EECE 5550
Mobile Robotics

Lecture 15: Markov Decision Processes

Derya Aksaray

Assistant Professor

Department of Electrical and Computer Engineering

Northeastern University

Recap

Planning

- A*: find the shortest path between two points
- We assume that any action has a known predictable outcome (go straight results in going straight)
- Is it true in reality?

Probabilistic robotics

- Can accommodate uncertainty in the process model and measurement model
- Beliefs in state estimation

Planning under uncertainty?

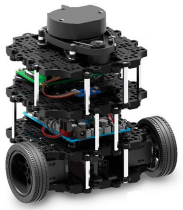
Deterministic vs. Stochastic

When an event occurs,

- **Deterministic**: there is only one outcome.
- **Stochastic:** there are multiple outcomes.

In the context of planning,

Deterministic



Move west

It always moves west

Stochastic

Due to some gust, actuation uncertainty, or variability on the terrain

Moves northwest

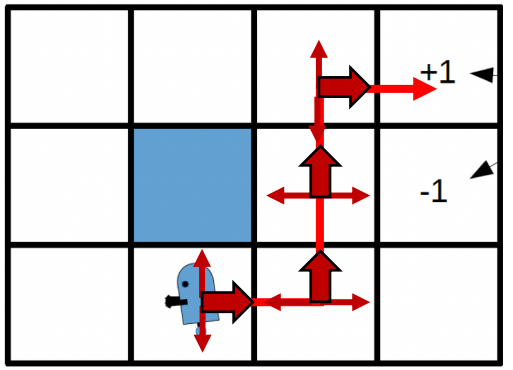
Moves west

Moves southwest

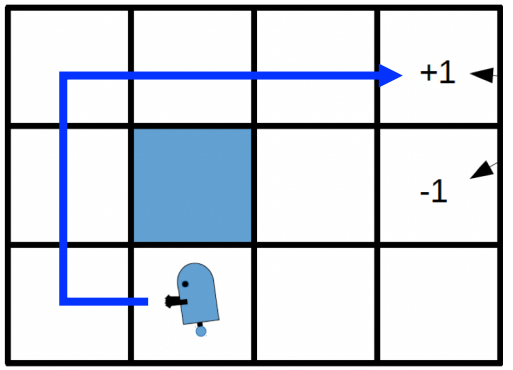
Planning example

- Desirable state (+1), undesirable state (-1), blue (obstacle)
- If we use A*:
- Now suppose that the robot has a stochastic dynamical model,

Desired action: Up



*no penalty in bumping the obstacle

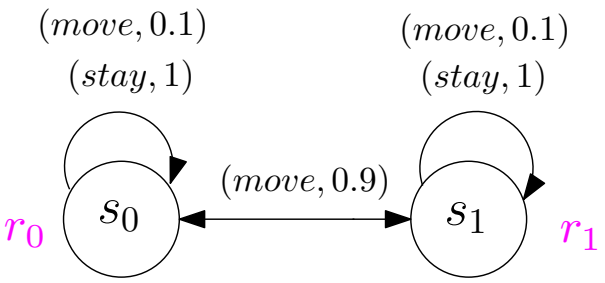


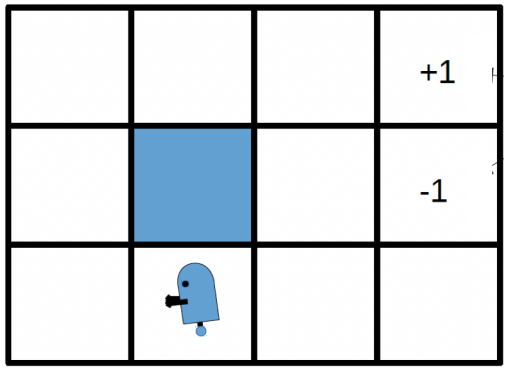
- A path that avoids the undesirable state can be longer:
- Uncertainty in dynamics should be considered in planning
  - Next: Markov Decision Process

Markov Decision Process (MDP)

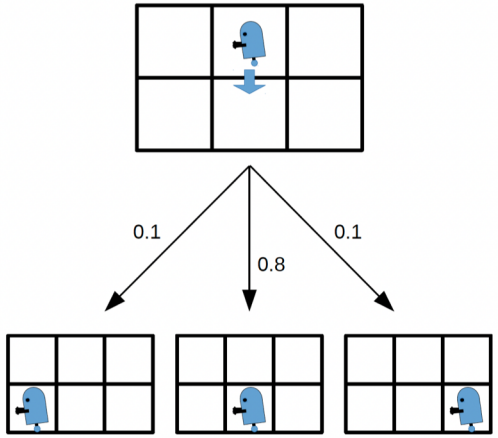
An MDP is a tuple $M = (S, A, P, R)$

- S: set of states
- A: set of motion primitives
- P: $S \times A \times S \rightarrow \mathbb{R}_{\geq 0}$ probabilistic transition function
- R: $S \times A \times S \rightarrow \mathbb{R}_{\geq 0}$ reward function

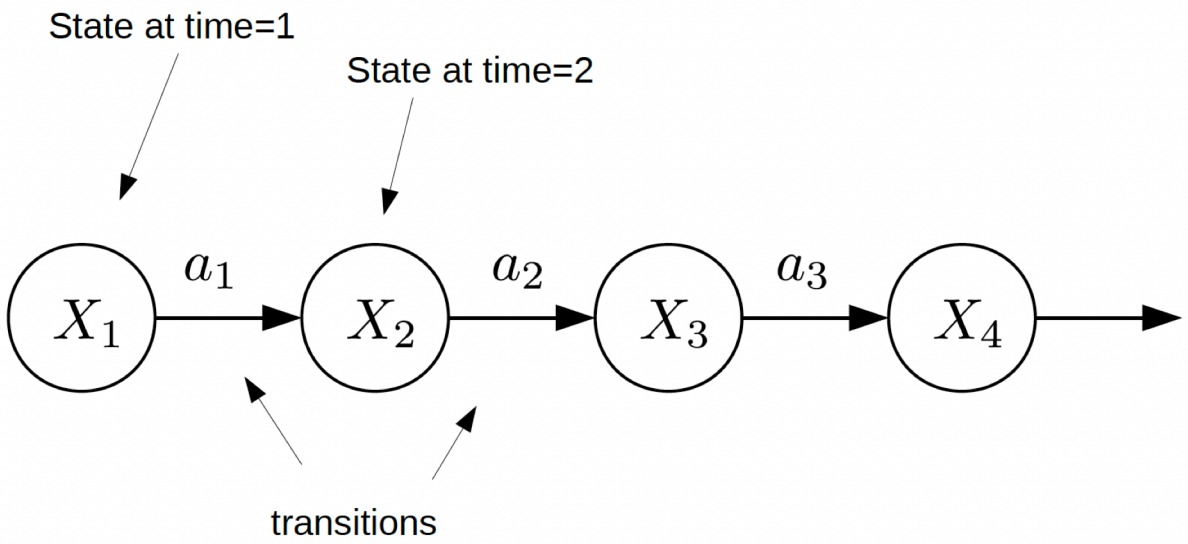




- Each cell is a state
- Up, right, left, down are actions.
- Go in intended direction 80% of the time
- Some states have rewards.



