Reinforcement Learning

Emre Ugur, BM 33

emre.ugur@boun.edu.tr

http://www.cmpe.boun.edu.tr/~emre/courses/cmpe462

cmpe462@listeci.cmpe.boun.edu.tr

Quiz

Kernel function for polynomials of degree q is

$$K(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^T \mathbf{y} + 1)^q$$

Find the mapping function Φ(x) for 2 dimensional input space and linear kernel function (q=1)

$$L_d = \sum_t \alpha^t - \frac{1}{2} \sum_t \sum_s \alpha^t \alpha^s r^t r^s \boldsymbol{\phi}(\boldsymbol{x}^t)^T \boldsymbol{\phi}(\boldsymbol{x}^s)$$

$$L_d = \sum_t \alpha^t - \frac{1}{2} \sum_t \sum_s \alpha^t \alpha^s r^t r^s K(\boldsymbol{x}^t, \boldsymbol{x}^s)$$

$$\begin{aligned} g(\boldsymbol{x}) &= \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}) = \sum_t \alpha^t r^t \boldsymbol{\phi}(\boldsymbol{x}^t)^T \boldsymbol{\phi}(\boldsymbol{x}) \\ &= \sum_t \alpha^t r^t K(\boldsymbol{x}^t, \boldsymbol{x}) \end{aligned}$$

Acknowledgements

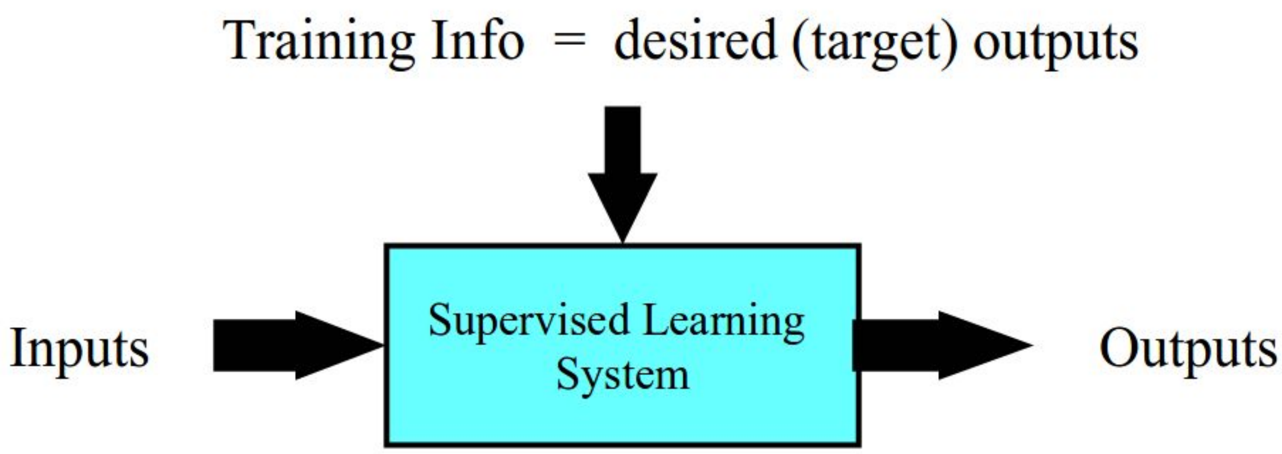
- These slides are adapted from lecture/textbook slides of
  - Dan Klein and Pieter Abbeel, CS188 Intro to AI at UC Berkeley.
  - David Silver, UCL Course on RL
  - Ethem Alpaydin, Introduction to Machine Learning
  - Sutton and. Barto, An Introduction to Reinforcement Learning,
  - Scott Niekum, CS 343: AI, The University of Texas at Austin

Learning methods

- Unsupervised learning: Learn clusters/groups without any label
    - K-means, mixture of gaussians...

Learning methods

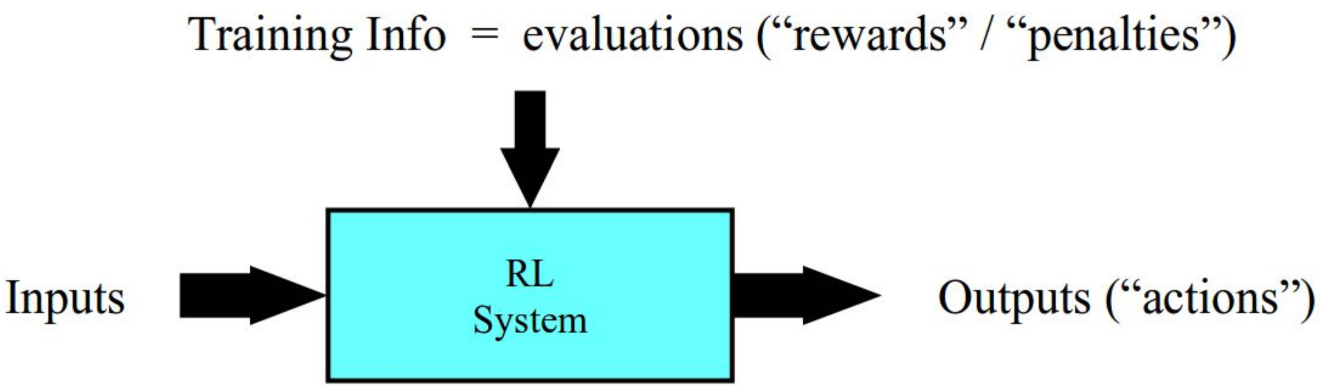
- Supervised learning: Given labeled examples, learn to predict label of new example.
    - Neural networks, decision trees, SVMs...
    - Classification or regression



$$\text{Error} = (\text{target output} - \text{actual output})$$

Learning methods

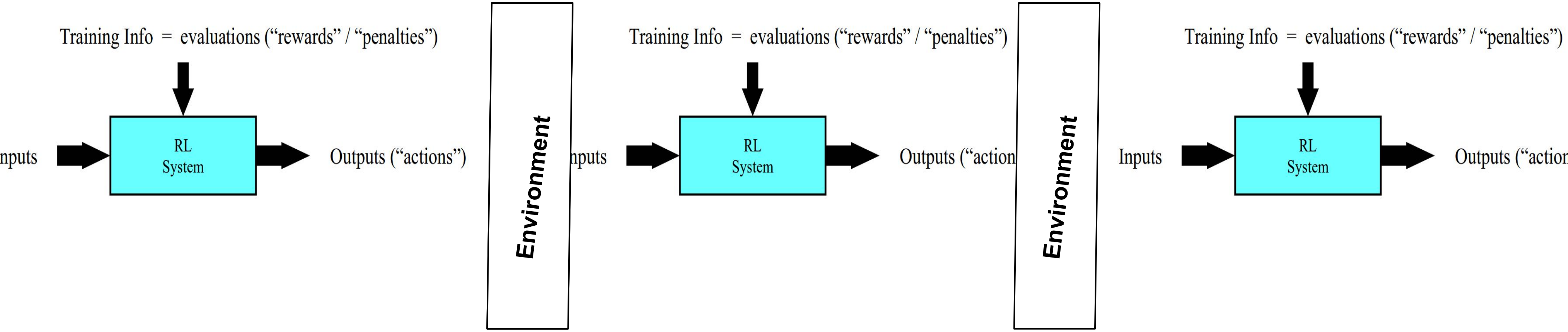
- Reinforcement learning:
  - Given rewards
  - Learn which actions to take in which situations
  - in order to maximize future cumulative reward



Objective: get as much reward as possible

Learning methods

- Reinforcement learning:
    - Given rewards
    - Learn which actions to take in which situations
    - in order to maximize **future cumulative reward**



Cumulative reward = Utility/Value = $Q = R_1 + R_2 + R_3 + R_4 \ldots$

Cumulative discounted reward = Utility/Value = $Q = R_1 + \gamma R_2 + \gamma^2 R_3 + \gamma^3 R_4 \ldots$

Reinforcement Learning

left, right, straight, left, left, left, straight **GOOD**

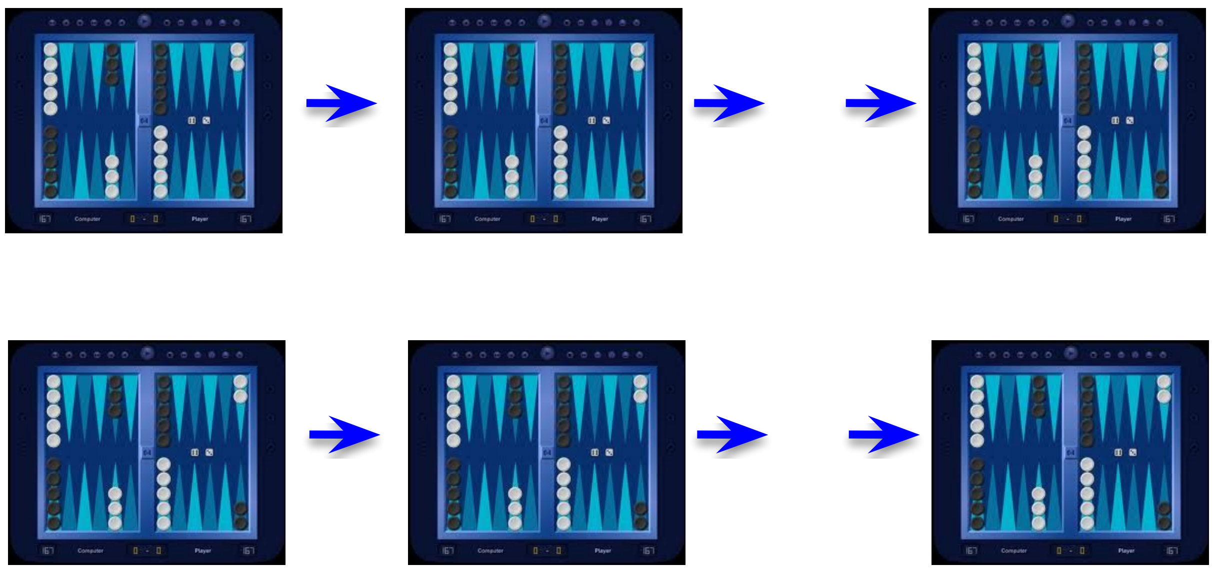
left, straight, straight, left, right, straight, straight **BAD**

left, right, straight, left, left, left, straight **18.5**

left, straight, straight, left, right, straight, straight **-3**

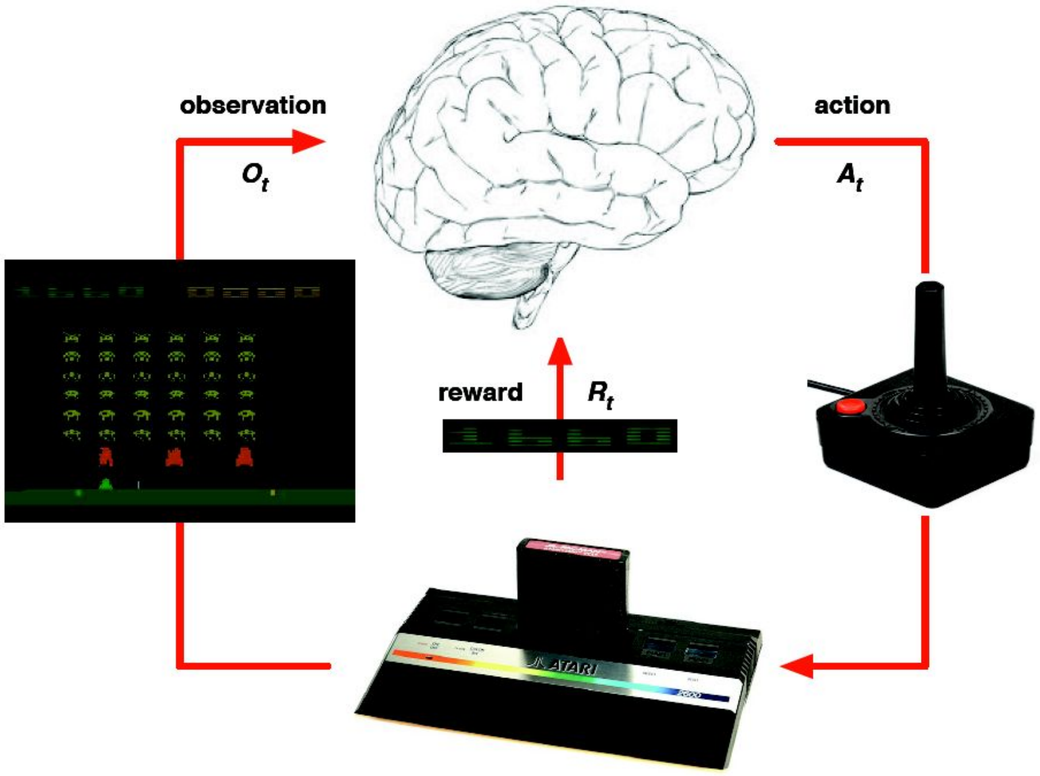
Given a *sequence* of examples/states and a *reward* after completing that sequence, learn to predict the action to take in for an individual example/state

Reinforcement Learning



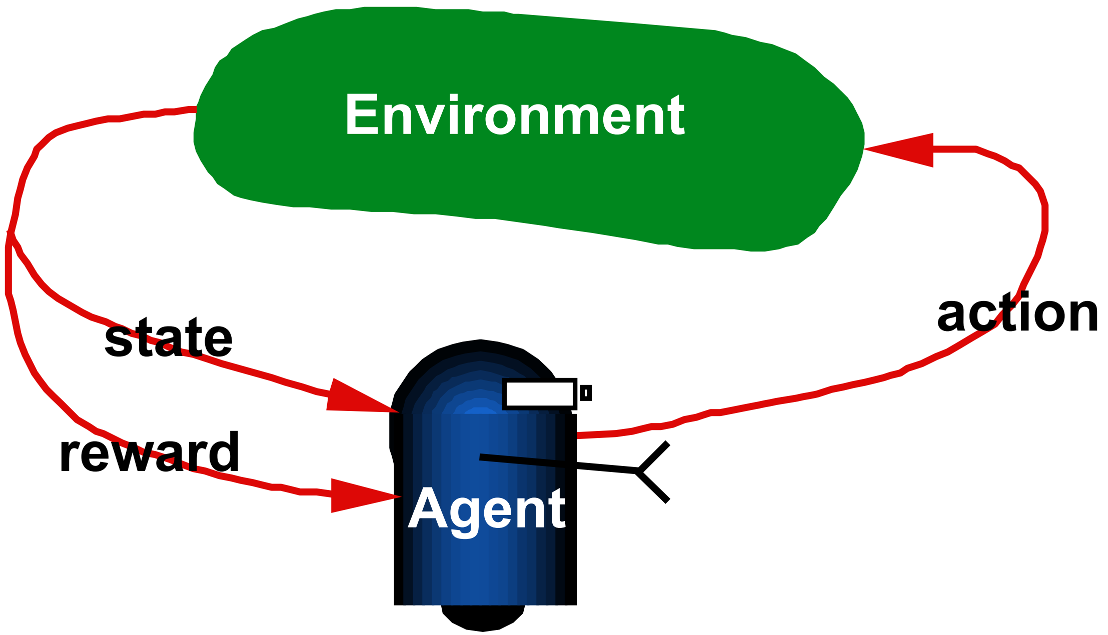
Given a *sequence* of examples/states and a *reward* after completing that sequence, learn to predict the action to take in for an individual example/state

