# Machine Learning

Lecture 15

Reinforcement Learning

## Outline

- Reinforcement Learning Problem
- Principle of optimality
- Markov Decision Process
- Monte-Carlo Methods
- Temporal Difference Learning

## Any definition

RL brings a way of programming agents by reward and punishment without specifying *how* the task is to be achieved. (Kaelbling,1996)

→Based on trial-error interactions

→A set of problems rather than a set of techniques

## Control Learning

Learning to choose actions

- Robot learning to dock to a battery station
- Learning to choose actions to optimize a factory output
- Learning to play Backgammon

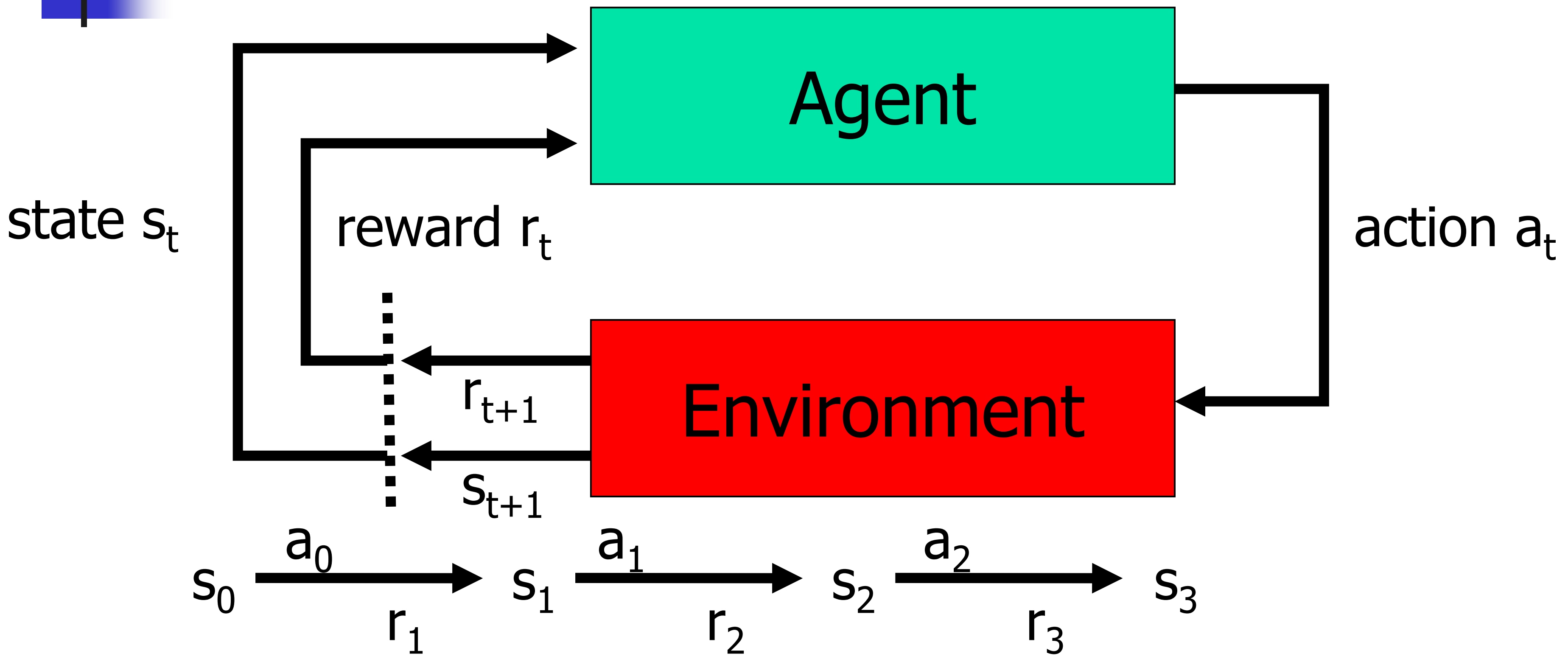
Problem characteristics:

- Delayed reward
- No direct feedback (error signal) for good and bad actions
- Opportunity for active exploration
- Possibly that state is only partially observable

## Learning to play Backgammon

- Immediate reward
  - +100 win
  - -100 loose
  - 0 for all other actions/states
- Trained by playing 1.5 million games against itself (Tesauro [1995])
- Now approximately equal to the best human player

## Reinforcement Learning Problem



Goal: Learn to choose actions $a_t$ that maximize future rewards $r_1+\gamma\, r_2+ \gamma^2\, r_3+\ldots$, where $0<\gamma<1$ is a discount factor

## *Models of Optimal Behavior:*

Agent tries to maximize one of the following:

*finite-horizon* Model:

$$E\left(\sum_{t=0}^{h} r_t\right) \; ;$$

*infinite-horizon* discounted model:

$$E\left(\sum_{t=0}^{\infty} \gamma^t r_t\right) \; .$$

*average-reward* model:

$$\lim_{h \to \infty} E\left(\frac{1}{h} \sum_{t=0}^{h} r_t\right) \; .$$

## Example 1: Slot Machine

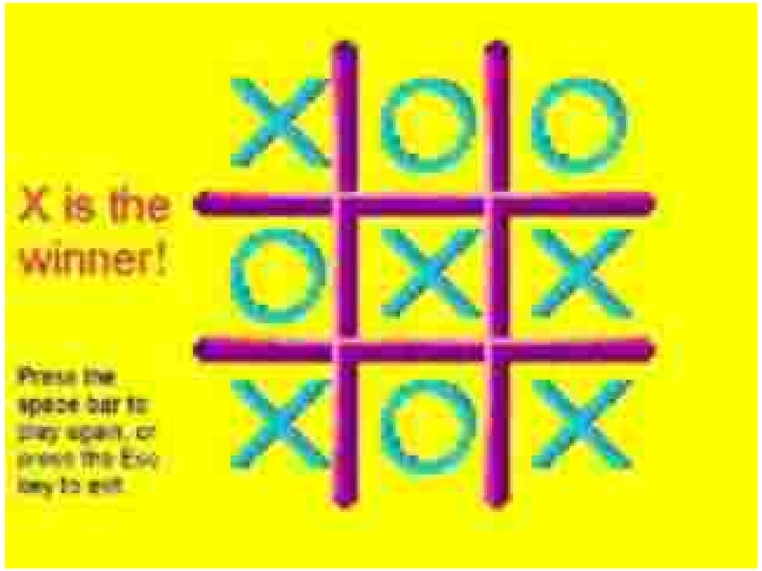
- State: configuration of slots
- Action: stopping time
- Reward: $$$

- Problem: Find $\pi: S \rightarrow A$ that
- maximizes R

## Example 2: Tic Tac Toe

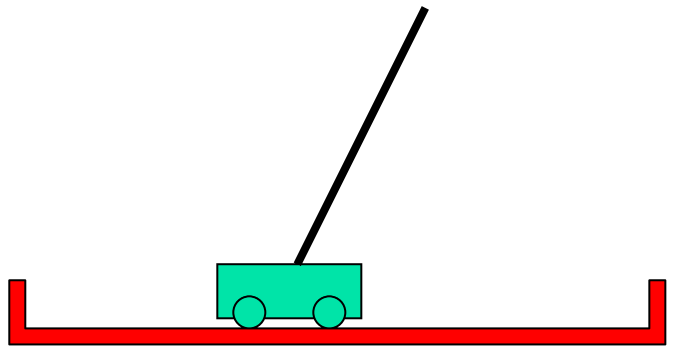
- State: board
- Action: next move
- Reward: 1 for win, -1 for loss, 0 for draw

- Problem: Find $\pi$:S$\rightarrow$A that
- maximizes R


X is the winner!
Press the space bar to play again, or press the Esc key to exit

## Example 3: Inverted Pendulum

- State: x(t),x′(t), q(t), q′(t)
- Action: Force F
  (bang bang)
- Reward: 1 for any step where pole balanced



- Problem: Find $\pi$:S→A that
- maximizes R

## Example 4: Mobile Robot

- State: location of robot, people
- Action: motion
- Reward: number of happy faces

- Problem: Find $\pi$:S$\rightarrow$A that
- maximizes R

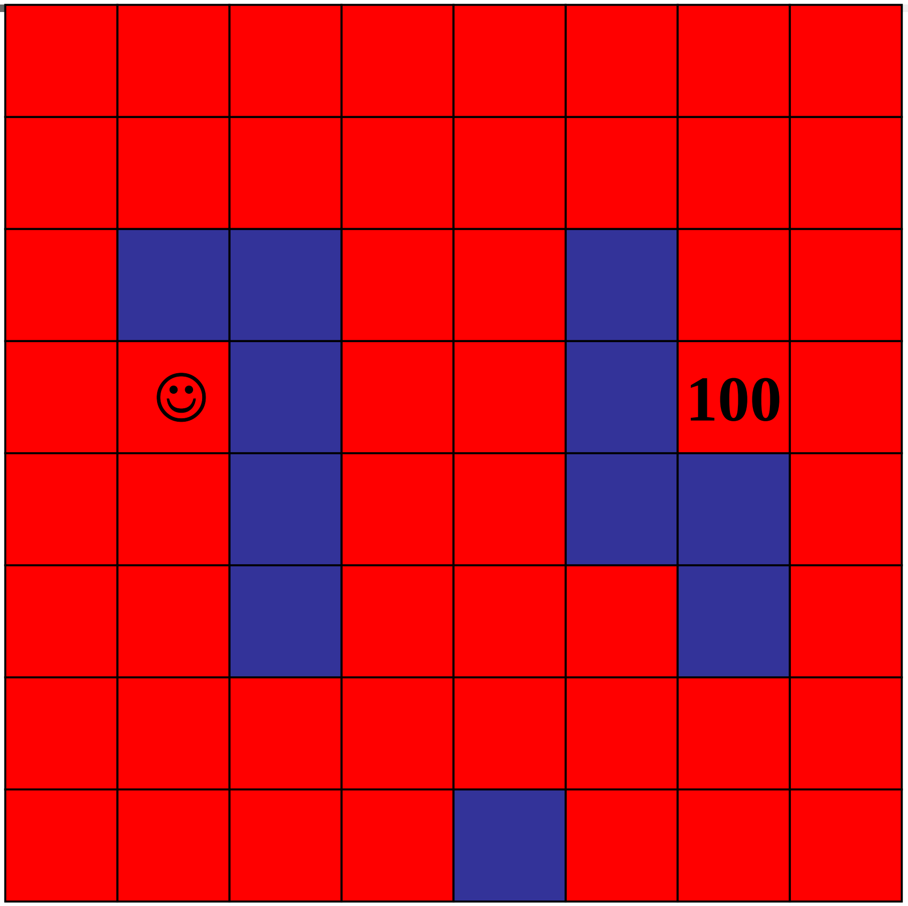
## Common Features of RL Problems

- ***Temporal Credit Assignment***
- Exploration/Exploitation
- Planning and Acting under Uncertainty