Markov assumption



$$p(X_t|a_{t-1}, X_{t-1}) = p(X_t|a_{t-1}, X_{t-1}, X_{t-2}, \ldots, X_1)$$

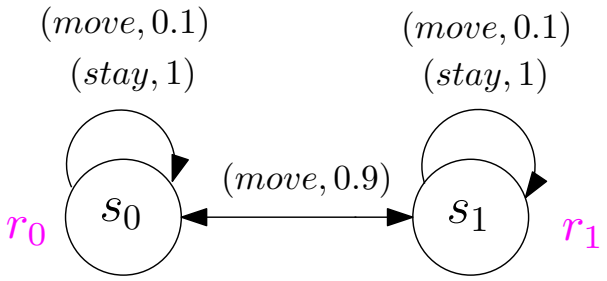
$$X_t \perp\!\!\!\perp X_{t-2}|X_{t-1}$$

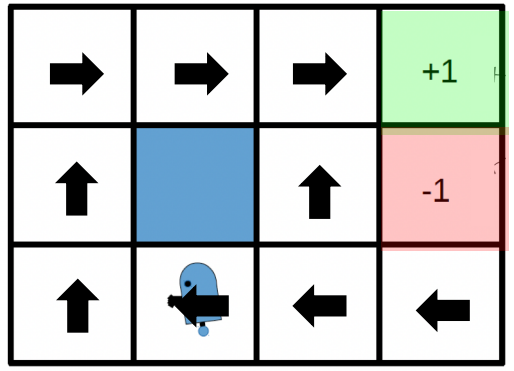
Markov Decision Process (MDP)

An MDP is a tuple $M = (S, A, P, R)$

- S: set of states
- A: set of motion primitives
- P: $S \times A \times S \rightarrow \mathbb{R}_{\geq 0}$ probabilistic transition function
- R: $S \times A \times S \rightarrow \mathbb{R}_{\geq 0}$ reward function

Policy π tells the agent what action to execute





Policy vs. Plan

Policies are more general than plans.

Plan:

- Specifies a sequence of actions to take
- Cannot react to unexpected outcome

Policy

- Tells you what action to take from any state

Example:

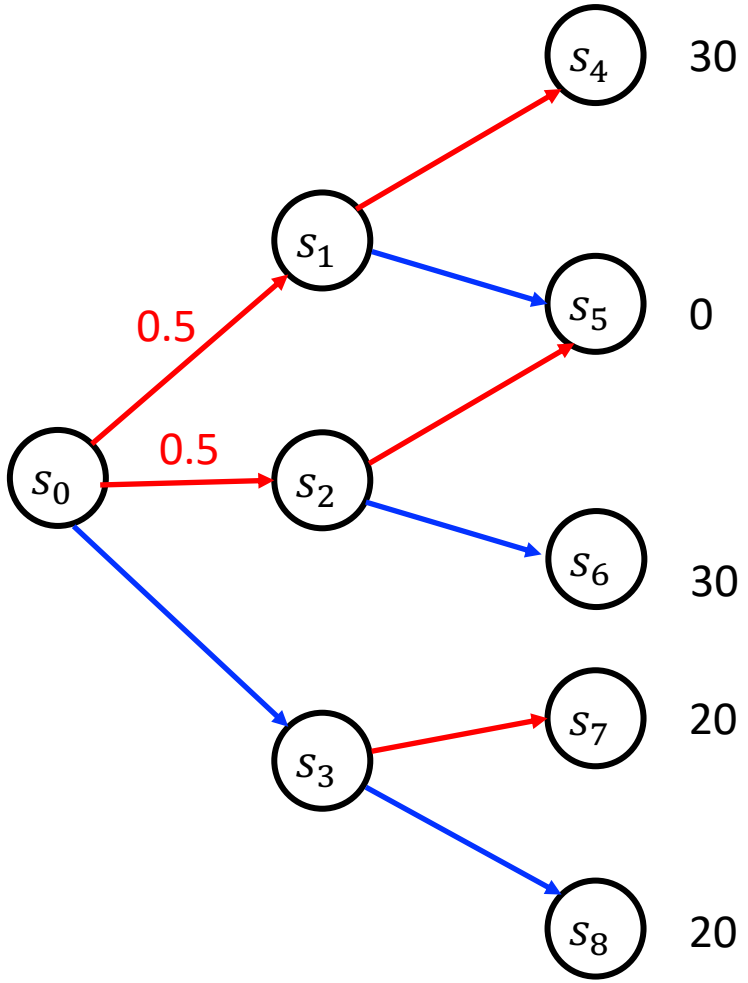
$U(red, red) = 15$

$U(red, blue) = 15$

$U(blue, red) = 20$

$U(blue, blue) = 20$

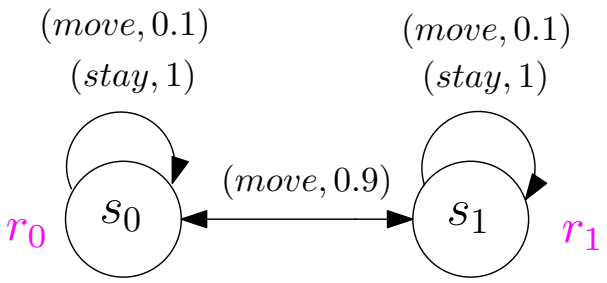
$The\ optimal\ policy\ can\ achieve\ U = 30.$



Markov Decision Process (MDP)