Atari example:



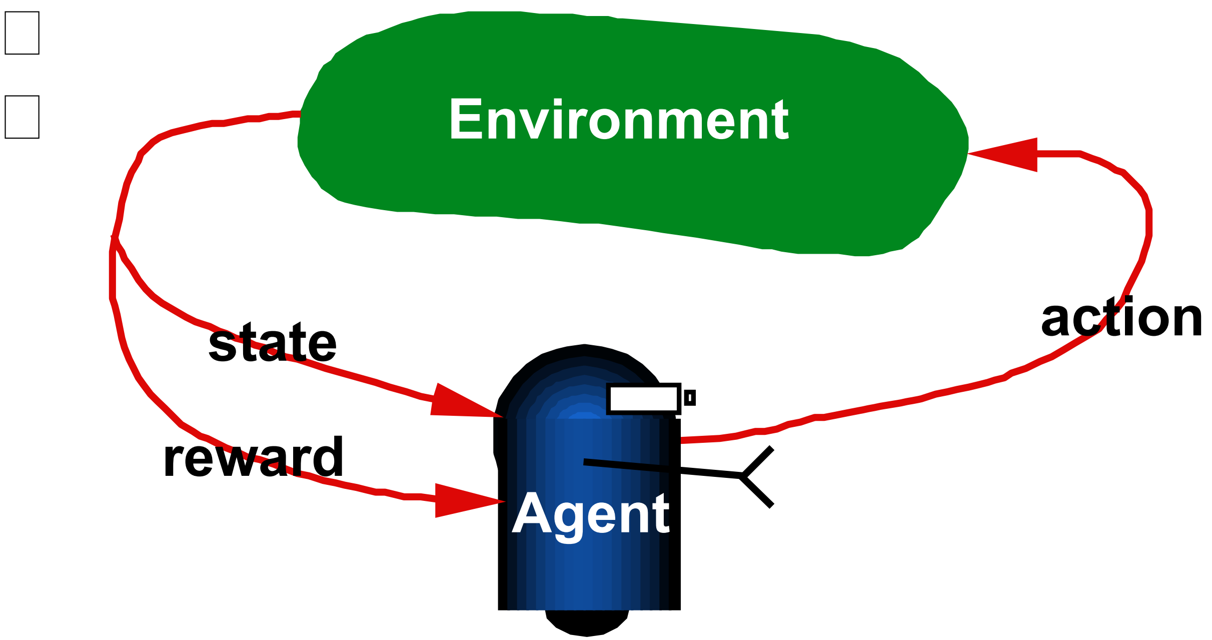
- Rules of the game are unknown
- Learn directly from interactive game-play
- Pick actions on joystick, see pixels and scores

Reinforcement learning



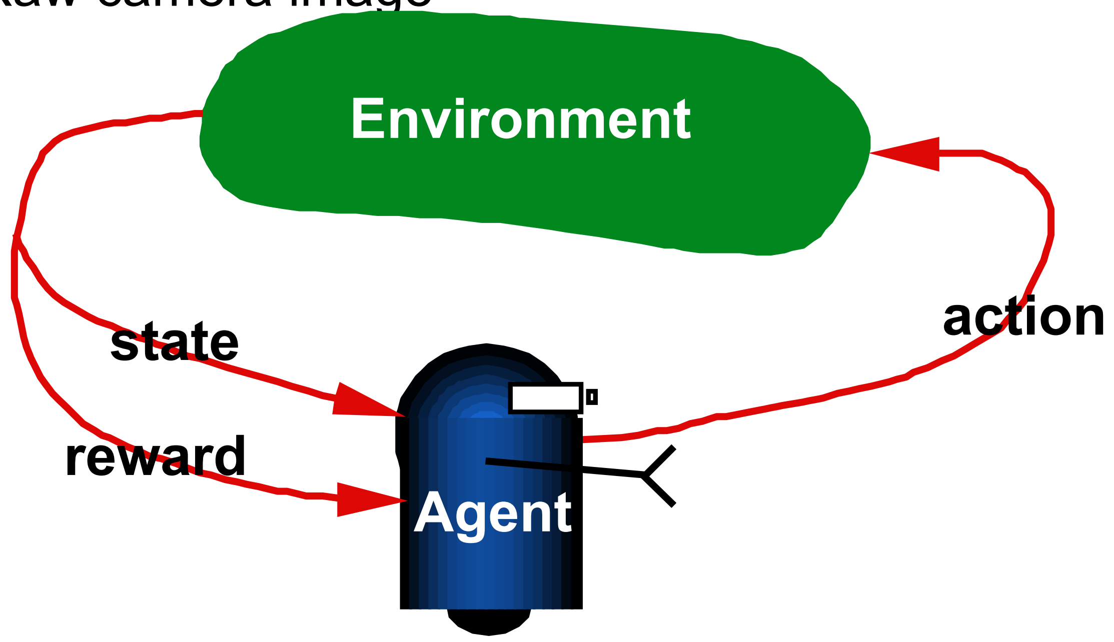
Reinforcement learning

- Action
    - Robot in 2d grid world: up, down, left, right
    - Backgammon agent: select a piece to move
    - Chess: select a piece+select movement
    - Autonomous car: steer left, steer right, accelerate, break



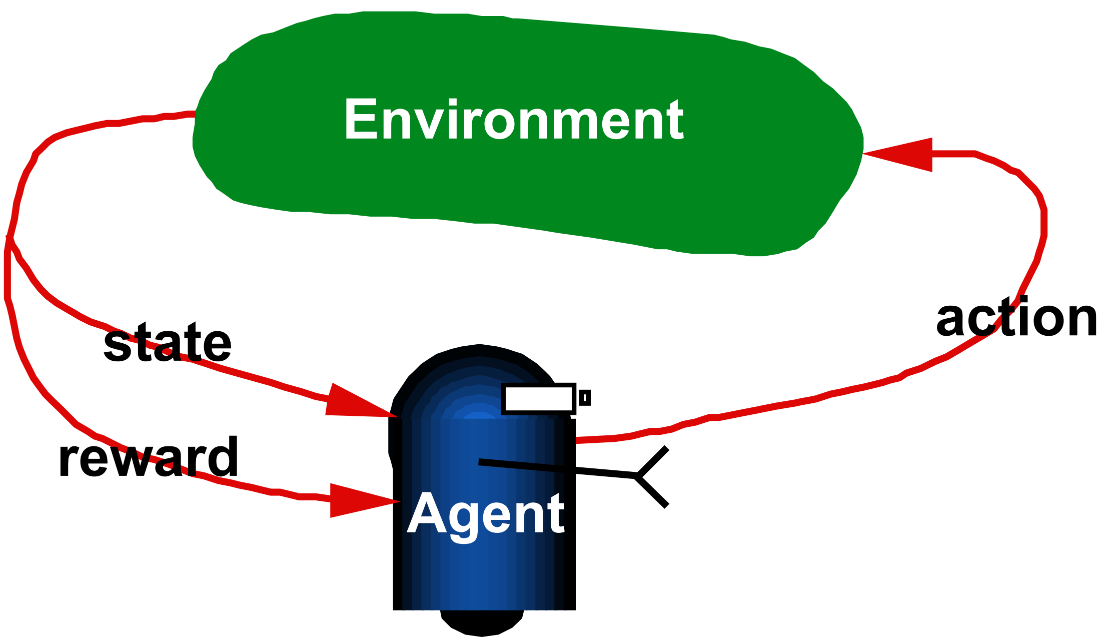
Reinforcement learning

- State
  - Robot in 2d grid world: grid index
  - Backgammon agent: piece configuration
  - Chess: piece configuration
  - Autonomous car: more complicated.
    - Position in the lane, velocity, cars around, pedestrians around
    - Raw camera image



Reinforcement learning

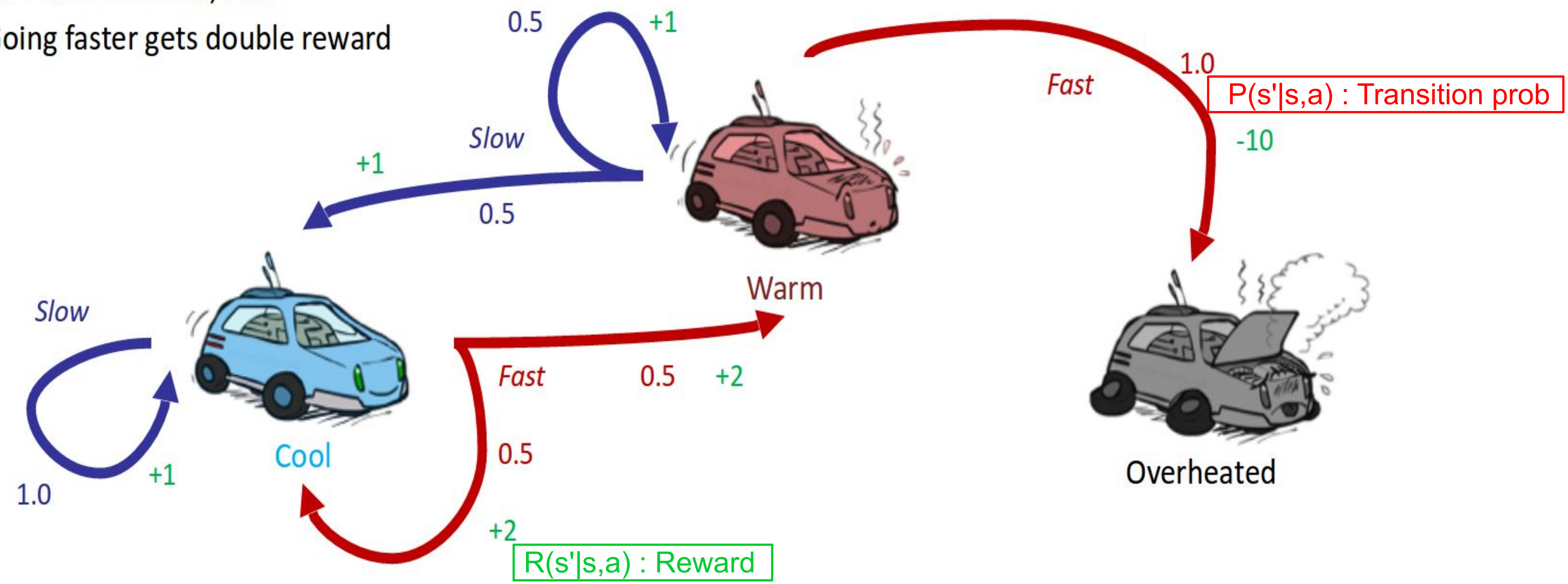
- Reward
  - Robot in 2d grid world: gold grid:+10, other grids:0
  - Backgammon agent: win:+1, lose:-1
  - Chess: win:+1, lose:-1
  - Autonomous car: crash:-10000, reach-target:10, other:0



