

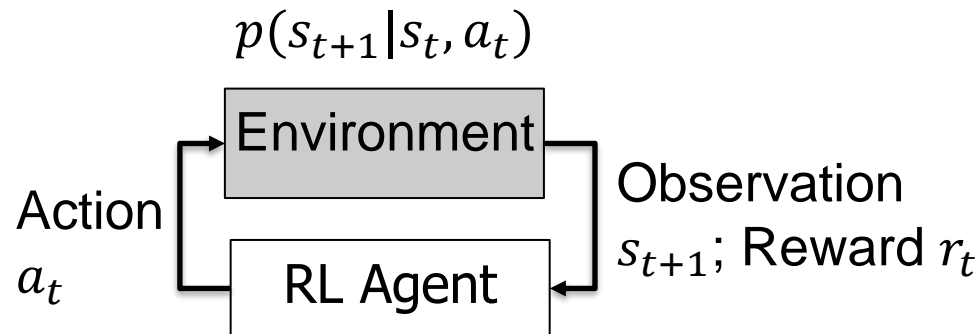
# L7.1 MDP Planning

Zonghua Gu 2020



# Reinforcement Learning (RL)

- The RL problem: find the optimal policy given the reward function.
  - Sometimes loss function is used, defined as negative of reward function, so minimizing loss is equivalent to maximizing reward.

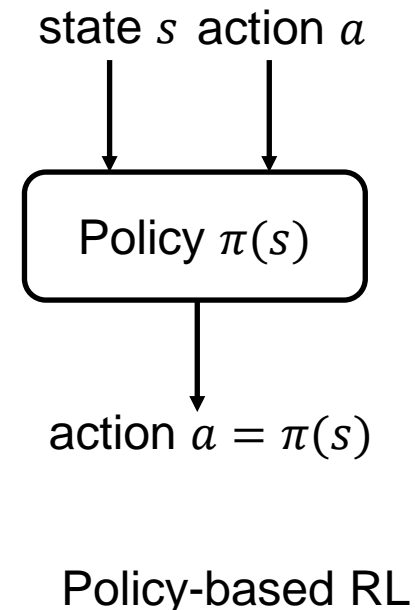
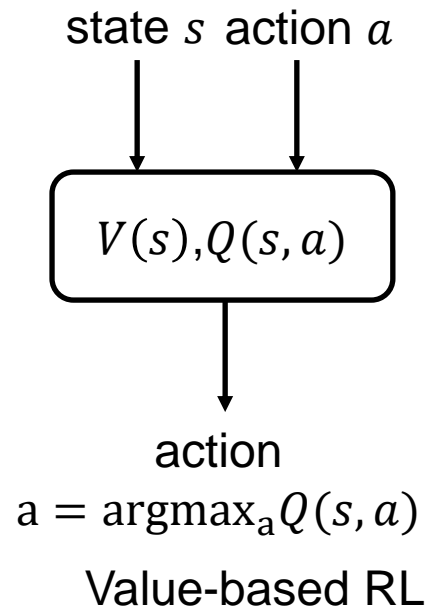
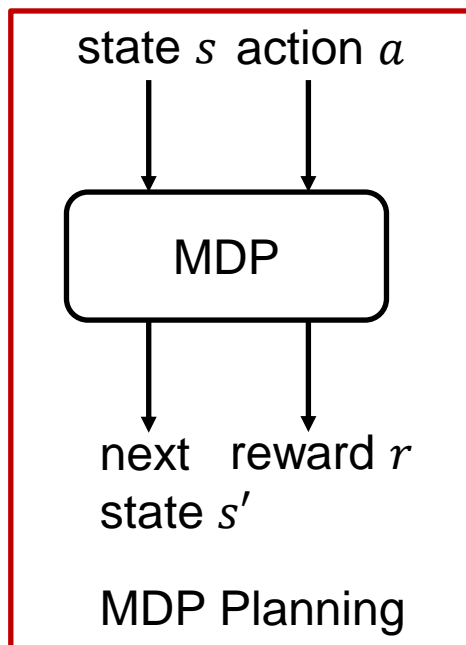


Goal: learn optimal policy  $\pi_\theta(a_t|s_t)$  that maximizes cumulative reward

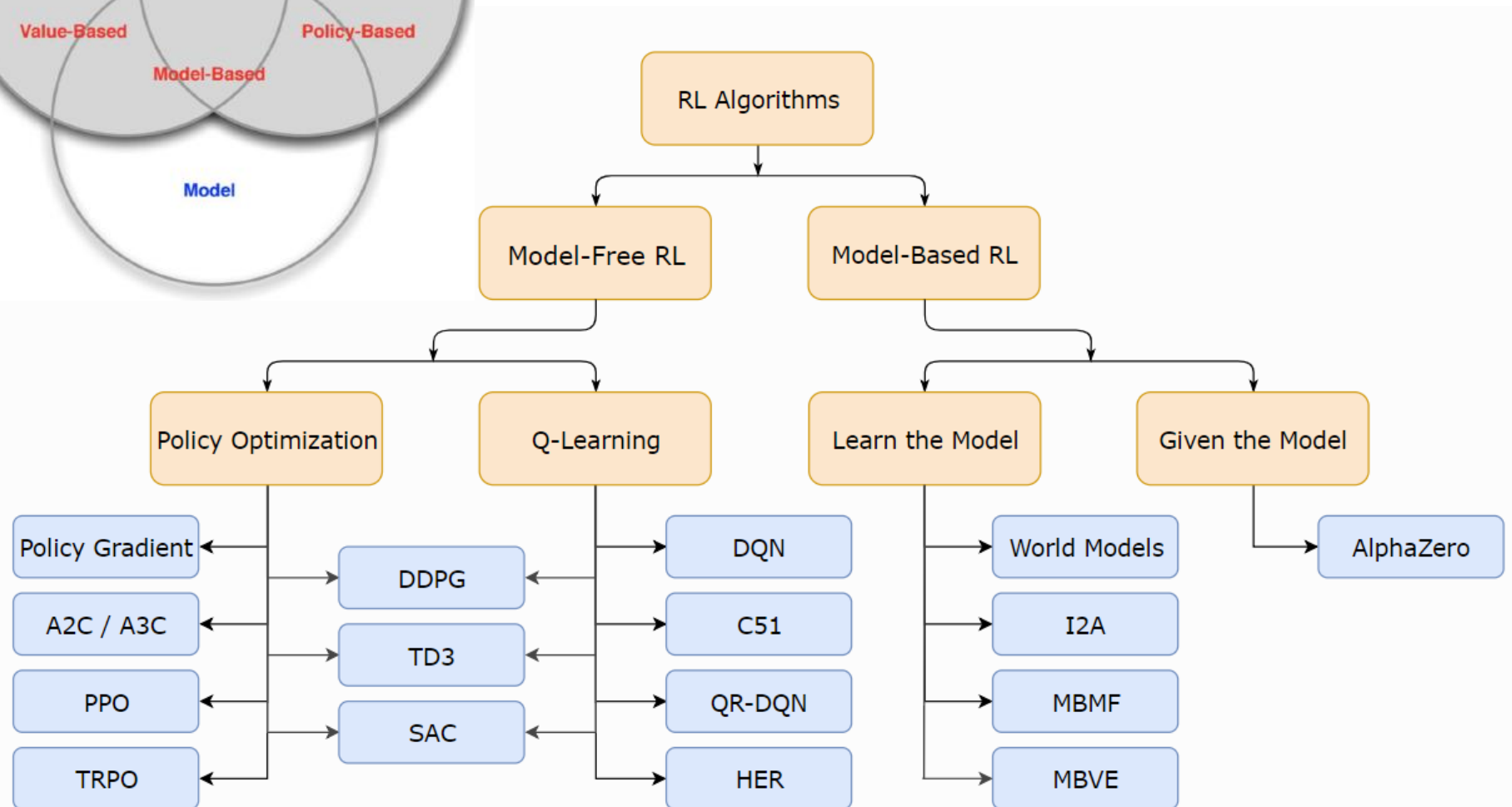
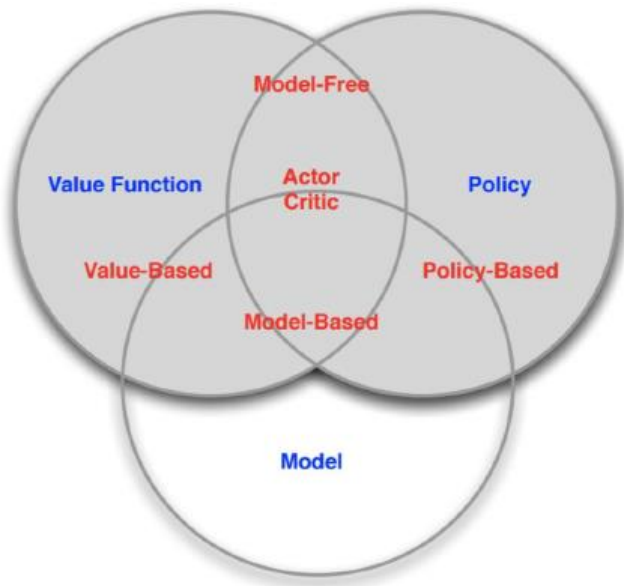
$$\max_{\pi} E(\sum_{t=0}^T \gamma^t r(s_t, a_t))$$

# Model-Based vs. Model-Free

- Model-Based RL: MDP planning
  - Learn MDP  $p(r, s' | s, a)$  (given current state  $s$  and action  $a$ , returns prob distribution of current reward  $r$  and next state  $s'$ ), then plan with Value Iteration or Policy Iteration
- Model-Free RL: Value-based and Policy-based
  - Learn value function  $V(s)$  or  $Q(s, a)$ , or policy function  $\pi(s)$  without learning MDP



# Overview of RL Algorithms

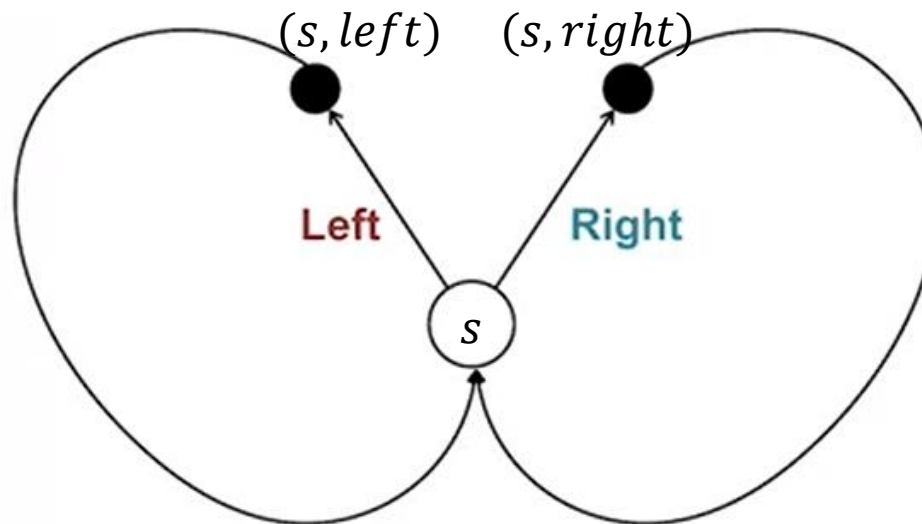


# Markov Decision Process (MDP)

- An MDP consists of:
  - Set of states  $S$
  - Start state  $s_0$
  - Set of actions  $A$
  - Transitions and rewards  $p(r, s' | s, a)$  (w. discount  $\gamma$ )
- Policy maps from states to actions:
  - Deterministic policy  $a = \pi(s)$  defines a deterministic action  $a$  for state  $s$ .
  - Stochastic policy  $\pi(a | s)$  defines a probability distribution over possible actions  $a$  for state  $s$ .
- Markov means that next state only depends on current state
  - $P(S_{t+1} = s' | S_t = s_t, A_t = a_t, S_{t-1} = s_{t-1}, A_{t-1} = a_{t-1}, \dots, S_0 = s_0, A_0 = a_0)$
  - $= P(S_{t+1} = s' | S_t = s_t, A_t = a_t)$
  - Given the present state, the future and the past are independent
  - e.g., for driving task, current vehicle position  $x$  as the state does not satisfy the Markov property, since the next state depends on not only  $x$ , but also velocity  $\dot{x}$ , acceleration  $\ddot{x}$ . If we redefine the state as vector  $[x, \dot{x}, \ddot{x}]^T$ , then it satisfies the Markov property.
  - Or, current snapshot of front camera view can be used as the state (e.g., NVIDIA's PilotNet), but some works use past  $N$  video frames as the state to capture more dynamics (e.g., Waymo's ChauffeurNet).

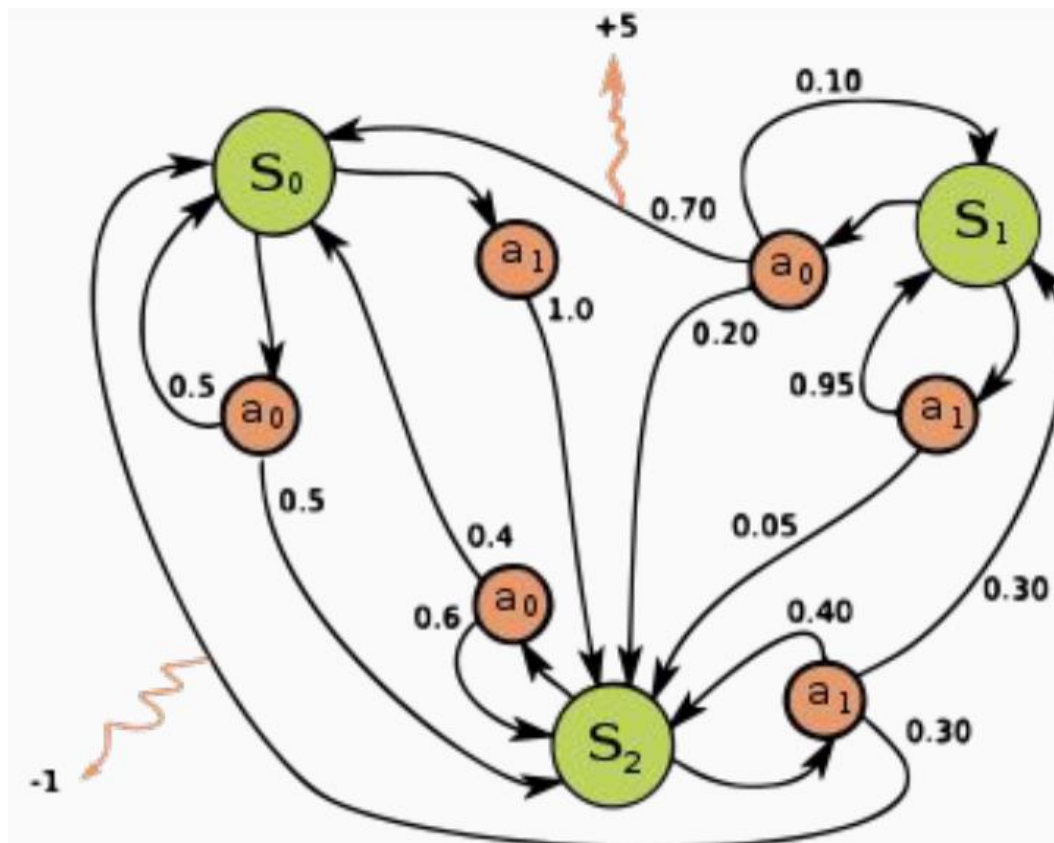
# MDP Quiz

- For this MDP with a single state  $s$  and two possible actions  $left$  and  $right$ . Are these valid policies?
  - 1)  $\pi(left|s) = \pi(right|s) = 0.5$  (goes left or right with equal probability. uniform random policy)
  - 2)  $\pi(left|s) = 1.0, \pi(right|s) = 0$  (always goes left)
  - 3) Alternating left and right, i.e., if previous action is left, then current action must be right, next action must be left, and so on.
  - ANS: 3) is not a valid policy, since it depends on the history of actions. To be a valid policy, the action must depend on the current state only.
- We can redefine the MDP' extended state to include the last action as part of it, then 3) is a valid policy for the new MDP.



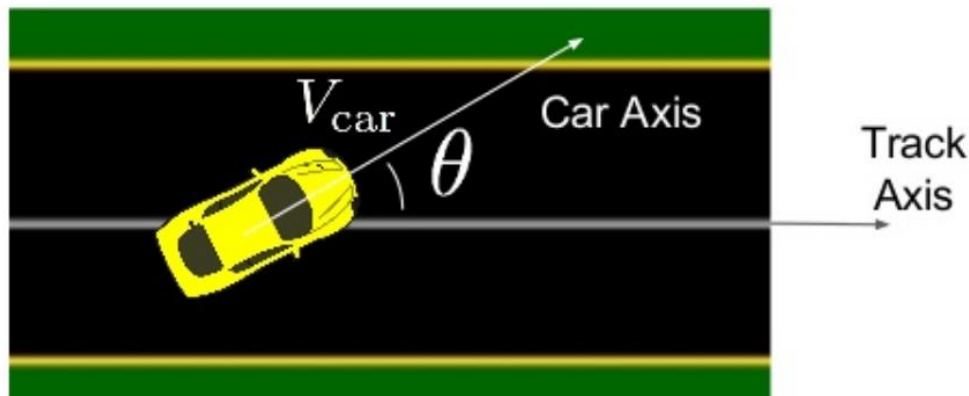
# An Example MDP

- Green nodes denote 3 states  $s_0, s_1, s_2$ ; Red nodes denote 2 possible actions  $a_0, a_1$  in each state. Each red node can also be denoted as  $(s, a)$ .
- Agent taking action  $a$  in state  $s$  may get different reward  $r$  and next state  $s'$ , denoted as state transition  $(s, a, r, s')$ , due to environment uncertainty (all rewards are 0 expect +5 and  $-1$  show in fig).
- State transition  $(s, a, r, s')$  is atomic. You can remove all red nodes and draw a single transition between green nodes.