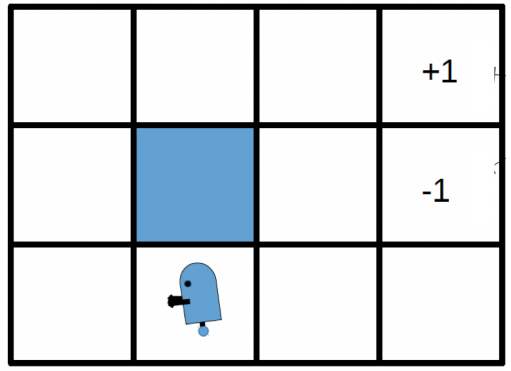
An MDP is a tuple $M = (S, A, P, R)$

- S: set of states
- A: set of motion primitives
- P: $S \times A \times S \rightarrow \mathbb{R}_{\geq 0}$ probabilistic transition function
- R: $S \times A \times S \rightarrow \mathbb{R}_{\geq 0}$ reward function



Policy π tells the agent what action to execute:

- Deterministic, $\pi(s): S \rightarrow A$
  - agent always executes the same action from a given state
- Stochastic, $\pi(a|s): S \rightarrow p(A)$
  - The probability of taking action $a \in A$ at state $s \in S$

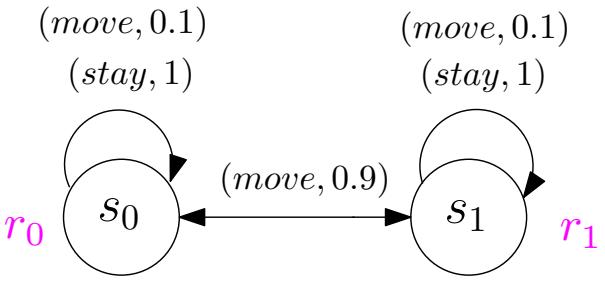


- Stationary policy, $\pi = (\pi, \pi, \pi, \dots)$
- Non-stationary policy, $\pi = (\pi_0, \pi_1, \pi_2, \dots)$

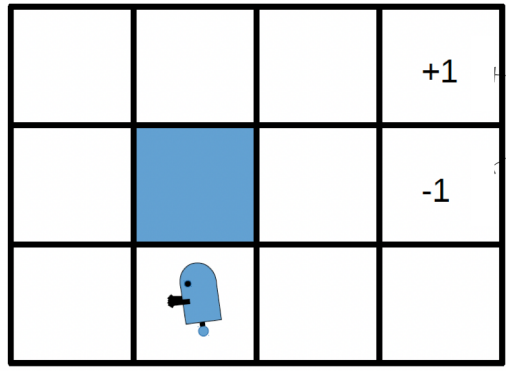
Markov Decision Process (MDP)

An MDP is a tuple $M = (S, A, P, R)$

- S: set of states
- A: set of motion primitives
- P: $S \times A \times S \rightarrow \mathbb{R}_{\geq 0}$ probabilistic transition function
- R: $S \times A \times S \rightarrow \mathbb{R}_{\geq 0}$ reward function



Policy π tells the agent what action to execute at each state.



Objective: Maximize expected reward

$$\pi^* = \underset{\pi}{\operatorname{argmax}}\, E^{\pi}\left[\sum_{t=0}^{T} r_t\right]$$

$$\pi^* = \underset{\pi}{\operatorname{argmax}}\, E^{\pi}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$$

γ: discount factor [0,1)

Reward design

Desired states (goals): positive reward

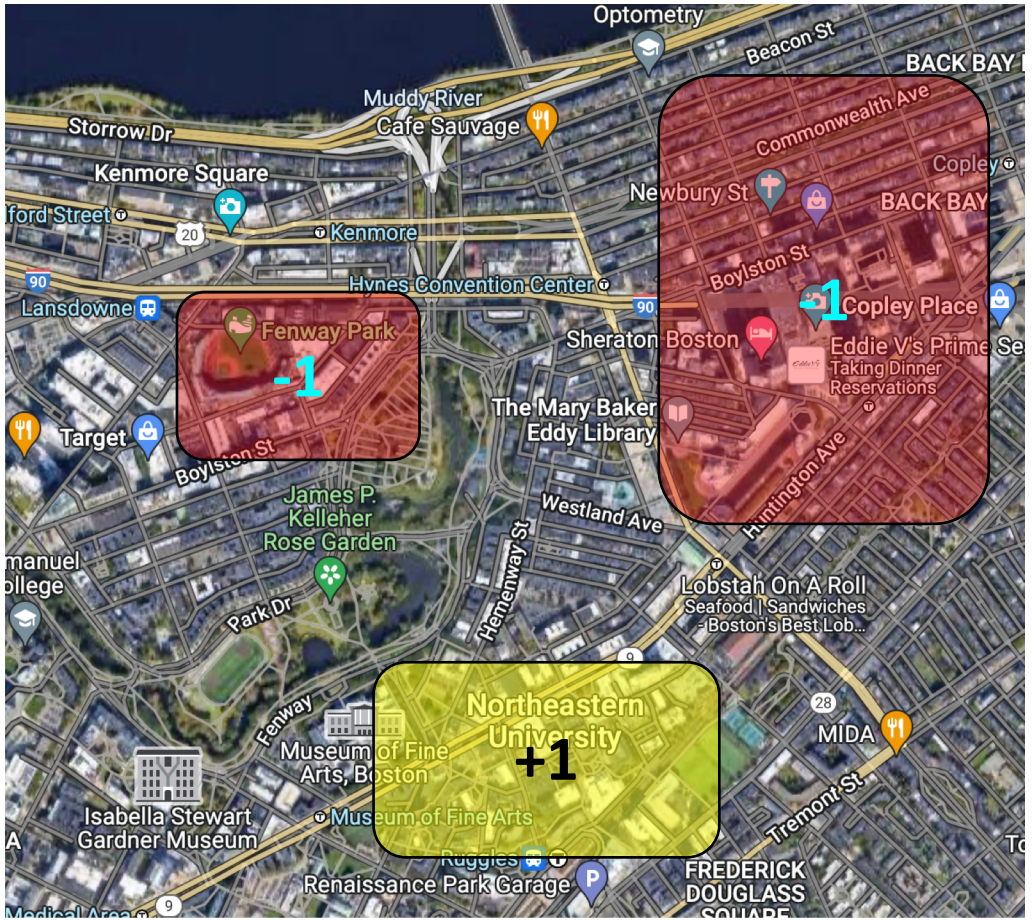
Undesired states: negative reward

Example: Surveillance task
Monitor yellow region as long as possible and
Avoid flying over the red regions

If the states in yellow are visited, receive +1 reward
If the states in red are visited, receive -1 reward
Otherwise, receive 0 reward

How do you give incentive to reach the goal faster?

Reward design for complex specifications?



Planning example – cont.