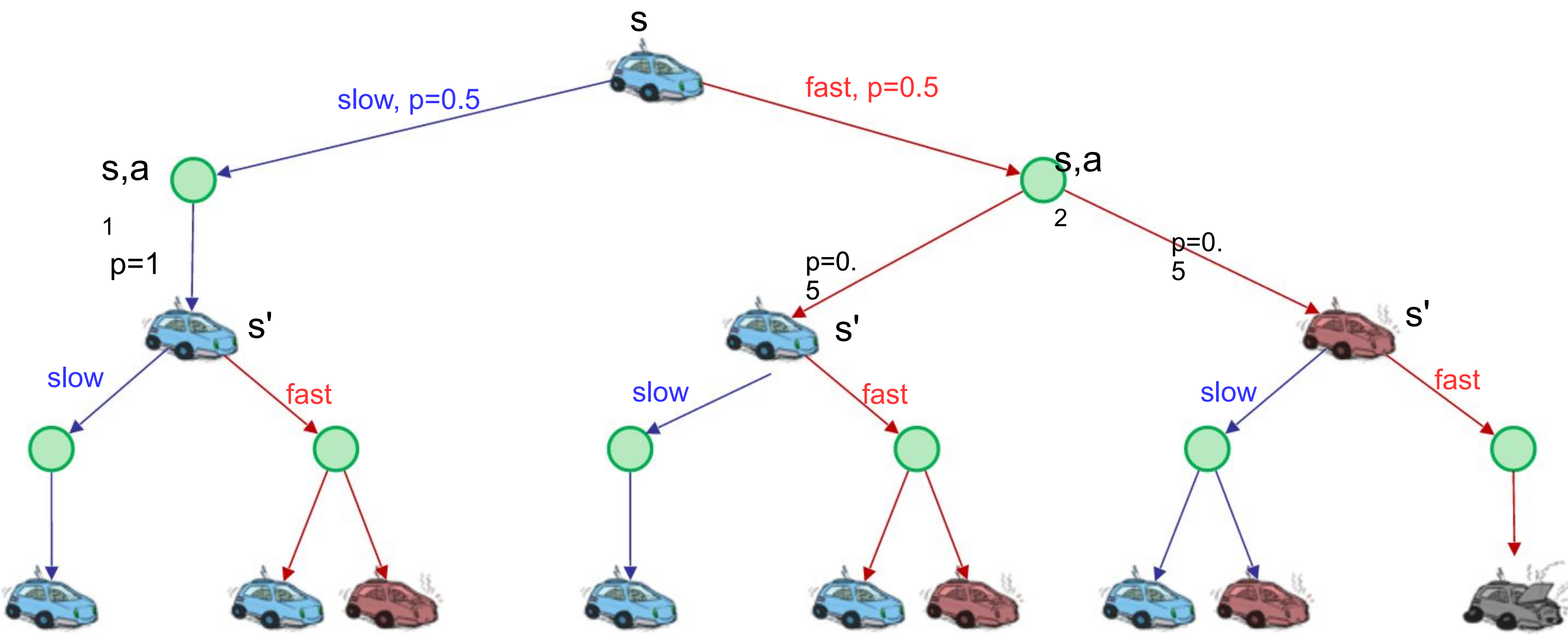
RL solves Markov Decision Process

Next state depends only to the current state and action

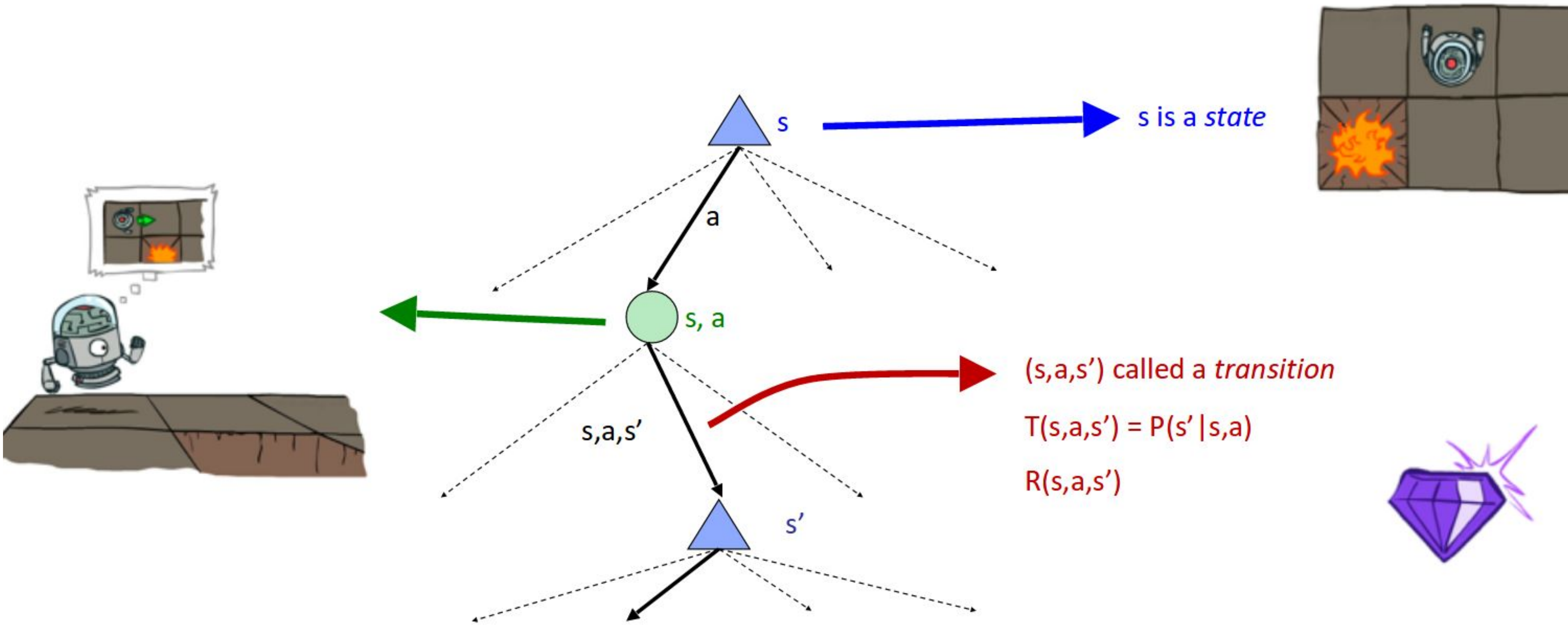
- A robot car wants to travel far, quickly
- Three states: Cool, Warm, Overheated
- Two actions: *Slow*, *Fast*
- Going faster gets double reward



Racing search tree



Search tree



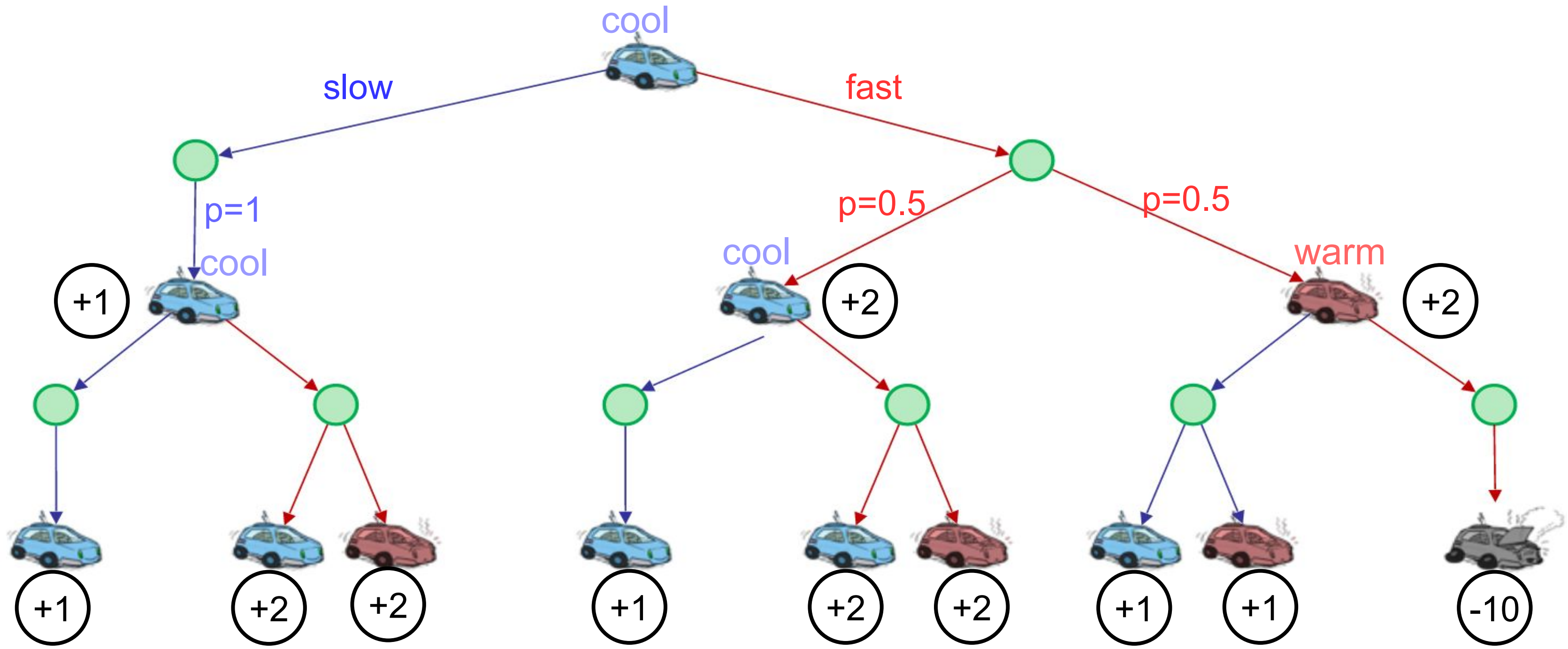
Major components of RL agent

- An RL agent may include one or more of these components:
  - Model: agent's representation of the environment
  - Value function: how good is each state and/or action
  - Policy: agent's behavior function

Major components of RL agent

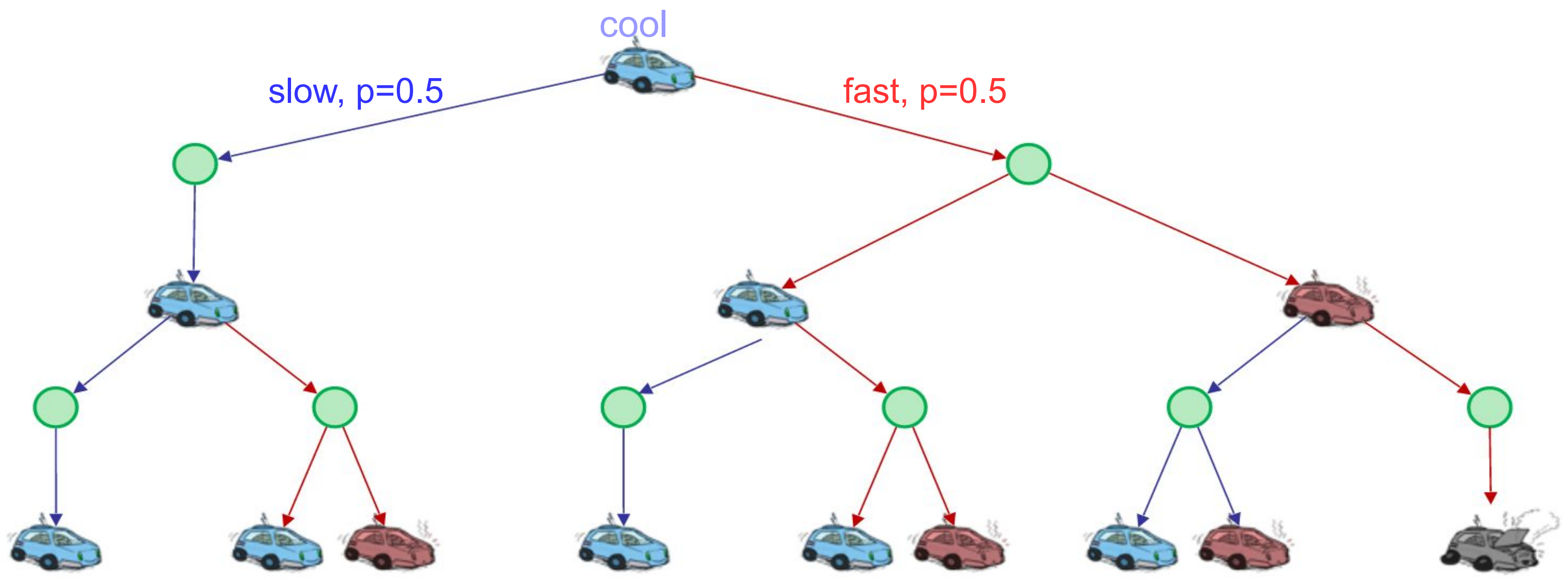
- A model predicts what the environment will do next
  - P predicts the next state
  - R predicts the next (immediate) reward, e.g.

$$\mathcal{P}_{ss'}^{a} = \mathbb{P}[S_{t+1} = s' \mid S_t = s, A_t = a]$$
$$\mathcal{R}_{s}^{a} = \mathbb{E}\left[R_{t+1} \mid S_t = s, A_t = a\right]$$



Major components of RL agent

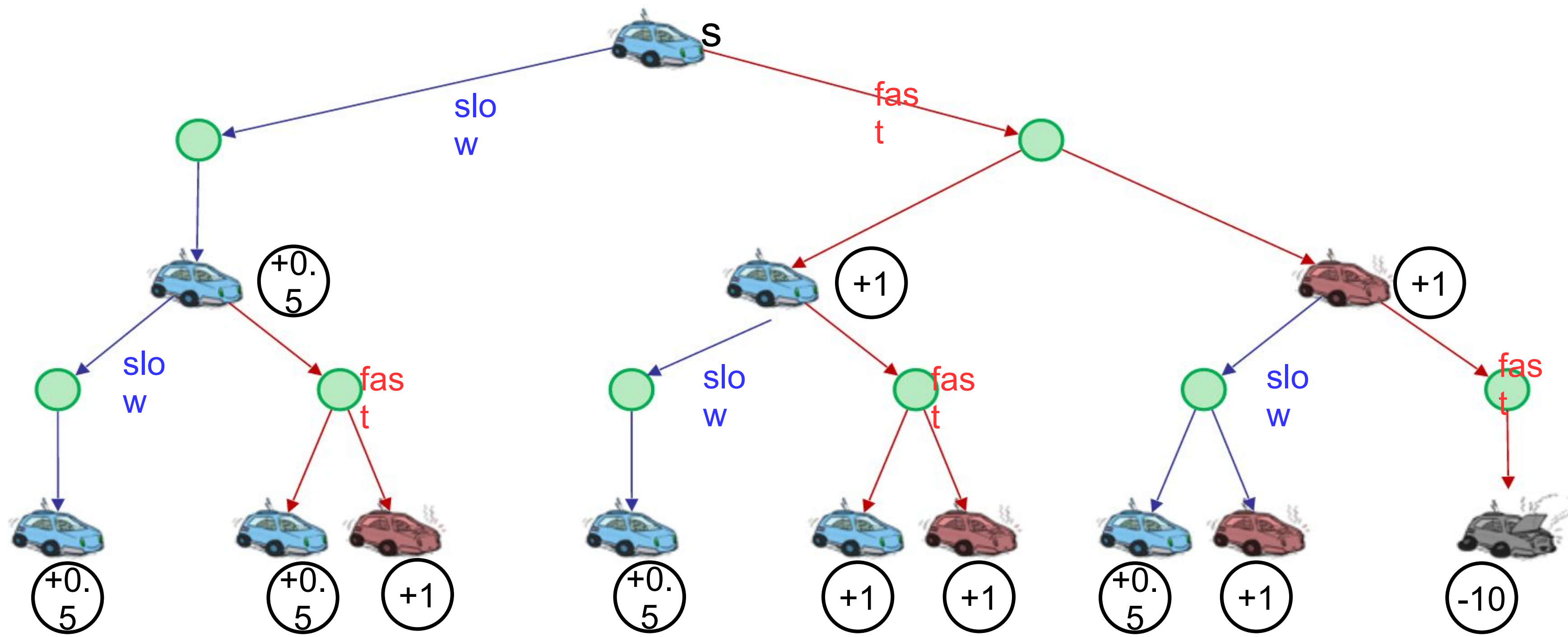
- A policy is the agent's behaviour
  - It is a map from state to action, e.g.
  - Deterministic policy: $a = \pi(s)$
    - e.g.: if this patch of room is **dirty**, I **clean** it.
  - Stochastic policy: $\pi(a|s) = P(A_t = a \mid S_t = s)$
    - if this patch is dirty, I clean it with 90%, or look for dirtier patches with 10%