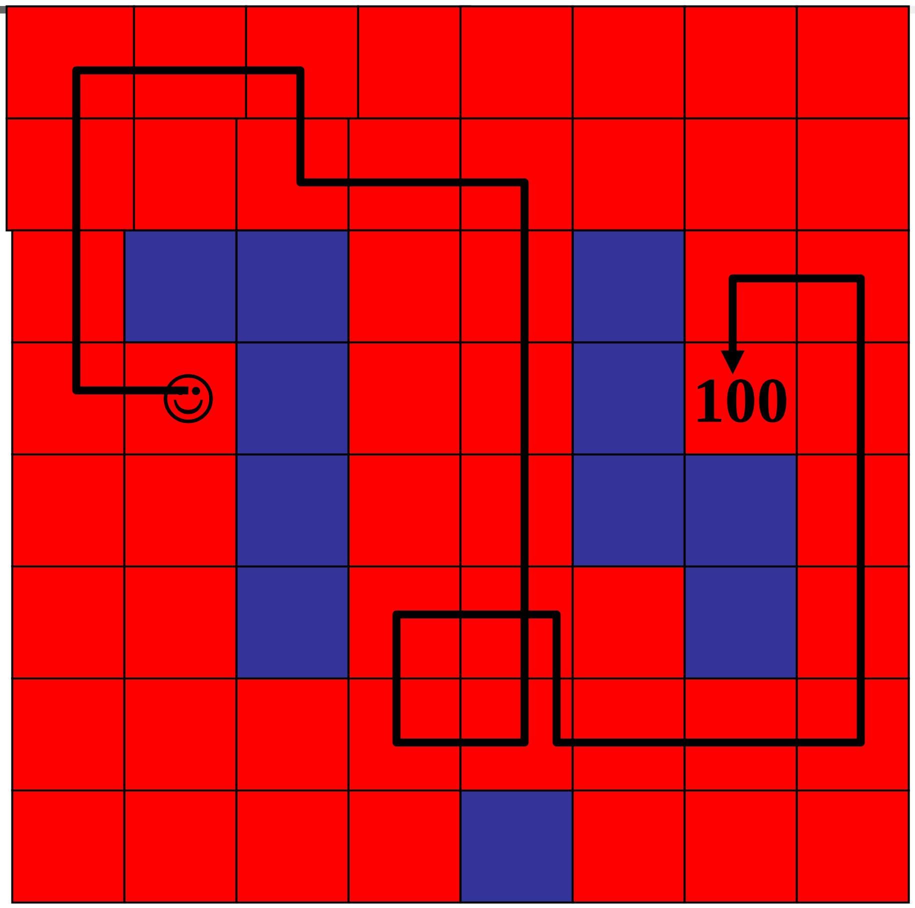
## The Principle of Optimality

- “an optimal set of decisions has the property that whatever the first decision is, the remaining decisions must be optimal w/ respect to the outcome…of the first decision.” –R. Bellman
- “if you don’t do the best you have with what you happen to have got, you will never do the best you might have done with what you should have had.” –R. Aris