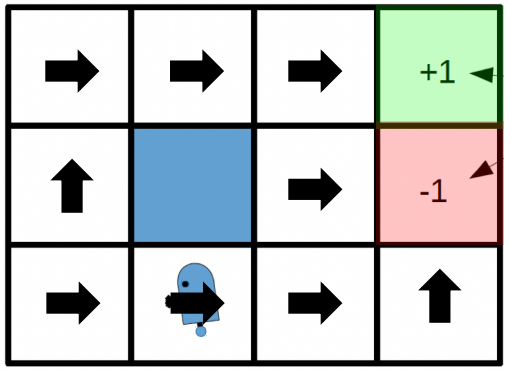
Objective

$$\pi^* = \underset{\pi}{\text{argmax}}\, E^{\pi}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)\right]$$

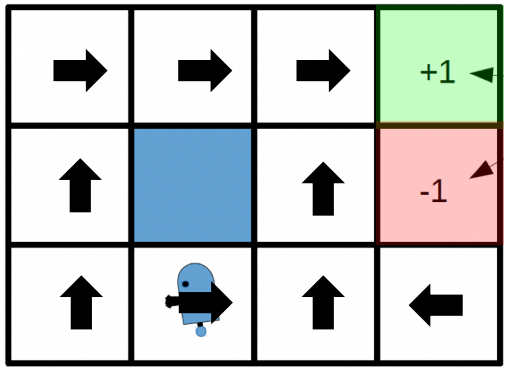
Let r be the reward in white cells.

$r = -2$

Leave white cells as soon as possible



The rewards impact the policy!

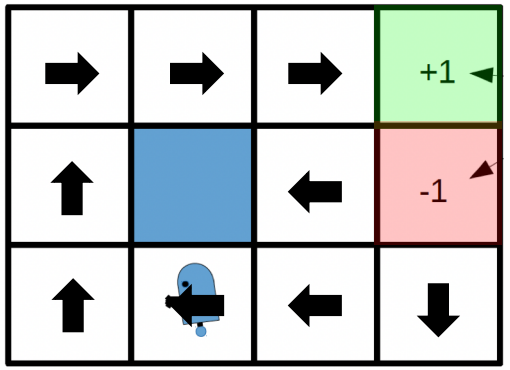


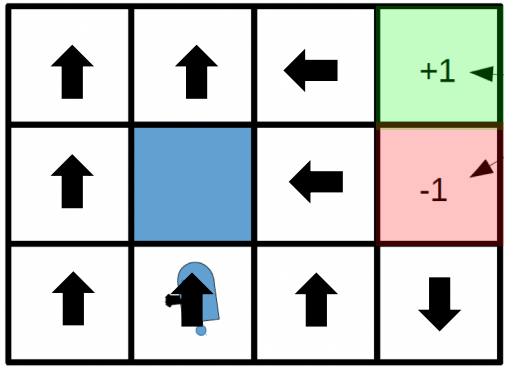
$r = -0.2$

Take shortcut, minor risks

$r = -0.01$

No risks are taken



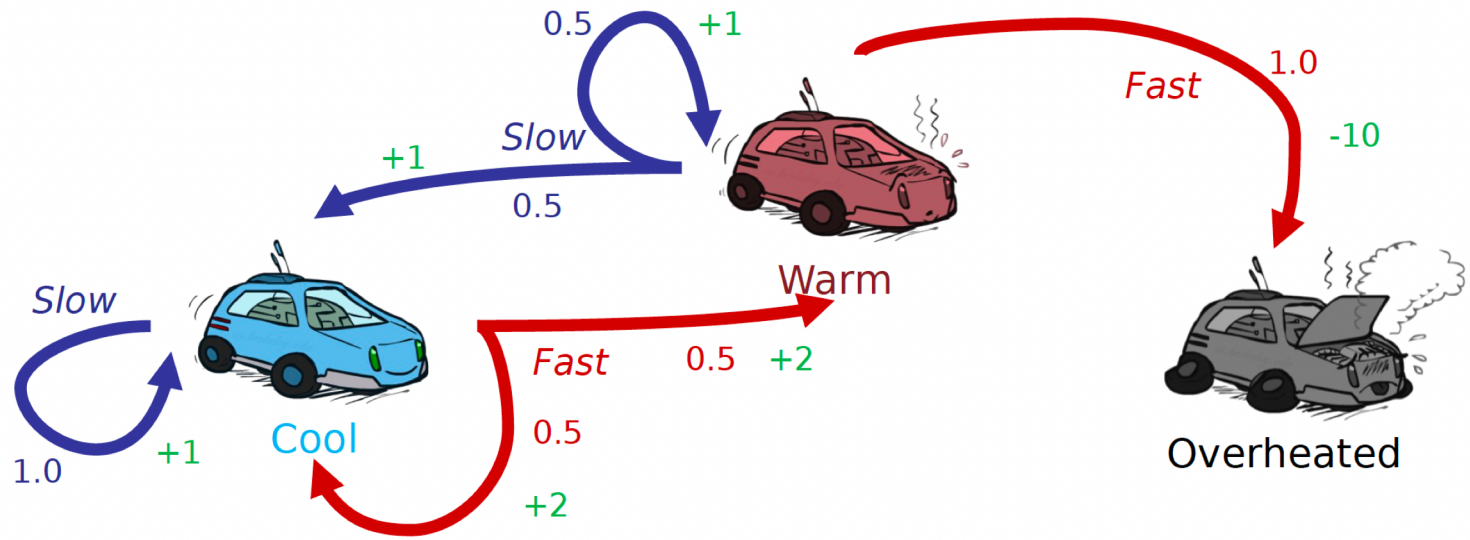


$r > 0$

Never leave (not a unique policy)

An example of an MDP

An autonomous car aiming to travel far quickly:



States?
Actions?
Probabilistic transition function?
Reward?

