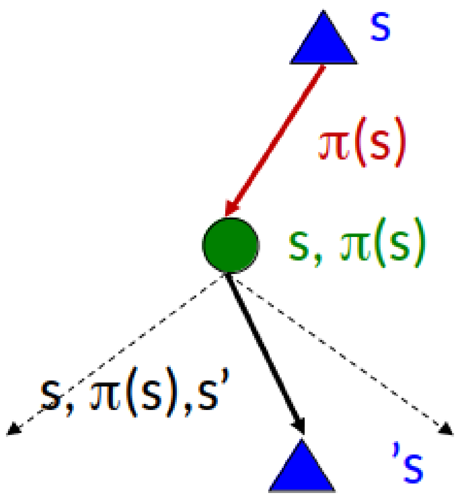
Policy evalution

- We are given a fixed policy π, not necessarily optimal.
- We want to find the associated V^π and Q^π values.
  - Defined as what if I follow this policy from now on, etc.
- Policy evaluation is easier than finding V^* and Q^* because we don't have the max - just do what the policy tells you to do!

Do the optimal action



Do what π says to do



Policy evaluation

- The Bellman equation for a fixed policy.

$$V^\pi = \sum_{s'} T(s, \pi(s), s') \left(R(s, \pi(s), s') + \gamma V^\pi(s')\right)$$

- Recursive, one step look ahead.
- Can we just solve it?
  - This time, yes! The max went away!
  - It is n equations, n variables, linear in the unknowns which are the $V^\pi(s)$ values.
  - Pick your favorite linear solver

Policy evaluation, solved iteratively

- We can also do the same trick as in value iteration:

$$V_0^\pi(s) = 0$$

$$V_{k+1}^\pi(s) \leftarrow \sum_{s'} T(s, \pi(s), s') \left(R(s, \pi(s), s') + \gamma V_k^\pi(s')\right)$$

- It converges, for the same reason as value iteration
- Efficiency $O(s^2)$ per iteration

Policy iteration

- Start with a random policy π_0
- Until no change in policy repeat
  - Evaluate π_k to values V^{π_k}
  - Create a new policy π_{k+1} using one-step look-ahead with V^{π_k} as future values

$$\pi^{k+1}(s) = \operatorname*{argmax}_a \sum_{s'} T(s, a, s') \left(R(s, a, s') + \gamma V^{\pi_k}(s') \right)$$

- It is still converges, and to the optimal policy
- Very often, it converges much faster

Comparing policy iteration with value iteration

- Value iteration
  - Every iteration updates the value and implicitly, the policy $O(S^2A)$
  - We don't track the policy explicitly, only extract it once at the end
- Policy iteration
  - We do several passes that update utilities with fixed policy $O(S^2)$
  - After the policy is evaluated, choose a new one $O(S^2A)$

Some of the things we did

- **Policy extraction**
  $V^* \Rightarrow \pi^*$
  $Q^* \Rightarrow \pi^*$
- **Policy evaluation**
  $\pi \Rightarrow V^\pi$
  $\pi \Rightarrow Q^\pi$
- **Value iteration**
  MDP $\Rightarrow V^*$ (by doing $V_0, V_1 \ldots$)
- **Policy iteration**
  MDP $\Rightarrow V^*$ (by doing $\pi_0, V^{\pi_0}, \pi_1, V^{\pi_1} \ldots$)