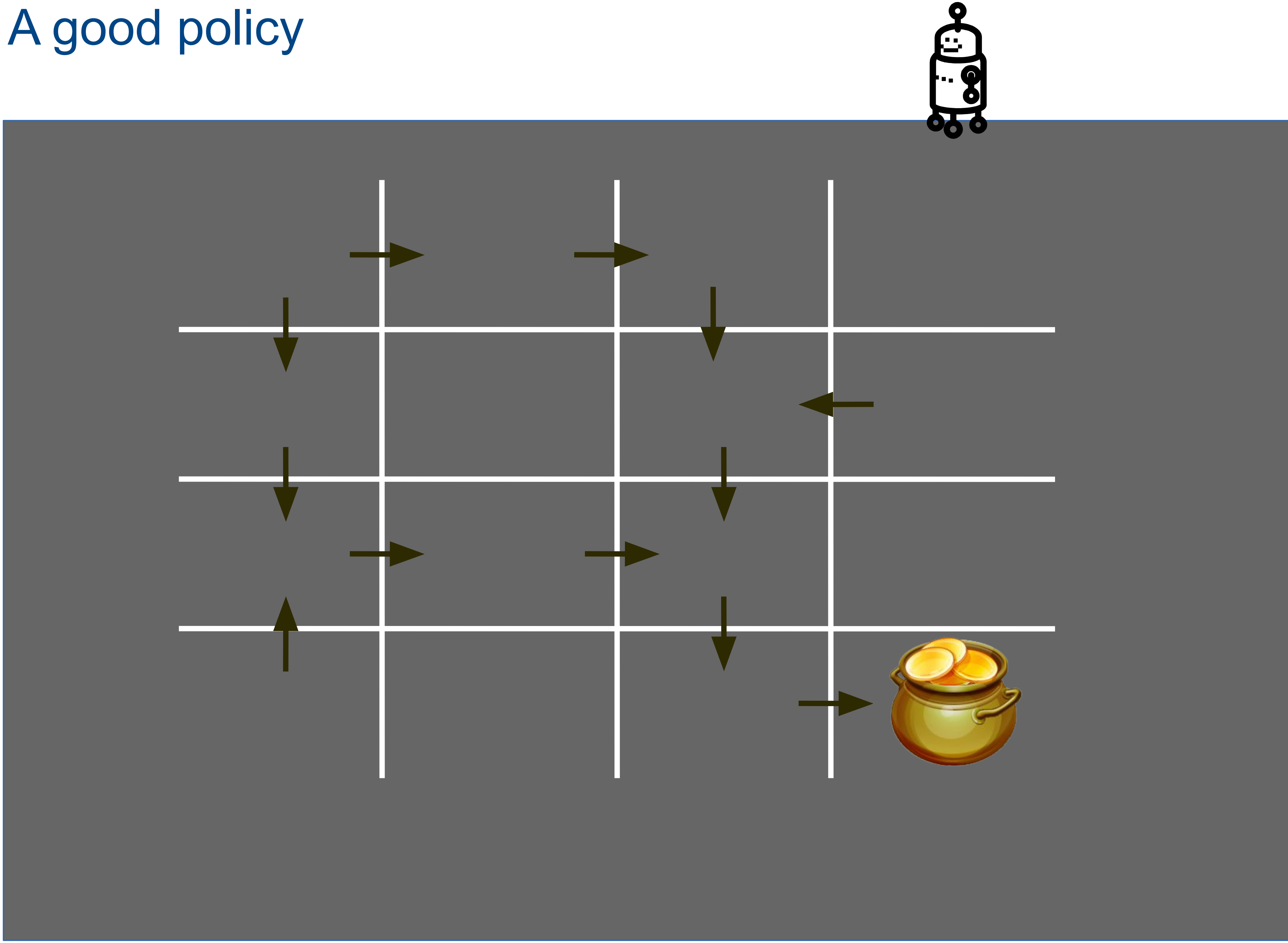
Major components of RL agent

- Value function is a prediction of future reward – cumulative reward
    - Used to evaluate the goodness/badness of states
    - And therefore to select between actions,

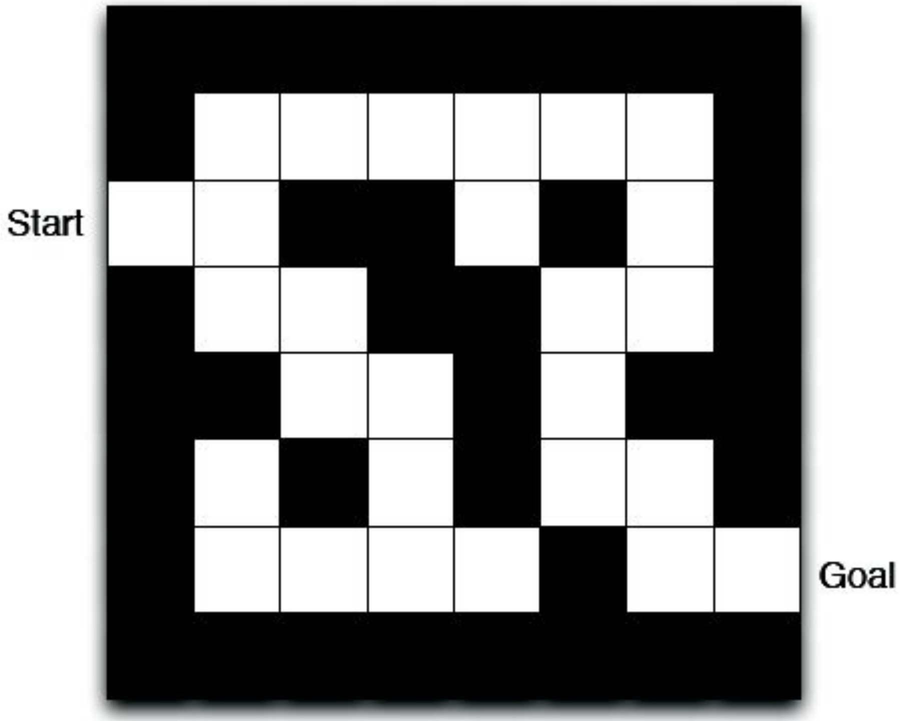
$$v_\pi(s) = \mathbb{E}_\pi\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \mid S_t = s\right]$$