Value functions

- The value of a state under policy π:

$$V^{\pi}(s) = E^{\pi}\left[\sum_{k=0}^{\infty} \gamma^{k} r_{t+k+1} \,|s_t = s\right]$$

$$= E^{\pi}[\gamma^{0} r_{t+1} + \gamma^{1} r_{t+2} + \gamma^{2} r_{t+3} + \cdots |s_t = s]$$

- The value of taking action a in state s under policy π:

$$Q^{\pi}(s,a) = E^{\pi}\left[\sum_{k=0}^{\infty} \gamma^{k} r_{t+k+1} \,|s_t = s, a_t = a\right]$$

$$= E^{\pi}[\gamma^{0} r_{t+1} + \gamma^{1} r_{t+2} + \gamma^{2} r_{t+3} + \cdots |s_t = s, a_t = a]$$

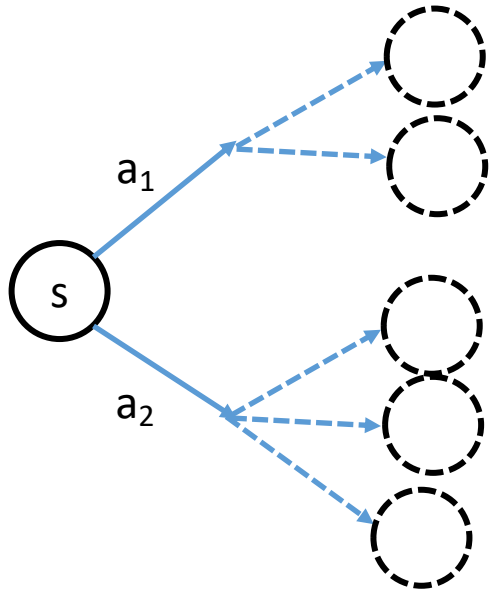
Bellman equation

$$V^{\pi}(s) = E^{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,|s_t = s\right]$$

$$= E^{\pi}\left[r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \,|s_t = s\right]$$

$$= \sum_a \pi(a|s) \sum_{s'} P(s,a,s')\left[R(s,a,s') + \gamma E^{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \,|s_{t+1} = s'\right]\right]$$

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



Potential next states

Optimal value functions

$$V^{\pi}(s) = E^{\pi}\left[\sum_{k=0}^{\infty} \gamma^{k} r_{t+k+1} \,|s_t = s\right] \qquad Q^{\pi}(s,a) = E^{\pi}\left[\sum_{k=0}^{\infty} \gamma^{k} r_{t+k+1} \,|s_t = s, a_t = a\right]$$

$$V^{*}(s) = \max_{\pi} V^{\pi}(s) \text{ for all } s \in S$$

$$Q^{*}(s,a) = \max_{\pi} Q^{\pi}(s,a) \text{ for all } s \in S \text{ and } a \in A$$

The relationship between V^{*} and Q^{*}:

$$V^{*}(s) = \max_{a \in A(s)} Q^{*}(s,a)$$

Optimal value functions

$$Q^*(s,a) = E\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,|s_t = s, a_t = a\right]$$

$$= E\left[r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \,|s_t = s, a_t = a\right]$$

$$Q^*(s,a) = E[r_{t+1} + \gamma V^*(s_{t+1}) | s_t = s, a_t = a]$$

Reward

Value of successor state

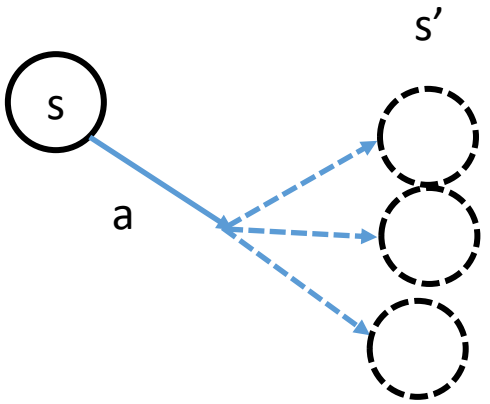
Optimal value functions – recursive form

$$Q^*(s,a) = E[r_{t+1} + \gamma V^*(s_{t+1}) | s_t = s, a_t = a]$$

Reward

Value of successor state

Potential next states



$$Q^*(s,a) = E\left[r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a') \,|\, s_t = s, a_t = a\right]$$

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

Optimal value functions – recursive form

$$V^*(s) = \max_{a \in A(s)} Q^*(s,a)$$

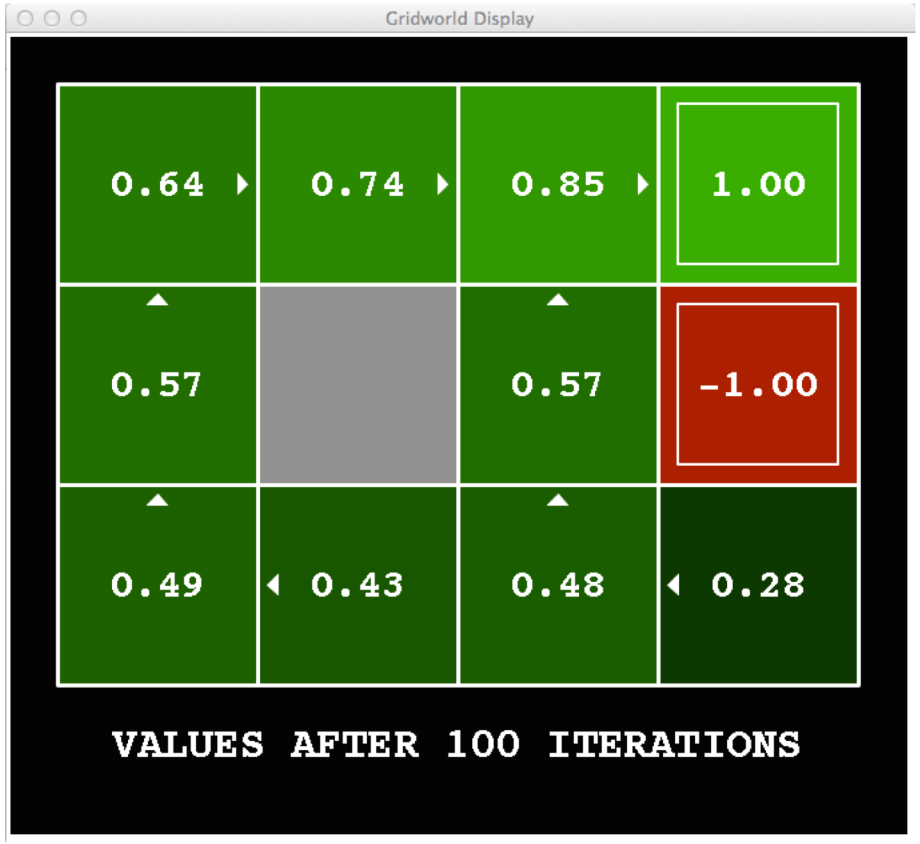
$$= \max_{a \in A(s)} E^{\pi^*}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,|s_t = s, a_t = a\right]$$