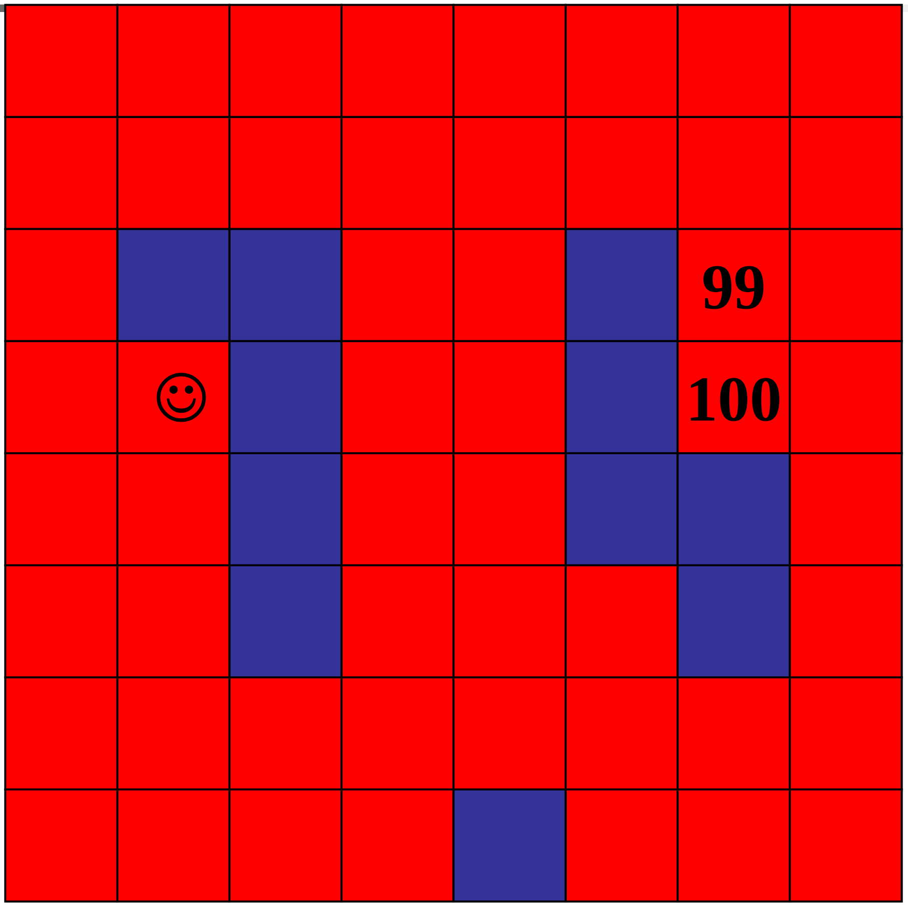
## Example



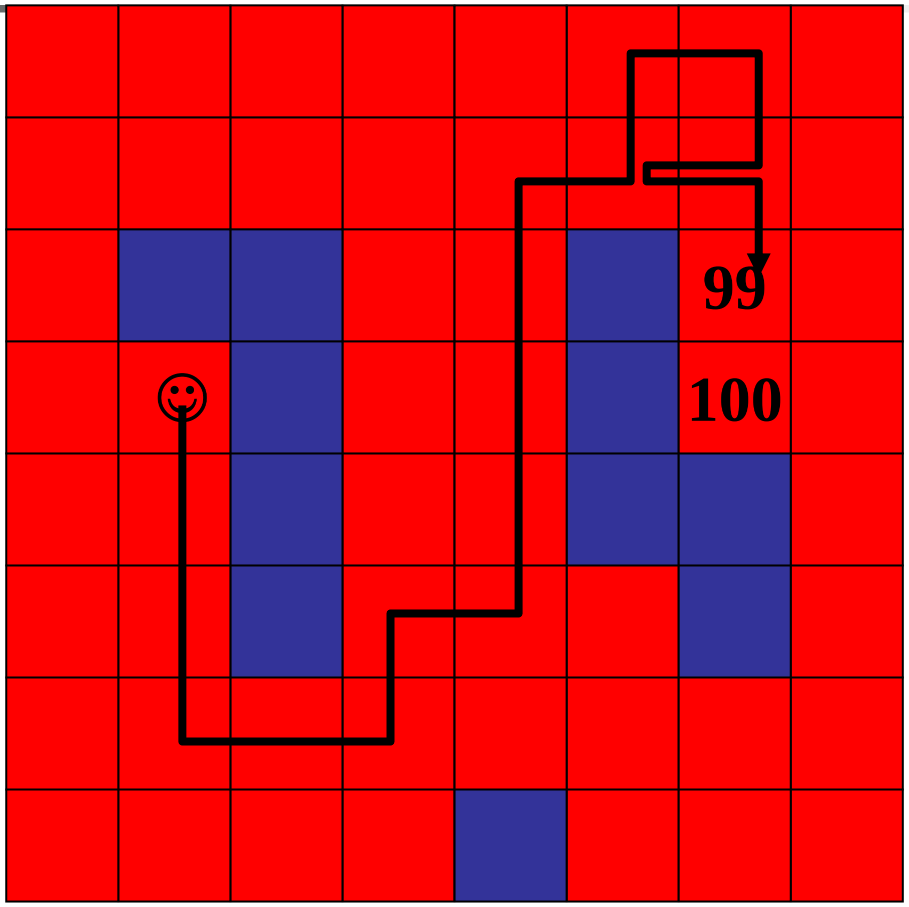
## Example



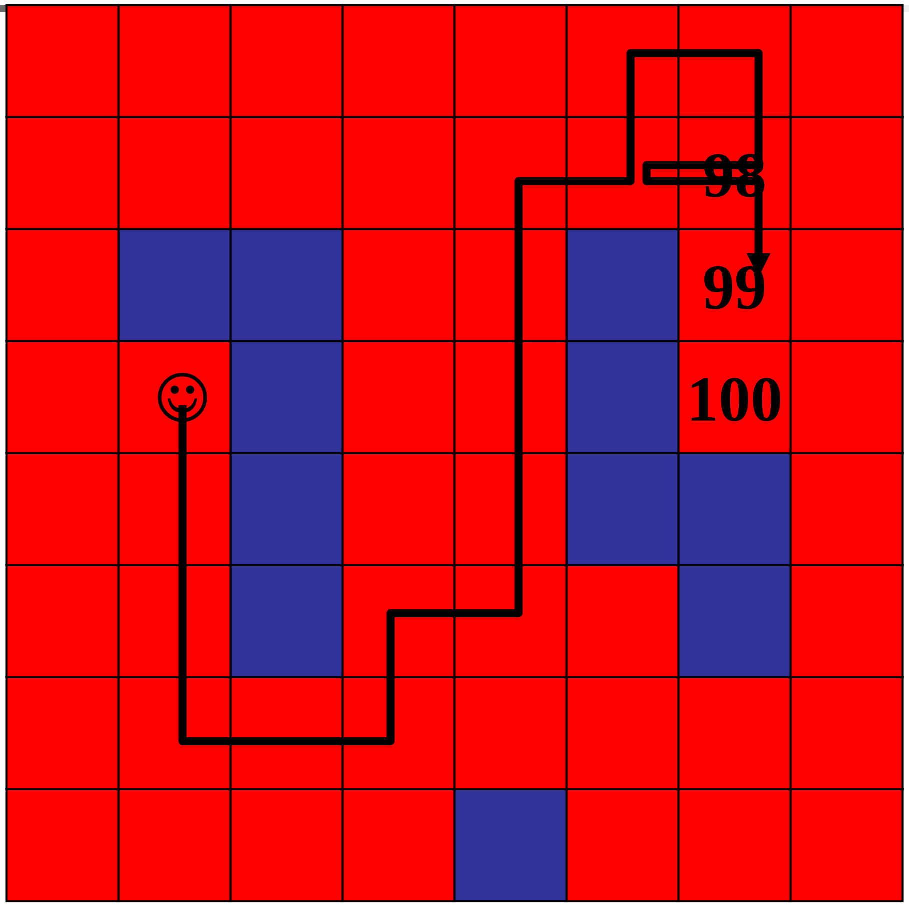
## Example



## Example



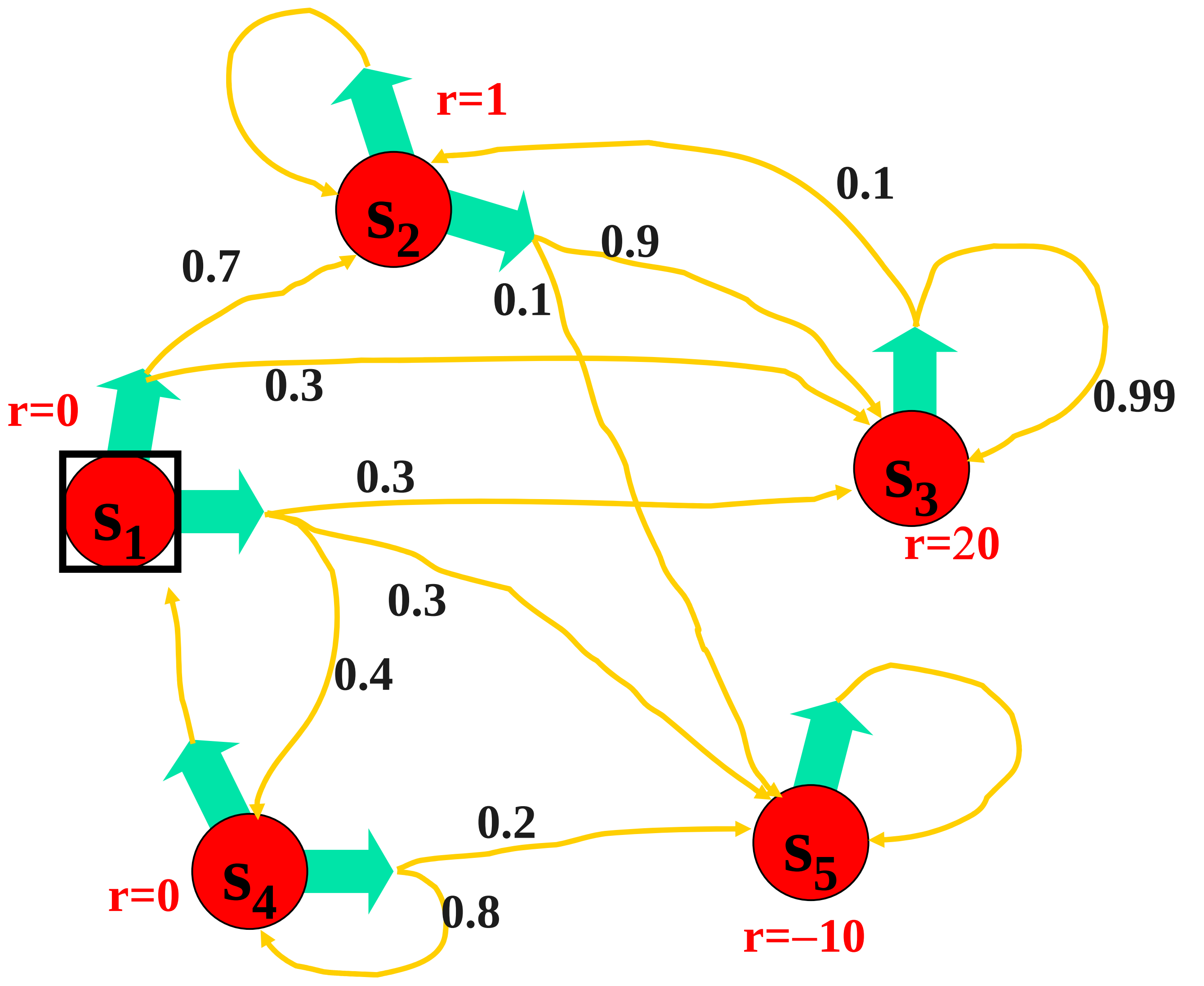
# Example



## Example

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 91 | 92 | 93 | 94 | 95 | 96 | 97 | 96 |
| 92 | 93 | 94 | 95 | 96 | 97 | 98 | 97 |
| 91 | 🟦 | 🟦 | 94 | 95 | 🟦 | 99 | 98 |
| 90 | ☺ | 🟦 | 93 | 94 | 🟦 | 100 | 99 |
| 89 | 88 | 🟦 | 92 | 93 | 🟦 | 🟦 | 98 |
| 88 | 89 | 🟦 | 91 | 92 | 93 | 🟦 | 97 |
| 89 | 90 | 91 | 92 | 93 | 94 | 95 | 96 |
| 88 | 89 | 90 | 91 | 🟦 | 93 | 94 | 95 |

# Markov Decision Processes



# Markov Decision Process (MDP)

- Finite set of states S
- Finite set of actions A
- At each time step the agent observes state $s_t \in S$ and chooses action $a_t \in A(s_t)$
- Then receives immediate reward $r_{t+1}$
- And state changes to $s_{t+1}$
- Markov assumption : $s_{t+1}=\delta(s_t,a_t)$ and $r_{t+1}=r(s_t,a_t)$
  - Next reward and state only depend on *current* state $s_t$ and action $a_t$
  - Functions $\delta(s_t,a_t)$ and $r(s_t,a_t)$ may be non-deterministic
  - Functions $\delta(s_t,a_t)$ and $r(s_t,a_t)$ not necessarily known to agent

## What’s Different from HMMs?

- Controllable Markov chain (HMMs are not controllable)
- Equally, an FSM with probabilistic transitions and full state output
- Objective: Learn a policy (HMMs learn models)
- State fully observable (state in HMMs is only partially observable)

# Why MDPs?

- Natural model of physical, apparently random phenomenom
- MDP’s obey the principle of optimality (under suitable restrictions on the reward function)
  - The Bellman equation: (fixed policy)

$$\mathrm{Q}(\mathrm{s},\mathrm{a}) = \mathrm{r}(\mathrm{s}) + \mathrm{E}_{\mathrm{s},\mathrm{a}}[Q(s',a')]$$

  - The Bellman equation:

$$\mathrm{Q}^{*}(\mathrm{s}) = \mathrm{r}(\mathrm{s}) + \max_{\mathrm{a}*} \mathrm{E}_{\mathrm{s}}[Q^{*}(s',a*)]$$

- There is a (almost unique) optimal *deterministic* mapping from states to actions

## Learning Task

Execute actions in the environment, observe results and

- Learn a policy $\pi_t(s,a) : S \rightarrow A$ from states $s_t \in S$ to actions $a_t \in A$ that maximizes the expected reward : $E[r_t + \gamma\ r_{t+1} + \gamma^2\ r_{t+2} + \ldots]$ from any starting state $s_t$
- $0 < \gamma < 1$ is the discount factor for future rewards
- Target function is $\pi_t(s,a) : S \rightarrow A$
- But there are no direct training examples of the form <s,a>
- Training examples are of the form <<s,a>,r>

## State Value Function

Consider deterministic environments, namely $\delta(s,a)$ and $r(s,a)$ are deterministic functions of s and a.