A good policy

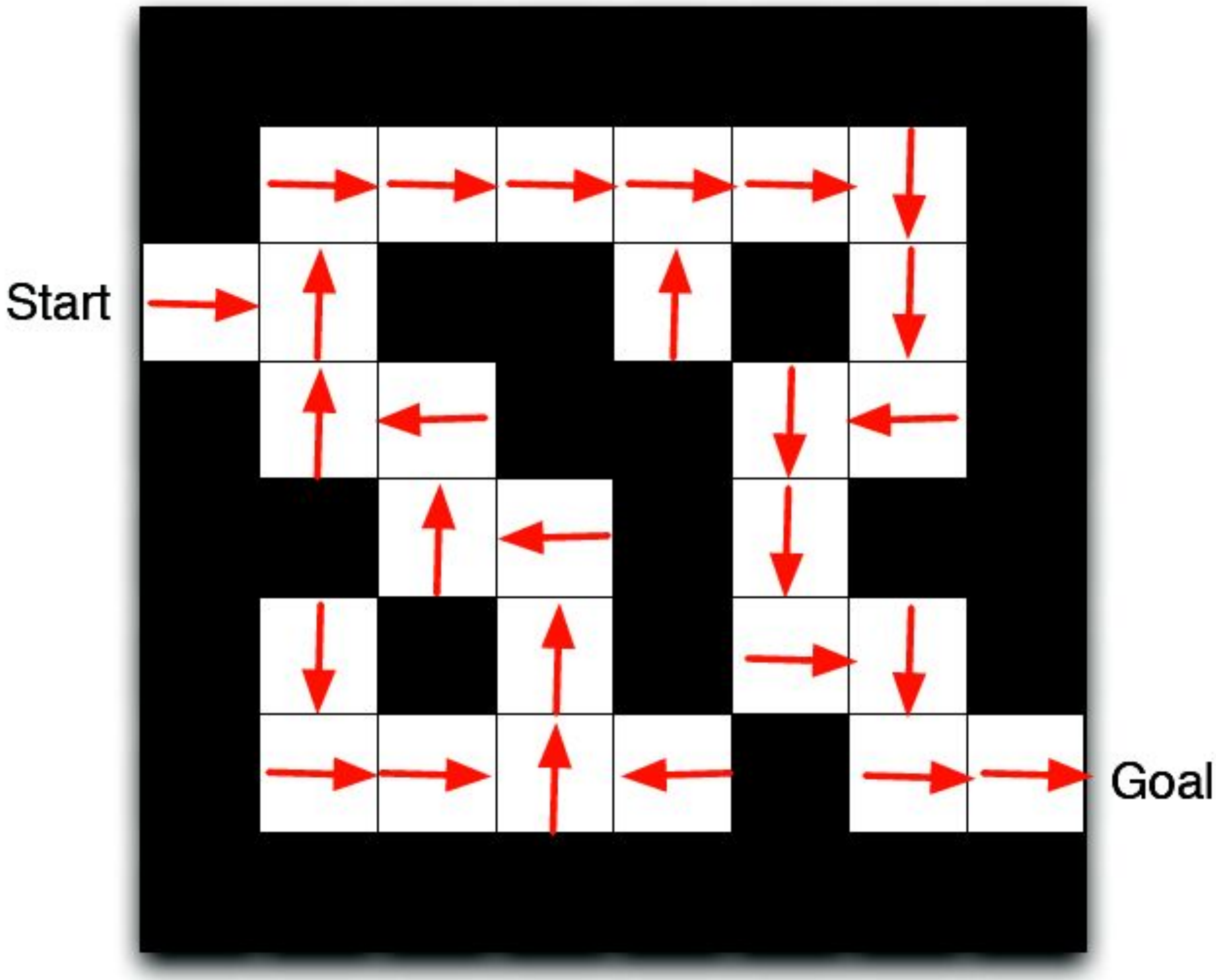


Maze Example



- Rewards: -1 per time-step
- Actions: N, E, S, W
- States: Agent's location

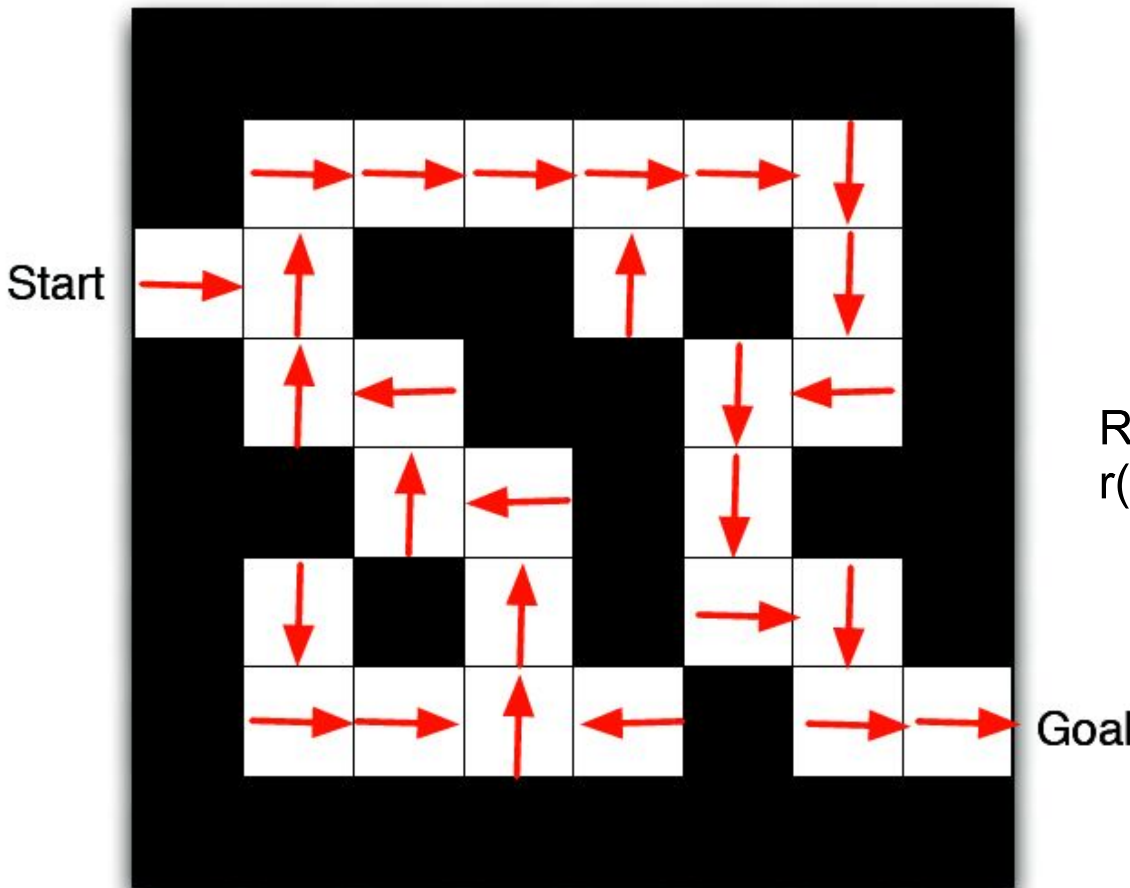
Maze Example: Policy



- Arrows represent policy $\pi(s)$ for each state s

Maze Example: Policy

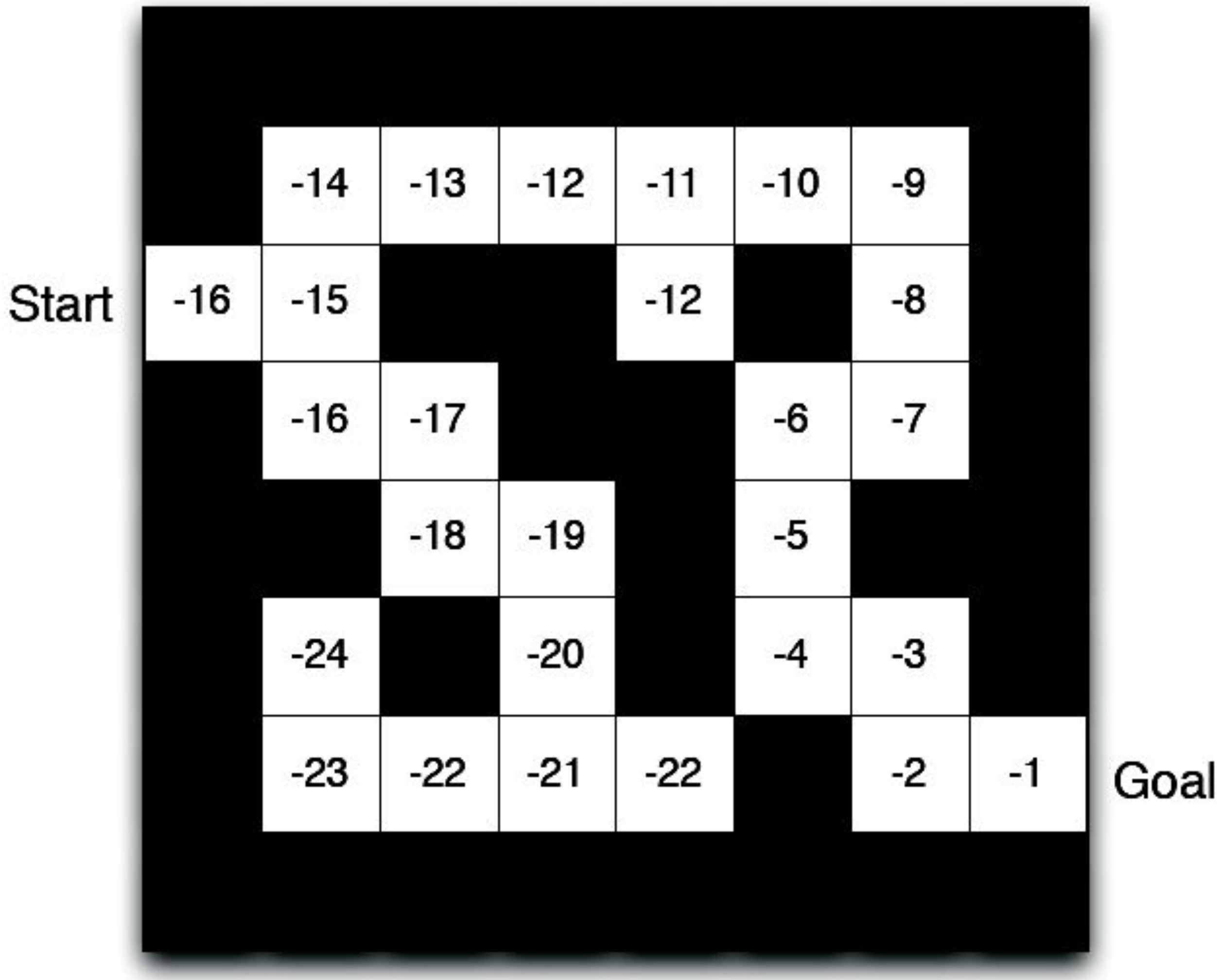
$$V^{\pi}(s_t) = E\left[r_{t+1} + r_{t+2} + \cdots + r_{t+T}\right] = E\left[\sum_{i=1}^{T} r_{t+i}\right]$$



Reward = -1 at each step, i.e. r(s,a)=-1

- Arrows represent policy $\pi(s)$ for each state s

Maze example: value function



- Numbers represent value $v_\pi(s)$ of each state s

Optimal qualities

- The value (utility) of a state s:

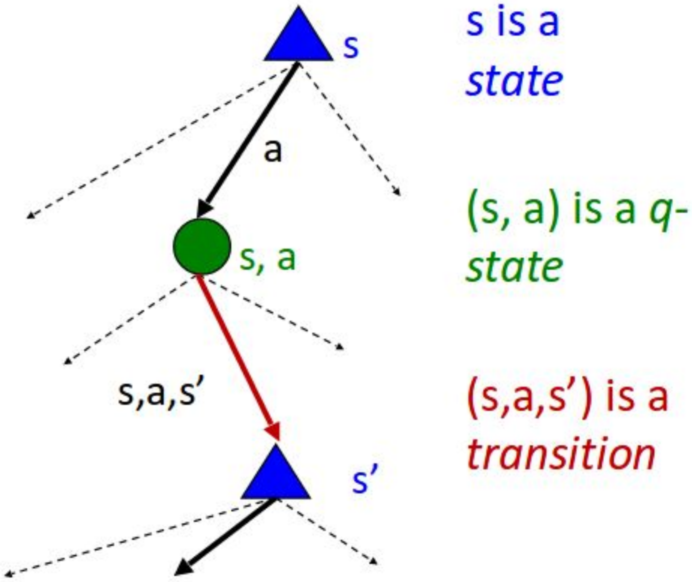
  $V^*(s)$ = expected utility starting in s and acting optimally

- The value (utility) of a q-state (s,a):

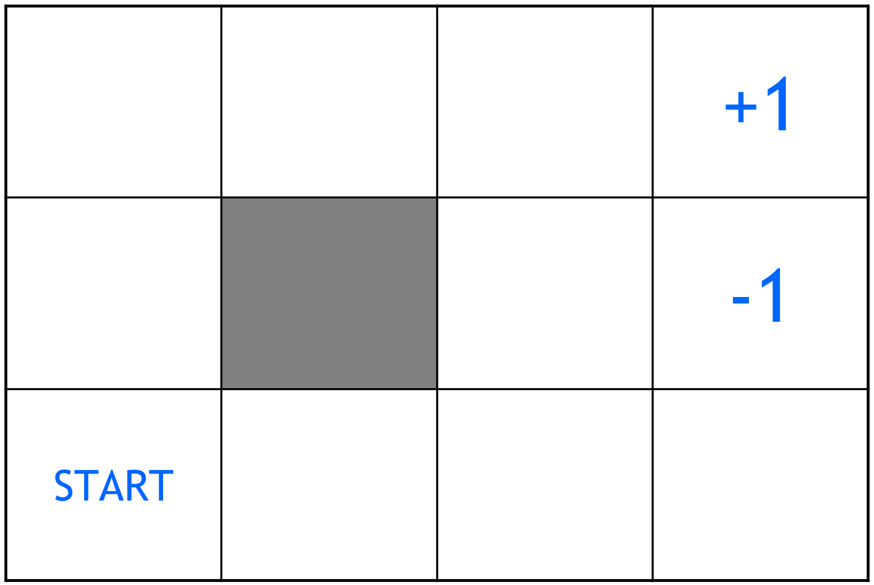
  $Q^*(s,a)$ = expected utility starting out having taken action a from state s and (thereafter) acting optimally

- The optimal policy:

  $\pi^*(s)$ = optimal action from state s



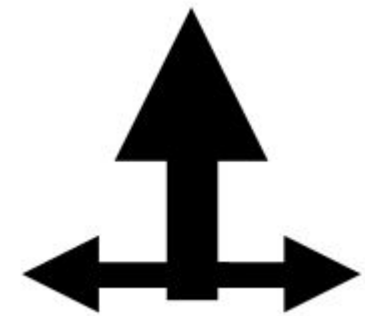
Robot in a stochastic room



actions: UP, DOWN, LEFT, RIGHT

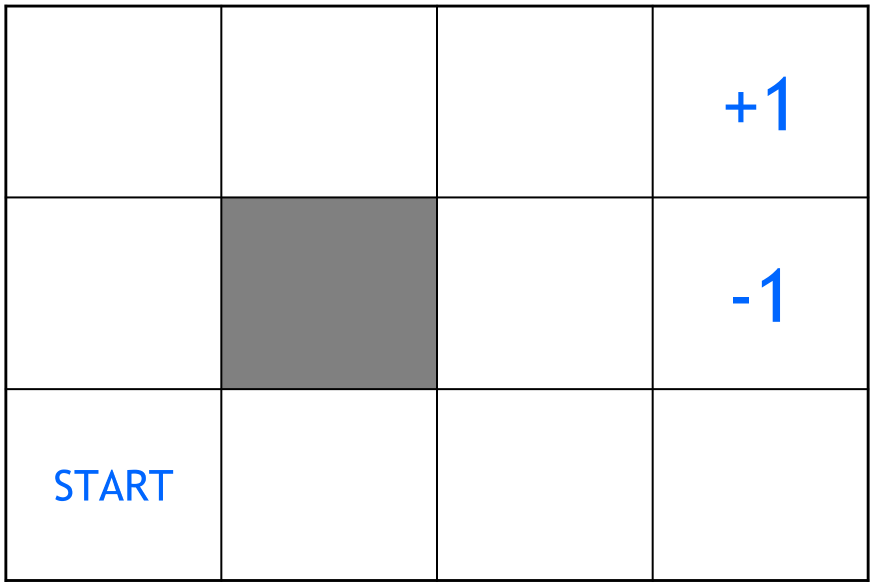
UP

80% move UP
10% move LEFT
10% move RIGHT



reward +1 at [4,3], -1 at [4,2]
reward -0.04 for each step

Robot if not stochastic



actions: UP, DOWN, LEFT, RIGHT

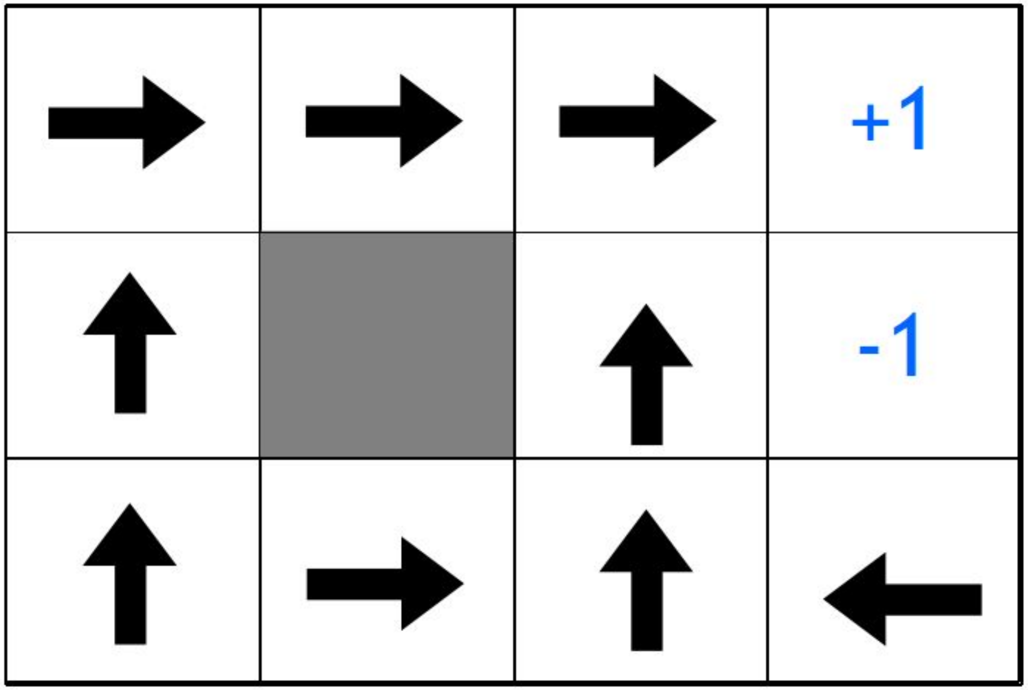
UP

100%   move UP

reward +1 at [4,3], -1 at [4,2]
reward -0.04 for each step

What is the solution?