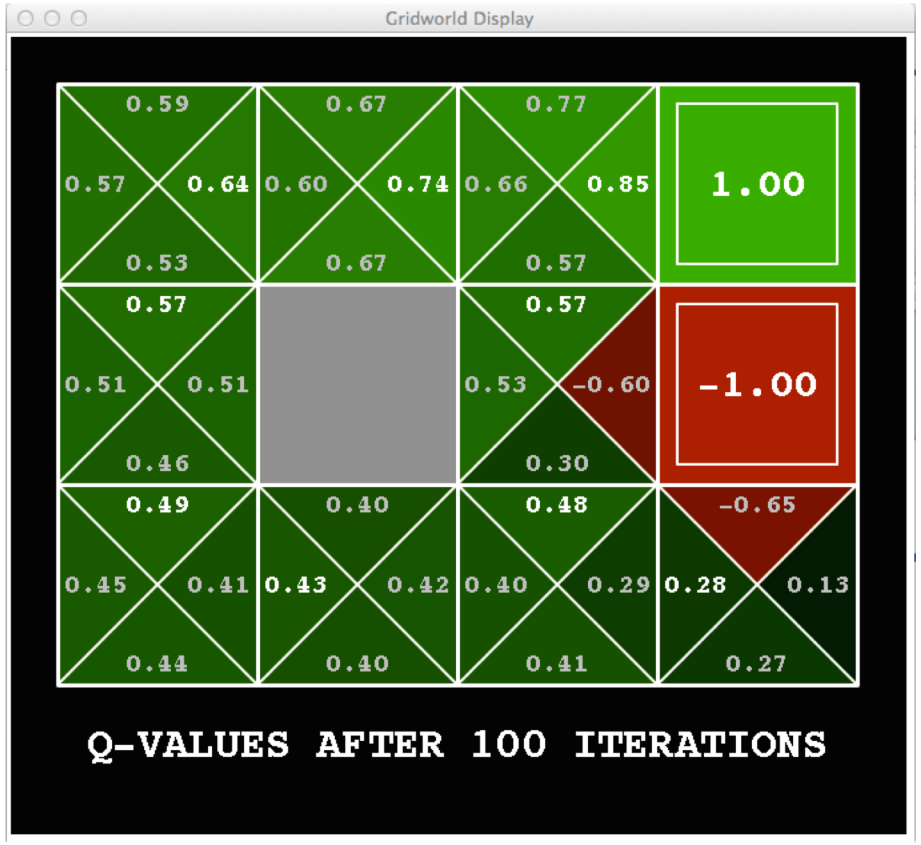
$$= \max_{a \in A(s)} E^{\pi^*}\left[r_{t+k+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \,|s_t = s, a_t = a\right]$$

$$= \max_{a \in A(s)} E^{\pi^*}[r_{t+k+1} + \gamma V^*(s_{t+1}) | s_t = s, a_t = a]$$

$$V^*(s) = \max_{a \in A(s)} \sum_{s'} P(s,a,s')\,[R(s,a) + \gamma V^*(s')]$$

Grid world example





Noise = 0.2

Discount = 0.9

Value iteration

$$V_{k+1}(s) = \max_a \sum_{s'} P(s,a,s') \left[R(s,a) + \gamma V_k(s')\right]$$

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

Repeat
  $\Delta \leftarrow 0$
  For each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V(s')\right]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, π, such that
  $\pi(s) = \arg\max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V(s')\right]$

Once it is converged, we will see

$$V^*(s) = \max_{a \in A(s)} \sum_{s'} P(s,a,s') \left[R(s,a) + \gamma V^*(s')\right]$$

Bellman equation for optimal value function

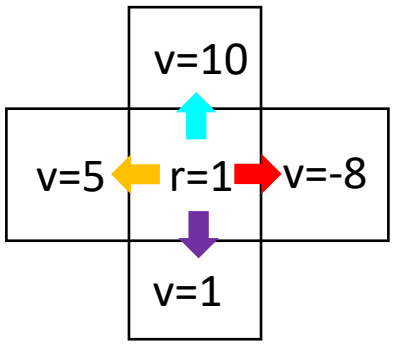
Value iteration - example

- Calculate the value of center cell

$$V_{k+1}(s) = \max_a \sum_{s'} P(s,a,s') \left[R(s,a) + \gamma V_k(s')\right]$$

Desired action: Up

Transition model



State-space
(v=value, r=reward)

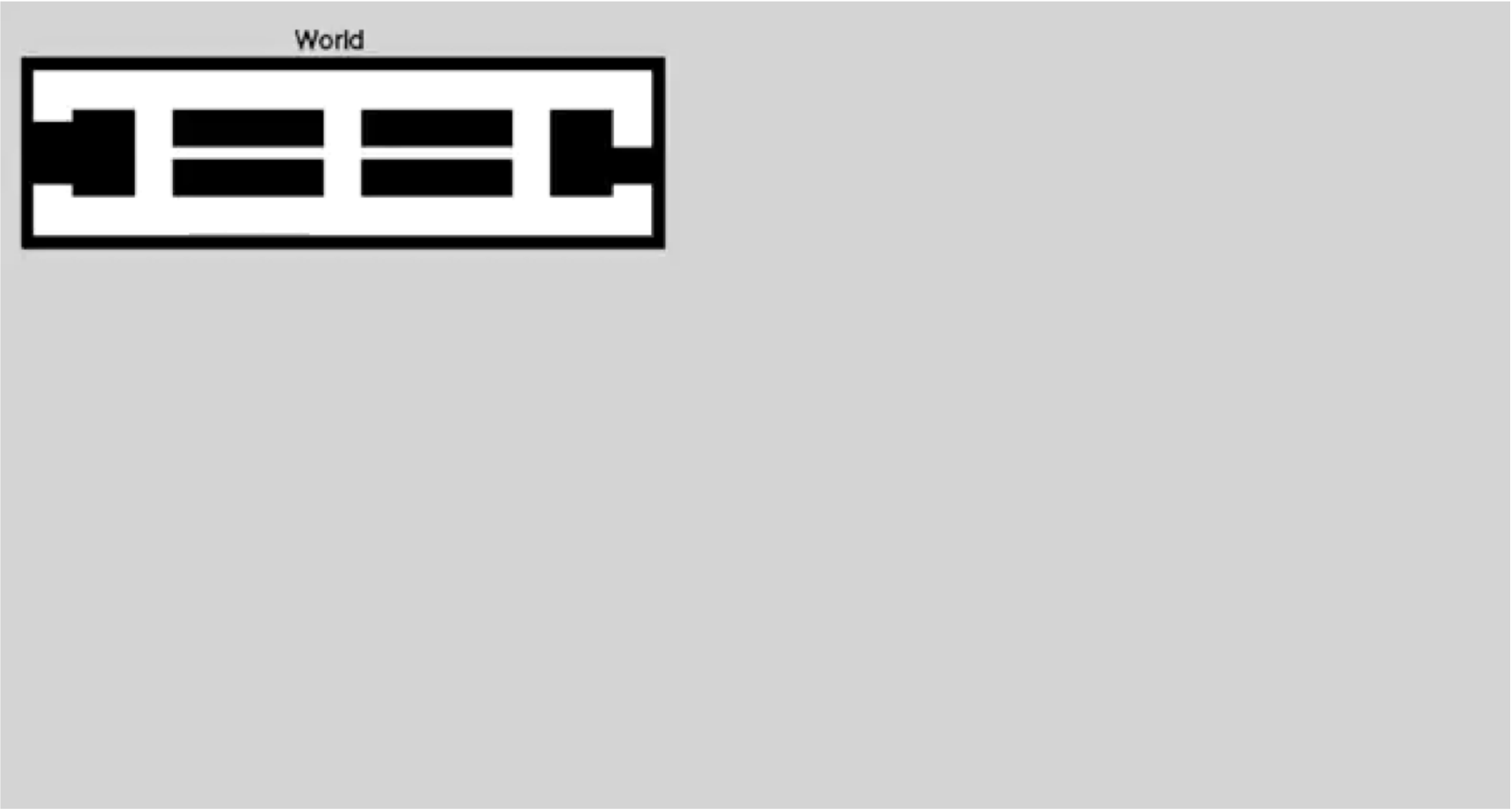
$$V = r + \max \begin{matrix} \{0.1\times1 + 0.8\times5 + 0.1\times10(\leftarrow), \\ 0.1\times5 + 0.8\times10 + 0.1\times-8(\uparrow), \\ 0.1\times10 + 0.8\times-8 + 0.1\times1(\rightarrow), \\ 0.1\times-8 + 0.8\times1 + 0.1\times5(\downarrow)\} \end{matrix}$$