For each policy $\pi(s,a) : S \rightarrow A$ the agent might adopt we define an evaluation function:

$$V^{\pi}(s)= r_t+\gamma\, r_{t+1}+ \gamma^2\, r_{t+2}+\ldots= \Sigma_{i=0}\, r_{t+i}\, \gamma^i$$

where $r_t$, $r_{t+1}$,… are generated by following the policy $\pi$ from start state s

Task: Learn the optimal policy $\pi^*$ that maximizes $V^{\pi}(s)$

$$\pi^* = \operatorname{argmax}_{\pi} V^{\pi}(s) ,\forall s$$

## Action Value Function

- State value function denotes the reward for starting in state s and following policy $\pi$.

$$V^{\pi}(s)= r_t+\gamma\ r_{t+1}+ \gamma^2\ r_{t+2}+\ldots= \Sigma_{i=0}\ r_{t+i}\ \gamma^i$$

- Action value function denotes the reward for starting in state s, taking action a and following policy $\pi$ afterwards.

$$Q^{\pi}(s,a)= r(s,a) + \gamma\ r_{t+1}+ \gamma^2\ r_{t+2}+\ldots= r(s,a) + \gamma\ V^{\pi}(\delta(s,a))$$

## Predicting the Value of a State

- Key idea: Learn a Value Function

  $Q: S \times A \rightarrow \Re$

  such that

  $$Q(s,a) = \sum \gamma^t r_t$$

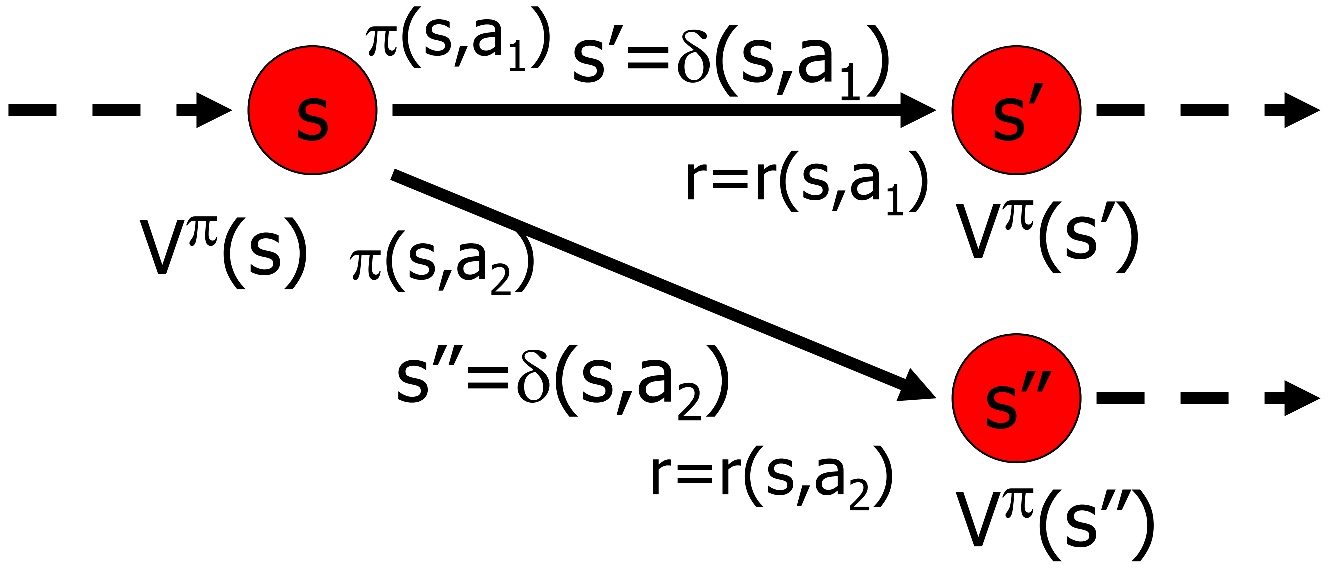
  A mapping that predicts what each state *s* is worth, given we take action *a* (Sum of future rewards.)

- Why: Duality between value functions and policies
  - By knowing the optimal policy, we can calculate the value of each state
  - By knowing the optimal value-function, we can calculate the optimal move

## Bellman Equation (Deterministic Case)

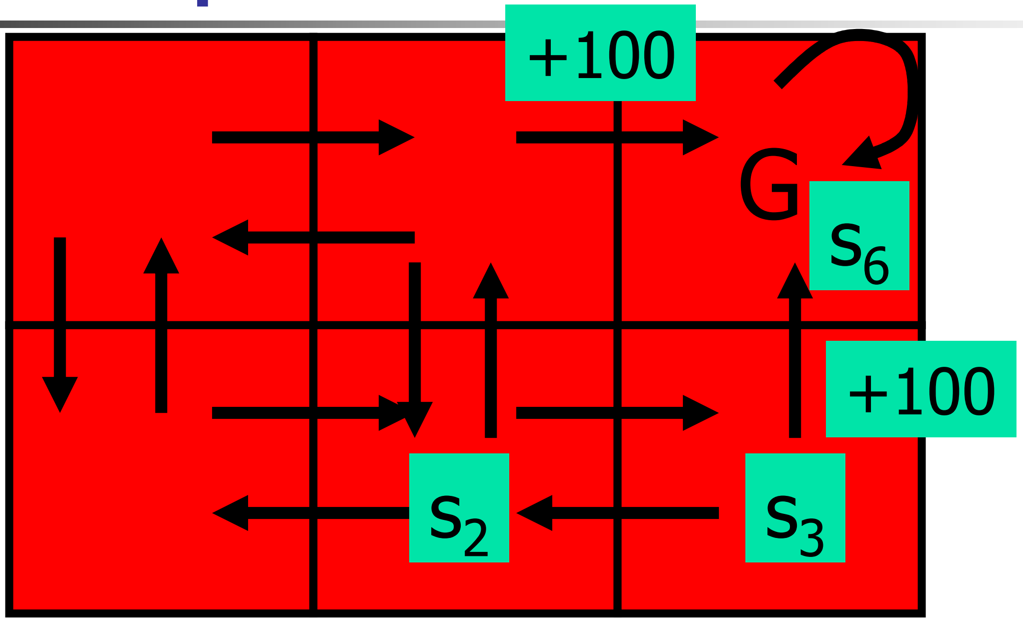
- $V^{\pi}(s) = r_t + \gamma\, r_{t+1} + \gamma^2\, r_{t+2} + \ldots$

  $= \Sigma_a\, \pi(s,a)\, (r(s,a) + + \gamma\, V^{\pi}(\delta(s,a)))$



Set of |s| linear equations, solve it directly or by policy evaluation.

## Example



G: terminal state, upon entering G agent obtains a reward of +100, remains in G forever and obtains no further rewards

Compute $V^{\pi}(s)$ for equil-probable policy $\pi(s,a)=1/|a|$

$V^{\pi}(s_3) = ½ \gamma V^{\pi}(s_2) + ½ (100 + \gamma V^{\pi}(s_6))$

# Iterative Policy Evaluation

- Instead of solving the Bellman equation directly one can use *iterative policy evaluation* by using the Bellman equation as an update rule.

$$V_{k+1}^{\pi}(s) = \Sigma_a \pi(s,a) (r(s,a) + \gamma V_k^{\pi}(\delta(s,a)))$$

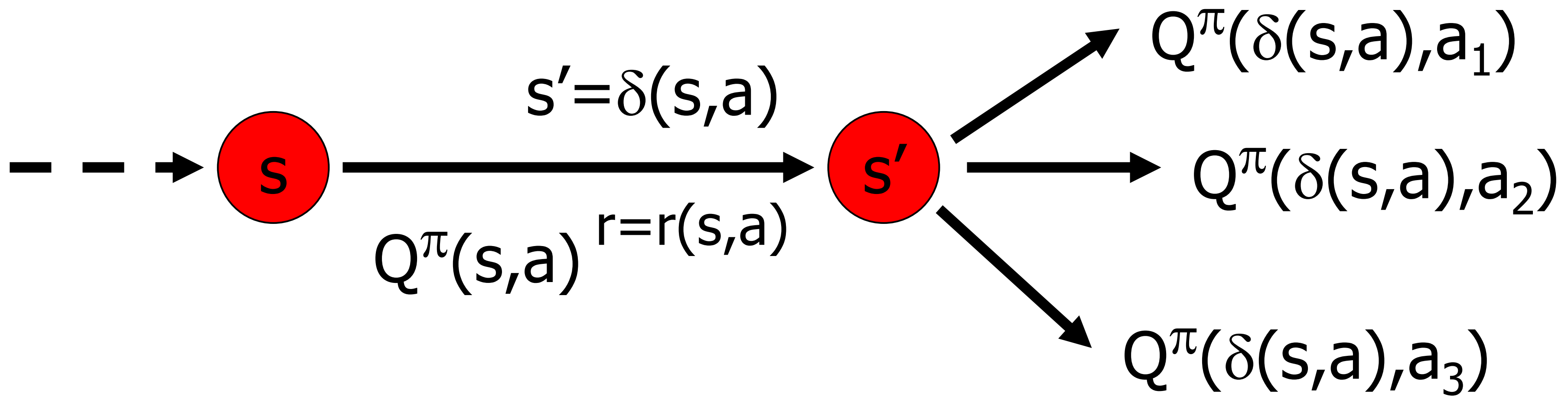
The sequence $V_k^{\pi}$ is guaranteed to converge to $V^{\pi}$

$\gamma=0.9$

| | | |
|---|---|---|
| $V^{\pi}_{50}=52$ | $V^{\pi}_{50}=66$ | $V^{\pi}_{50}=0$ |
| $V^{\pi}_{50}=49$ | $V^{\pi}_{50}=57$ | $V^{\pi}_{50}=76$ |

## Bellman Equation (Deterministic Case)

- $Q^{\pi}(s,a) = r(s,a) + \gamma \sum_{a'} \pi(\delta(s,a),a') \, Q^{\pi}(\delta(s,a),a')$