Robot if not stochastic



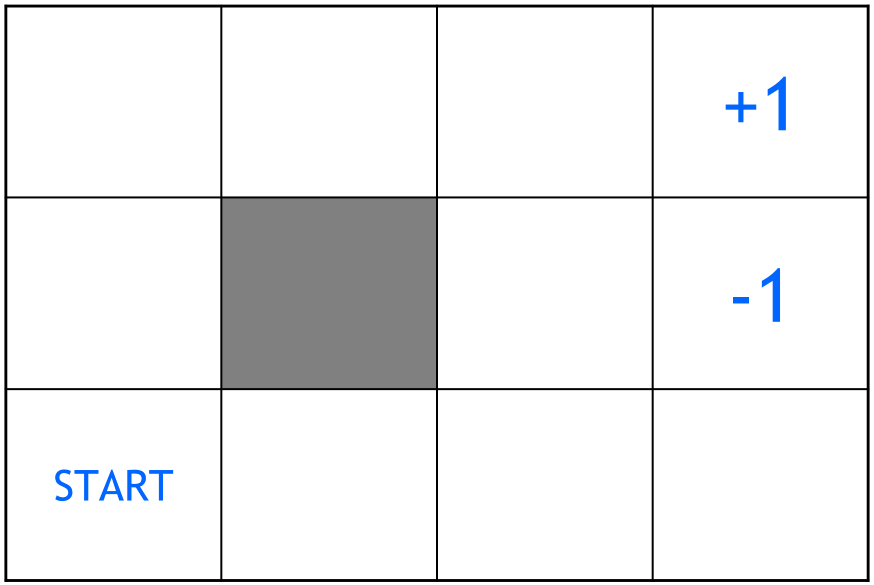
actions: UP, DOWN, LEFT, RIGHT

UP

100%   move UP

reward +1 at [4,3], -1 at [4,2]
reward -0.04 for each step

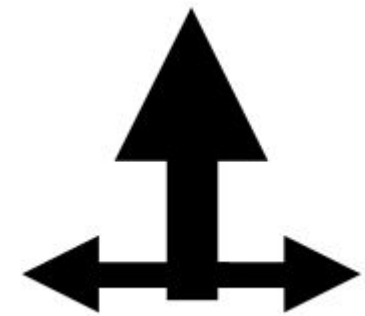
Robot in a stochastic room



actions: UP, DOWN, LEFT, RIGHT

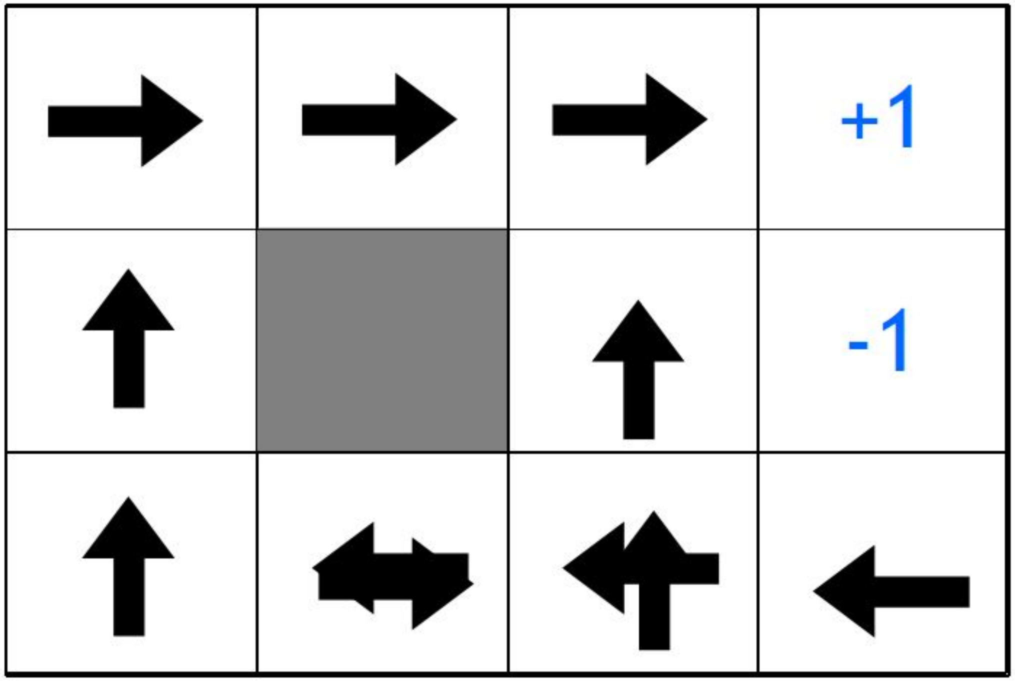
UP

80% move UP
10% move LEFT
10% move RIGHT



reward +1 at [4,3], -1 at [4,2]
reward -0.04 for each step

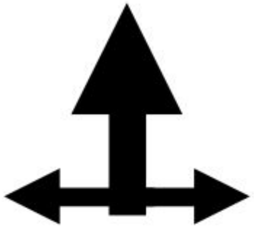
Robot in stochastic room



actions: UP, DOWN, LEFT, RIGHT

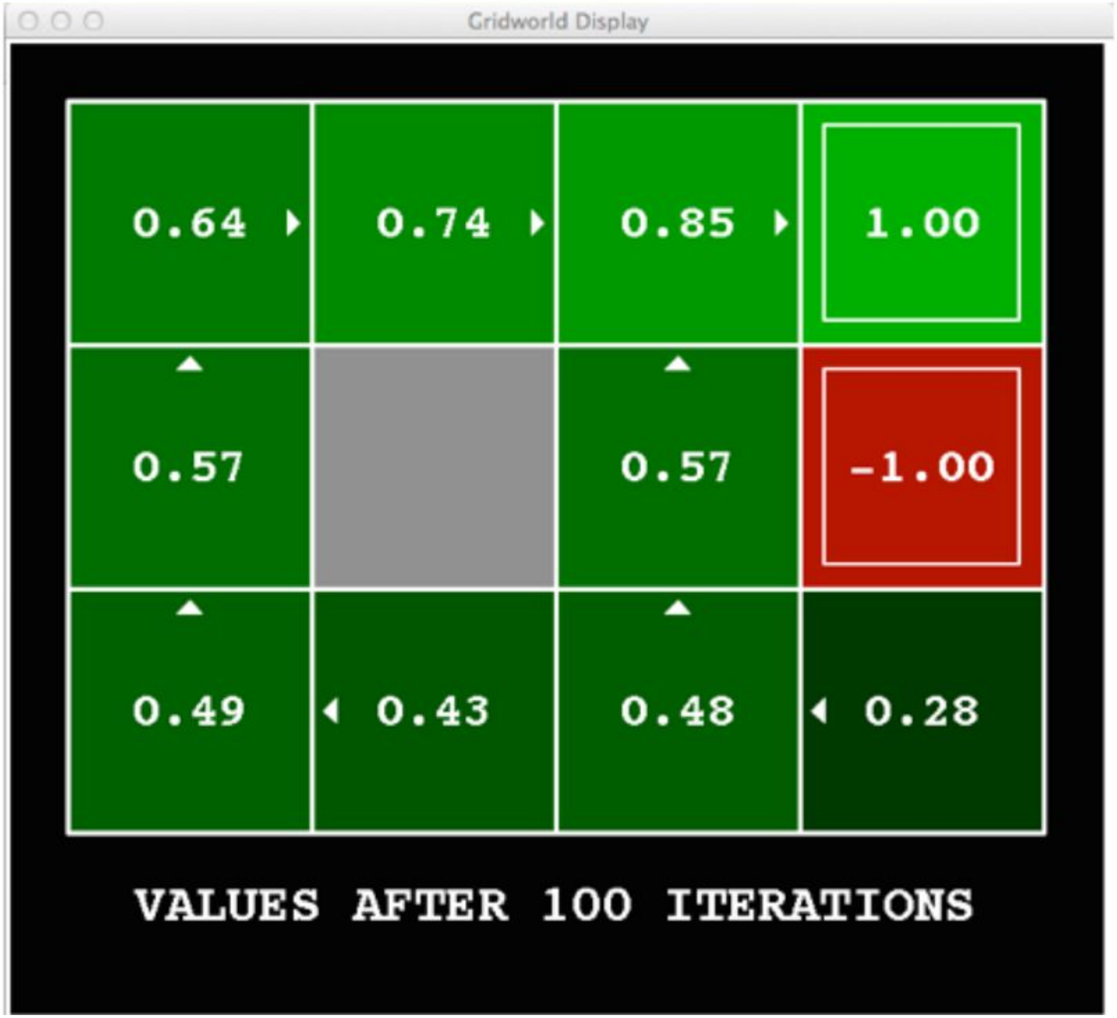
UP

80%	move UP
10%	move LEFT
10%	move RIGHT

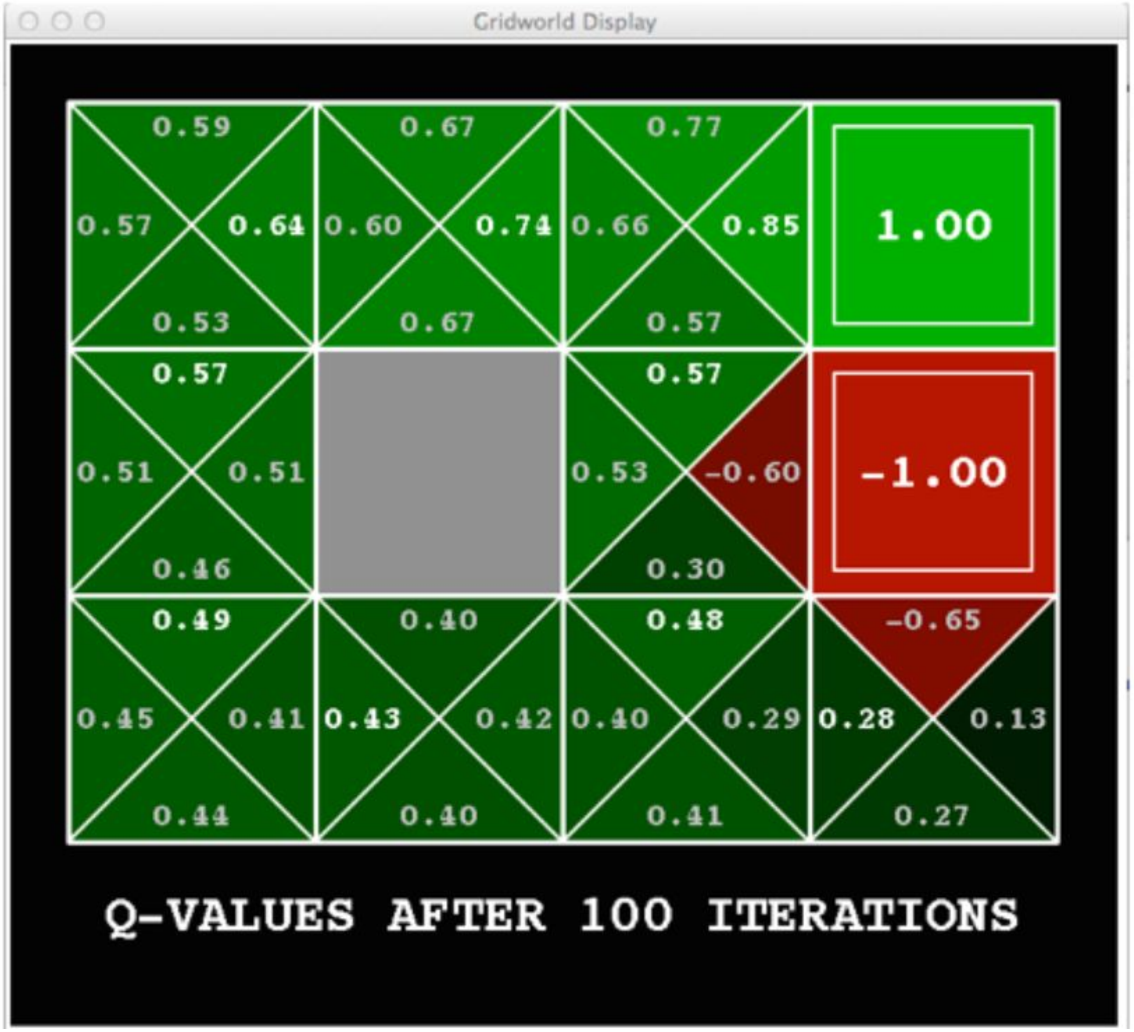


reward +1 at [4,3], -1 at [4,2]
reward -0.04 for each step

Gridworld V Values



Gridworld Q Values



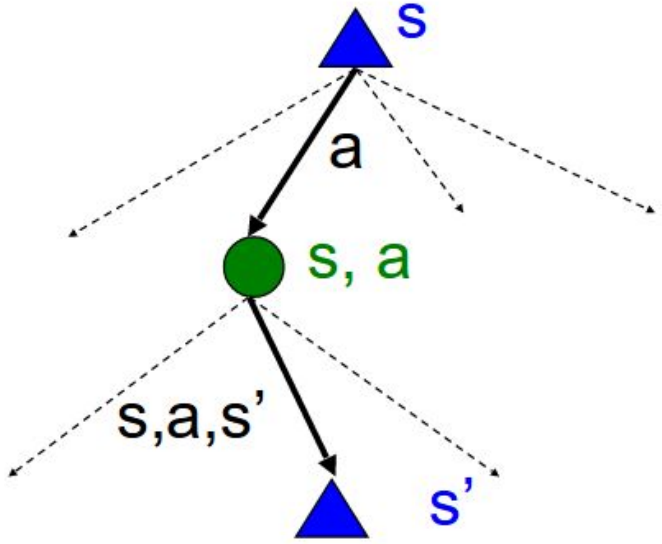
Values of states

- Recursive definition of (optimal value)

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

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

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

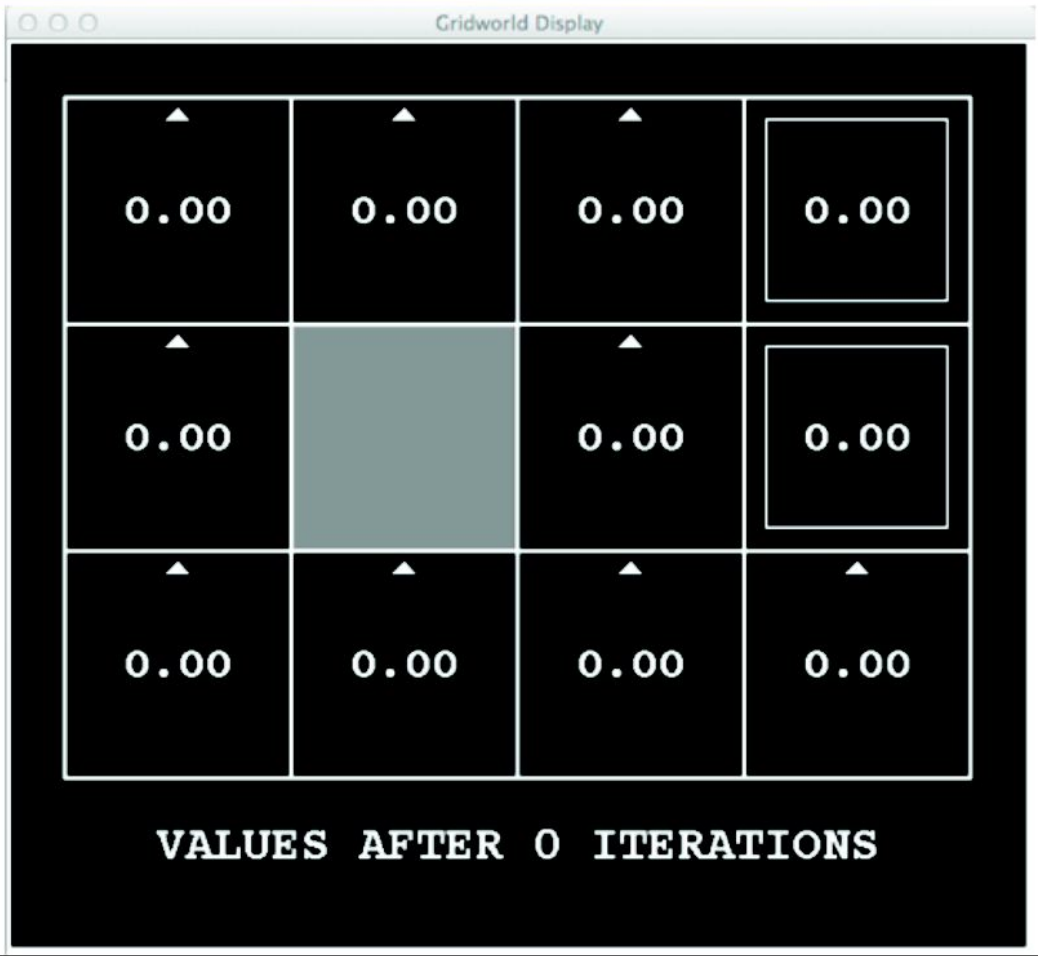


"Bellman equation"