# RL Reward Function

- For the vehicle in left fig:
  - state: Pose of ego-car ( $x, y, \theta$ ) and environment map; action: Steering wheel/brake/acceleration
- Possible reward function:  $R_t = w_1 V_{car} \cos \theta - w_2 |cte|$ 
  - Weight sum to maximum longitudinal velocity (first term), and minimize cross-track error (distance to lane center)
  - This is an example of dense reward (e.g., at every time step), as opposed to sparse reward (e.g., only at the end of each episode)
- Compare with twiddle() :
  - twiddle() can be viewed as an RL algorithm (policy gradient), that learns PID parameters with sparse reward (cost function is cumulative cross-track error (cte), computed at the end of each simulation episode, as sum of squares of ctes for N timesteps. But it is very crude:
  - It does not use the numeric value of cte, only its relative size (if  $err < best\_err$ );
  - Cost function does not include heading angle  $\theta$ ;
  - if the track is very long and irregular, then we can make the reward denser, to adjust PID parameters every K timesteps instead of at the end of each episode.



```
if err < best_err:
    best_err = err
    dp[i] *= 1.1
else:
    p[i] -= 2 * dp[i]
    robot = make_robot()
    x_trajectory, y_trajectory, err = run(robot, p)

    if err < best_err:
        best_err = err
        dp[i] *= 1.1
    else:
        p[i] += dp[i]
        dp[i] *= 0.9
```

twiddle()



# Amazon DeepRacer

- Amazon Web Services (AWS) launched DeepRacer in 2018 for training AD algorithms with RL
  - <https://aws.amazon.com/deepracer/>
- You can train RL algorithm in the simulator on AWS cloud, but it costs money after some free time.
- They hold competitions, both online and in real-world. 1/10<sup>th</sup> scale race car costs USD \$349.



# Params for Writing Reward Function

all\_wheels\_on\_track

x

y

distance\_from\_center

is\_left\_of\_center

is\_reversed

heading

progress

steps

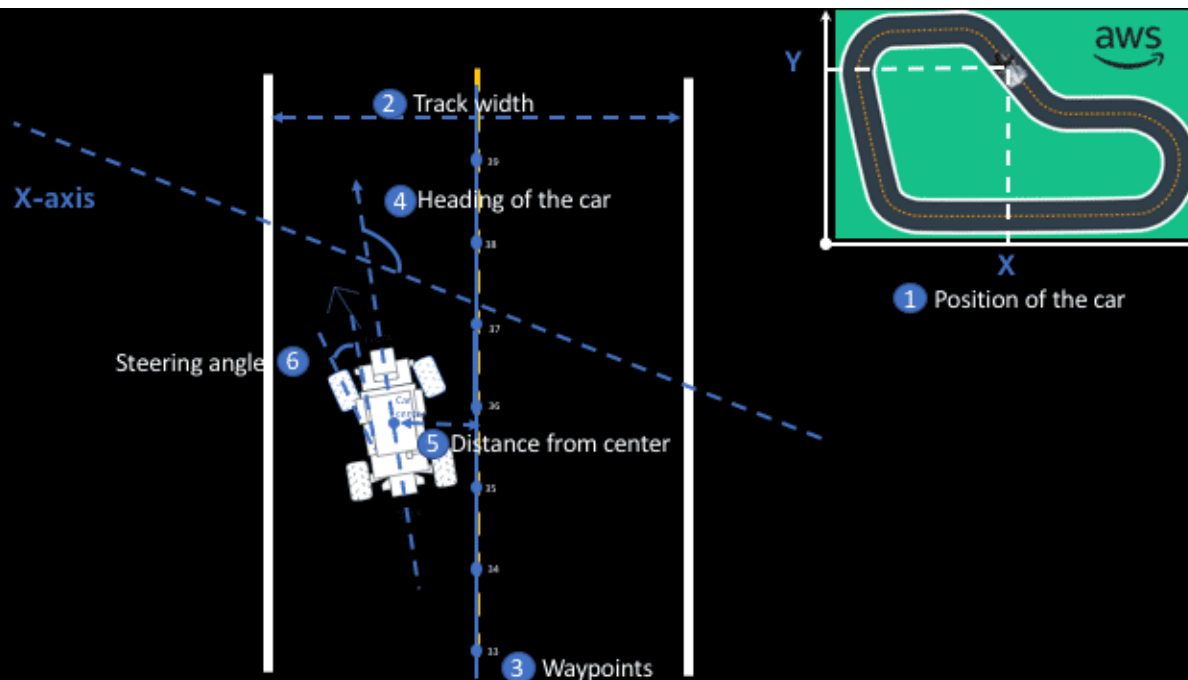
speed

steering\_angle

track\_width

waypoints

closest\_waypoints

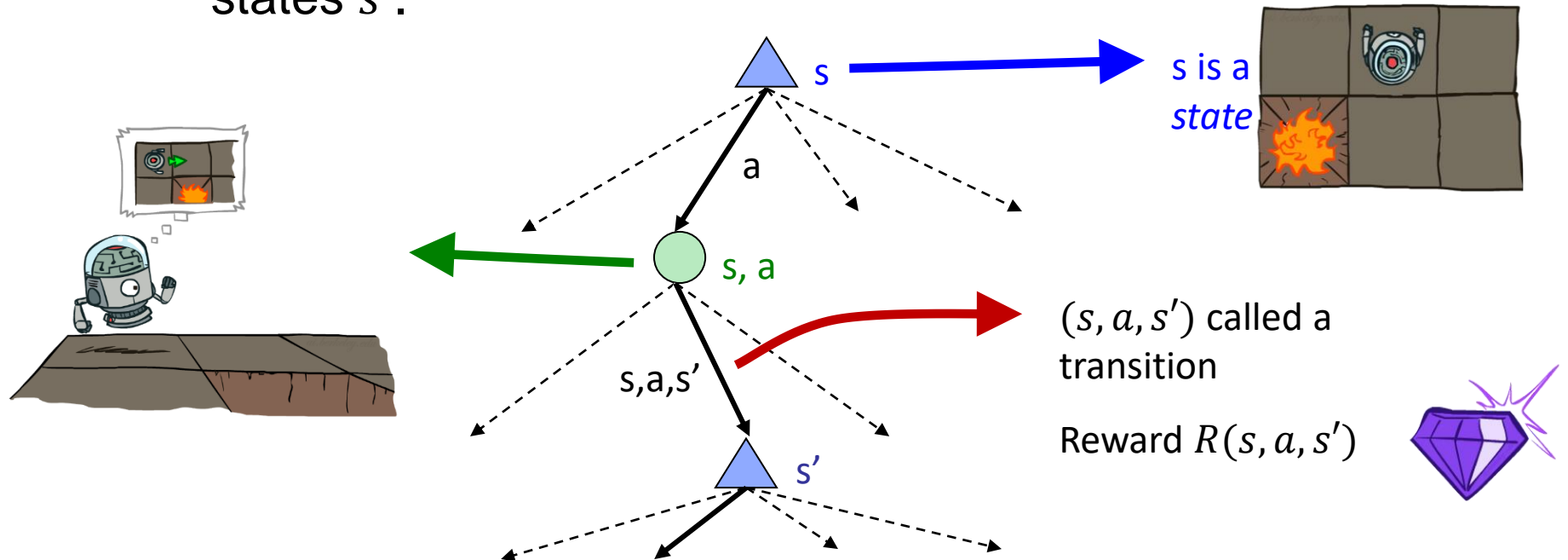


# Example Reward Function

- ```
def reward_function(params):  
    """Example of penalize steering, which helps mitigate zig-zag behaviors"""  
    # Read input parameters  
    distance_from_center = params['distance_from_center']  
    track_width = params['track_width']  
    steering = abs(params['steering_angle']) # Only need the absolute steering angle  
  
    # Calculate 3 markers that are at varying distances away from the center line  
    marker_1 = 0.1 * track_width  
    marker_2 = 0.25 * track_width  
    marker_3 = 0.5 * track_width  
  
    # Give higher reward if the agent is closer to center line and vice versa  
    if distance_from_center <= marker_1:  
        reward = 1  
    elif distance_from_center <= marker_2:  
        reward = 0.5  
    elif distance_from_center <= marker_3:  
        reward = 0.1  
    else:  
        reward = 1e-3 # likely crashed/ close to off track  
  
    # Steering penalty threshold, change the number based on your action space setting  
    ABS_STEERING_THRESHOLD = 15  
  
    # Penalize reward if the agent is steering too much  
    if steering > ABS_STEERING_THRESHOLD:  
        reward *= 0.8  
  
    return float(reward)
```
- A more realistic and complex reward function: <https://www.middleware-solutions.fr/2019/08/14/an-introduction-to-aws-deepracer>

# MDP Search Tree

- Each MDP state  $s$  projects a search tree starting from it.
- In general, both policy and environment may be stochastic
  - Policy  $\pi(a|s)$ : probability distribution over possible actions  $a$  from state  $s$ .
  - Environment  $p(r, s'|s, a)$ : if agent takes action  $a$  in state  $s$ , env gives probability distribution over reward  $r$  and next states  $s'$ .



# Preventing Infinite Rewards

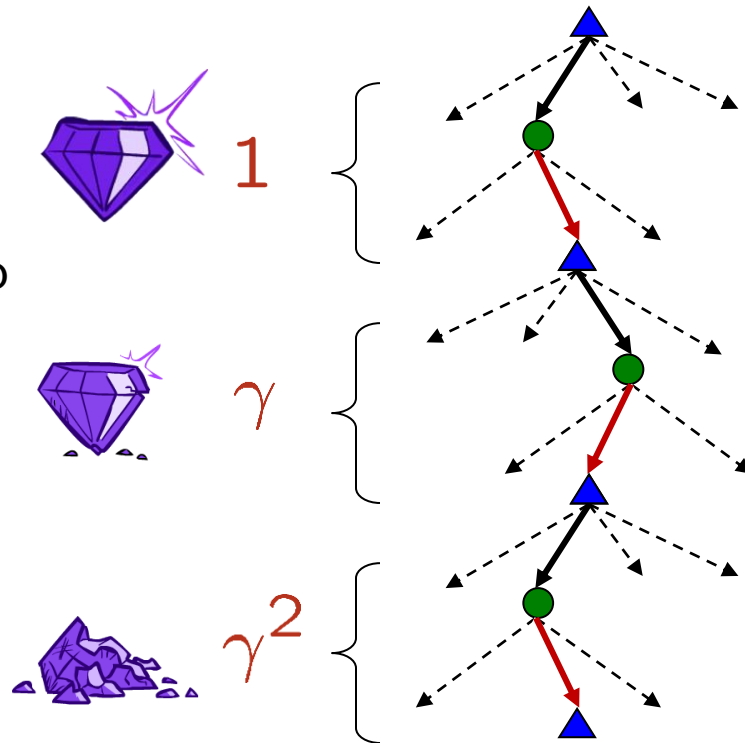
- Problem: What if the game lasts forever? Do we get infinite rewards? No. Possible solutions:
- Finite horizon: (limit search tree depth)
  - Terminate episodes after a fixed  $T$  timesteps (e.g. life)
  - Gives nonstationary policies ( $\pi$  depends on time left)
    - If you know you are going to die next month, you are likely to take very different actions today than if you will die 50 years later.
- Discounting: use  $0 \leq \gamma \leq 1$ 
  - $U([r_0, \dots, r_\infty]) = \sum_{t=0}^{\infty} \gamma^t r_t \leq \sum_{t=0}^{\infty} \gamma^t r_{max} = \frac{r_{max}}{1-\gamma}$
  - Smaller  $\gamma$  means smaller “horizon” – shorter term focus
- Absorbing state: guarantee that for every policy, a terminal state will eventually be reached

# Discounting

- How to discount?
  - Each time we descend a level, we multiply in the discount once

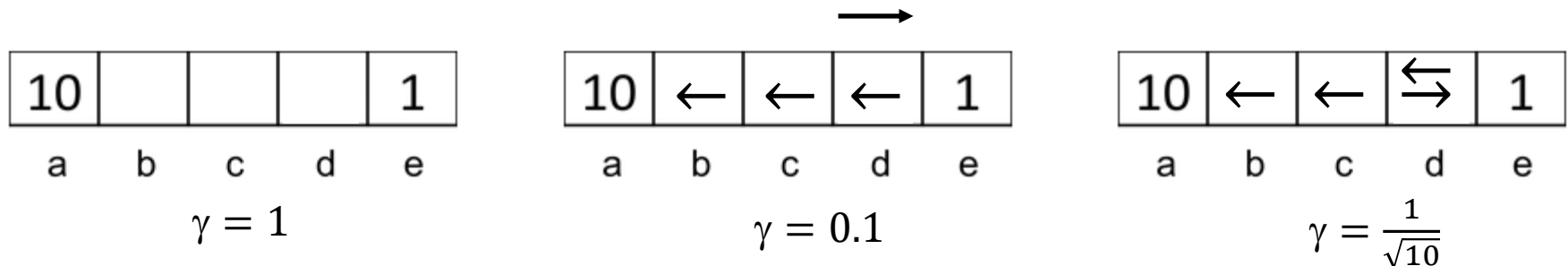
- Why discount?
  - Think of it as a  $1 - \gamma$  chance of ending the process at every step
  - Also helps our algorithms converge

- Example:  $\gamma = 0.5$ 
  - $U([1,2,3]) = 1 * 1 + 0.5 * 2 + 0.25 * 3 < U([3,2,1])$



# Discounting Example

- Given:
  - Actions: East, West, and Exit (only available in exit states a, e)
  - Transitions: deterministic
- For  $\gamma = 1$ , optimal policy in each state is always moving left
  - From state d, reward of going West is  $\gamma^3 \cdot 10 = 10$ , larger than reward of going East  $\gamma \cdot 1 = 1$
- For  $\gamma = 0.1$ , optimal policy in each state is below
  - From state d, reward from going West is  $\gamma^3 \cdot 10 = 0.01$ , less than reward from going East  $\gamma \cdot 1 = 0.1$ .
- For which  $\gamma$  are West and East equally good when in state d?
  - $\gamma^3 \cdot 10 = \gamma \cdot 1 \Rightarrow \gamma = \frac{1}{\sqrt{10}}$



Important

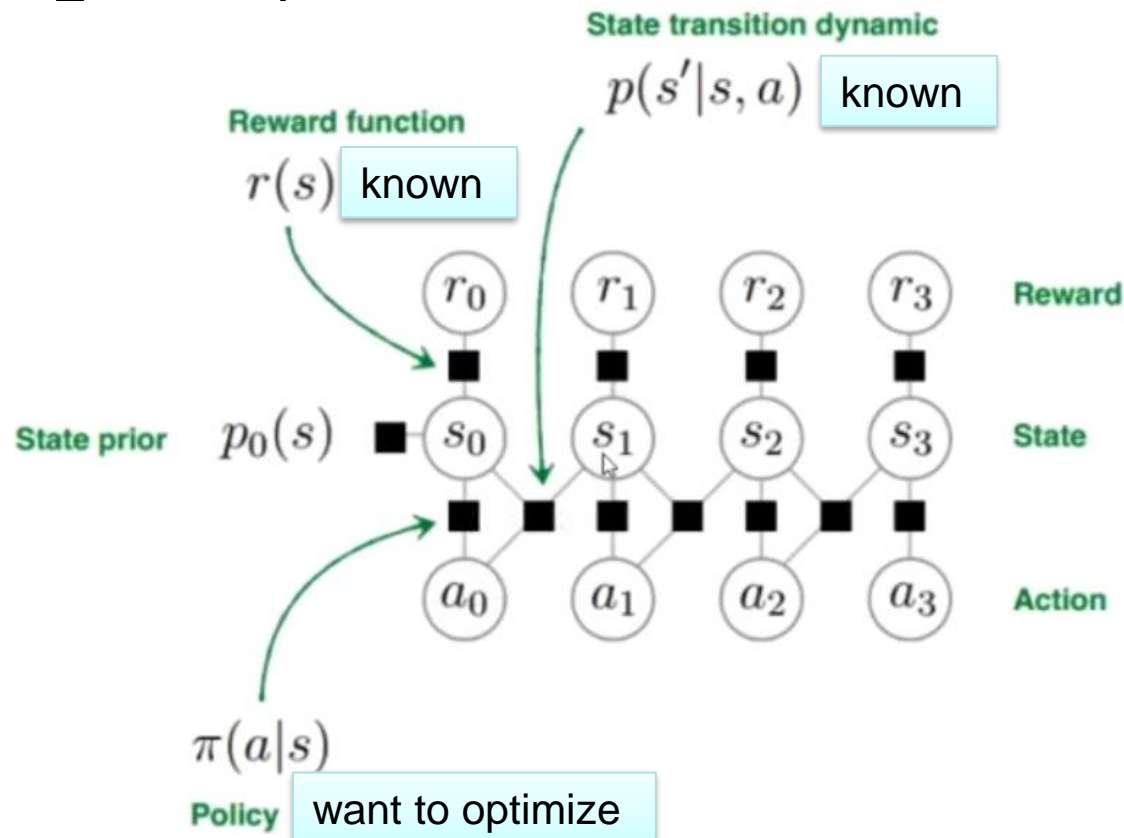
# The Big Picture

| Problem                                                                                        | Bellman Equation                                                | Algo (known MDP)                                | Algo (unknown MDP, sample-based)                                                                                      |
|------------------------------------------------------------------------------------------------|-----------------------------------------------------------------|-------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------|
| Prediction (compute $v_{\pi}(s)$ )                                                             | Bellman Exp. Equation for $v$                                   | Policy Evaluation (PE)                          | MC Prediction, TD/TD( $\lambda$ ) (on-policy)                                                                         |
| Control (compute $v_{\pi}(s)$ , then $\pi(s) = \operatorname{argmax}_a q(s, a)$ for known MDP) | Bellman Exp. Equation for $v$ + Greedy Policy Improvement (GPI) | Policy Iteration (PI=PE+GPI)                    | Cannot do GPI, since cannot get $Q(s, a)$ from $V(s)$ without MDP                                                     |
| Control (compute $v_*(s)$ , then $\pi^*(s) = \operatorname{argmax}_a q_*(s, a)$ for known MDP) | Bellman Opt. Equation for $v$                                   | Value Iteration (VI) (a form of Generalized PI) | Cannot compute $V^*(s)$ w. sample-based method due to $\max_a$ in front; cannot get $Q(s, a)$ from $V(s)$ without MDP |
| Control (compute $q_*(s, a)$ , then $\pi^*(s) = \operatorname{argmax}_a q_*(s, a)$ )           | Bellman Opt. Equation for $q$                                   | Q Value Iteration (QVI)                         | MC control, Sarsa (on-policy)<br>Q Learning, Expected Sarsa (off-policy)                                              |



# Known MDP

- In this lecture, we assume known MDP, and use dynamic programming to solve Bellman Equations and find the optimal policy (no learning here).