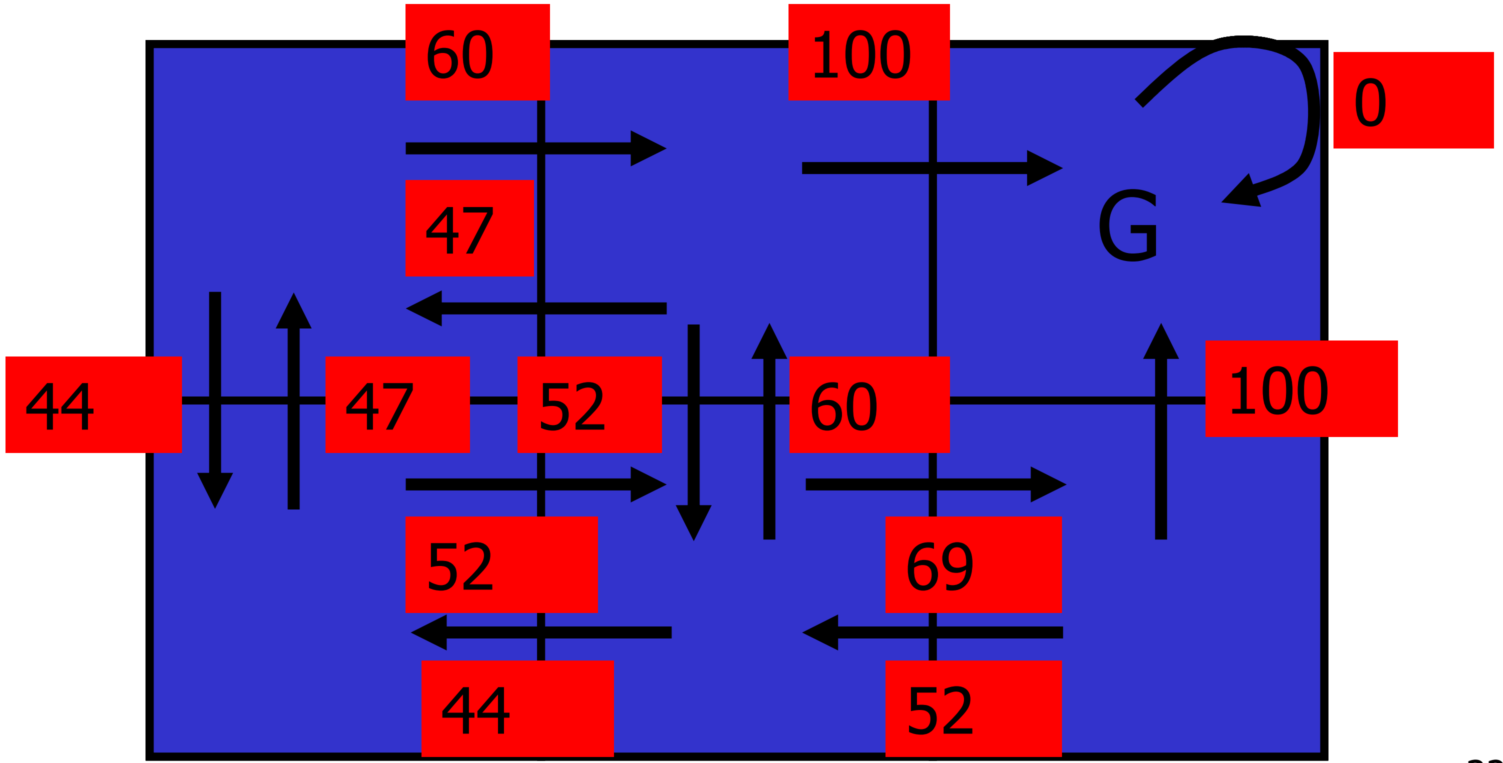


# Iterative Policy Evaluation

- Bellman equation as an update rule for action-value function:

$$Q_{k+1}^{\pi}(s,a) = r(s,a) + \gamma \sum_{a'} \pi(\delta(s,a),a')\, Q_k^{\pi}(\delta(s,a),a')$$

$\gamma=0.9$



# Optimal Value Functions

- $V^*(s) = \max_\pi V^\pi(s)$
- $Q^*(s,a) = \max_\pi Q^\pi(s,a)$

Bellman optimality equations

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

  $= \max_a ( r(s,a) + \gamma V^*(\delta(s,a)) )$
- $Q^*(s,a) = r(s,a) + V^*(\delta(s,a)))$

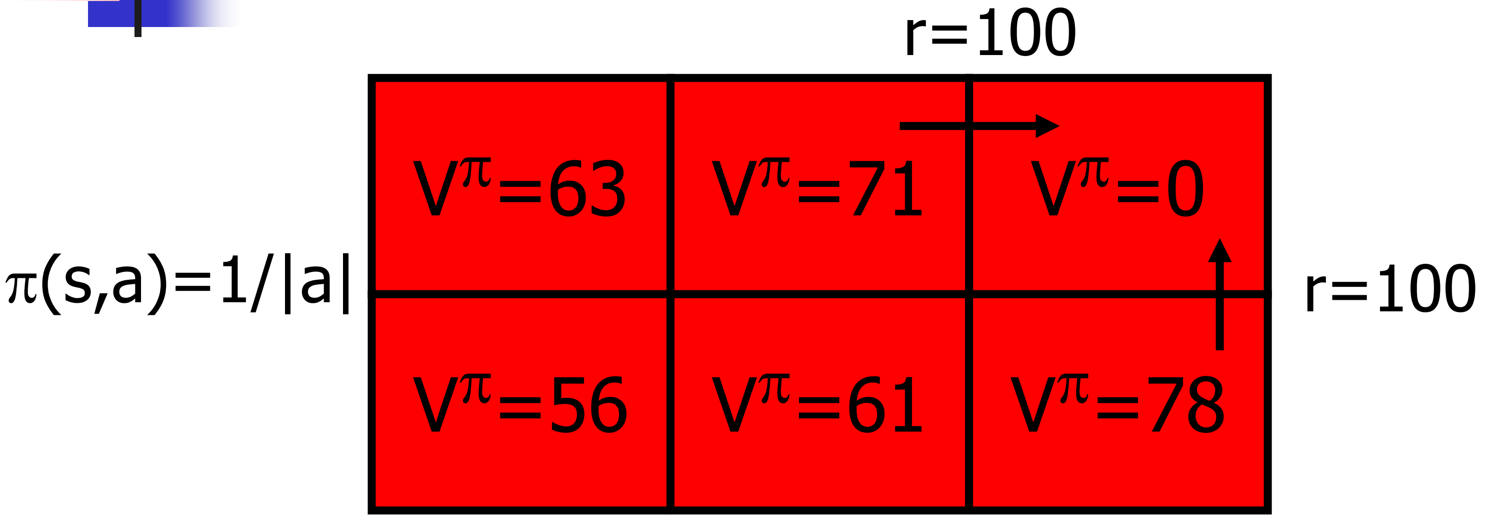
  $= r(s,a) + \gamma \max_{a'} Q^*(\delta(s,a),a')$

## Policy Improvement

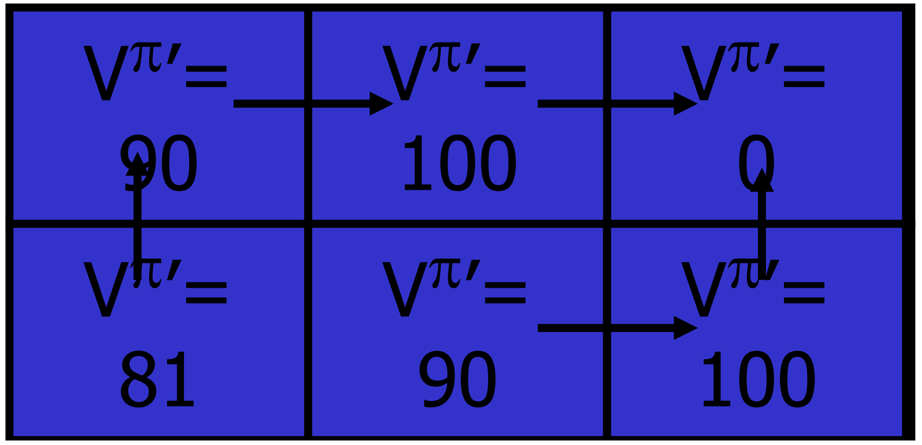
- Suppose we have determined the value function $V^\pi$ for an arbitrary deterministic policy $\pi$.
- For some state s we would like to know if it is better to choose an action $a \neq \pi(s)$.
- Select a and follow the existing policy $\pi$ afterwards gives us reward $Q^\pi(s,a)$
- If $Q^\pi(s,a) > V^\pi$ then a is obviously better than $\pi(s)$
- Therefore choose new policy $\pi'$ as

  $\pi'(s) = \text{argmax}_a\, Q^\pi(s,a) = \text{argmax}_a\, r(s,a) + \gamma\, V^\pi(\delta(s,a))$