stat e	V*

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

k=0

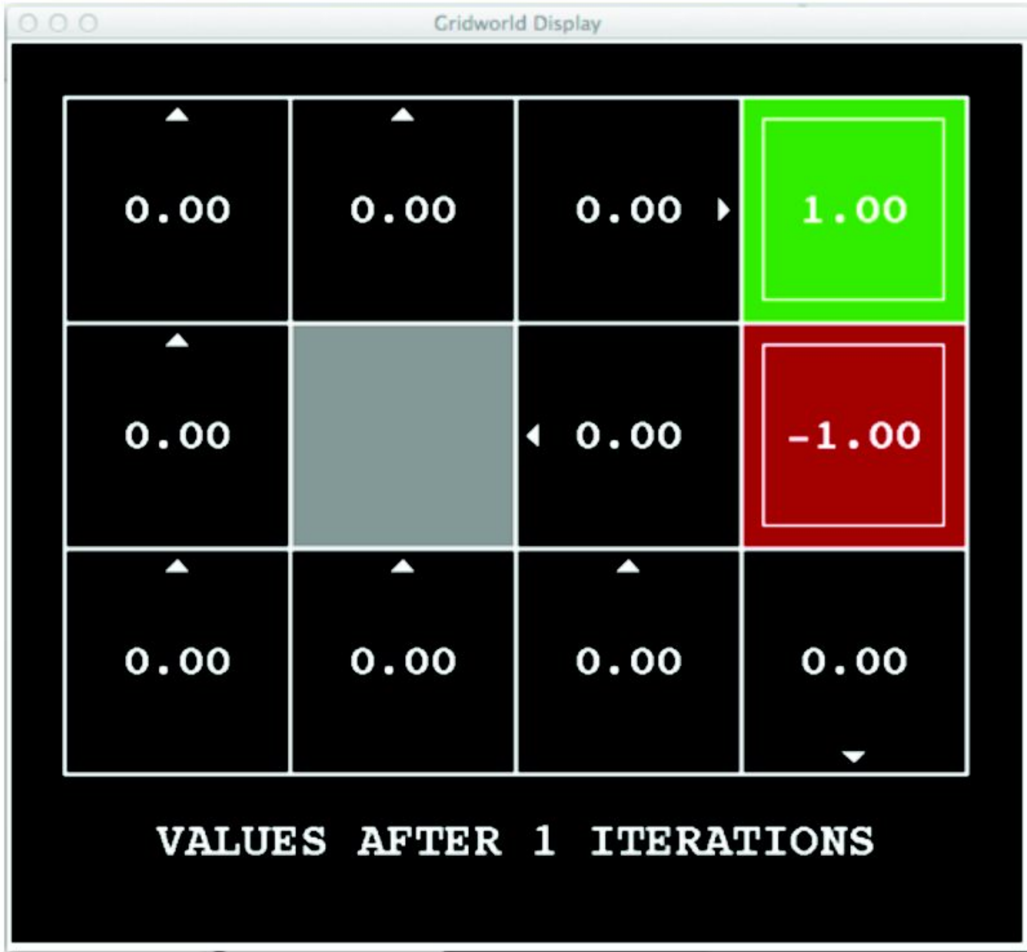


Noise = 0.2
Discount = 0.9
Living reward = 0

state	V^*

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

k=1



Noise = 0.2
Discount = 0.9
Living reward = 0

stat	V
e	*

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

k=2

Noise = 0.2
Discount = 0.9
Living reward = 0

state	V^*

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

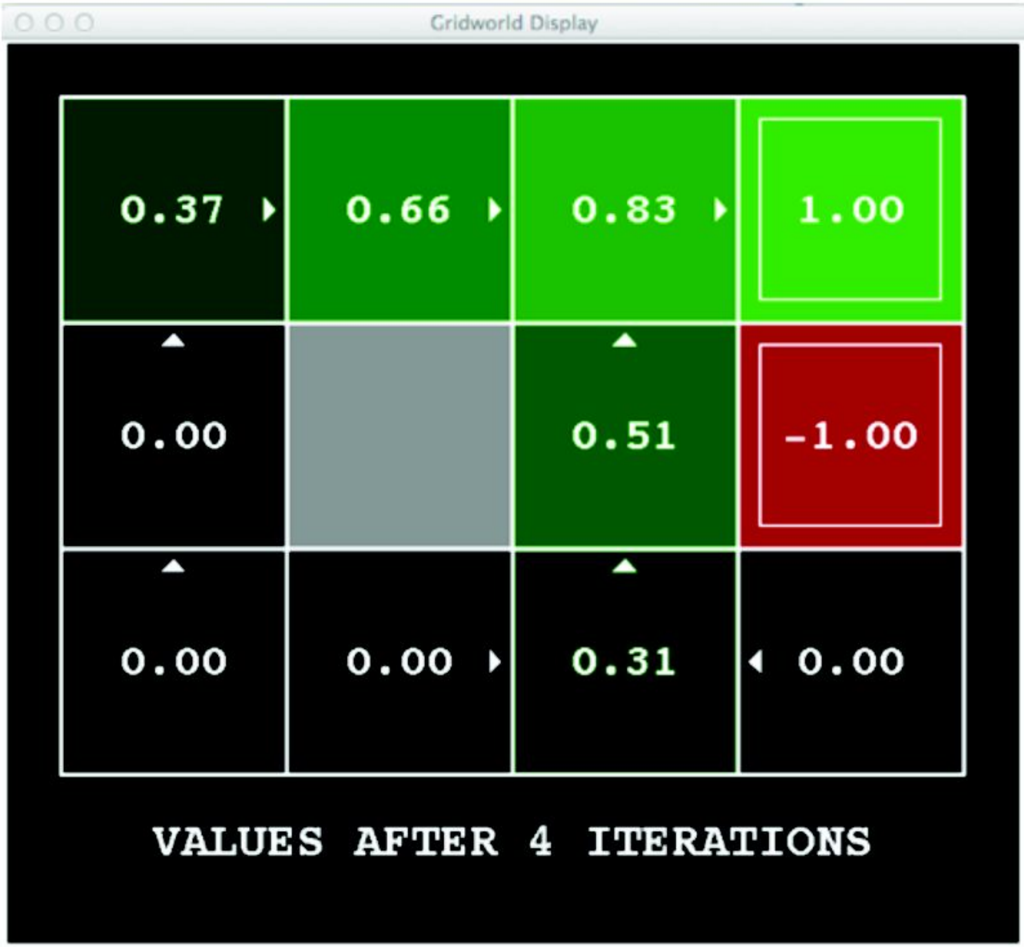
k=3

Noise = 0.2
Discount = 0.9
Living reward = 0

state	V^*

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

k=4



Noise = 0.2
Discount = 0.9
Living reward = 0

state	V^*

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

k=5

Noise = 0.2
Discount = 0.9
Living reward = 0

stat e	V *

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

k=6

Noise = 0.2
Discount = 0.9
Living reward = 0

state	V^*

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

k=7

Noise = 0.2
Discount = 0.9
Living reward = 0

stat e	V *

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

k=8

Noise = 0.2
Discount = 0.9
Living reward = 0

state	V^*

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

k=9

Noise = 0.2
Discount = 0.9
Living reward = 0

state	V^*

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

k=10

Noise = 0.2
Discount = 0.9
Living reward = 0

stat e	V *

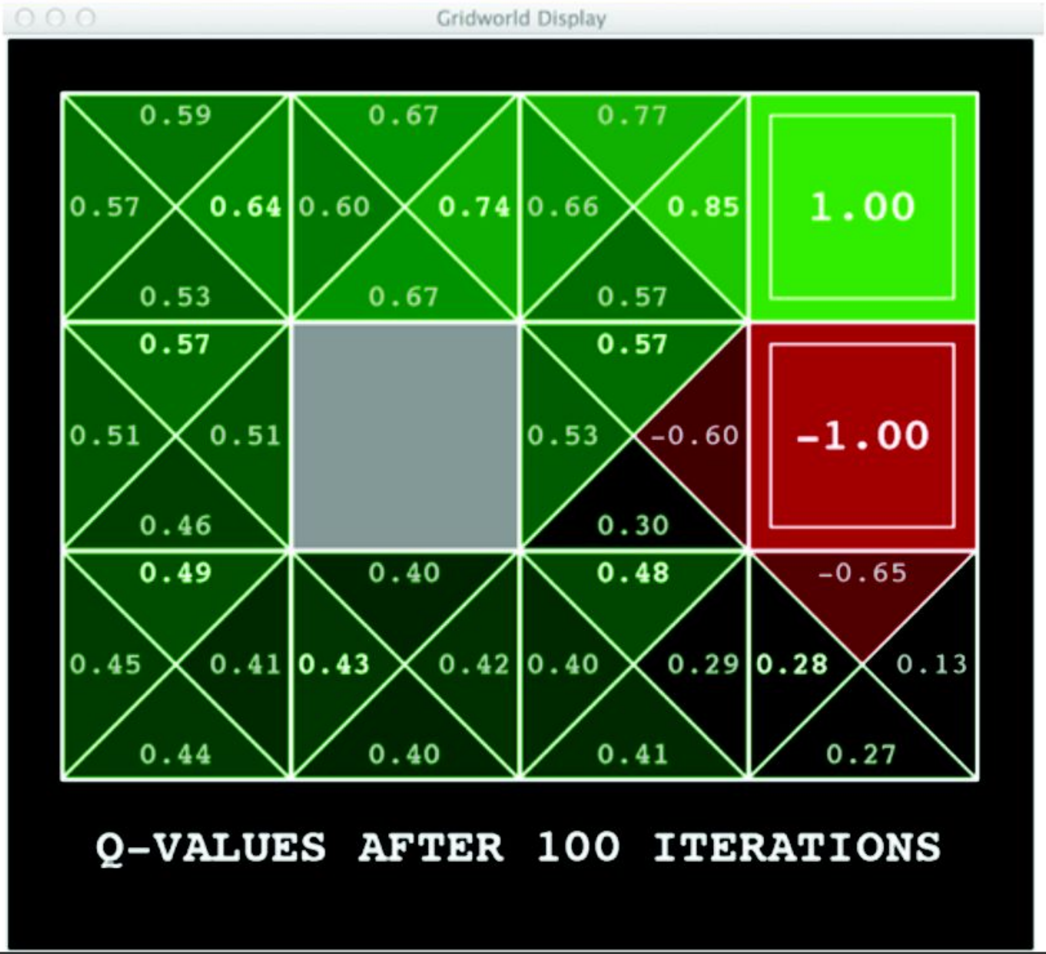
$$V^*(s) = \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^*(s')\right]$$

k=100

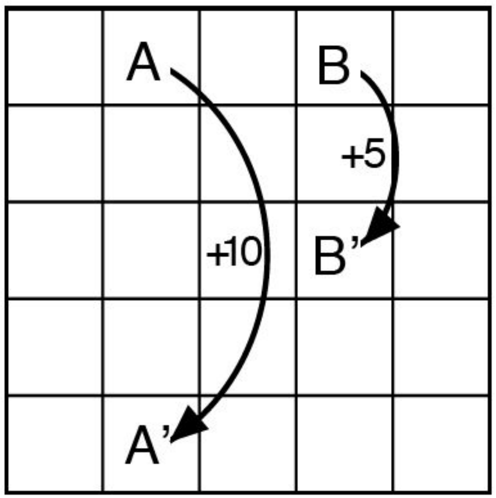
Noise = 0.2
Discount = 0.9
Living reward = 0

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

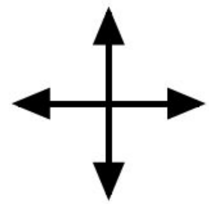
Gridworld: Q*



Gridworld Example: Prediction



(a)

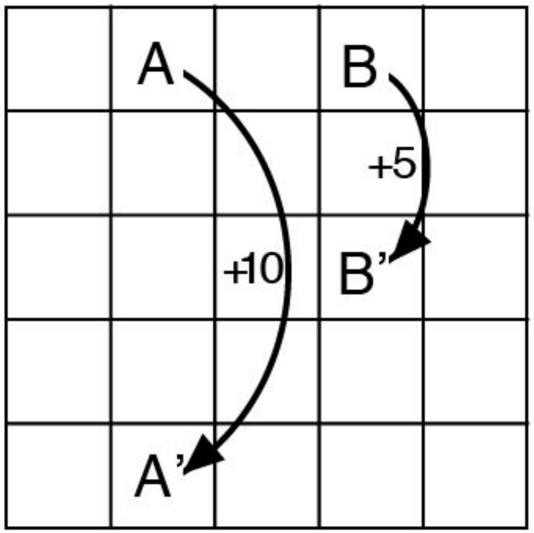


3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

(b)

What is the value function for the uniform random policy?

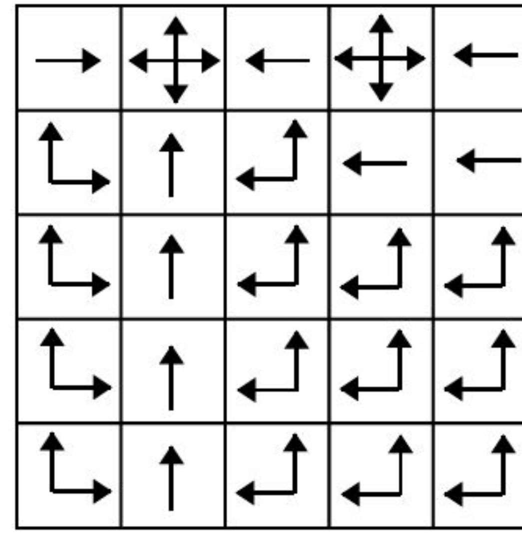
Gridworld Example: Control



a) gridworld

22.0	24.4	22.0	19.4	17.5
19.8	22.0	19.8	17.8	16.0
17.8	19.8	17.8	16.0	14.4
16.0	17.8	16.0	14.4	13.0
14.4	16.0	14.4	13.0	11.7

b) v_*



c) π_*

What is the optimal value function over all possible policies?

What is the optimal policy?

Learning utility, i.e. V(s) or Q(s,a)

- If the model is known
  - Dynamic Programming: solve a set of equations
- If the model is not known
  - Learn the model
    - Dynamic programming
  - Do not learn the model
    - From samples: Directly evaluate Q values from runs
    - From samples and policy: Use Temporal Difference to learn Q values

Example to Illustrate Model-Based vs. Model-Free: Expected Age

Goal: Compute expected age of cs188 students

Known P(A)

$$E[A] = \sum_a P(a) \cdot a \qquad = 0.35 \times 20 + \ldots$$

Without P(A), instead collect samples $[a_1, a_2, \ldots a_N]$

Unknown P(A): "Model Based"

$$\hat{P}(a) = \frac{\text{num}(a)}{N}$$

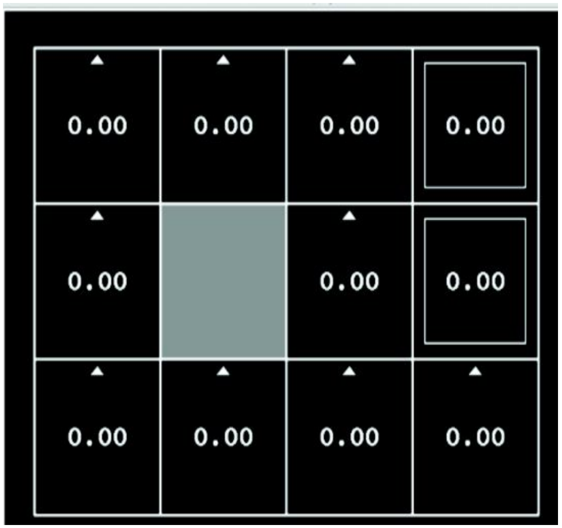
$$E[A] \approx \sum_a \hat{P}(a) \cdot a$$

Unknown P(A): "Model Free"

$$E[A] \approx \frac{1}{N} \sum_i a_i$$

Known Model: Known transition probabilities and reward

- Transition probabilities and rewards are known.