# Bellman Expectation Equations

- Bellman Expectation Equation for State Value Function:
- $v_{\pi}(s) = \sum_a \pi(a|s) \sum_{r,s'} p(r, s'|s, a) [r + \gamma v_{\pi}(s')]$   
 $= \mathbb{E}_{\pi}[R_{t+1} + \gamma v_{\pi}(S_{t+1}) | S_t = s]$ 
  - Expected value starting from state  $s$  and following policy  $\pi$ .
- Bellman Expectation Equation for Action Value Function
- $q_{\pi}(s, a) = \sum_{r,s'} p(r, s'|s, a) [r + \gamma \sum_{a'} \pi(a'|s') q_{\pi}(s', a')]$   
 $= \mathbb{E}_{\pi}[R_{t+1} + \gamma v_{\pi}(S_{t+1}) | S_t = s, A_t = a]$ 
  - Expected value starting from state  $s$ , taking action  $a$ , and thereafter following policy  $\pi$ .
- Relating  $v_{\pi}(s)$  and  $q_{\pi}(s, a)$ :
  - $v_{\pi}(s) = \sum_a \pi(a|s) q_{\pi}(s, a)$
  - $q_{\pi}(s, a) = \sum_{r,s'} p(r, s'|s, a) [r + \gamma v_{\pi}(s')]$

Important

# Bellman Optimality Equations

- Bellman Optimality Equation for Optimal State Value Function:
- $$v_*(s) = \max_a \sum_{r,s'} p(r, s'|s, a) [r + \gamma v_*(s')]$$

$$= \max_a \mathbb{E}_\pi [R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a]$$
  - Max value starting from state  $s$  and following the greedy policy  $\pi(s) = \operatorname{argmax}_a q_*(s, a)$
- Bellman Optimality Equation for Optimal Action Value Function
- $$q_*(s, a) = \sum_{r,s'} p(r, s'|s, a) [r + \gamma \max_{a'} q_*(s', a')] = \mathbb{E}_\pi [R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a]$$
  - Max value starting from state  $s$ , taking action  $a$ , and thereafter following the greedy policy  $\pi(s) = \operatorname{argmax}_a q_*(s, a)$
- Relating  $v_*(s)$  and  $q_*(s, a)$ :
  - $$v_*(s) = \max_a q_*(s, a); q_*(s, a) = \sum_{r,s'} p(r, s'|s, a) [r + \gamma v_*(s')]$$

Handwritten equations and diagram illustrating the Bellman Optimality Equations:

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

$$Q(s, a) = R(s, a) + \gamma \sum_{s'} T(s, a, s') \max_{a'} Q(s', a')$$

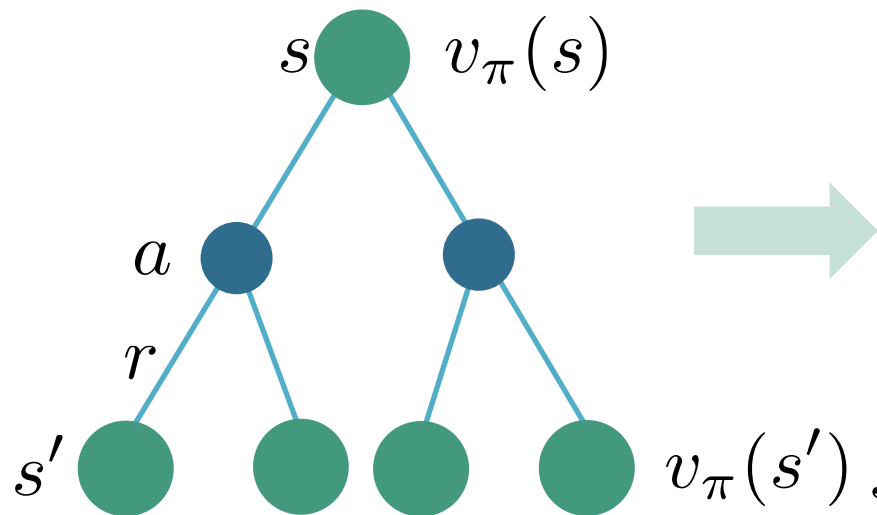
The diagram shows a sequence of states  $s_1, s_2, s_3$  and actions  $a_1, a_2$ . Transitions are labeled with probabilities  $T(s, a, s')$ . Rewards  $R(s, a)$  are shown at each state. The diagram illustrates the recursive nature of the Bellman equations for state value  $V(s)$  and action value  $Q(s, a)$ .

- Notations in left fig:
- $$\sum_{s'} T(s, a, s') [\dots] = \sum_{r,s'} p(r, s'|s, a) [\dots]$$
- $$R(s, a) = R_t = \sum_{s'} T(s, a, s') r$$

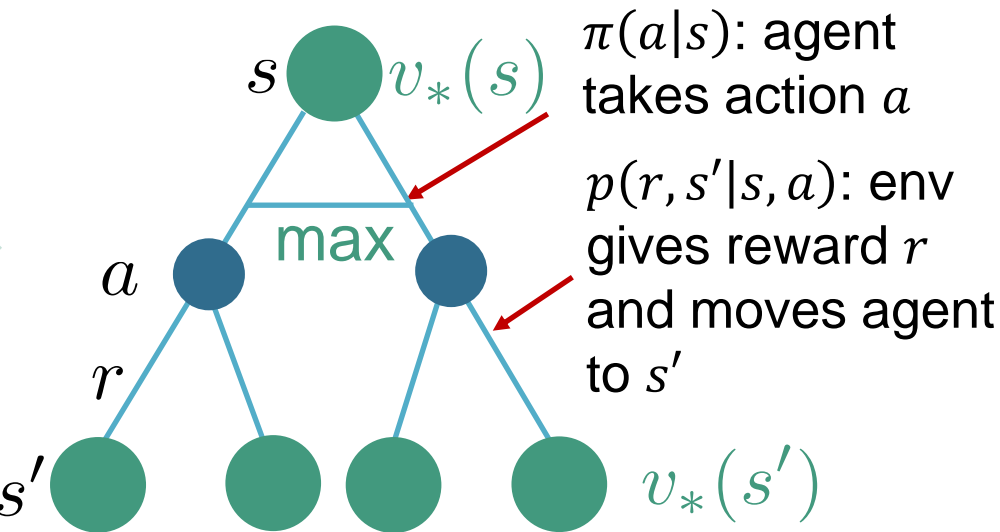
# Simplified Bellman Equations

- Bellman Equations:
  - $v_{\pi}(s) = \sum_a \pi(a|s) q_{\pi}(s, a); q_{\pi}(s, a) = \sum_{r,s'} p(r, s'|s, a) [r + \gamma v_{\pi}(s')]$
  - $v_{*}(s) = \max_a q_{*}(s, a); q_{*}(s, a) = \sum_{r,s'} p(r, s'|s, a) [r + \gamma v_{*}(s')]$
- For Deterministic Env: there is only one possible  $(r, s')$  for a given  $(s, a)$ , it can be simplified to (we use  $R_s^a$  instead of  $r$  to emphasize that reward is specific to this  $(s, a)$ ):
  - $q_{\pi}(s, a) = R_s^a + \gamma v_{\pi}(s')$
  - $q_{*}(s, a) = R_s^a + \gamma v_{*}(s')$
- In the back diagrams, BF denotes Branching Factor of backup diagram.

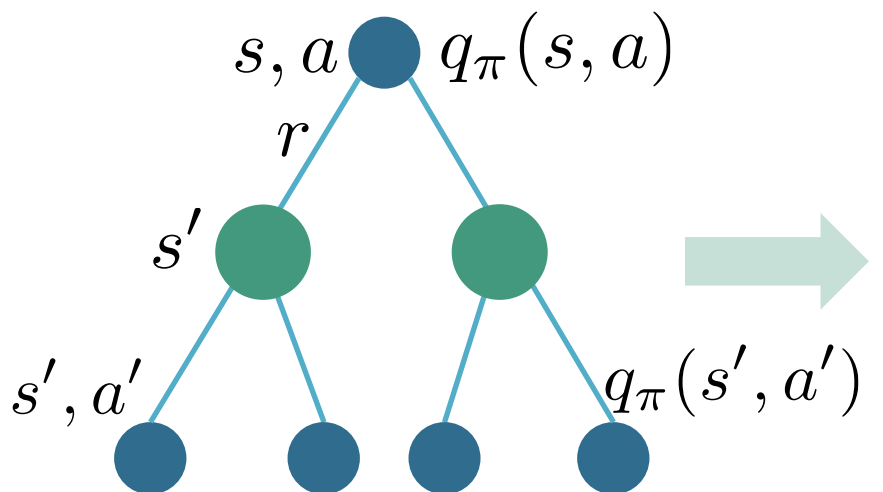
# Backup Diagrams



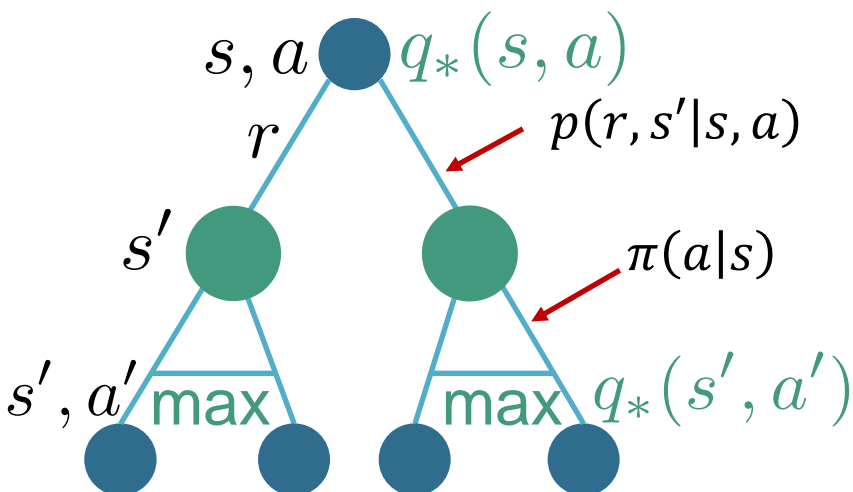
Bellman Exp Eqn for  $v_\pi(s)$



Bellman Opt Eqn for  $v_*(s)$



Bellman Exp Eqn for  $q_\pi(s, a)$



Bellman Opt Eqn for  $q_*(s, a)$

$\pi(a|s)$ : agent takes action  $a$   
 $p(r, s'|s, a)$ : env gives reward  $r$  and moves agent to  $s'$

## $v(s)$ vs. $q(s, a)$

- **Can always** go from  $q_\pi(s, a)$  to  $v_\pi(s)$ , or from  $q_*(s, a)$  to  $v_*(s)$ :
  - $v_\pi(s) = \sum_a \pi(a|s) q_\pi(s, a)$ ;  $v_*(s) = \max_a q_*(s, a)$
- **Can always** get optimal policy  $\pi_*(s) = \operatorname{argmax}_a q_*(s, a)$
- With known MDP model ( $p(r, s'|s, a)$ ): **can** go from  $v_\pi(s)$  to  $q_\pi(s, a)$ , or from  $v_*(s)$  to  $q_*(s, a)$ :
  - $q_\pi(s, a) = \sum_{r, s'} p(r, s'|s, a) [r + \gamma v_\pi(s')]$
  - $q_*(s, a) = \sum_{r, s'} p(r, s'|s, a) [r + \gamma v_*(s')]$
  - **Can** get optimal policy  $\pi_*(s) = \operatorname{argmax}_a q_*(s, a)$
- With unknown MDP (unknown  $p(r, s'|s, a)$ , i.e., model-free) : **cannot** go from  $v_\pi(s)$  to  $q_\pi(s, a)$ , or from  $v_*(s)$  to  $q_*(s, a)$ 
  - **Cannot** get optimal policy  $\pi_*(s)$  from  $v_*(s)$
- $q(s, a)$  contains more information than  $v(s)$ .

# Policy Evaluation

- Also called the Prediction Problem
- Predict Value Function for given policy  $\pi$  by solving Bellman Exp. Equation for State Value Function
  - $v_{\pi}(s) = \sum_a \pi(a|s) \sum_{r,s'} p(r, s'|s, a) [r + \gamma v_{\pi}(s')] = \mathbb{E}_{\pi}[R_{t+1} + \gamma v_{\pi}(S_{t+1}) | S_t = s]$
- A set of linear equations that can be solved efficiently for small system
  - # unknowns = # equations = # states

# Iterative Policy Evaluation

- If # states are too large, it may be too expensive to solve the set of linear Bellman Exp. Equations. We can use Iterative Policy Evaluation and solve the recursive Bellman equations with iterative updates:
  - $v(s) = \text{Bellman}(v(s')) \Rightarrow v_k(s) \leftarrow \text{Bellman}(v_{k-1}(s'))$
- In-place updates:
  - For faster convergence, update  $v_k(s)$  using the value of  $v_k(s')$  that has been updated in the same iteration, instead of  $v_{k-1}(s')$  from the previous iteration.
  - $v_k(s) \leftarrow \text{Bellman}(v_k(s'))$

## 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  $V(s)$ , for all  $s \in \mathcal{S}^+$ , arbitrarily except that  $V(\text{terminal}) = 0$

Loop:

$\Delta \leftarrow 0$

Loop for each  $s \in \mathcal{S}$ :

$v \leftarrow V(s)$

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

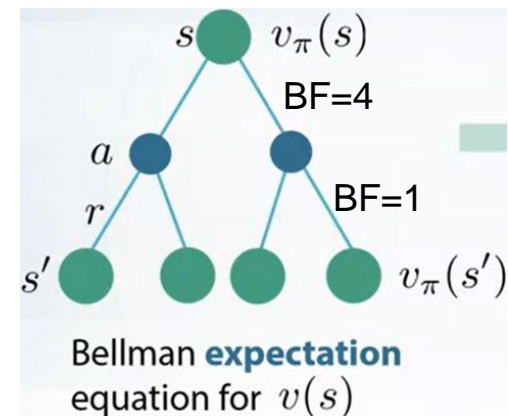
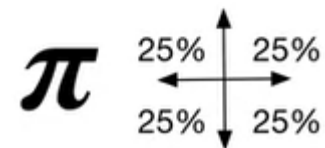
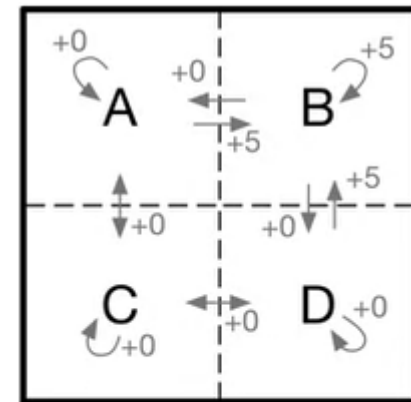
$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

until  $\Delta < \theta$



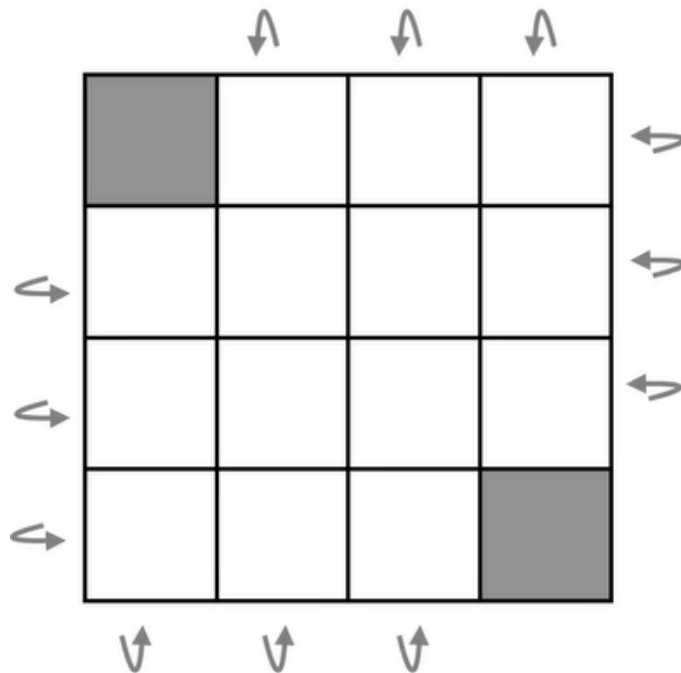
# Grid World1: Policy Evaluation

- Deterministic env:** Agent in state  $s \in \{A, B, C, D\}$  taking action  $a \in \{l, r, u, d\}$  always moves to the next state in the movement direction, unless it is blocked by the walls. Discount factor  $\gamma = 0.7$ .
- Random policy:** Agent in state  $s \in \{A, B, C, D\}$  takes a random action  $a \in \{l, r, u, d\}$  with equal probability of 0.25 each.
- Bellman Exp. Equation (simplified for det env) :
  - $v_{\pi}(s) = \sum_a \pi(a|s) q_{\pi}(s, a) ; q_{\pi}(s, a) = R_s^a + \gamma v_{\pi}(s')$
- $$v_{\pi}(A) = 0.25(q_{\pi}(A, l) + q_{\pi}(A, r) + q_{\pi}(A, u) + q_{\pi}(A, d)) = 0.5 \cdot 0.7v_{\pi}(A) + 0.25 \cdot (5 + 0.7v_{\pi}(B)) + 0.25 \cdot 0.7v_{\pi}(C)$$
  - $q_{\pi}(A, l) = q_{\pi}(A, u) = 0 + 0.7v_{\pi}(A)$
  - $q_{\pi}(A, r) = 5 + 0.7v_{\pi}(B)$
  - $q_{\pi}(A, d) = 0 + 0.7v_{\pi}(C)$
- $$v_{\pi}(B) = 0.25(q_{\pi}(B, l) + q_{\pi}(B, r) + q_{\pi}(B, u) + q_{\pi}(B, d)) = 0.25 \cdot 0.7v_{\pi}(A) + 0.5 \cdot (5 + 0.7v_{\pi}(B)) + 0.25 \cdot 0.7v_{\pi}(D)$$
  - $q_{\pi}(B, l) = 0 + 0.7v_{\pi}(A)$
  - $q_{\pi}(B, r) = q_{\pi}(A, u) = 5 + 0.7v_{\pi}(B)$
  - $q_{\pi}(B, d) = 0 + 0.7v_{\pi}(D)$
- $$v_{\pi}(C) = 0.25(q_{\pi}(C, l) + q_{\pi}(C, r) + q_{\pi}(C, u) + q_{\pi}(C, d)) = 0.25 \cdot 0.7v_{\pi}(A) + 0.5 \cdot 0.7v_{\pi}(C) + 0.25 \cdot 0.7v_{\pi}(D)$$
  - $q_{\pi}(C, l) = q_{\pi}(C, d) = 0 + 0.7v_{\pi}(C)$
  - $q_{\pi}(C, r) = 0 + 0.7v_{\pi}(D)$
  - $q_{\pi}(C, u) = 0 + 0.7v_{\pi}(A)$
- $$v_{\pi}(D) = 0.25(q_{\pi}(D, l) + q_{\pi}(D, r) + q_{\pi}(D, u) + q_{\pi}(D, d)) = 0.25 \cdot (5 + 0.7v_{\pi}(B)) + 0.25 \cdot 0.7v_{\pi}(C) + 0.5 \cdot 0.7v_{\pi}(D)$$
  - $q_{\pi}(D, l) = 0 + 0.7v_{\pi}(C)$
  - $q_{\pi}(D, r) = q_{\pi}(D, d) = 0 + 0.7v_{\pi}(D)$
  - $q_{\pi}(D, u) = 5 + 0.7v_{\pi}(B)$
- Solution:**  $v_{\pi}(A) = 4.2, v_{\pi}(B) = 6.1, v_{\pi}(C) = 2.2, v_{\pi}(D) = 4.2$



