# CSCI 561
# Foundation for Artificial Intelligence

## 07 - Reinforcement Learning
## 08 – Learn to Search and Play Games

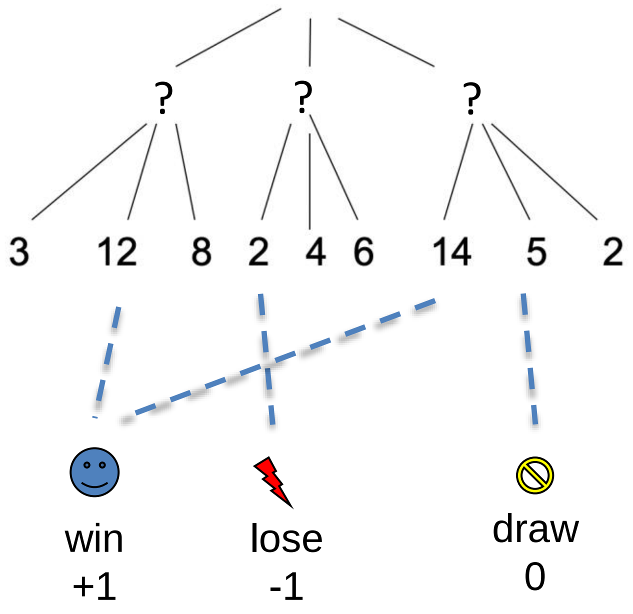
Professor Wei-Min Shen
University of Southern California

# Outline

- Motivation
  - Agent and Environment (Search & Games)
- States, actions, utility, rewards, policy
- Utility value iteration
- Policy Iterations
- Reinforcement Learning
  - Model-based
  - Model-free
- Q-Learning
  - State space for advanced game playing