# Example

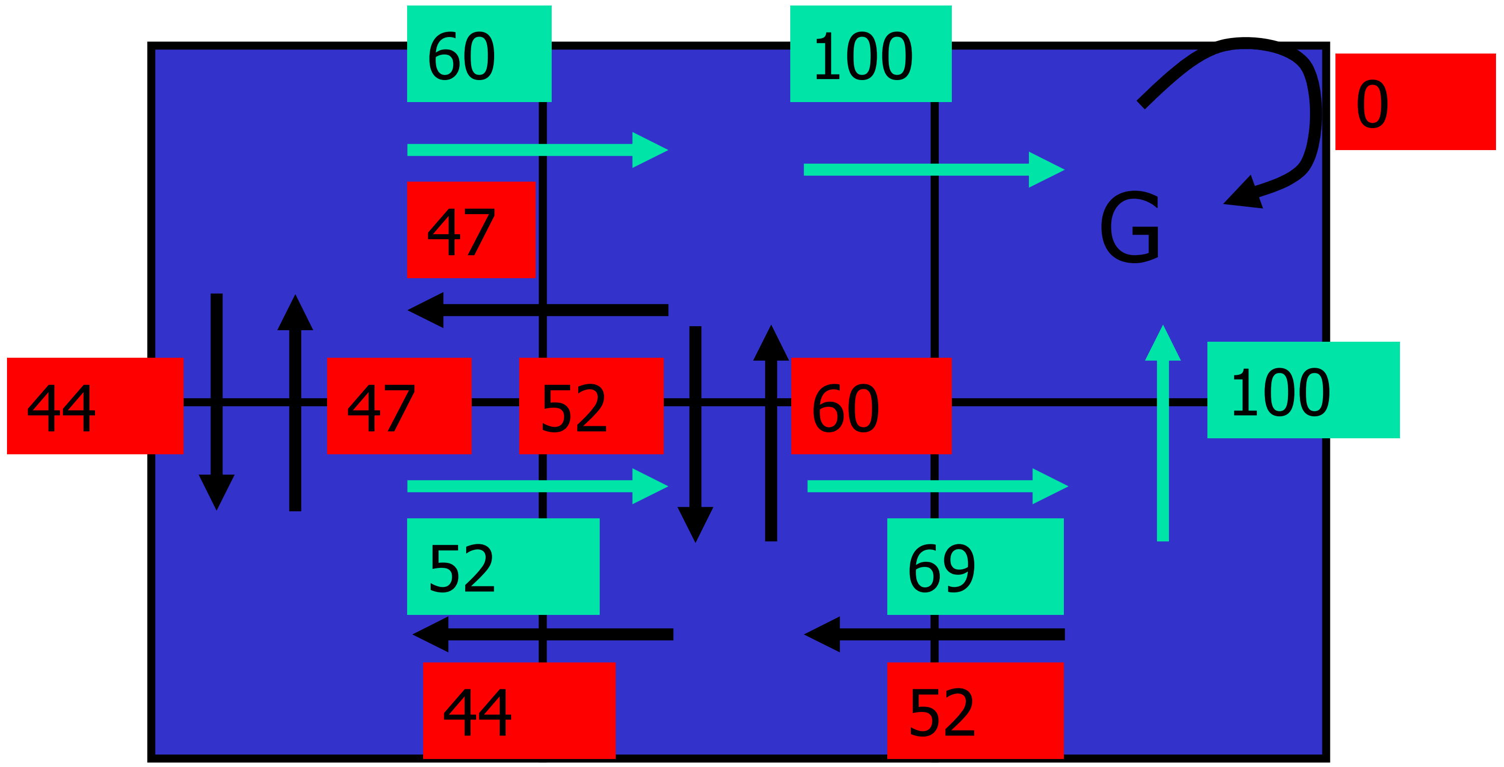


$$\pi'(s)=\text{argmax}_a\ r(s,a)+\gamma V^\pi(\delta(s,a))$$



## Example

$$\pi'(s) = \mathrm{argmax}_a \, Q^{\pi}(s,a)$$



## Generalized Policy Iteration

- Intertwine *policy evaluation* with *policy improvement*

$$\pi_0 \xrightarrow{E} V^{\pi 0} \xrightarrow{I} \pi_1 \xrightarrow{E} V^{\pi 1} \xrightarrow{I} \pi_2 \xrightarrow{E} V^{\pi 2} \xrightarrow{I} \cdots \xrightarrow{I} \pi^* \xrightarrow{E} V^{\pi^*}$$

$$\begin{tikzcd}
\pi \arrow[r, bend left=50, "\text{evaluation}", "V \to V^{\pi}"'] & V \arrow[l, bend left=50, "\text{improvement}", "\pi \to \text{greedy}(V)"']
\end{tikzcd}$$

## Value Iteration (Q-Learning)

- Idea: do not wait for policy evaluation to converge, but improve policy after each iteration.

$$V_{k+1}^{\pi}(s) = \max_a (r(s,a) + \gamma V_k^{\pi}(\delta(s,a)))$$

or

$$Q_{k+1}^{\pi}(s,a) = r(s,a) + \gamma \max_{a'} Q_k^{\pi}(\delta(s,a),a')$$

Stop when $\forall_s |V_{k+1}^{\pi}(s) - V_k^{\pi}(s)| < \varepsilon$

or $\forall_{s,a} |Q_{k+1}^{\pi}(s,a) - Q_k^{\pi}(s,a)| < \varepsilon$

## Non-Deterministic Case

- State transition function $\delta(s,a)$ no longer deterministic but probabilistic given by

$$P(s'|s,a) = Pr\{s_{t+1}=s'|s_t=s, a_t=a\}$$

*Transition probability* that given a current state s and action a the next state is s'.

- Reward function r(s,a) no longer deterministic but probabilistic given by

$$R(s',s,a) = E\{r_{t+1}|s_t=s, a_t=a, s_{t+1}=s'\}$$

- $P(s'|s,a)$ and $R(s',s,a)$ completely specify MDP.

# Bellman Equation (Non-Deterministic Case)

- $Q^{\pi}(s,a) = \Sigma_{s'} P(s'|s,a) [R(s',s,a) + \gamma V^{\pi}(s')]$
- $V^{\pi}(s) = \Sigma_a \pi(s,a) \Sigma_{s'} P(s'|s,a) [R(s',s,a) + \gamma V^{\pi}(s')]$
- $Q^{\pi}(s,a) = \Sigma_{s'} P(s'|s,a) [R(s',s,a) + \gamma \Sigma_{a'} \pi(s',a') Q^{\pi}(s',a')]$

Bellman optimality equations:

- $V^{*}(s) = \max_a \Sigma_{s'} P(s'|s,a) [R(s',s,a) + \gamma V^{*}(s')]$
- $Q^{*}(s,a) = \Sigma_{s'} P(s'|s,a) [R(s',s,a) + \gamma \max_{a'} Q^{*}(s',a')]$

## Value Iteration (Q-Learning)

$V_{k+1}^{\pi}(s) = \max_a \Sigma_{s'} P(s'|s,a) [R(s',s,a) + \gamma V_k^{\pi} (s')]$

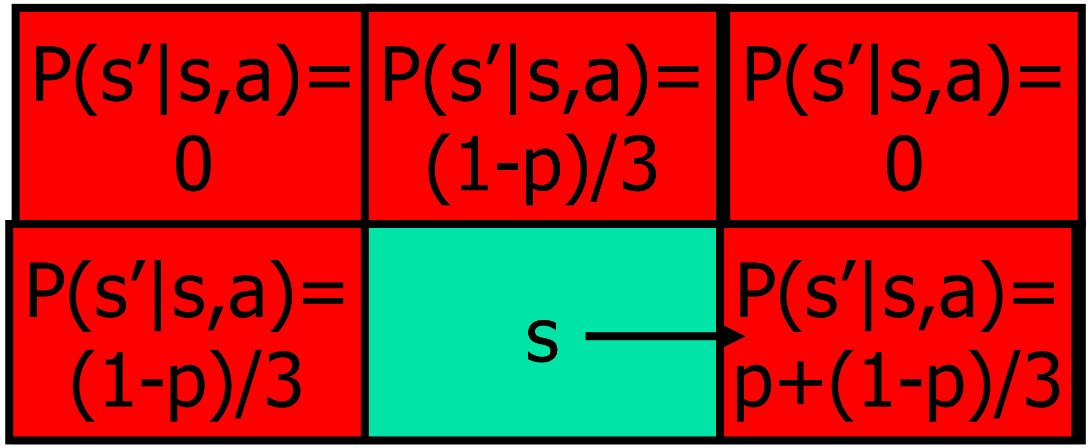
or

$Q_{k+1}^{\pi}(s,a) = \Sigma_{s'} P(s'|s,a) [R(s',s,a)+ \gamma \max_{a'} Q_k^{\pi}(s',a')]$

Stop when $\forall_s |V_{k+1}^{\pi}(s)- V_k^{\pi}(s)| < \varepsilon$

or $\forall_{s,a} |Q_{k+1}^{\pi}(s,a)- Q_k^{\pi}(s,a)| < \varepsilon$

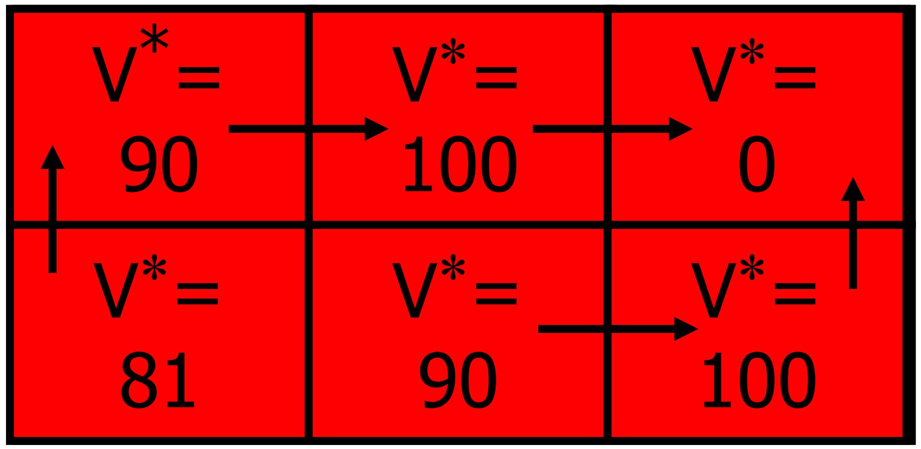
## Example



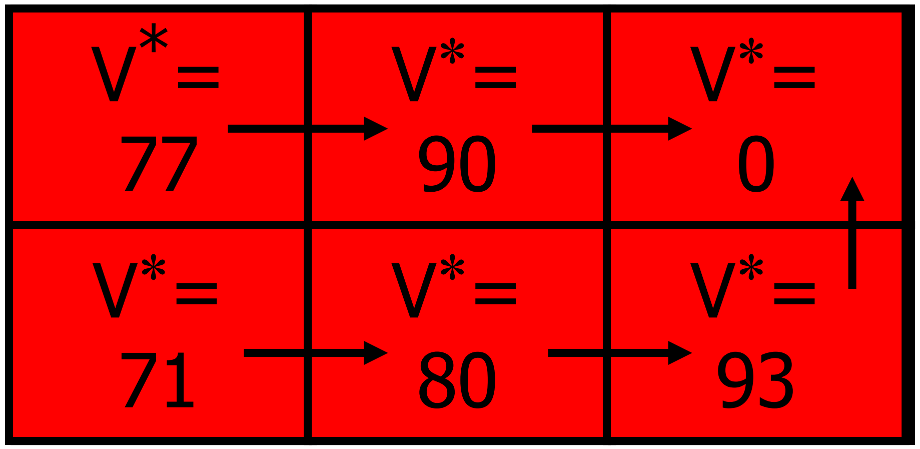
- Now assume that actions a are non-deterministic, with probability p agent moves to the correct square indicated by a, with probability (1-p) agent moves to a random neighboring square.