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

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

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

$V^*(s_0)$= ...

$V^*(s_1)$= ...

$V^*(s_2)$= …

...

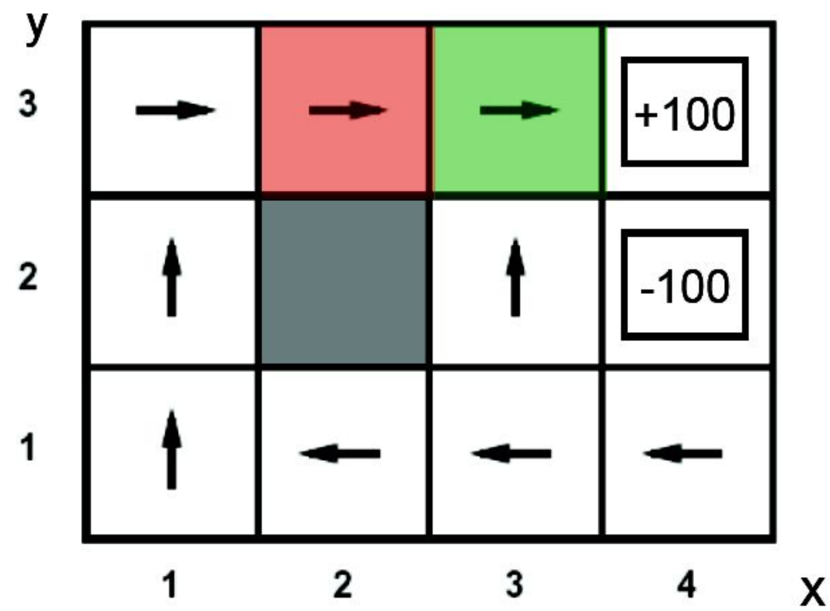
$V^*(s_n)$= ...

- Solve the set of equations with n equations and n unknowns

Unknown model: Learn the model
Estimate transition probability from samples

- Episodes:

(1,1) up -1	(1,1) up -1
(1,2) up -1	(1,2) up -1
(1,2) up -1	(1,3) right -1
(1,3) right -1	(2,3) right -1
(2,3) right -1	(3,3) right -1
(3,3) right -1	(3,2) up -1
(3,2) up -1	(4,2) exit -100
(3,3) right -1	(done)
(4,3) exit +100	
(done)	



$\gamma = 1$

T(<3,3>, right, <4,3>) = 1 / 3

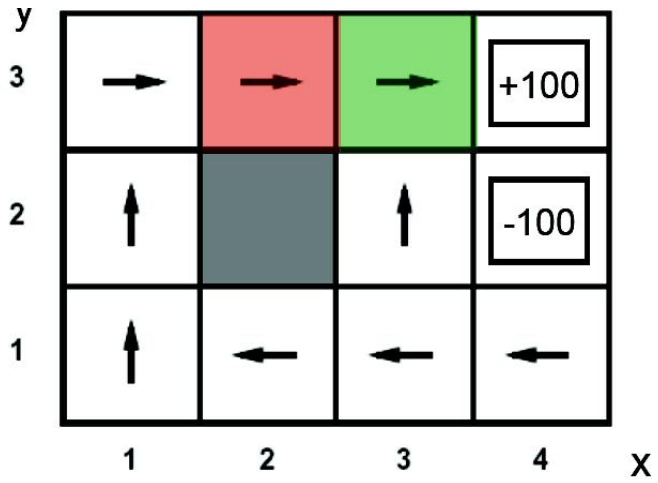
T(<2,3>, right, <3,3>) = 2 / 2

Unknown model: do not learn the model: Direct evalution of Value function

- Repeatedly execute the policy π
- Estimate the value of the state s as the average over all times the state s was visited of the sum of discounted rewards accumulated from state s onwards

  - Episodes:

(1,1) up -1	(1,1) up -1
(1,2) up -1	(1,2) up -1
(1,2) up -1	(1,3) right -1
(1,3) right -1	(2,3) right -1
(2,3) right -1	(3,3) right -1
(3,3) right -1	(3,2) up -1
(3,2) up -1	(4,2) exit -100
(3,3) right -1	(done)
(4,3) exit +100	
(done)	



$\gamma = 1$, $R = -1$

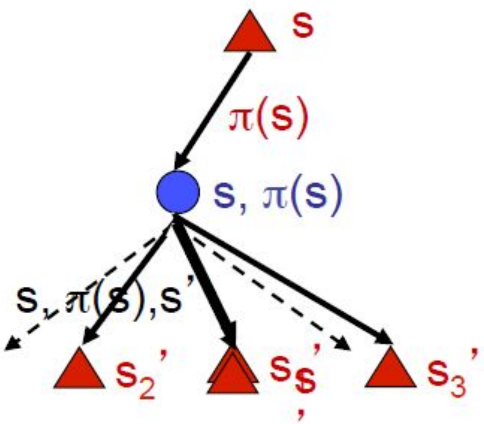
V(2,3) ~ (96 + -103) / 2 = -3.5

V(3,3) ~ (99 + 97 + -102) / 3 = 31.3

Unknown model :Sample-Based Policy Evaluation

$$V_{i+1}^{\pi}(s) \leftarrow \sum_{s'} T(s,\pi(s),s')[R(s,\pi(s),s') + \gamma V_i^{\pi}(s')]$$

- Who needs T and R? Approximate the expectation with samples of s’ (drawn from T!)



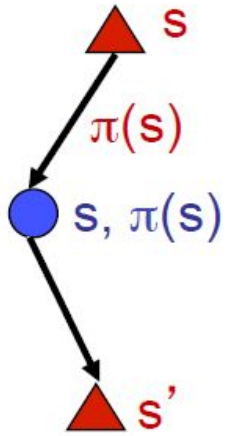
$$sample_1 = R(s,\pi(s),s_1') + \gamma V_i^{\pi}(s_1')$$
$$sample_2 = R(s,\pi(s),s_2') + \gamma V_i^{\pi}(s_2')$$
$$\ldots$$
$$sample_k = R(s,\pi(s),s_k') + \gamma V_i^{\pi}(s_k')$$

$$V_{i+1}^{\pi}(s) \leftarrow \frac{1}{k}\sum_i sample_i$$

Almost! But we can’t rewind time to get sample after sample from state s.

Unknown model :Sample-Based Policy Evaluation Temporal Difference Learning

- Big idea: learn from every experience!
  - Update V(s) each time we experience (s,a,s',r)
  - Likely s' will contribute updates more often



- Temporal difference learning
  - Policy still fixed!
  - Move values toward value of whatever successor occurs: running average!

Sample of V(s): $$sample = R(s, \pi(s), s') + \gamma V^{\pi}(s')$$

Update to V(s): $$V^{\pi}(s) \leftarrow (1-\alpha)V^{\pi}(s) + (\alpha)sample$$

Same update: $$V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha(sample - V^{\pi}(s))$$

Q-learning

- Q-Learning: sample-based Q-value iteration
- Learn Q*(s,a) values
  - Receive a sample (s,a,s',r)
  - Consider your old estimate: $Q(s,a)$
  - Consider your new sample estimate:

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

$$sample = R(s,a,s') + \gamma \max_{a'} Q(s',a')$$

  - Incorporate the new estimate into a running average:

$$Q(s,a) \leftarrow (1-\alpha) Q(s,a) + (\alpha) \left[sample \right]$$

Q-learning algorithm

Q	s	a
?	0	0
?	0	1
?	1	0
?	.....	

Initialize all $Q(s, a)$ arbitrarily
For all episodes
  Initalize s
  Repeat
    Choose a using policy derived from Q, $a = \pi(s)$
    Take action a, observe r and s'
    Update $Q(s, a)$:
      $Q(s, a) \leftarrow Q(s, a) + \eta(r + \gamma \max_{a'} Q(s', a') - Q(s, a))$
    $s \leftarrow s'$
  Until s is terminal state

Figure 18.5 Q learning, which is an off-policy temporal difference algorithm.

Q-learning algorithm

Q	s	a
?	0	0
?	0	1
?	1	0
?	…..	

Always executes the action that it thinks "best"

Initialize all $Q(s,a)$ arbitrarily
For all episodes
    Initalize s
    Repeat
        Choose a using policy derived from Q,   a = π (s)
        Take action a, observe r and s'
        Update $Q(s,a)$:
            $Q(s,a) \leftarrow Q(s,a) + \eta(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$
        $s \leftarrow s'$
    Until s is terminal state

Figure 18.5 Q learning, which is an off-policy temporal difference algorithm.

Q-learning algorithm

Q	s	a
?	0	0
?	0	1
?	1	0
?	…..	

Initialize all $Q(s, a)$ arbitrarily
For all episodes
    Initalize s
    Repeat
        Choose a using policy derived from Q, e.g., ϵ-greedy
        Take action a, observe r and s'
        Update $Q(s, a)$:
            $Q(s, a) \leftarrow Q(s, a) + \eta(r + \gamma \max_{a'} Q(s', a') - Q(s, a))$
        $s \leftarrow s'$
    Until s is terminal state

or softmax: $\boxed{P(a|s) = \frac{\exp Q(s, a)}{\sum_{b \in \mathcal{A}} \exp Q(s, b)}}$

Figure 18.5 Q learning, which is an off-policy temporal difference algorithm.

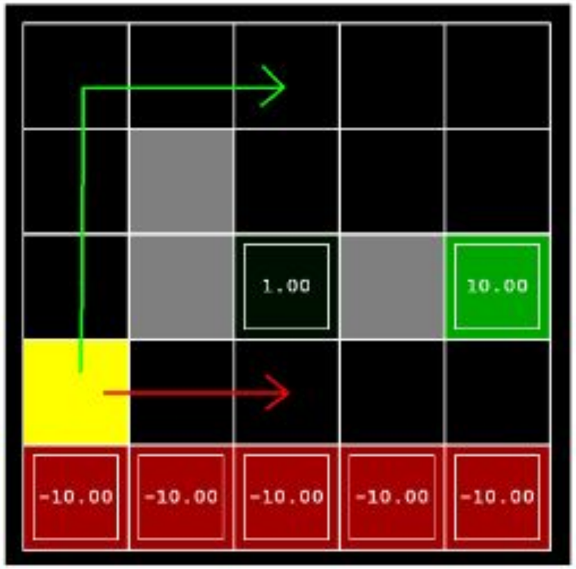
Exploration Strategies

- ε-greedy: With pr ε,choose one action at random uniformly; and choose the best action with pr 1-ε
- Probabilistic:

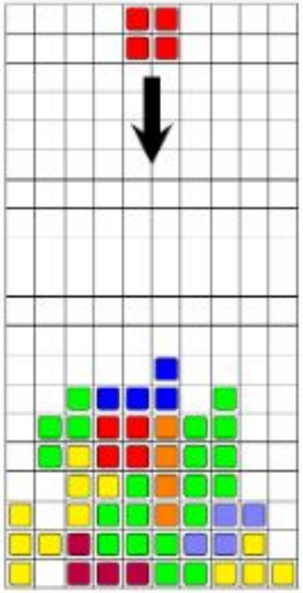
$$P(a \mid s) = \frac{\exp Q(s,a)}{\sum_{b=1}^{\mathrm{A}} \exp Q(s,b)}$$

Can tabular methods scale?

- Discrete environments



Gridworld
10^1

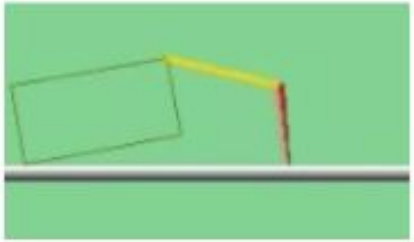


Tetris
10^60

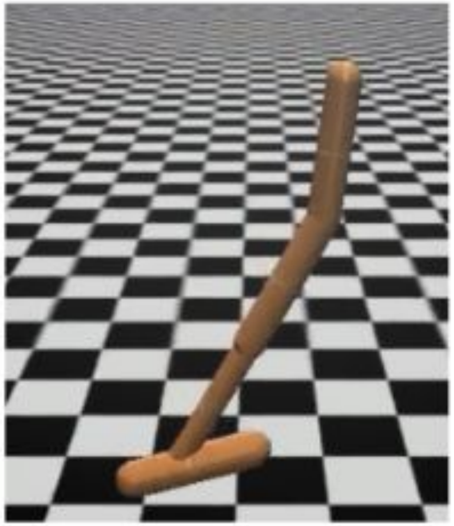
Atari
10^308 (ram) 10^16992 (pixels)

Can tabular methods scale?

- Continuous environments (by crude discretization)



Crawler
10^2



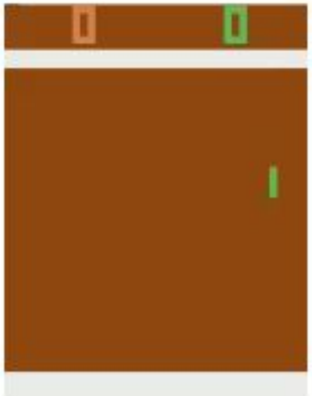
Hopper
10^10

Humanoid
10^100

Generalizing Across States

- Basic Q-Learning keeps a table of all q-values
- In realistic situations, we cannot possibly learn about every single state!
  - Too many states to visit them all in training
  - Too many states to hold the q-tables in memory
- Instead, we want to generalize:
  - Learn about some small number of training states from experience
  - Generalize that experience to new, similar situations
  - This is a fundamental idea in machine learning, and we'll see it over and over again

Deep Reinforcement Learning



Pong

Enduro

Beamrider

Q*bert

- From pixels to actions
- Same algorithm (with effective tricks)
- CNN function approximator, w/ 3M free parameters