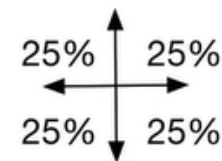
# Iterative Policy Evaluation Example

- An episodic MDP with terminal states with  $v(s) = 0$  located in the top left and bottom right corners. Reward  $R = -1$  for every transition. Discount factor  $\gamma = 1$ . Four possible actions in each state: up, down, left, and right. Environment is deterministic. If the action would move the agent off the grid, it instead leaves the agent in the same state.



$$R = -1$$

$$\gamma = 1$$



**Policy**

# Iterative Policy Evaluation Example

- Bellman Exp. Equation:  $v_{\pi}(s) = \sum_a \pi(a|s) q_{\pi}(s, a)$ ;  $q_{\pi}(s, a) = R_s^a + \gamma v_{\pi}(s')$
- 1<sup>st</sup> sweep:
  - for all states  $v_{\pi}(s) = 0.25((-1 + 0) + (-1 + 0) + (-1 + 0) + (-1 + 0)) = -1$
- 2<sup>nd</sup> sweep:
  - For states marked -2:  $v_{\pi}(s) = 0.25((-1 - 1) + (-1 - 1) + (-1 - 1) + (-1 - 1)) = -2$
  - For states marked -1.7:  $v_{\pi}(s) = 0.25((-1 + 0) + (-1 - 1) + (-1 - 1) + (-1 - 1)) = -1.75$
- 3<sup>rd</sup> sweep:
  - For states marked -2.4:  $v_{\pi}(s) = 0.25((-1 + 0) + (-1 - 2) + (-1 - 1.75) + (-1 - 2)) = -2.43$
  - For states marked -2.9:  $v_{\pi}(s) = 0.25((-1 - 1.7) + (-1 - 2) + (-1 - 1.7) + (-1 - 2)) = -2.85$
  - For states marked -3:  $v_{\pi}(s) = 0.25((-1 - 2) + (-1 - 2) + (-1 - 2) + (-1 - 2)) = -3$
- Influence of terminal states gradually spreads through the entire grid.

|   |   |   |   |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |

Initial

|    |    |    |    |
|----|----|----|----|
| 0  | -1 | -1 | -1 |
| -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | 0  |

After 1<sup>st</sup> sweep

|      |      |      |      |
|------|------|------|------|
| 0    | -1.7 | -2   | -2   |
| -1.7 | -2   | -2   | -2   |
| -2   | -2   | -2   | -1.7 |
| -2   | -2   | -1.7 | 0    |

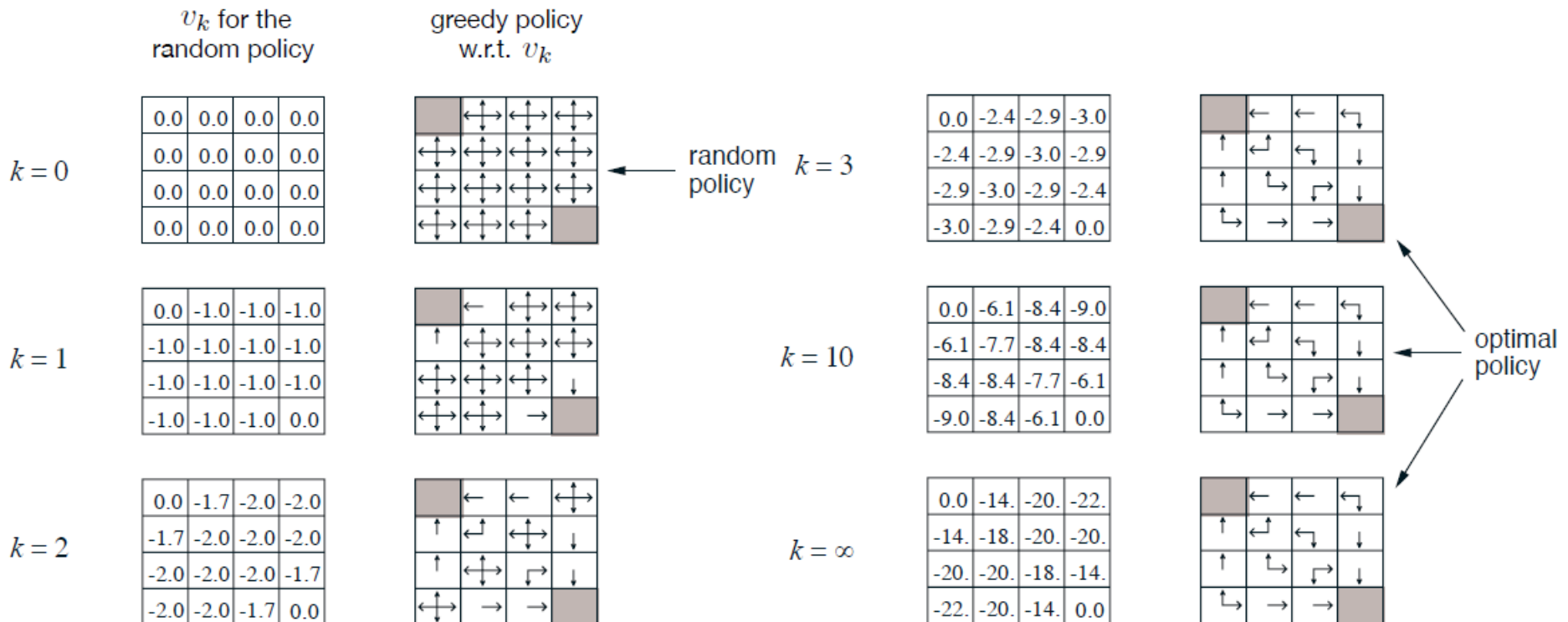
After 2<sup>nd</sup> sweep

|      |      |      |      |
|------|------|------|------|
| 0    | -2.4 | -2.9 | -3   |
| -2.4 | -2.9 | -3   | -2.9 |
| -2.9 | -3   | -2.9 | -2.4 |
| -3   | -2.9 | -2.4 | 0    |

After 3<sup>rd</sup> sweep<sub>27</sub>

# Iterative Policy Evaluation Results

- Figure 4.1: Convergence of iterative policy evaluation on a small gridworld. The left column is the sequence of approximations of the state-value function for the random policy (all actions equally likely). The right column is the sequence of greedy policies corresponding to the value function  $v_{\pi}(s)$  estimates (arrows are shown for all actions achieving the max  $v_{\pi}(s)$ ). All policies after  $k = 3$  iterations are optimal.
- In this case, one sweep of PE is followed by one Policy Improvement, so it just happens by chance to get the optimal policy.



# Policy Iteration

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  $s \in \mathcal{S}$

## 2. Policy Evaluation

Loop:

$\Delta \leftarrow 0$

Loop for each  $s \in \mathcal{S}$ :

$v \leftarrow V(s)$

$V(s) \leftarrow \sum_{s',r} p(s',r|s,\pi(s)) [r + \gamma V(s')]$

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

until  $\Delta < \theta$  (a small positive number determining the accuracy of estimation)

Repeat until policy converges:

Policy Evaluation: Estimate state value function  $v_\pi$  for some fixed policy  $\pi$  with Iterative Policy Evaluation (or solving linear equations).

Policy Improvement: generate new policy based on the newly estimated  $v_\pi$ :  $\pi = \text{greedy}(v_\pi)$ .

## 3. Policy Improvement

$\text{policy-stable} \leftarrow \text{true}$

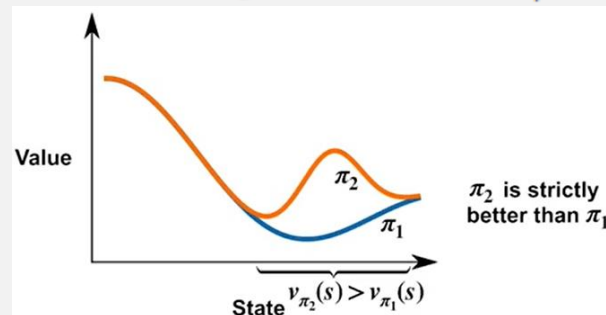
For each  $s \in \mathcal{S}$ :

$\text{old-action} \leftarrow \pi(s)$

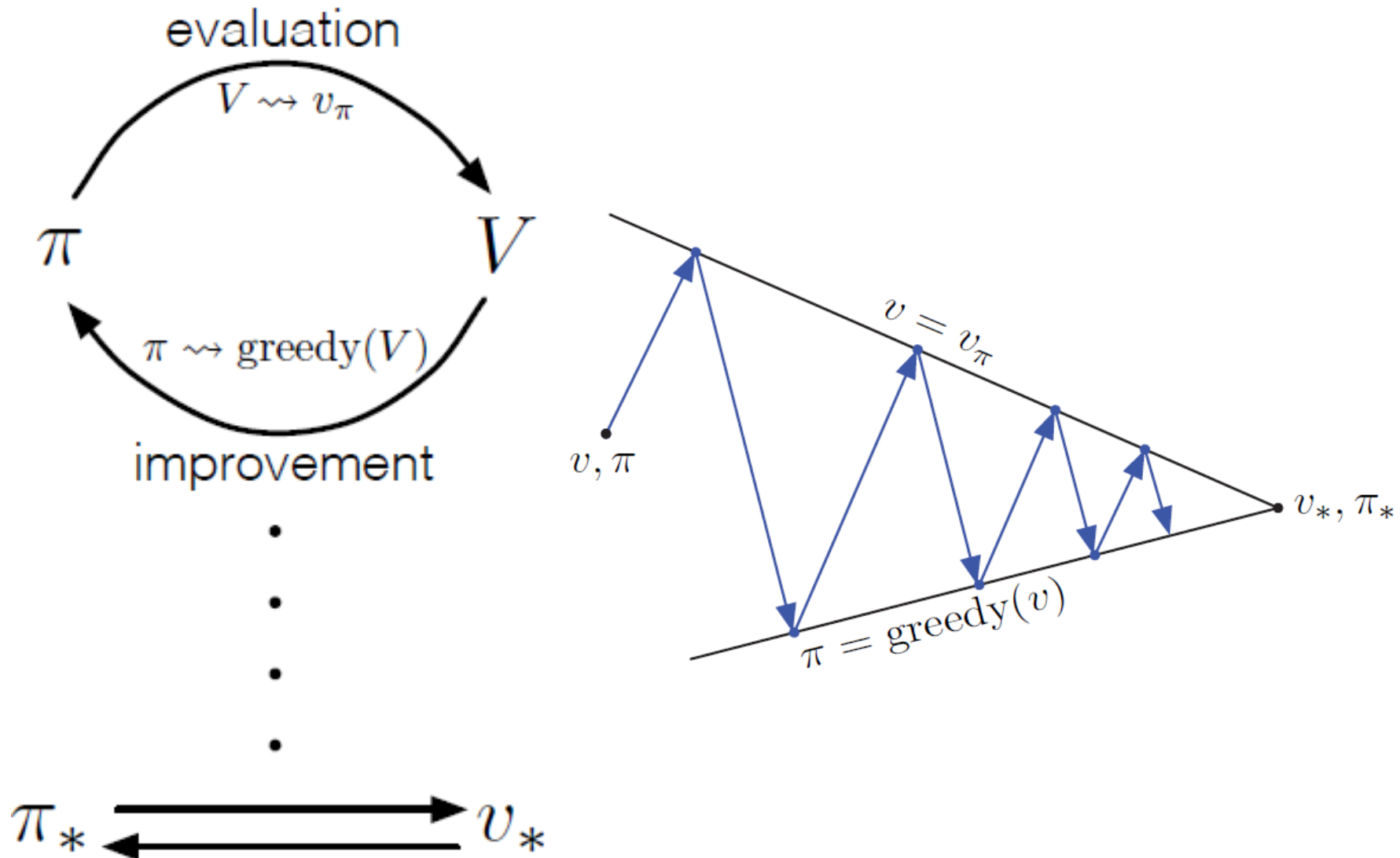
$\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s',r|s,a) [r + \gamma V(s')]$

If  $\text{old-action} \neq \pi(s)$ , then  $\text{policy-stable} \leftarrow \text{false}$

If  $\text{policy-stable}$ , then stop and return  $V \approx v_*$  and  $\pi \approx \pi_*$ ; else go to 2

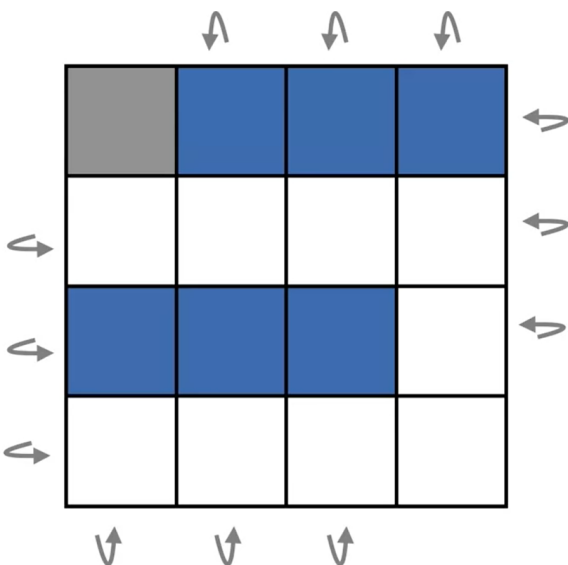


# Policy Iteration Illustration



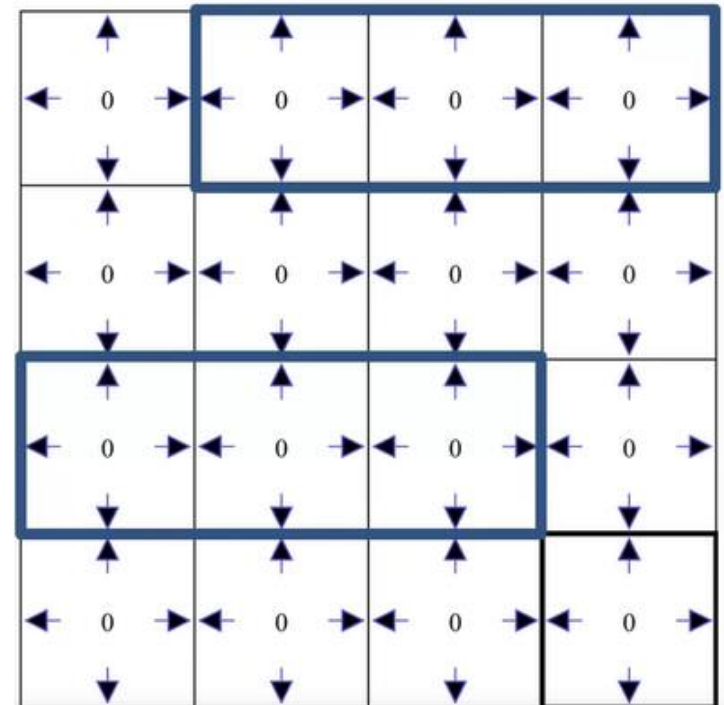
# Policy Iteration Example

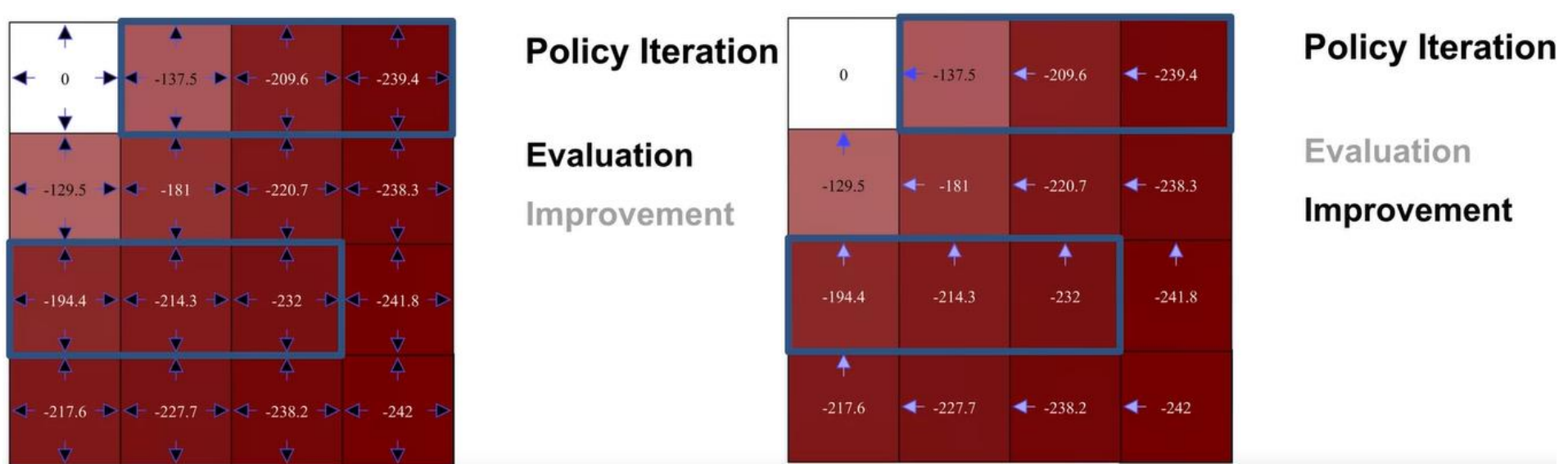
- An episodic MDP with terminal states with  $v(s) = 0$  located in the top left corner. Blue states are bad states with large negative reward.
- Start with uniform random policy.
- Initialize  $v(s) = 0$  for all  $s$ .