$$= r + \max\{5.1(\leftarrow), 7.7(\uparrow), -5.3(\rightarrow), 0.5(\downarrow)\}$$

$$= 1 + 7.7$$

$$= 8.7$$

Grid world example



https://www.youtube.com/watch?v=gThGerajccM

Policy iteration

1. Initialization
   $V(s) \in \Re$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

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

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   $b \leftarrow \pi(s)$
   $\pi(s) \leftarrow \arg\max_a \sum_{s'} \mathcal{P}_{ss'}^{a} \left[\mathcal{R}_{ss'}^{a} + \gamma V(s') \right]$
   If $b \neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop; else go to 2

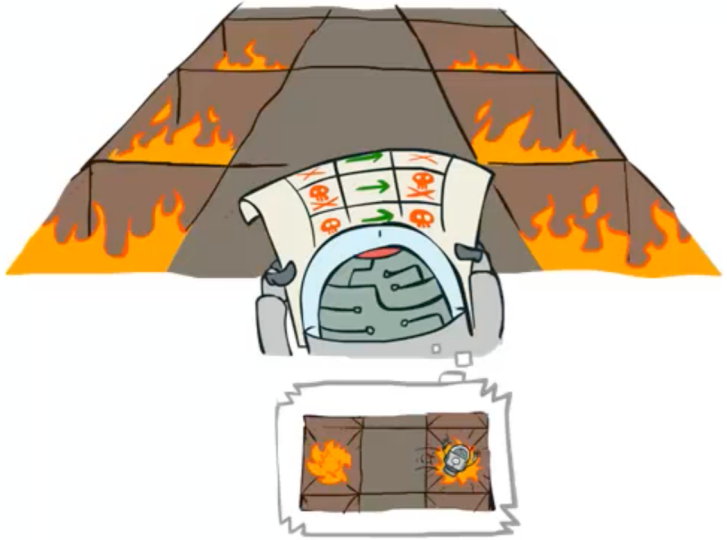
$$\pi_0 \xrightarrow{\text{E}} V^{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} V^{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \cdots \xrightarrow{\text{I}} \pi^* \xrightarrow{\text{E}} V^*$$

It converges to the optimal policy.

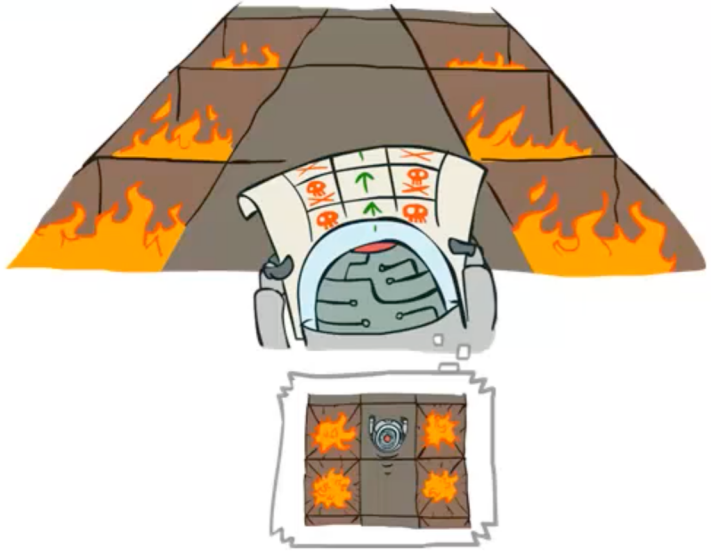
In some cases, it is faster than value iteration.

Policy iteration (example)

Always Go Right



Always Go Forward



Policy iteration (example)

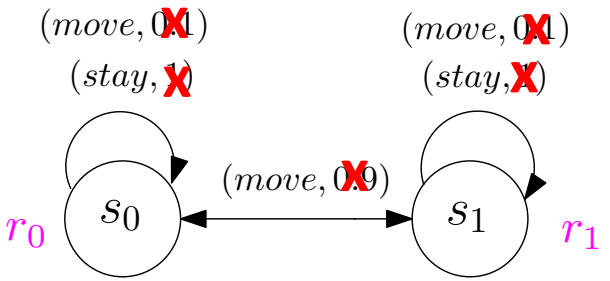
Always Go Right

Always Go Forward

Summary

Policy Iteration	Value Iteration
• Starts with a random policy • Algorithm is more complex • Guaranteed to converge • Cheaper to compute • Requires few iterations to converge • Faster	• Starts with a random value function • Algorithm is simpler • Guaranteed to converge • More expensive to compute • Requires more iterations to converge • Slower