## Example

Deterministic optimal value function



Non-deterministic optimal value function p=0.5



## Reinforcement Learning

- What if the transition probabilities P(s'|s,a) and reward function R(s',s,a) are unknown?
- Can we still learn V(s), Q(s,a) and identify and optimal policy $\pi$(s,a)?
- The answer is yes. Consider the observed rewards $r_t$ and state transitions $s_{t+1}$as training samples drawn from the true underlying probability functions R(s',s,a) and P(s'|s,a).
- Use approximate state V(s) and action value Q(s,a) functions

## Model Free Methods

- Models of the environment:
- T: $S \times A \rightarrow \Pi(S)$ and $R : S \times A \rightarrow R$

- Do we know them? Do we have to know them?

- Monte Carlo Methods
- Adaptive Heuristic Critic
- Q Learning

# Monte Carlo Methods

- *Idea*:
- Hold statistics about rewards for each state
- Take the average
- This is the V(s)

- Based only on experience ☺
- Assumes episodic tasks ☹
- (Experience is divided into episodes and all episodes will terminate regardless of the actions selected.)

- Incremental in episode-by-episode sense not step-by-step sense.

Problem:  Unvisited <s, a> pairs
(problem of ***maintaining exploration***)

For every <s, a> make sure that:
P(<s, a> selected as a start state and action) >0

(Assumption of ***exploring starts*** )

## Monte Carlo Method

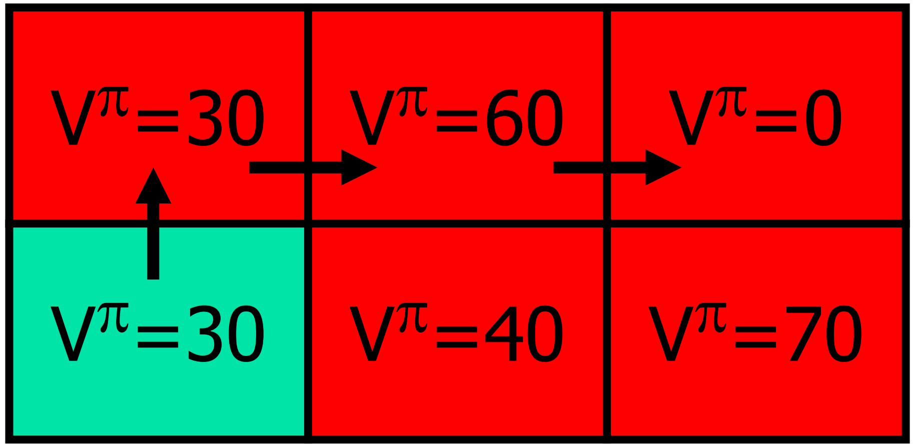
- Initialize:
  - $\pi \leftarrow$ policy to be evaluated
  - $V(s) \leftarrow$ an arbitrary state-value function
  - $Return(s) \leftarrow$ an empty list, for all $s \in S$
- Repeat forever
  - Generate an episode using $\pi$
  - For each state s appearing in the episode:
    - $R \leftarrow$ return following the first occurence of s
    - Append R to Returns(s)
    - $V(s) \leftarrow$ average(Returns(s))

## Monte Carlo Method

- $V(s_t) \leftarrow V(s_t) + \alpha\,[R_t - V(s_t)]$

where $R_t = r_{t+1} + \gamma\, r_{t+2} + \gamma^2\, r_{t+3} + \ldots$

is the observed reward after time t and $\alpha$ is a constant step-size parameter

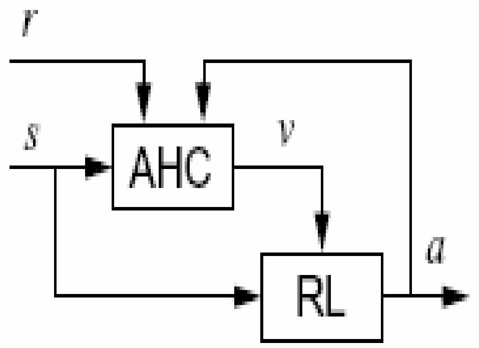


$V(s_t) \leftarrow 30 + 0.1\,[0 + 0.9*0 + 0.9^2 * 100 - 30] = 35.1$

## ADAPTIVE HEURISTIC CRITIC & TD(λ)

How the AHC learns, TD(0) algorithm:

$$V(s) := V(s) + \alpha(r + \gamma V(s') - V(s))$$



Architecture for the adaptive heuristic critic.

AHC : TD Algorithm

# Q LEARNING

Q values in Value Iteration:

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

But we don't know $R(s,a)$ and $T(s,a,s')$

Instead use the following :

$$Q(s,a) := Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

Decayed ***α*** properly? Q values will converge. (Singh 1994)

## Q-LEARNING CRITICS:

- Simpler than AHC learning
- Q-Learning is exploration sensitive
- Analog to value iteration in MDP
- Most popular Model free learning algorithm

## Temporal Difference Learning

- Monte-Carlo:
  - $V(s_t) \leftarrow V(s_t) + \alpha\, [R_t - V(s_t)]$
  - target for V(s) : $E\,\{R_t \mid s_t = s\}$

Must wait until the end of the episode to update V

- Temporal Difference (TD):
  - $V(s_t) \leftarrow V(s_t) + \alpha\, [r_{t+1} + \gamma\, V(s_{t+1}) - V(s_t)]$
  - target for V(s) : $E\,\{r_{t+1} + \gamma\, V(s_{t+1}) \mid s_t = s\}$

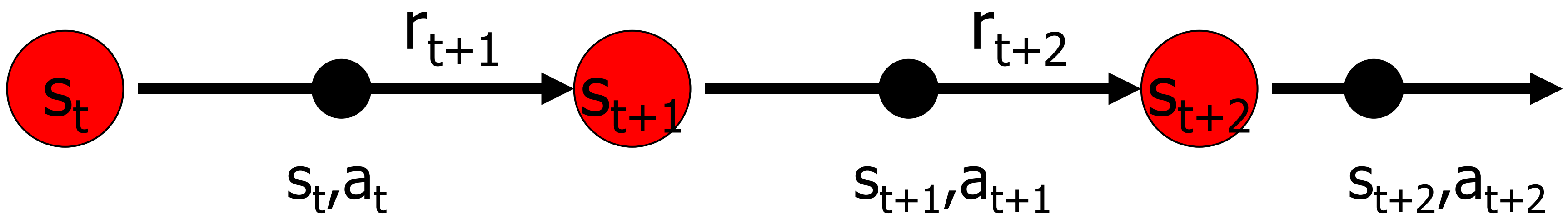
TD method is *bootstrapping* by using the existing estimate of the next state $V(s_{t+1})$ for updating $V(s_t)$

## TD(0) : Policy Evaluation

```
Initialize:
  ▪ $\pi \leftarrow$ policy to be evaluated
  ▪ $V(s) \leftarrow$ an arbitrary state-value function
Repeat for each episode
  ▪ Initialize s
  ▪ Repeat for each step of episode
      ▪ $a \leftarrow$ action given by $\pi$ for s
      ▪ Take action a, observe reward r, and next state s'
      ▪ $V(s) \leftarrow V(s_t) + \alpha\, [r + \gamma\, V(s') - V(s)]$
      ▪ $s \leftarrow s'$
  ▪ Until s is terminal
```

# TD(0): Policy Iteration

- $Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha\,[r_{t+1} + \gamma\, Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)]$



The update rule uses a quintuple of events $(s_t,a_t,r_{t+1},s_{t+1},a_{t+1})$, therefore called SARSA.

Problem: Unlike in the deterministic case we can not choose

A completely greedy policy $\pi(s)=\max_a Q(s,a)$, as due to the unknown transition and reward functions $\delta(s,a)$ and $r(s,a)$ we can not be sure if another action might eventually turn out to be better.