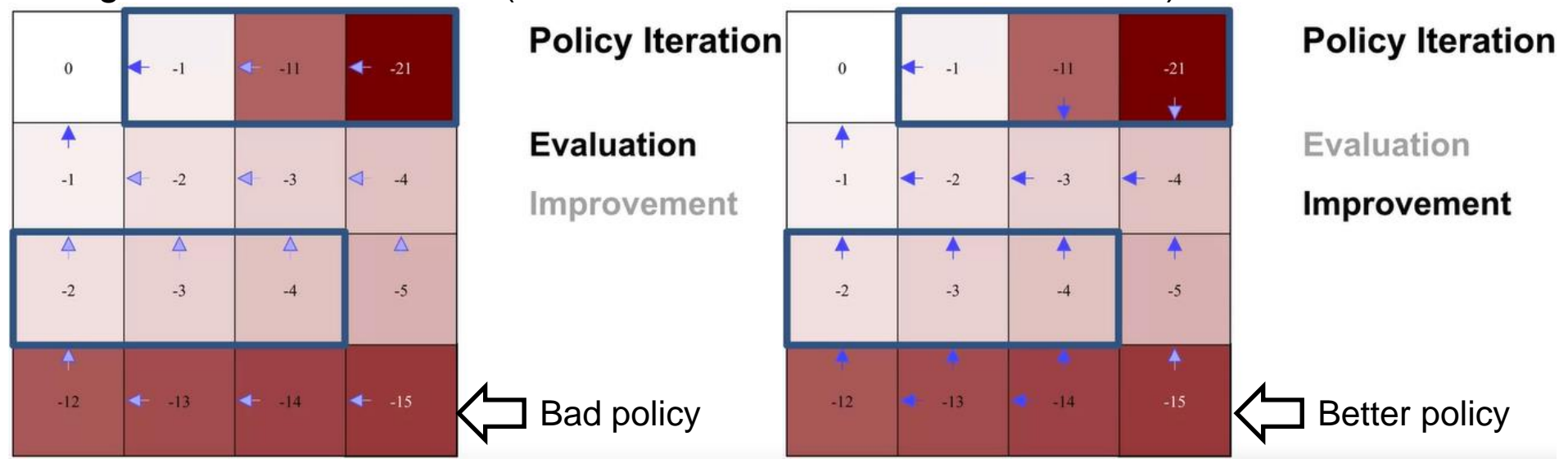
$$R = \begin{cases} -10 & \text{Blue states} \\ -1 & \text{Other states} \end{cases}$$

$$\gamma = 1$$



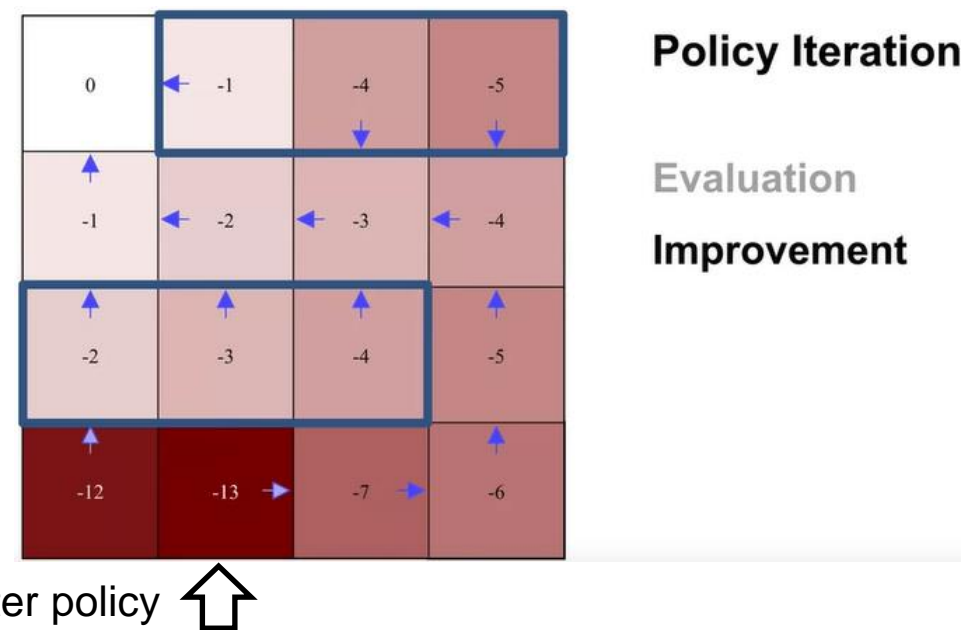
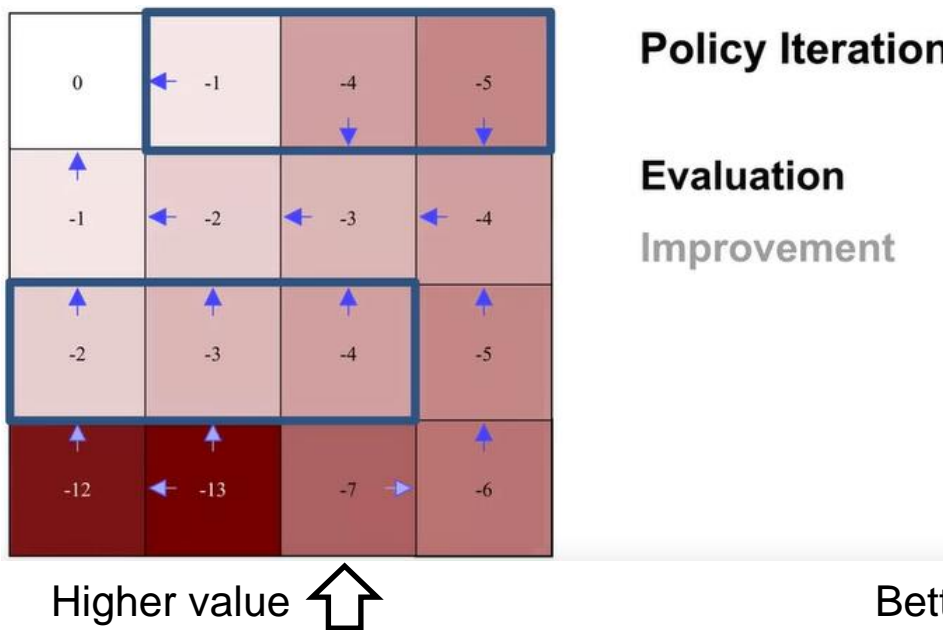
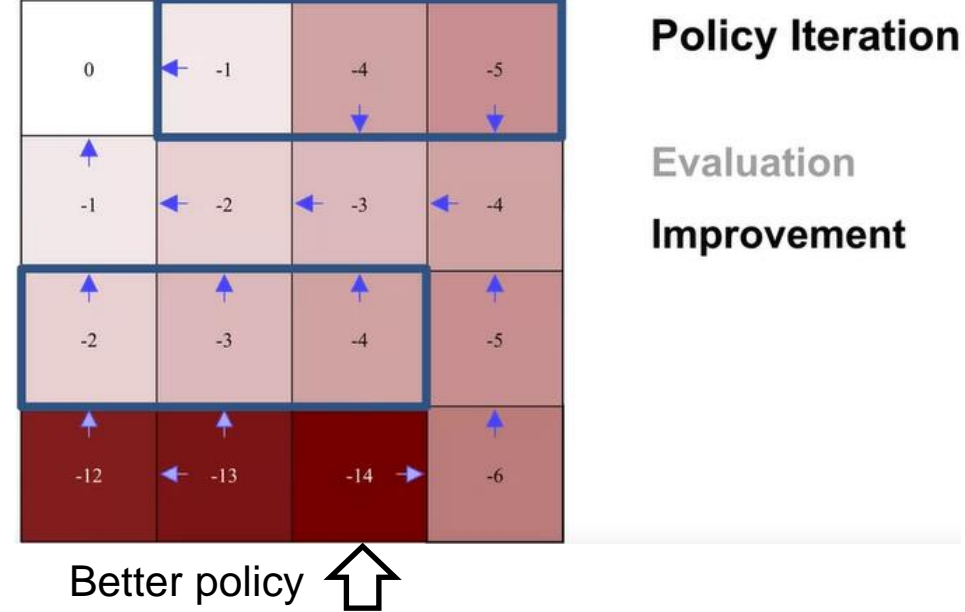
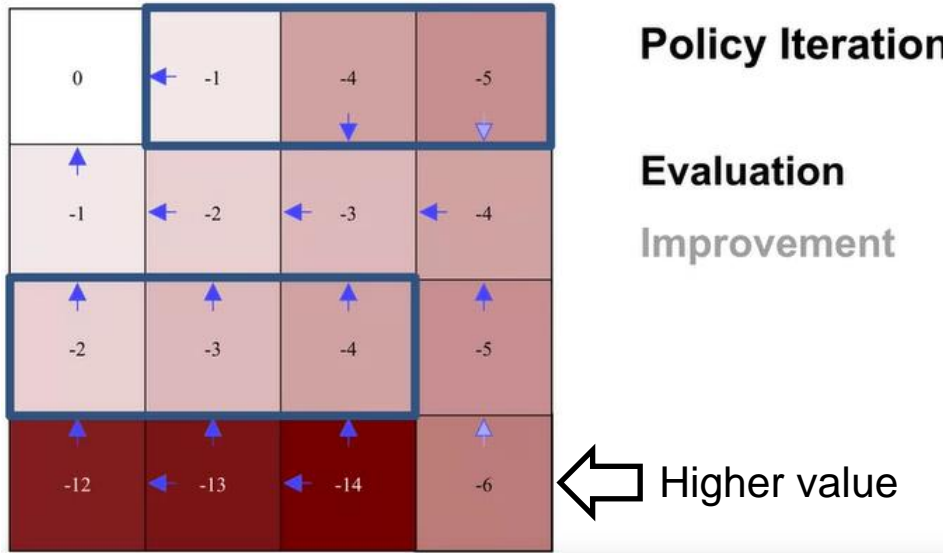


After 1<sup>st</sup> sweep of PE+PI. Most state's policies are not good since they go towards blue states (darker colors denote low-value states).



After 2<sup>nd</sup> sweep of PE+PI. Policies are much better, but a few states' policies still go towards blue states.



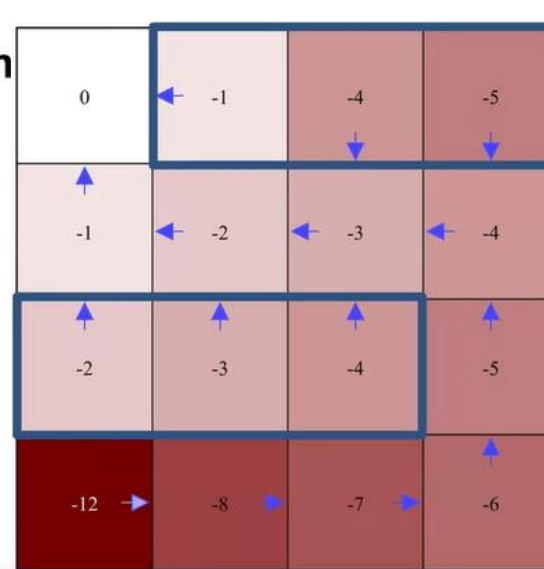




↑ Higher value

Policy Iteration

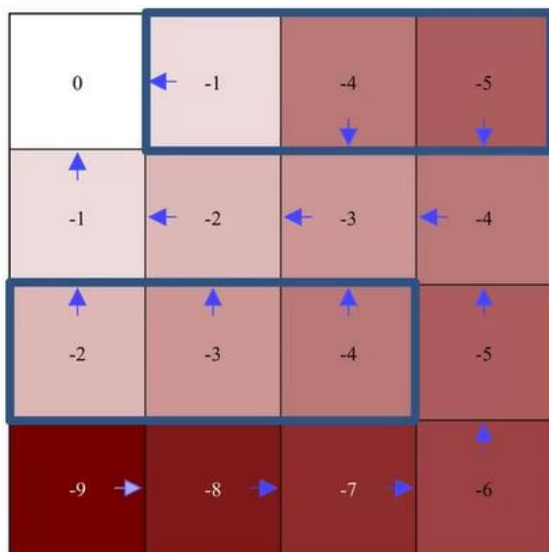
Evaluation  
Improvement



Better policy ↑

Policy Iteration

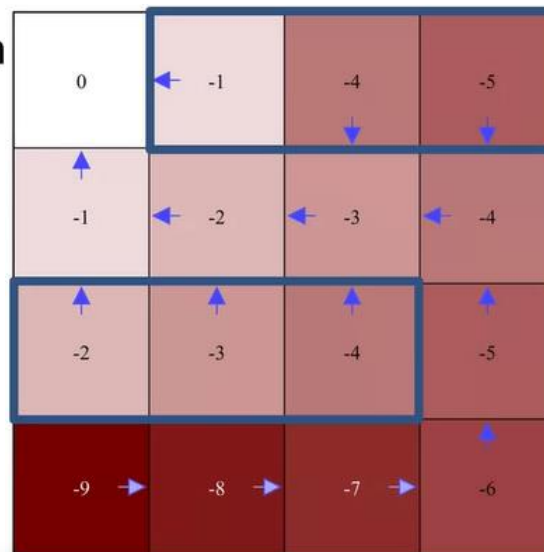
Evaluation  
Improvement



Value functions converged  
(May not converge in general. Policy often stable long before value functions converge.)

Policy Iteration

Evaluation  
Improvement



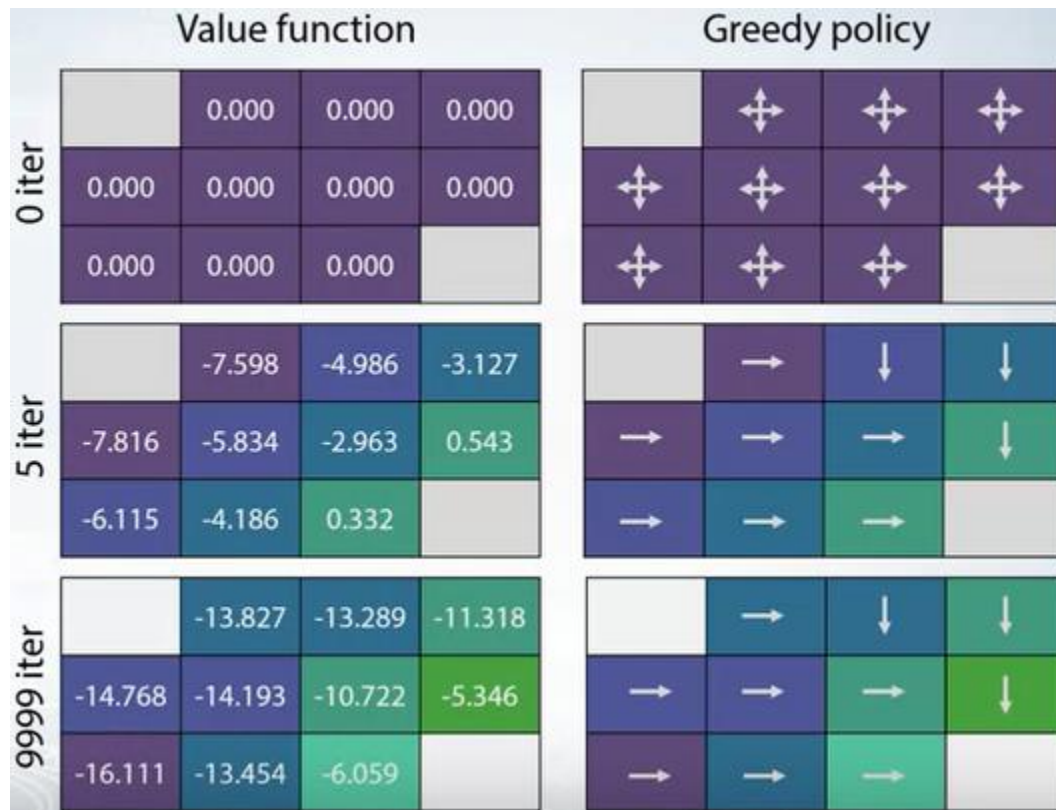
Policy stable, hence it is optimal policy)

Policy Iteration

Evaluation  
Improvement

# Another Policy Iteration Example

- The greedy policy converges to the optimal policy after 5<sup>th</sup> iteration of Policy Iteration.
- Precise values of  $v(s)$  are not necessary for computing optimal policy with Policy Iteration.



# Value Iteration

- Obtain optimal value function by solving Bellman Opt. Equation for State Value Function
  - $v_*(s) = \max_a \sum_{r,s'} p(r,s'|s,a) [r + \gamma v_*(s')]$
- Optimal policy  $\pi_*(s)$  is output at the end; intermediate value function  $v(s)$  may not correspond to any valid policy.

## 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  $V(\text{terminal}) = 0$

Loop:

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

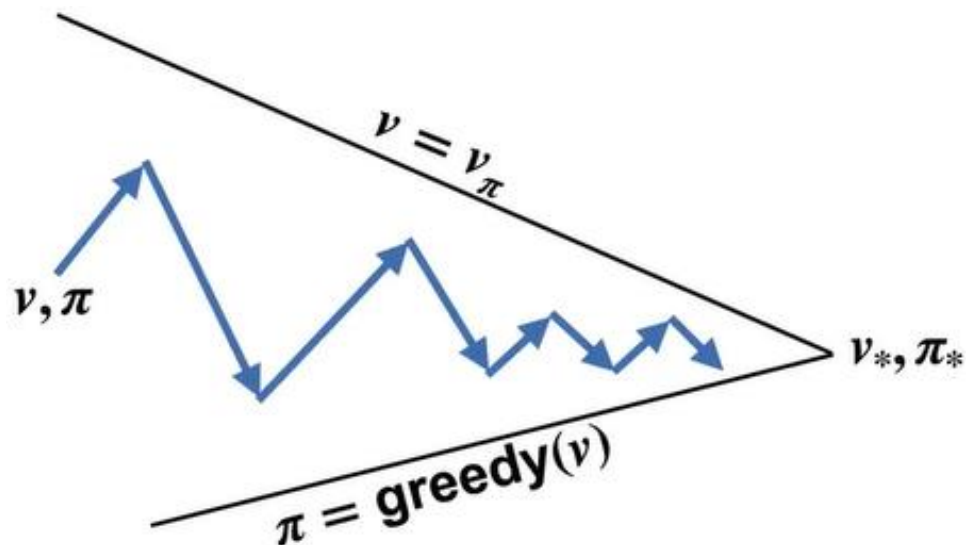
Output a deterministic policy,  $\pi \approx \pi_*$ , such that  
$$\pi(s) = \operatorname{argmax}_a \sum_{s',r} p(s',r|s,a) [r + \gamma V(s')]$$

# Generalized Policy Iteration (GPI)

- GPI: 1. evaluate given policy; 2. Improve policy by acting greedily w.r.t its value function. GPI is guaranteed to converge to the optimal deterministic policy  $\pi^*$ .
- Two special cases of GPI:
- Policy Iteration (PI):
  - 1. Policy evaluation until convergence; 2. Improve policy
- Value Iteration (VI):
  - 1. Policy evaluation with one single iteration; 2. improve policy (implicitly)

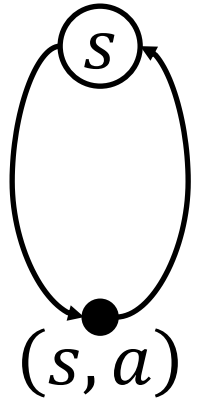
# PI vs. VI: Computational Complexity

- PI is slower per sweep (cycle), with complexity  $O(|A||S|^2 + |S|^3)$ , and requires fewer sweeps.
- VI is faster per sweep, with complexity  $O(|A||S|^2)$ , and requires more sweeps.
- Modified Policy Iteration: run Policy Evaluation for # steps between 1 for VI, and that needed for convergence of  $v_\pi(s)$ , may achieve best efficiency.