Reinforcement learning



An MDP is a tuple $M = (S, A, P, R)$

- S: set of states
- A: set of motion primitives
- P: $S \times A \times S \rightarrow \mathbb{R}_{\geq 0}$ probabilistic transition function (unknown)
- R: $S \times A \rightarrow \mathbb{R}_{\geq 0}$ reward function (known or unknown)

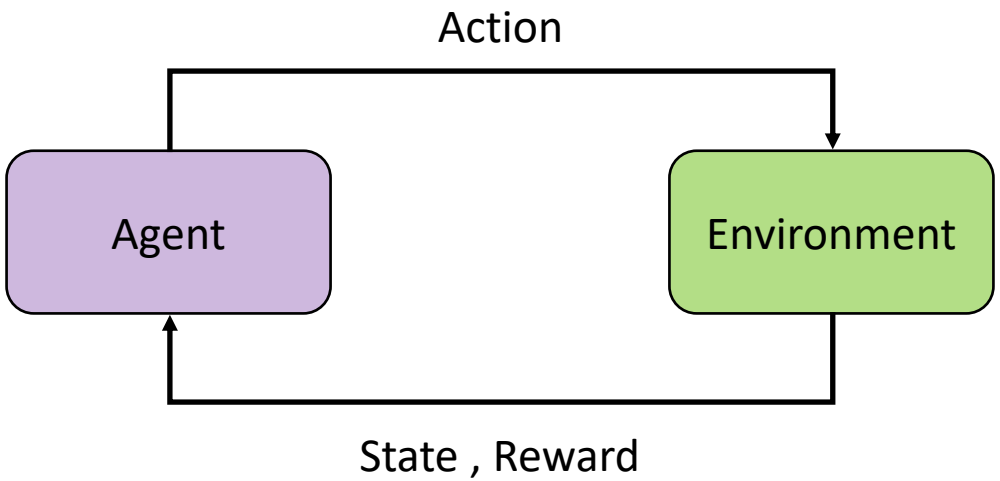
Policy π tells the agent what action to execute.

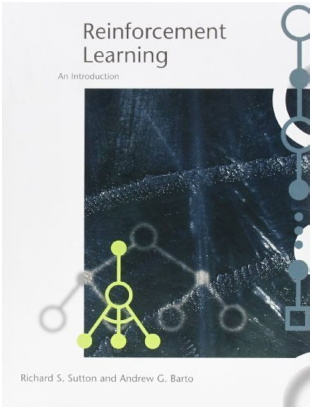
Objective: Maximize expected reward

$$\pi^* = \underset{\pi}{\operatorname{argmax}}\, E^{\pi}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right] \qquad \pi^* = \underset{\pi}{\operatorname{argmax}}\, E^{\pi}\left[\sum_{t=0}^{T} r_t\right]$$

Reinforcement Learning - Overview

Goal: Discover an optimal behavior through trial-and-error interactions with the environment





https://inst.eecs.berkeley.edu/~cs188/sp20/assets/files/SuttonBartoIPRLBook2ndEd.pdf

Reward: Designer of the task provides feedback in terms of a scalar function

Learn how to take actions to maximize reward

Robot Locomotion

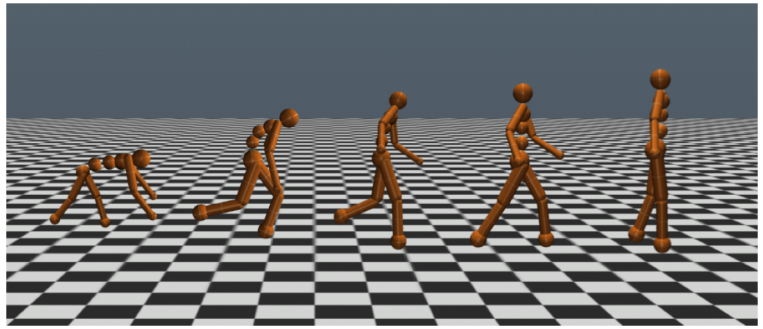
Goal: Robot learns to move forward

Comparison of
RL algorithms
in Humanoid-v2
using CleanRL

State: Angle and position of the joints

Action: Torques applied on joints

Reward: 1 at each time step upright and forward movement



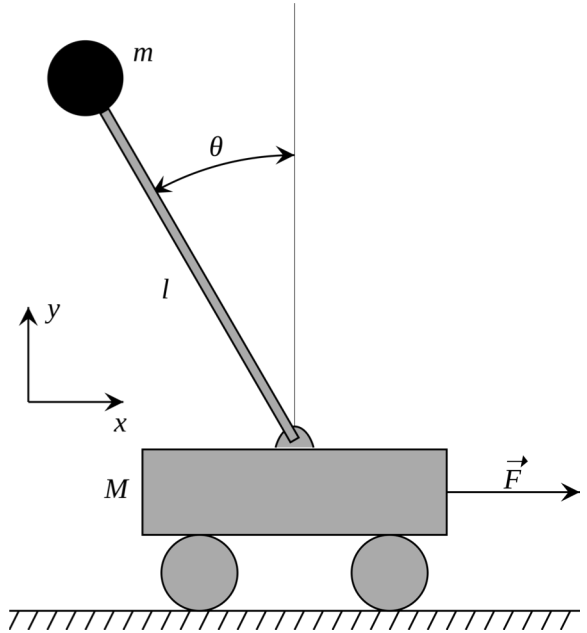
Cart-Pole Problem

Goal: Learn to balance a pole on top of a movable cart

State: angle, angular speed, position, hor. velocity

Action: horizontal force applied on the cart

Reward: 1 at each time step if the pole is upright



Atari Game

Goal: Complete the game with the highest score

State: Raw pixel inputs of the game state

Action: Left, Right, Up, Down, etc.

Reward: Score increase/decrease at each time step

Trajectory planning

Goal: Monitor a desired region as long as possible under a constrained behavior

Reinforcement Learning Algorithms

Value-based

- Maximize long-term return of the current states under a policy π

Policy-based

- Find a policy leading to each action performed in every state to return max reward

Model-based

- Create a virtual model to perform learning

Q-Learning

Value-based, model-free learning to maximize expected discounted reward

(State, *s*) → (Action, *a*) → (Next state, *s'*) → (Reward, *r*) → (Update Q-function)

$$Q(s,a) := (1-\alpha)Q(s,a) + \alpha[r + \gamma \max_{a^* \in A} Q(s', a^*)]$$

Learning rate

Reward

Max receivable future reward

Convergence: If each action *a* is implemented for infinite num. of times and α decays appropriately, then *Q* converges to Q^* w.p.1.

Optimal Policy

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

Q vs. Deep-Q learning