## $\varepsilon$-greedy policy

- Soft policy: $\pi(s,a) > 0$ for all $s \in S$, $a \in A(s)$

Non-zero probability off choosing every possible action

- $\varepsilon$-greedy policy: Most of the time with probability $(1-\varepsilon)$ follow the optimal policy

  $\pi(s) = \max_a Q(s,a)$

  but with probability e pick a random action:

  $\pi(s,a) \geq \varepsilon/|A(s)|$

- Let $\varepsilon \rightarrow 0$ go to zero as $t \rightarrow \infty$ for example $\varepsilon = 1/t$ so that $\varepsilon$-greedy policy converges to the optimal deterministic policy

## SARSA Policy Iteration

```
Initialize Q(s,a) arbitrarily:
Repeat for each episode
  ▪ Initialize s
  ▪ Choose a from using $\varepsilon$-greedy policy derived from Q
  ▪ Repeat for each step of episode
      ▪ Take action a, observe reward r, and next state s′
      ▪ $Q(s,a) \leftarrow Q(s,a) + \alpha\, [r + \gamma\, Q(s',a') - Q(s,a)]$
      ▪ $s \leftarrow s'$, $a \leftarrow a'$
  ▪ Until s is terminal
```

## Q-Learning (Off-Policy TD)

- Approximates the optimal value functions V*(s) or Q*(s,a) independent of the policy being followed.
- The policy determines which state-action pairs are visited and updated

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha \, [r_{t+1} + \gamma \max_{a'} Q(s_{t+1},a') - Q(s_t,a_t)]$$

## Q-Learning Off-Policy Iteration

```
Initialize Q(s,a) arbitrarily:
Repeat for each episode
  ▪ Initialize s
  ▪ Choose a from s using $\varepsilon$-greedy policy derived from Q
  ▪ Repeat for each step of episode
      ▪ Take action a, observe reward r, and next state s'
      ▪ $Q(s,a) \leftarrow Q(s,a) + \alpha\ [r + \gamma\ \max_{a'} Q(s',a') - Q(s,a)]$
      ▪ $s \leftarrow s'$
  ▪ Until s is terminal
```

## TD versus Monte-Carlo

So far TD uses one-step look-ahead but why not use 2-steps or n-steps look-ahead to update Q(s,a).

- 2-step look-ahead

$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha\, [r_{t+1} + \gamma\, r_{t+2} + \gamma^2\, Q(s_{t+2},a_{t+2}) - Q(s_t,a_t)]$

- N-step look-ahead

$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha\, [r_{t+1} + \gamma\, r_{t+2} + \ldots + \gamma^{n-1}\, r_{t+n} + \gamma^n\, Q(s_{t+n},a_{t+n}) - Q(s_t,a_t)]$

- Monte-Carlo method

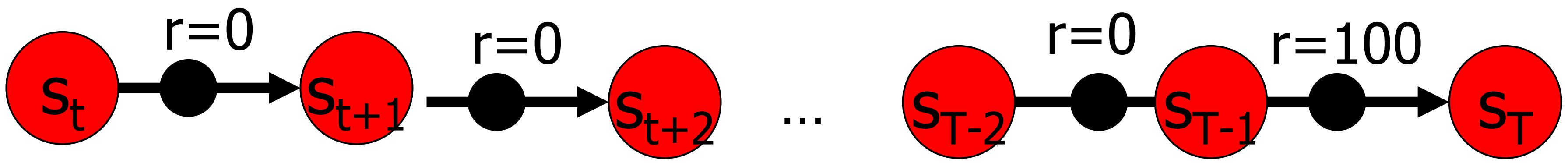
$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha\, [r_{t+1} + \gamma\, r_{t+2} + \ldots + \gamma^{N-1}\, r_{t+N} - Q(s_t,a_t)]$

(compute total reward $R^T$ at the end of the episode)

## Temporal Difference Learning

Drawback of one-step look-ahead:

- Reward is only propagated back to the successor state (takes long time to finally propagate to the start state)



Initial V:

V(s)=0 V(s)=0 V(s)=0 V(s)=0 V(s)=0

After first epsiode:

V(s)=0 V(s)=0 V(s)=0 V(s)=0 V(s)=$\alpha$*100

After second epsiode:

V(s)=0 V(s)=0 V(s)=0 V(s)=$\alpha * \gamma$*100 V(s)=$\alpha$*100+..

## Monte-Carlo Method

Drawback of Monte-Carlo method

- Learning only takes place after an episode terminated
- Performs a random walk until goal state is discovered for the first time as all state-action values seem equal
- It might take long time to find the goal state by random walk
- TD-learning actively explores state space if each action receives a small default penalty

## N-Step Return

Idea: blend TD learning with Monte-Carlo method

Define:

$R_t^{(1)} = r_{t+1} + \gamma V_t(s_{t+1})$

$R_t^{(2)} = r_{t+1} + \gamma r_{t+2} + \gamma^2 V_t(s_{t+2})$

…

$R_t^{(n)} = r_{t+1} + \gamma r_{t+2} + \ldots + \gamma^{n-1} r_{t+n} + \gamma^n V_t(s_{t+n})$

The quantity $R_t^{(n)}$ is called the n-step return at time t.

## TD($\lambda$)

- n-step backup : $V_t(s_t) \leftarrow V_t(s_t) + \alpha\, [R_t^{(n)} - V_t(s_t)]$
- TD($\lambda$) : use average of n-step returns for backup

$R_t^{\lambda} = (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} R_t^{(n)}$

$R_t^{\lambda} = (1-\lambda) \sum_{n=1}^{T-t-1} \lambda^{n-1} R_t^{(n)} + \lambda^{T-t-1} R_t$

(if $s_T$ is a terminal state)

The weight of the n-step return decreases with a factor of $\lambda$

TD(0): one-step temporal difference method

TD(1) : Monte-Carlo method

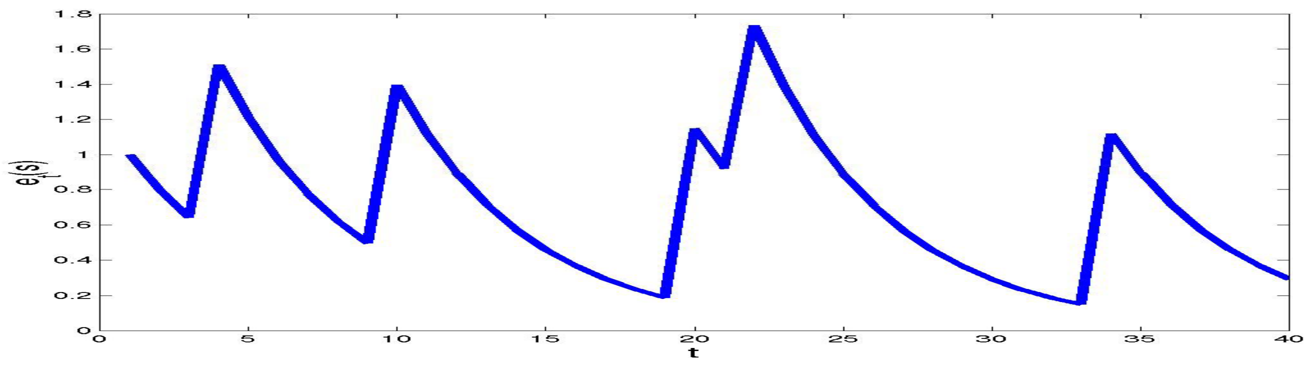
# Eligibility Traces

Practical implementation of TD($\lambda$):

With each state-action pair associate an eligibility trace $e_t(s,a)$

On each step, the eligibility trace for all state-action pairs decays by a factor $\gamma\lambda$ and the eligibility trace for the one state and action visited on the step is incremented by 1.

$$e_t(s,a) = \begin{cases} \gamma \lambda e_{t-1}(s,a) + 1 & \text{if } s=s_t \text{ and } a=a_t \\ \gamma \lambda e_{t-1}(s,a) & \text{otherwise} \end{cases}$$



## On-line TD($\lambda$)

```
Initialize $Q(s,a)$ arbitrarily and $e(s,a)=0$ for all s,a:
Repeat for each episode
  Initialize s,a
  Repeat for each step of episode
    Take action a, observe r, s'
    Choose a' from s' using policy derived from Q ($\varepsilon$-greedy)
    $\delta \leftarrow r + \gamma Q(s',a') - Q(s,a)$
    $e(s,a) \leftarrow e(s,a) + 1$
    For all s,a:
    $Q(s,a) \leftarrow Q(s,a) + \alpha \delta e(s,a)$
    $e(s,a) \leftarrow \gamma \lambda e(s,a)$
    $s \leftarrow s'$, $a \leftarrow a'$
  Until s is terminal
```

## Function Approximation

- So far we assumed that the action value function Q(s,a) is represented as a table.
- Limited to problems with a small number of states and actions
- For large state spaces a table based representation requires large memory and large data sets to fill them accurately
- Generalization: Use any supervised learning algorithm to estimate Q(s,a) from a limited set of action value pairs
  - Neural Networks (Neuro-Dynamic Programming)
  - Linear Regression
  - Nearest Neighbors

## Function Approximation

- Minimize the mean-squared error between training examples $Q_t(s_{t,},a_t)$ and the true value function $Q^\pi(s,a)$

$$\Sigma_{s \in S} P(s) \, [Q^\pi(s,a) - Q_t(s_{t,},a_t)]^2$$

- Notice that during policy iteration P(s) and $Q^\pi(s,a)$ change over time
- Parameterize $Q^\pi(s,a)$ by a vector $\theta=(\theta_1,\ldots, \theta_n)$ for example weights in a feed-forward neural network

## Stochastic Gradient Descent

- Use gradient descent to adjust the parameter vector $\theta$ in the direction that reduces the error for the current example

$$\theta_{t+1} = \theta_t + \alpha \, [Q^{\pi}(s_t,a_t) - Q_t(s_t,a_t)] \nabla_{\theta t} \, Q_t(s_t,a_t)$$

- The target output $q_t$ of the t-th training example is not the true value of QP but some approximation of it.

$$\theta_{t+1} = \theta_t + \alpha \, [v_t - Q_t(s_t,a_t)] \nabla_{\theta t} \, Q_t(s_t,a_t)$$

- Still converges to a local optimum if $E\{v_t\}=Q^{\pi}(s_t,a_t)$ if $a \to 0$ for $t \to \infty$