# Toy Example

- An MDP with a single state  $s$  and single action  $a$ , with reward  $r(s, a) = 1, \gamma = 0.9$
- From definition:  $v_\pi(s) = v_*(s) = 1 + \gamma + \gamma^2 + \dots = \frac{1}{1-\gamma} = 10$
- Policy Iteration:
  - Policy Evaluation:
    - $v_\pi(s) = \sum_a \pi(a|s) q_\pi(s, a) = \sum_a \pi(a|s) (1 + \gamma v_\pi(s)) = 1 + \gamma v_\pi(s)$
    - $v_\pi(s) = \frac{1}{1-\gamma} = 10$
    - Policy Improvement: policy  $\pi(s) = a$  is now stable, since there is only one possible action  $a$ .
- Value Iteration:
  - $v_*(s) = \max_a q_*(s, a) = \max_a (1 + \gamma v_*(s)) = 1 + \gamma v_*(s)$
  - $v_*(s) = \frac{1}{1-\gamma} = 10$
  - $\pi_*(s) = \operatorname{argmax}_a q_*(s, a) = a$



# MiniGW Example

- Policy Iteration for Deterministic Environment
- Policy Iteration for Stochastic Environment
  - Policy Evaluation with analytic solution, not IPE
- Value Iteration for Deterministic Environment
- Value Iteration for Stochastic Environment
  - Bellman Opt. Equation solved iteratively w. in-place updates
- Model-based Learning

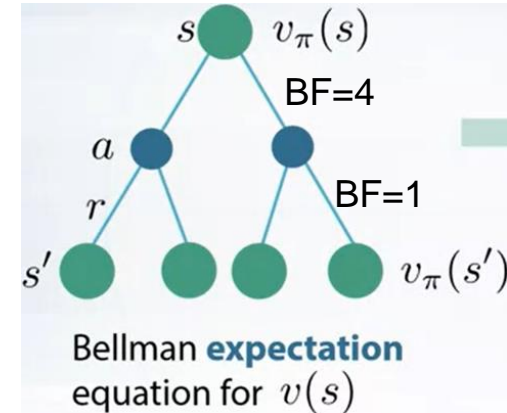
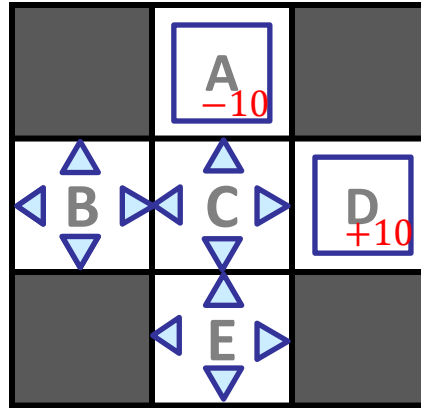


# MiniGW Setup



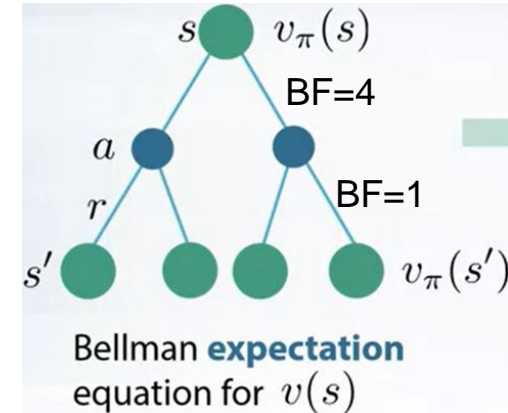
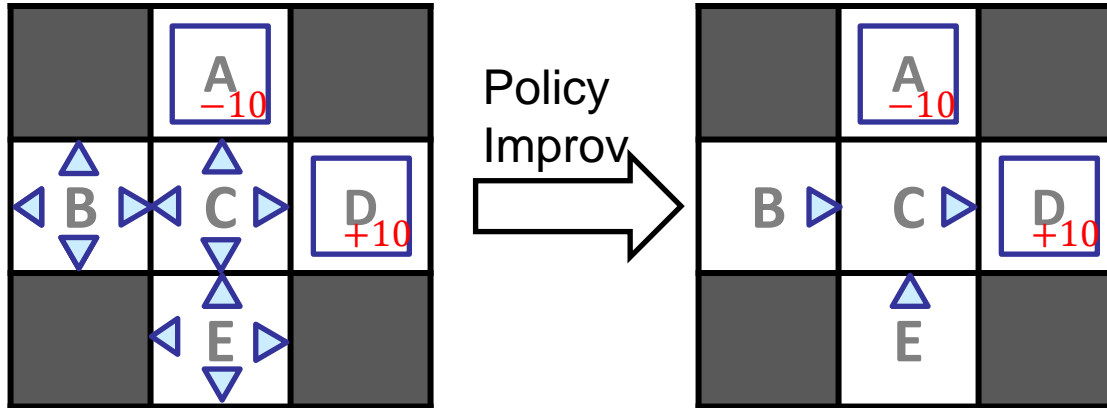
- All transitions  $(s, a, r, s')$  for  $s \in \{B, C, E\}$ ,  $a \in \{l, r, u, d\}$  have the same reward  $r = -1$ .
- $A$  and  $D$  are terminal states with fixed value function  $v(A) = -10$ ,  $v(D) = 10$ 
  - Or equivalently, you can think of an extra terminal state  $x$  with  $v(x) = 0$ , the only action in state  $A$  or  $D$  is *exit* that leads to  $x$  with  $R_A^a = -10$ ,  $R_D^a = 10$ .
- Dark squares denote obstacles which the agent cannot move into.
- Discount factor  $\gamma = 1$ .
- **Deterministic env:** Agent in state  $s \in \{B, C, E\}$  taking action  $a \in \{l, r, u, d\}$  always moves to the next state in the movement direction, unless it is blocked by an obstacle.
- **Stochastic env:** Agent in state  $s \in \{B, C, E\}$  taking action  $a \in \{l, r, u, d\}$  moves to the next state in the movement direction w. prob 0.8, or to the left or right side, each w. prob 0.1. If it is blocked by an obstacle in any direction, then it stays in the same state with prob of moving in the blocked direction.

# Iter1 Policy Evaluation of Random Policy



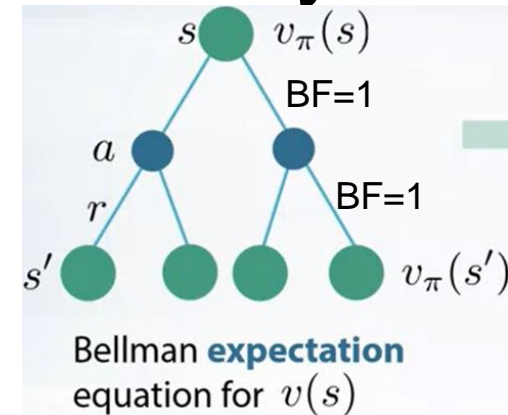
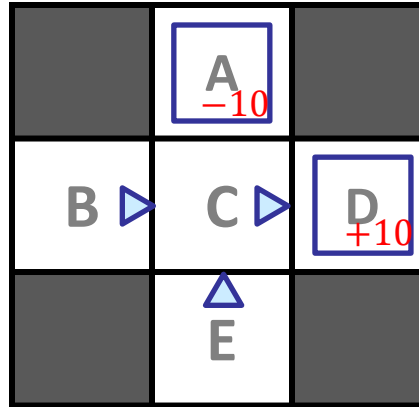
- Random policy:  $\pi(l|s) = \pi(r|s) = \pi(u|s) = \pi(d|s) = 0.25$
- Policy Evaluation for Det Env:  $v_\pi(s) = \sum_a \pi(a|s) q_\pi(s, a)$ ;  $q_\pi(s, a) = r + \gamma v_\pi(s')$
- Set of equations:
- $v_\pi(C) = 0.25(q_\pi(C, l) + q_\pi(C, r) + q_\pi(C, u) + q_\pi(C, d)) = 0.25(-4 + v_\pi(B) + v_\pi(E))$ 
  - $q_\pi(C, l) = -1 + 1.0v_\pi(B)$
  - $q_\pi(C, r) = -1 + 1.0v(D) = -1 + 10 = 9$
  - $q_\pi(C, u) = -1 + 1.0v(A) = -1 - 10 = -11$
  - $q_\pi(C, d) = -1 + 1.0v_\pi(E)$
- $v_\pi(B) = 0.25(q_\pi(B, l) + q_\pi(B, r) + q_\pi(B, u) + q_\pi(B, d)) = 0.25(-4 + 3.0v_\pi(B) + v_\pi(C))$ 
  - $q_\pi(B, l) = q_\pi(B, u) = q_\pi(B, d) = -1 + 1.0v_\pi(B)$
  - $q_\pi(B, r) = -1 + 1.0v_\pi(C)$
- $v_\pi(E) = 0.25(q_\pi(E, l) + q_\pi(E, r) + q_\pi(E, u) + q_\pi(E, d)) = 0.25(-4 + 3.0v_\pi(E) + v_\pi(C))$ 
  - $q_\pi(E, l) = q_\pi(E, r) = q_\pi(E, d) = -1 + 1.0v_\pi(E)$
  - $q_\pi(E, u) = -1 + 1.0v_\pi(C)$
- Solution:  $v_\pi(C) = -6, v_\pi(B) = -10, v_\pi(E) = -10$

# Iter1 Policy Improvement



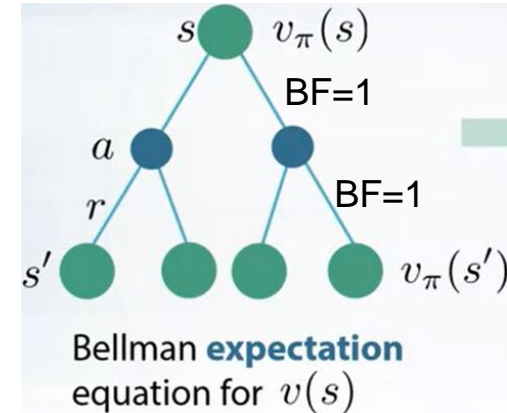
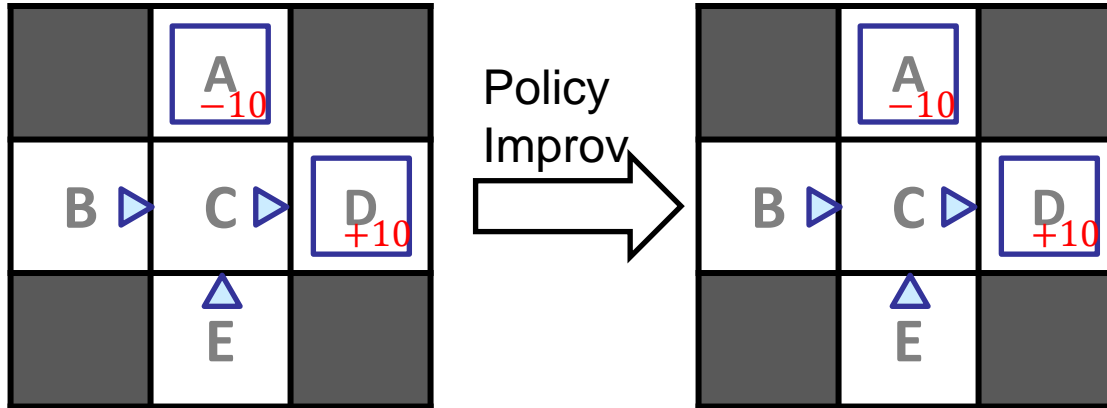
- Plug in values from PE:  $v_\pi(C) = -6, v_\pi(B) = -10, v_\pi(E) = -10$ , to get new policy  $\pi'$
- $\pi'(C) = \operatorname{argmax}_{a \in \{l, r, u, d\}} q_\pi(C, a) = r$ 
  - $q_\pi(C, l) = -1 + 1.0v_\pi(B) = -1 - 10 = -11$
  - $q_\pi(C, r) = -1 + 1.0v(D) = -1 + 10 = 9$
  - $q_\pi(C, u) = -1 + 1.0v(A) = -1 - 10 = -11$
  - $q_\pi(C, d) = -1 + 1.0v_\pi(E) = -1 - 10 = -11$
- $\pi'(B) = \operatorname{argmax}_{a \in \{l, r, u, d\}} q_\pi(B, a) = r$ 
  - $q_\pi(B, l) = q_\pi(B, u) = q_\pi(B, d) = -1 + 1.0v_\pi(B) = -1 - 10 = -11$
  - $q_\pi(B, r) = -1 + 1.0v_\pi(C) = -1 - 6 = -7$
- $\pi'(E) = \operatorname{argmax}_{a \in \{l, r, u, d\}} q_\pi(E, a) = u$ 
  - $q_\pi(E, l) = q_\pi(E, r) = q_\pi(E, d) = -1 + 1.0v_\pi(E) = -1 - 10 = -11$
  - $q_\pi(E, u) = -1 + 1.0v_\pi(C) = -1 - 6 = -7$

# Iter2 Policy Evaluation of Det Policy



- Det policy:  $\pi(r|B) = 1; \pi(r|C) = 1; \pi(u|C) = 1$
- Policy Evaluation for Det Env:  $v_\pi(s) = \sum_a \pi(a|s) q_\pi(s, a); q_\pi(s, a) = r + \gamma v_\pi(s')$
- Set of equations:
- $v_\pi(C) = 1.0 q_\pi(C, r)$ 
  - $q_\pi(C, r) = -1 + 1.0 v_\pi(D) = -1 + 1.0 \cdot 10 = 9$
- $v_\pi(B) = 1.0 q_\pi(B, r)$ 
  - $q_\pi(B, r) = -1 + 1.0 v_\pi(C)$
- $v_\pi(E) = 1.0 q_\pi(E, u)$ 
  - $q_\pi(E, u) = -1 + 1.0 v_\pi(C)$
- Solution:
  - $v_\pi(C) = 9; v_\pi(B) = 8; v_\pi(E) = 8$

# Iter2 Policy Improvement

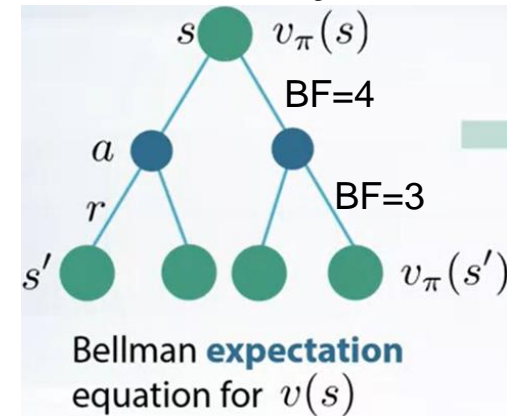
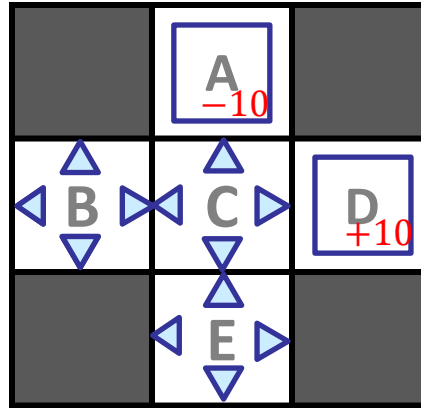


- Plug in values from PE:  $v_\pi(C) = 9, v_\pi(B) = 8, v_\pi(E) = 8$ , to get new policy  $\pi'$
- $\pi'(C) = \operatorname{argmax}_{a \in \{l, r, u, d\}} q_\pi(C, a) = r$ 
  - $q_\pi(C, l) = -1 + 1.0v_\pi(B) = -1 + 8 = 7$
  - $q_\pi(C, r) = -1 + 1.0v_\pi(D) = -1 + 10 = 9$
  - $q_\pi(C, u) = -1 + 1.0v_\pi(A) = -1 - 10 = -11$
  - $q_\pi(C, d) = -1 + 1.0v_\pi(E) = -1 + 8 = 7$
- $\pi'(B) = \operatorname{argmax}_{a \in \{l, r, u, d\}} q_\pi(B, a) = r$ 
  - $q_\pi(B, l) = q_\pi(B, u) = q_\pi(B, d) = -1 + 1.0v_\pi(B) = 7$
  - $q_\pi(B, r) = -1 + 1.0v_\pi(C) = -1 + 9 = 8$
- $\pi'(E) = \operatorname{argmax}_{a \in \{l, r, u, d\}} q_\pi(E, a) = u$ 
  - $q_\pi(E, l) = q_\pi(E, r) = q_\pi(E, d) = -1 + 1.0v_\pi(E) = 7$
  - $q_\pi(E, u) = -1 + 1.0v_\pi(C) = -1 + 9 = 8$
- New policy is now stable ( $\pi' = \pi$ ), so we have found the optimal policy.

# MiniGW Example

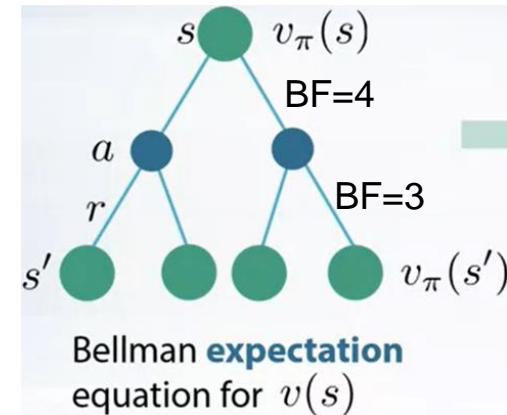
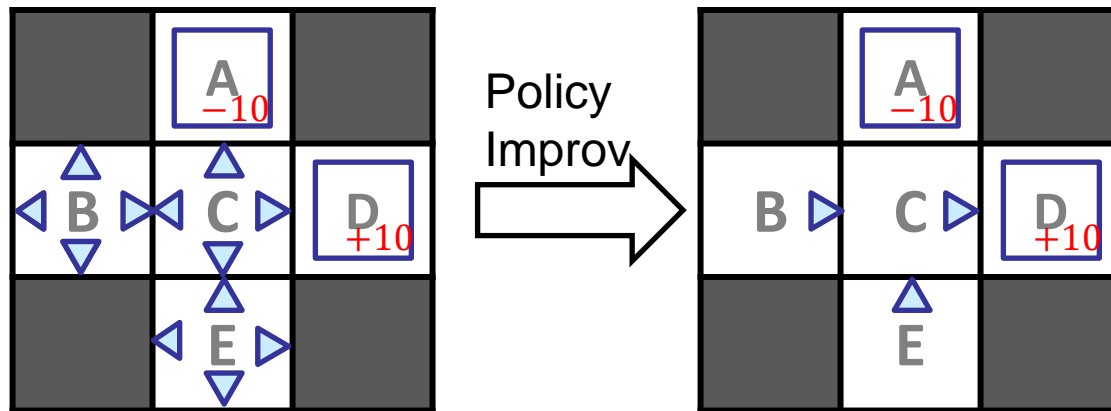
- Policy Iteration for Deterministic Environment
- Policy Iteration for Stochastic Environment
  - Policy Evaluation with analytic solution, not IPE
- Value Iteration for Deterministic Environment
- Value Iteration for Stochastic Environment
  - Bellman Opt. Equation solved iteratively w. in-place updates
- Model-based Learning

# Iter1 Policy Evaluation of Random Policy



- Random policy:  $\pi(l|s) = \pi(r|s) = \pi(u|s) = \pi(d|s) = 0.25$
- Policy Evaluation:  $v_\pi(s) = \sum_a \pi(a|s) q_\pi(s, a)$ ;  $q_\pi(s, a) = \sum_{r, s'} p(r, s'|s, a) [r + \gamma v_\pi(s')]$
- Set of equations:
- $v_\pi(C) = 0.25(q_\pi(C, l) + q_\pi(C, r) + q_\pi(C, u) + q_\pi(C, d)) = 0.25(-4 + v_\pi(B) + v_\pi(E))$ 
  - $q_\pi(C, l) = -1 + 0.8v_\pi(B) + 0.1v(A) + 0.1v_\pi(E) = -2 + 0.8v_\pi(B) + 0.1v_\pi(E)$
  - $q_\pi(C, r) = -1 + 0.8v(D) + 0.1v(A) + 0.1v_\pi(E) = 6 + 0.1v_\pi(E)$
  - $q_\pi(C, u) = -1 + 0.8v(A) + 0.1v_\pi(B) + 0.1v(D) = -8 + 0.1v_\pi(B)$
  - $q_\pi(C, d) = -1 + 0.8v_\pi(E) + 0.1v_\pi(B) + 0.1v(D) = 0.8v_\pi(E) + 0.1v_\pi(B)$
- $v_\pi(B) = 0.25(q_\pi(B, l) + q_\pi(B, r) + q_\pi(B, u) + q_\pi(B, d)) = 0.25(-4 + 3.0v_\pi(B) + 1.0v_\pi(C))$ 
  - $q_\pi(B, l) = -1 + 1.0v_\pi(B)$
  - $q_\pi(B, r) = -1 + 0.8v_\pi(C) + 0.2v_\pi(B)$
  - $q_\pi(B, u) = q_\pi(B, d) = -1 + 0.9v_\pi(B) + 0.1v_\pi(C)$
- $v_\pi(E) = 0.25(q_\pi(E, l) + q_\pi(E, r) + q_\pi(E, u) + q_\pi(E, d)) = 0.25(-4 + 3.0v_\pi(E) + 1.0v_\pi(C))$ 
  - $q_\pi(E, l) = q_\pi(B, r) = -1 + 0.9v_\pi(E) + 0.1v_\pi(C)$
  - $q_\pi(E, u) = -1 + 0.8v_\pi(C) + 0.2v_\pi(E)$
  - $q_\pi(E, d) = -1 + 1.0v_\pi(E)$
- Solution:
  - $V_\pi(C) = -6, V_\pi(B) = -10, V_\pi(E) = -10$

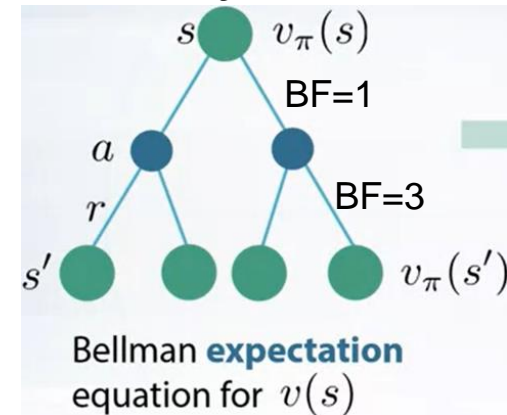
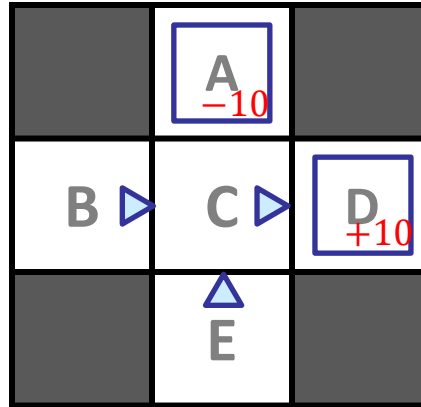
# Iter1 Policy Improvement



- Plug in values from PE:  $v_\pi(C) = -6, v_\pi(B) = -10, v_\pi(E) = -10$ , to get new policy  $\pi'$
- $\pi'(C) = \operatorname{argmax}_{a \in \{l, r, u, d\}} q_\pi(C, a) = r$ 
  - $q_\pi(C, l) = -1 + 1.0v_\pi(B) = -1 - 10 = -11$
  - $q_\pi(C, r) = -1 + 1.0v_\pi(D) = -1 + 10 = 9$
  - $q_\pi(C, u) = -1 + 1.0v_\pi(A) = -1 - 10 = -11$
  - $q_\pi(C, d) = -1 + 1.0v_\pi(E) = -1 - 10 = -11$
- $\pi'(B) = \operatorname{argmax}_{a \in \{l, r, u, d\}} q_\pi(B, a) = r$ 
  - $q_\pi(B, l) = q_\pi(B, u) = q_\pi(B, d) = -1 + 1.0v_\pi(B) = -11$
  - $q_\pi(B, r) = -1 + 1.0v_\pi(C) = -1 - 6 = -7$
- $\pi'(E) = \operatorname{argmax}_{a \in \{l, r, u, d\}} q_\pi(E, a) = u$ 
  - $q_\pi(E, l) = q_\pi(E, r) = q_\pi(E, d) = -1 + 1.0v_\pi(E) = -11$
  - $q_\pi(E, u) = -1 + 1.0v_\pi(C) = -1 - 6 = -7$

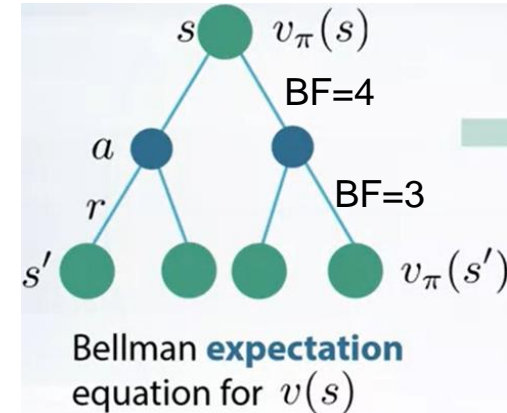
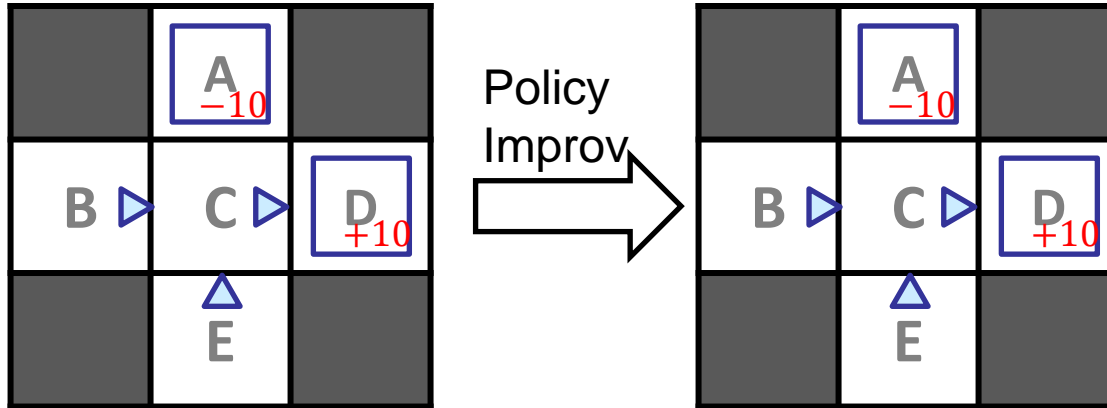


# Iter2 Policy Evaluation of Det Policy



- Det policy:  $\pi(r|B) = 1; \pi(r|C) = 1; \pi(u|C) = 1$
- Policy Evaluation:  $v_{\pi}(s) = \sum_a \pi(a|s) q_{\pi}(s, a); q_{\pi}(s, a) = r + \gamma \sum_{s'} p(s'|s, a) v_{\pi}(s')$
- Set of equations:
- $v_{\pi}(C) = 1.0 q_{\pi}(C, r)$ 
  - $q_{\pi}(C, r) = -1 + 0.8 v_{\pi}(D) + 0.1 v_{\pi}(A) + 0.1 v_{\pi}(E) = -1 + 0.8 \cdot 10 + 0.1(-10) + 0.1 v_{\pi}(E) = 6 + 0.1 v_{\pi}(E)$
- $v_{\pi}(B) = 1.0 q_{\pi}(B, r)$ 
  - $q_{\pi}(B, r) = -1 + 0.8 v_{\pi}(C) + 0.2 v_{\pi}(B)$
- $v_{\pi}(E) = 1.0 q_{\pi}(E, u)$ 
  - $q_{\pi}(E, u) = -1 + 0.8 v_{\pi}(C) + 0.2 v_{\pi}(E)$
- Solution:
  - $V_{\pi}(C) = 6.53, V_{\pi}(B) = 5.3, V_{\pi}(E) = 5.3$

# Iter2 Policy Improvement

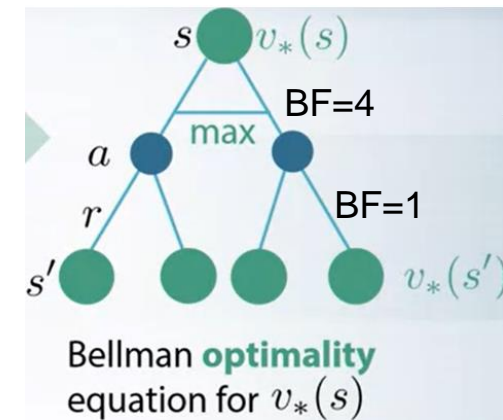
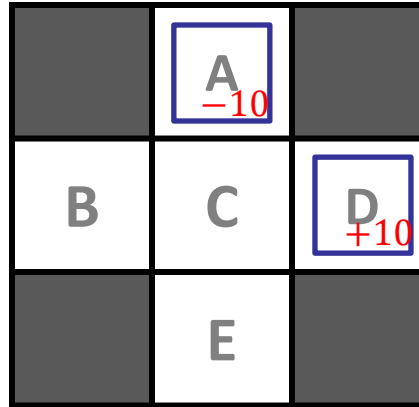


- Plugging in values from PE:  $V_\pi(C) = 6.53, V_\pi(B) = 5.3, V_\pi(E) = 5.3$ , we get new policy  $\pi'$
- $\pi'(C) = \operatorname{argmax}_{a \in \{l, r, u, d\}} q_\pi(C, a) = r$ 
  - $q_\pi(C, l) = -1 + 1.0v_\pi(B) = -1 + 5.3 = 4.3$
  - $q_\pi(C, r) = -1 + 1.0v_\pi(D) = -1 + 10 = 9$
  - $q_\pi(C, u) = -1 + 1.0v_\pi(A) = -1 - 10 = -11$
  - $q_\pi(C, d) = -1 + 1.0v_\pi(E) = -1 + 5.3 = 4.3$
- $\pi'(B) = \operatorname{argmax}_{a \in \{l, r, u, d\}} q_\pi(B, a) = r$ 
  - $q_\pi(B, l) = q_\pi(B, u) = q_\pi(B, d) = -1 + 1.0v_\pi(B) = -1 + 5.3 = 4.3$
  - $q_\pi(B, r) = -1 + 1.0v_\pi(C) = -1 + 6.53 = 5.53$
- $\pi'(E) = \operatorname{argmax}_{a \in \{l, r, u, d\}} q_\pi(E, a) = u$ 
  - $q_\pi(E, l) = q_\pi(E, r) = q_\pi(E, d) = -1 + 1.0v_\pi(E) = 4.3$
  - $q_\pi(E, u) = -1 + 1.0v_\pi(C) = -1 + 6.53 = 5.53$
- New policy is now stable ( $\pi' = \pi$ ), so we have found the optimal policy.

# MiniGW Example

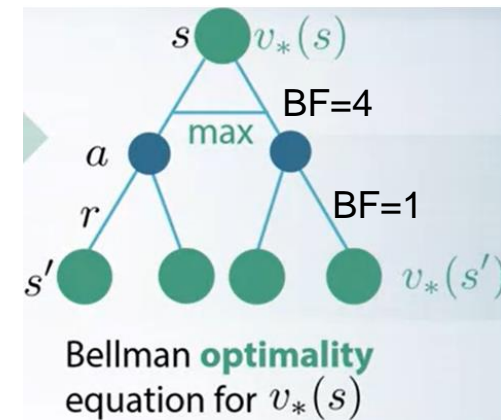
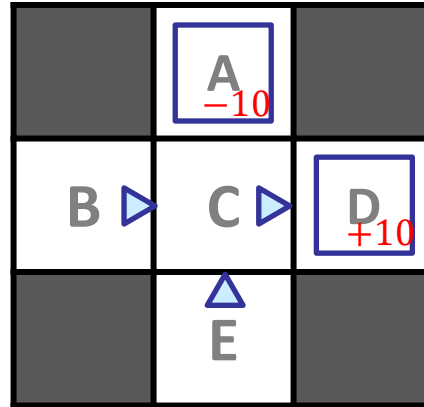
- Policy Iteration for Deterministic Environment
- Policy Iteration for Stochastic Environment
  - Policy Evaluation with analytic solution, not IPE
- Value Iteration for Deterministic Environment
- Value Iteration for Stochastic Environment
  - Bellman Opt. Equation solved iteratively w. in-place updates
- Model-based Learning

# Value Iteration w. Det Env: 1<sup>st</sup> Iteration



- Value Iteration:  $v_{k+1}(s) = \max_a q_{k+1}(s, a) = \max_a [r + \gamma v_k(s')]$  w. **in-place updates**.
- Initialize  $v_0(B) = v_0(C) = v_0(E) = 0$ .
- $v_1(C) = \max_a q_1(C, a) = \max(-1, 9, -11, -1) = 9$ 
  - $q_1(C, l) = r + 1.0v_0(B) = -1 + 1.0 \cdot 0 = -1$
  - $q_1(C, r) = r + 1.0v(D) = -1 + 1.0 \cdot 10 = 9$
  - $q_1(C, u) = r + 1.0v(A) = -1 + 1.0(-10) = -11$
  - $q_1(C, d) = r + 1.0v_0(E) = -1 + 1.0 \cdot 0 = -1$
- $v_1(B) = \max_a q_1(B, a) = \max(-1, 8, -1, -1) = 8$ 
  - $q_1(B, l) = q_1(B, u) = q_1(B, d) = r + 1.0v_0(B) = -1 + 1.0 \cdot 0 = -1$
  - $q_1(B, r) = r + 1.0v_1(C) = -1 + 1.0 \cdot 9 = 8$  ( $v_1(C)$  computed in the current iteration is used instead of  $v_0(C)$ )
- $v_1(E) = \max_a q_1(E, a) = \max(-1, 8, -1, -1) = 8$ 
  - $q_1(E, l) = q_1(E, r) = q_1(E, d) = r + 1.0v_0(E) = -1 + 1.0 \cdot 0 = -1$
  - $q_1(E, u) = r + 1.0v_1(C) = -1 + 1.0 \cdot 9 = 8$  ( $v_1(C)$  is used)

# Value Iteration w. Det Env: 2<sup>nd</sup> Iteration



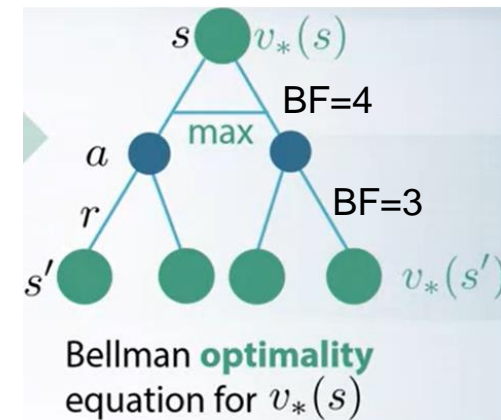
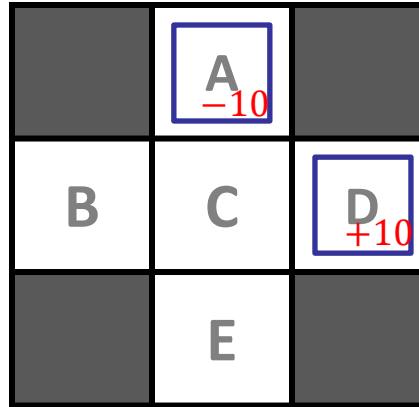
- Value Iteration:  $v_{k+1}(s) = \max_a q_{k+1}(s, a) = \max_a [r + \gamma v_k(s')]$  w. **in-place updates**.
- We have now  $v_1(B) = v_1(E) = 8, v_1(C) = 9$ .
- $v_2(C) = \max_a q_2(C, a) = \max(7, 9, -11, 7) = 9$ 
  - $q_2(C, l) = r + 1.0v_1(B) = -1 + 1.0 \cdot 8 = 7$
  - $q_2(C, r) = r + 1.0v(D) = -1 + 1.0 \cdot 10 = 9$
  - $q_2(C, u) = r + 1.0v(A) = -1 + 1.0(-10) = -11$
  - $q_2(C, d) = r + 1.0v_1(E) = -1 + 1.0 \cdot 8 = 7$
- $v_2(B) = \max_a q_2(B, a) = \max(7, 8, 7, 7) = 8$ 
  - $q_2(B, l) = q_2(B, u) = q_2(B, d) = r + 1.0v_1(B) = -1 + 1.0 \cdot 8 = 7$
  - $q_2(B, r) = r + 1.0v_2(C) = -1 + 1.0 \cdot 9 = 8$  ( $v_2(C)$  is used instead of  $v_1(C)$ )
- $v_2(E) = \max_a q_2(E, a) = \max(7, 8, 7, 7) = 8$ 
  - $q_2(E, l) = q_2(E, r) = q_2(E, d) = r + 1.0v_1(E) = -1 + 1.0 \cdot 0 = -1$
  - $q_2(E, u) = r + 1.0v_2(C) = -1 + 1.0 \cdot 9 = 8$  ( $v_2(C)$  is used)
- Value iteration has converged, so  $v_*(\cdot) = v_2(\cdot)$ . We can get the optimal policy:
  - $\pi_*(C) = \operatorname{argmax}_{a \in \{l, r, u, d\}} q_*(C, a) = r$ ;  $\pi_*(B) = \operatorname{argmax}_{a \in \{l, r, u, d\}} q_*(B, a) = r$ ;  $\pi_*(E) = \operatorname{argmax}_{a \in \{l, r, u, d\}} q_*(E, a) = u$

- $v_*(s) = \max_a q_*(s, a); q_*(s, a) = \sum_{r,s'} p(r, s'|s, a) [r + \gamma v_*(s')]$
- $q_*(C, l) = \sum_{r,s'} p(r, s'|s, a) [r + \gamma v_*(s')]$

# MiniGW Example

- Policy Iteration for Deterministic Environment
- Policy Iteration for Stochastic Environment
  - Policy Evaluation with analytic solution, not IPE
- Value Iteration for Deterministic Environment
- Value Iteration for Stochastic Environment
  - Bellman Opt. Equation solved iteratively w. in-place updates
- Model-based Learning

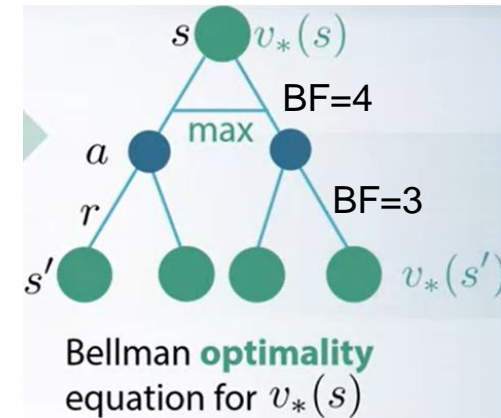
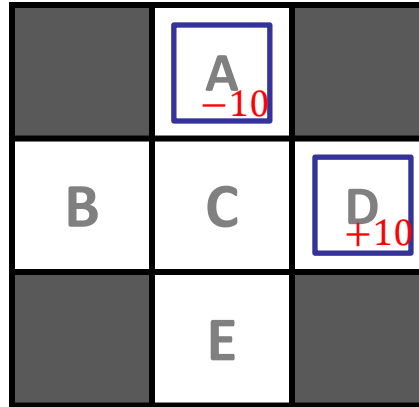
# Value Iteration w. Sto Env: 2<sup>nd</sup> Iteration



- Value Iteration:  $v_{k+1}(s) = \max_a q_{k+1}(s, a) = \max_a [r + \gamma \sum_{s'} p(s'|s, a) v_k(s')]$  w. **in-place updates**.
- We have now:  $v_1(B) = v_1(E) = 3.8, v_1(C) = 6$ .
- $v_2(C) = \max_a q_2(C, a) = \max(-2, 6, -8, 0) = 6$ 
  - $q_2(C, l) = r + .8v_1(B) + .1v(A) + .1v_1(E) = -1 + .8 \cdot 3.8 + .1(-10) + .1 \cdot 3.8 = 1.42$
  - $q_2(C, r) = r + .8v(D) + .1v(A) + .1v_0(E) = -1 + .8 \cdot 10 + .1(-10) + .1 \cdot 3.8 = 6$
  - $q_2(C, u) = r + .8v(A) + .1v_1(B) + .1v(D) = -1 + .8(-10) + .1 \cdot 0 + .1 \cdot 10 = -8$
  - $q_2(C, d) = r + .8v_1(E) + .1v_1(B) + .1v(D) = -1 + .8 \cdot 3.8 + .1 \cdot 0 + .1 \cdot 10 = 3.04$
- $v_2(B) = \max_a q_2(B, a) = \max(2.8, 4.56, 3.02, 3.02) = 4.56$ 
  - $q_2(B, l) = r + 1.0v_1(B) = -1 + 1.0 \cdot 3.8 = 2.8$
  - $q_2(B, r) = r + .8v_2(C) + .2v_1(B) = -1 + .8 \cdot 6 + .2 \cdot 3.8 = 4.56$
  - $q_2(B, u) = q_2(B, d) = r + .9v_1(B) + .1v_1(C) = -1 + .9 \cdot 3.8 + .1 \cdot 6 = 3.02$
- $v_2(E) = \max_a q_2(E, a) = \max(2.8, 4.56, 3.02, 3.02) = 4.56$ 
  - $q_1(E, d) = r + 1.0v_1(E) = -1 + 1.0 \cdot 3.8 = 2.8$
  - $q_2(E, u) = r + 0.8v_2(C) + 0.2v_1(E) = -1 + 0.8 \cdot 6 + 0.2 \cdot 3.8 = 4.56$
  - $q_2(E, l) = q_2(E, r) = r + .9v_1(E) + .1v_2(C) = -1 + 0.9 \cdot 3.8 + .1 \cdot 6 = 3.02$
- 1<sup>st</sup> sweep has converged, so  $q_*(\cdot) = q_1(\cdot)$ . We can get the optimal policy:
  - $\pi_*(C) = \operatorname{argmax}_{a \in \{l, r, u, d\}} q_*(C, a) = r$ ;  $\pi_*(B) = \operatorname{argmax}_{a \in \{l, r, u, d\}} q_*(B, a) = r$ ;  $\pi_*(E) = \operatorname{argmax}_{a \in \{l, r, u, d\}} q_*(E, a) = u$

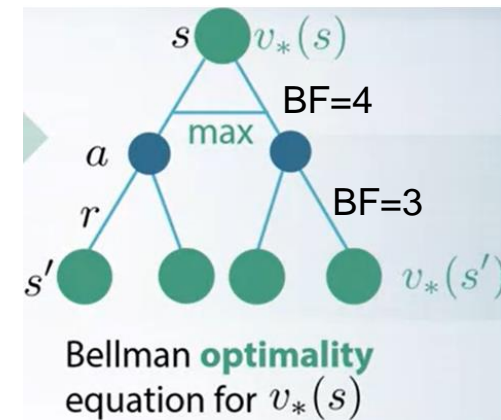
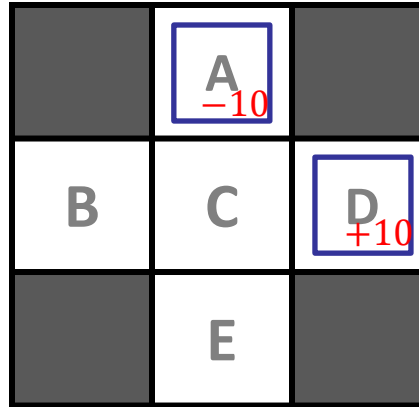


# Value Iteration w. Sto Env: 1<sup>st</sup> Iteration



- Value Iteration:  $v_{k+1}(s) = \max_a q_{k+1}(s, a) = \max_a [r + \gamma \sum_{s'} p(s'|s, a) v_k(s')]$  w. **in-place updates**.
- Initialize  $v_0(B) = v_0(C) = v_0(E) = 0$ .
- $v_1(C) = \max_a q_1(C, a) = \max(-2, 6, -8, 0) = 6$ 
  - $q_1(C, l) = r + 0.8v_0(B) + 0.1v_0(A) + 0.1v_0(E) = -1 + 0.8 \cdot 0 + 0.1(-10) + 0.1 \cdot 0 = -2$
  - $q_1(C, r) = r + 0.8v_0(D) + 0.1v_0(A) + 0.1v_0(E) = -1 + 0.8 \cdot 10 + 0.1(-10) + 0.1 \cdot 0 = 6$
  - $q_1(C, u) = r + 0.8v_0(A) + 0.1v_0(B) + 0.1v_0(D) = -1 + 0.8(-10) + 0.1 \cdot 0 + 0.1 \cdot 10 = -8$
  - $q_1(C, d) = r + 0.8v_0(E) + 0.1v_0(B) + 0.1v_0(D) = -1 + 0.8 \cdot 0 + 0.1 \cdot 0 + 0.1 \cdot 10 = 0$
- $v_1(B) = \max_a q_1(B, a) = \max(-1, 3.8, -0.4, -0.4) = 3.8$ 
  - $q_1(B, l) = r + 1.0v_0(B) = -1 + 1.0 \cdot 0 = -1$
  - $q_1(B, r) = r + 0.8v_1(C) + 0.2v_0(B) = -1 + 0.8 \cdot 6 + 0.2 \cdot 0 = 3.8$
  - $q_1(B, u) = q_1(B, d) = r + 0.9v_0(B) + 0.1v_1(C) = -1 + 0.9 \cdot 0 + 0.1 \cdot 6 = -0.4$
- $v_1(E) = \max_a q_1(E, a) = \max(-1, 3.8, -0.4, -0.4) = 3.8$ 
  - $q_1(E, l) = q_1(E, r) = r + 0.9v_0(E) + 0.1v_1(C) = -1 + 0.9 \cdot 0 + 0.1 \cdot 6 = -0.4$
  - $q_1(E, u) = r + 0.8v_1(C) + 0.2v_0(E) = -1 + 0.8 \cdot 6 + 0.2 \cdot 0 = 3.8$
  - $q_1(E, d) = r + 1.0v_0(E) = -1 + 1.0 \cdot 0 = -1$

# Value Iteration w. Sto Env: 2<sup>nd</sup> Iteration



- Value Iteration:  $v_{k+1}(s) = \max_a q_{k+1}(s, a) = \max_a [r + \gamma \sum_{s'} p(s'|s, a) v_k(s')]$  w. **in-place updates**.
- We have now:  $v_1(B) = v_1(E) = 3.8, v_1(C) = 6$ .
- $v_2(C) = \max_a q_2(C, a) = \max(1.42, 6.38, -7.62, 3.42) = 6.38$ 
  - $q_2(C, l) = r + .8v_1(B) + .1v(A) + .1v_1(E) = -1 + .8 \cdot 3.8 + .1(-10) + .1 \cdot 3.8 = 1.42$
  - $q_2(C, r) = r + .8v(D) + .1v(A) + .1v_1(E) = -1 + .8 \cdot 10 + .1(-10) + .1 \cdot 3.8 = 6.38$
  - $q_2(C, u) = r + .8v(A) + .1v_1(B) + .1v(D) = -1 + .8(-10) + .1 \cdot 3.8 + .1 \cdot 10 = -7.62$
  - $q_2(C, d) = r + .8v_1(E) + .1v_1(B) + .1v(D) = -1 + .8 \cdot 3.8 + .1 \cdot 3.8 + .1 \cdot 10 = 3.42$
- $v_2(B) = \max_a q_2(B, a) = \max(2.8, 4.864, 3.058, 3.058) = 4.864$ 
  - $q_2(B, l) = r + 1.0v_1(B) = -1 + 1.0 \cdot 3.8 = 2.8$
  - $q_2(B, r) = r + .8v_2(C) + .2v_1(B) = -1 + .8 \cdot 6.38 + .2 \cdot 3.8 = 4.864$
  - $q_2(B, u) = q_2(B, d) = r + .9v_1(B) + .1v_2(C) = -1 + .9 \cdot 3.8 + .1 \cdot 6.38 = 3.058$
- $v_2(E) = \max_a q_2(E, a) = \max(2.8, 4.864, 3.058, 3.058) = 4.864$ 
  - $q_1(E, d) = r + 1.0v_1(E) = -1 + 1.0 \cdot 3.8 = 2.8$
  - $q_2(E, u) = r + 0.8v_2(C) + 0.2v_1(E) = -1 + 0.8 \cdot 6.38 + 0.2 \cdot 3.8 = 4.864$
  - $q_2(E, l) = q_2(E, r) = r + .9v_1(E) + .1v_2(C) = -1 + 0.9 \cdot 3.8 + .1 \cdot 6.38 = 3.058$

# Value Iteration w. Sto Env: after Convergence

- Value functions  $v_*(s)$  converge after 6 iterations (shown below), with  $v_*(C) = 6.53$ ,  $v_*(B) = v_*(C) = 5.28$ . We can get the optimal policy:
  - $\pi_*(C) = \operatorname{argmax}_{a \in (l,r,u,d)} q_*(C, a) = \operatorname{argmax}_{a \in (l,r,u,d)} \sum_{r,s'} p(r, s' | s, a) [r + \gamma v_*(s')] = r$ 
    - We can obtain  $q_*(s, a)$  from  $v_*(s)$ , but we can also keep track of  $q_*(s, a)$ , which are computed during the process of computing  $v_*(s)$ .
  - $\pi_*(B) = \operatorname{argmax}_{a \in (l,r,u,d)} q_*(B, a) = r$ ;
  - $\pi_*(E) = \operatorname{argmax}_{a \in (l,r,u,d)} q_*(E, a) = u$

values:

```
-----
0.00|-10.00| 0.00|
-----
3.80| 6.00| 10.00|
-----
0.00| 3.80| 0.00|
```

values:

```
-----
0.00|-10.00| 0.00|
-----
5.25| 6.52| 10.00|
-----
0.00| 5.25| 0.00|
```

values:

```
-----
0.00|-10.00| 0.00|
-----
4.86| 6.38| 10.00|
-----
0.00| 4.86| 0.00|
```

values:

```
-----
0.00|-10.00| 0.00|
-----
5.27| 6.52| 10.00|
-----
0.00| 5.27| 0.00|
```

values:

```
-----
0.00|-10.00| 0.00|
-----
5.16| 6.49| 10.00|
-----
0.00| 5.16| 0.00|
```

values:

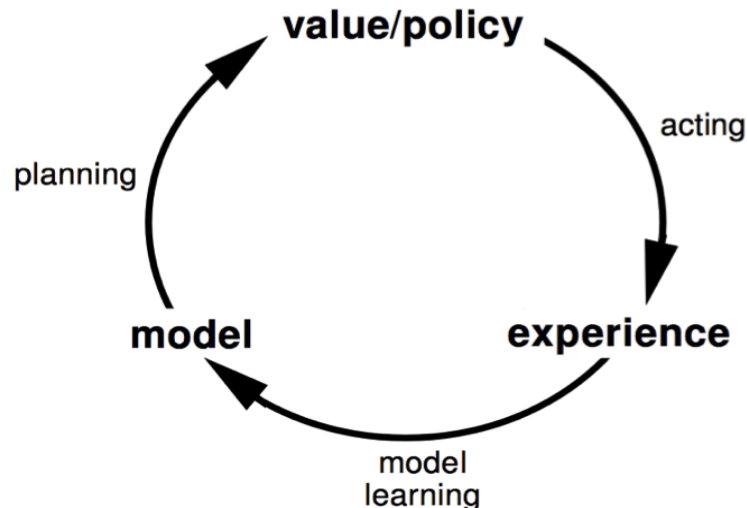
```
-----
0.00|-10.00| 0.00|
-----
5.28| 6.53| 10.00|
-----
0.00| 5.28| 0.00|
```

# MiniGW Example

- Policy Iteration for Deterministic Environment
- Policy Iteration for Stochastic Environment
  - Policy Evaluation with analytic solution, not IPE
- Value Iteration for Deterministic Environment
- Value Iteration for Stochastic Environment
  - Bellman Opt. Equation solved iteratively w. in-place updates
- **Model-based Learning**

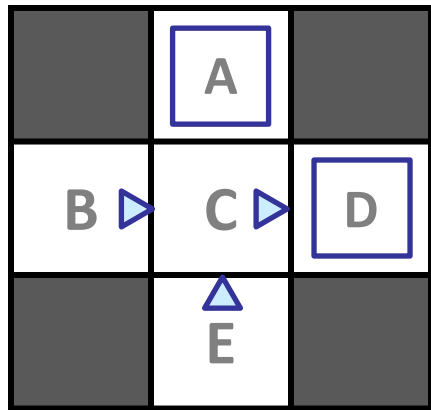
# Model-Based RL

- If MDP is not available, we can use Model-Based RL:
- Step 1: Learn empirical MDP model
  - Estimate the model  $p(r, s' | s, a)$  by executing some policy  $\pi$  (maybe random), and keeping track of outcomes  $r, s'$  for each  $s, a$  in the observed episodes.
- Step 2: Do planning w. the learned MDP for the optimal policy
  - Dynamic Programming w. Value Iteration or Policy Iteration



# MiniGW: Model Learning

Input Policy  $\pi$



Assume:  $\gamma = 1$

Observed Episodes (Training)

Episode 1

$(B, r, C, -1)$   
 $(C, r, D, -1)$   
 $(D, \text{exit}, x, 10)$

Episode 2

$(B, r, C, -1)$   
 $(C, r, D, -1)$   
 $(D, \text{exit}, x, +10)$

Episode 3

$(E, u, C, -1)$   
 $(C, r, D, -1)$   
 $(D, \text{exit}, x, 10)$

Episode 4

$(E, u, C, -1)$   
 $(C, r, A, -1)$   
 $(A, \text{exit}, x, -10)$

Learned Model

$p(s'|s, a)$

$p(-1, C|B, r) = 1.0$   
 $p(-1, D|C, r) = 0.75$   
 $p(-1, A|C, r) = 0.25$   
 $p(10, x|D, \text{exit}) = 1.0$   
 $p(-10, x|A, \text{exit}) = 1.0$

- In the 4 episodes, we see 4 transitions from  $(s = C, a = r)$ . 3 of them go to next state  $s' = D$ , and one goes to next state  $s' = A$ , each w. reward  $-1$ . Hence  $p(-1, D|C, r) = 0.75$ ;  $p(-1, A|C, r) = 0.25$ .