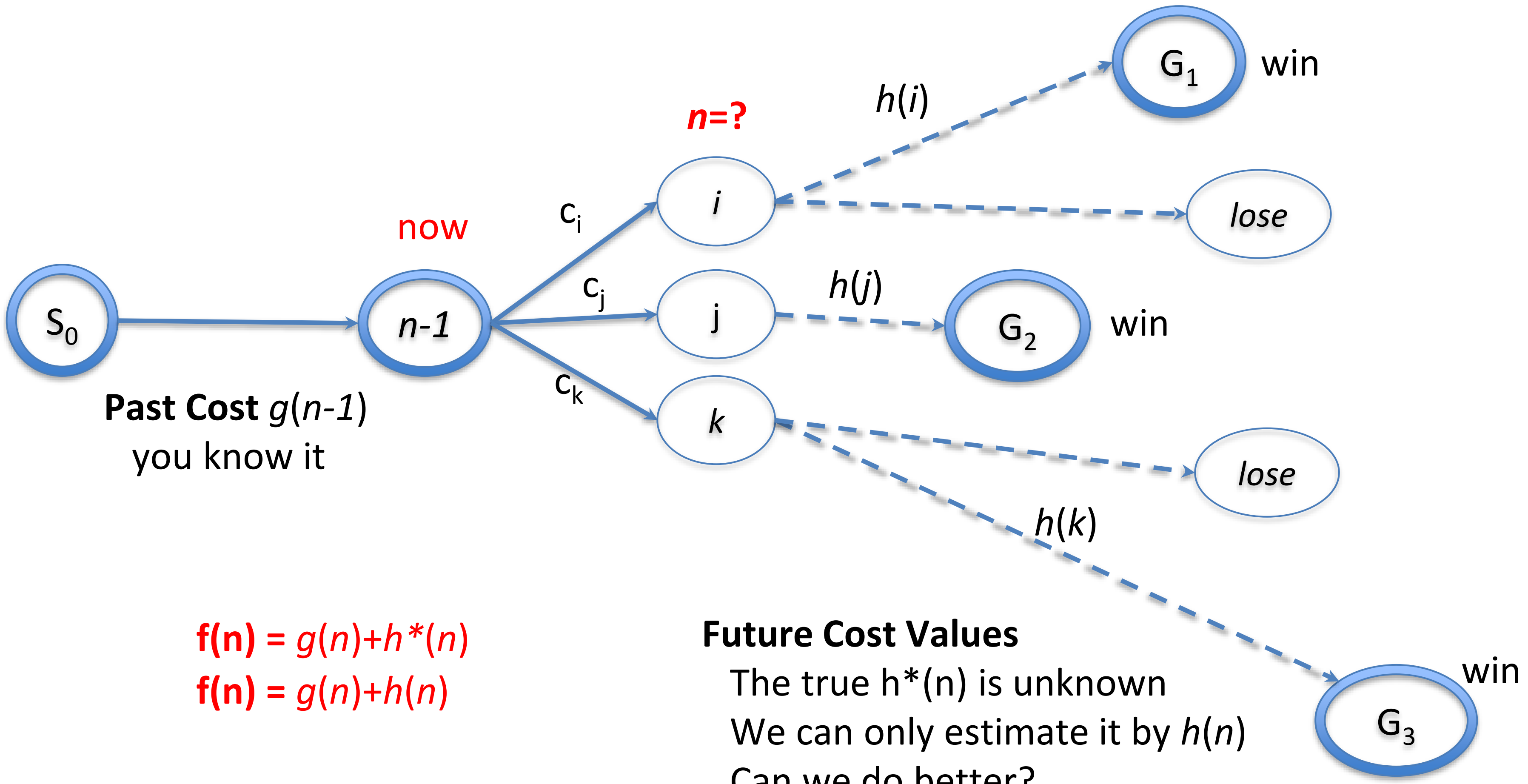
## A Key Question

- In all search and game playing problems, what is the key information that you wish to have for finding the optimal solution?
  - What you wish to have but may not always have?



# The True Future Values

(Only if we would know them !)

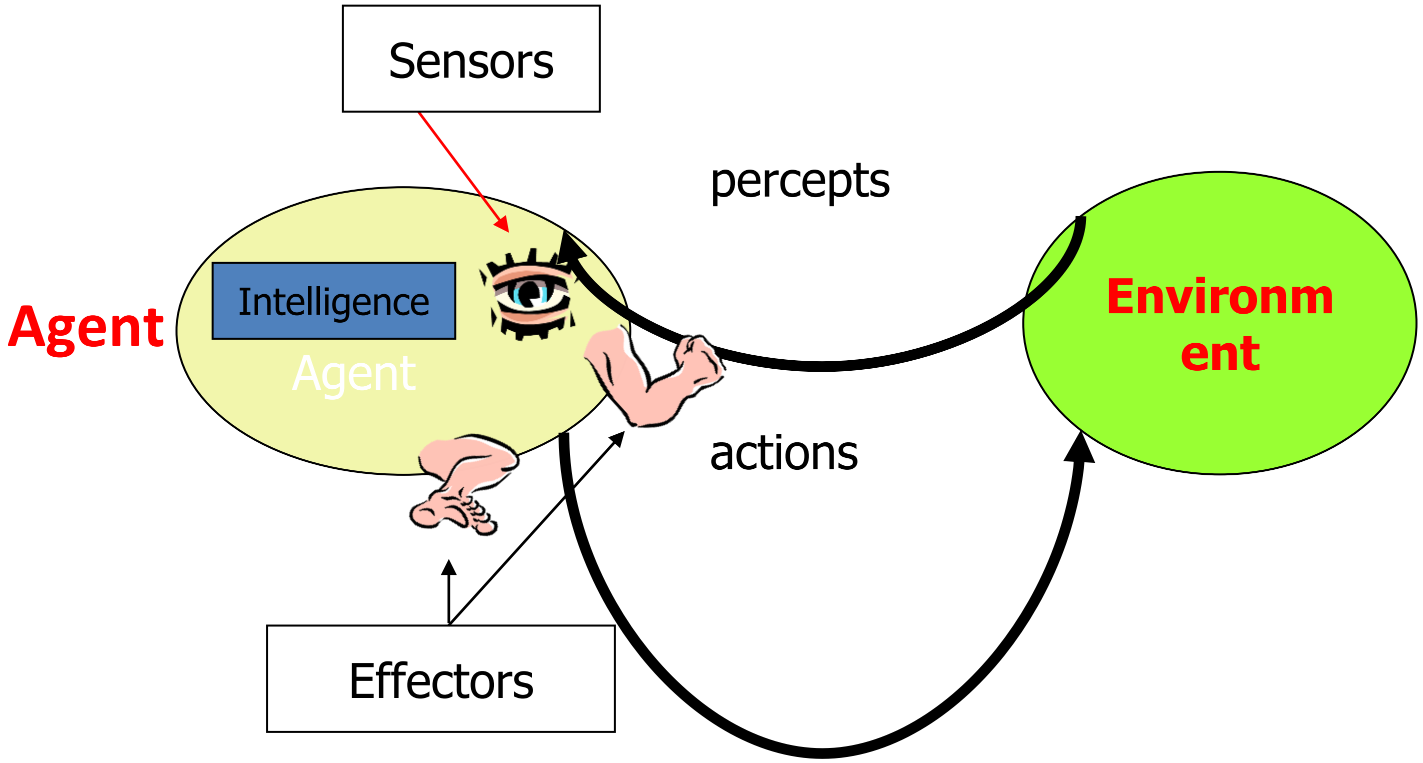


**Past Cost** $g(n\text{-}1)$
you know it

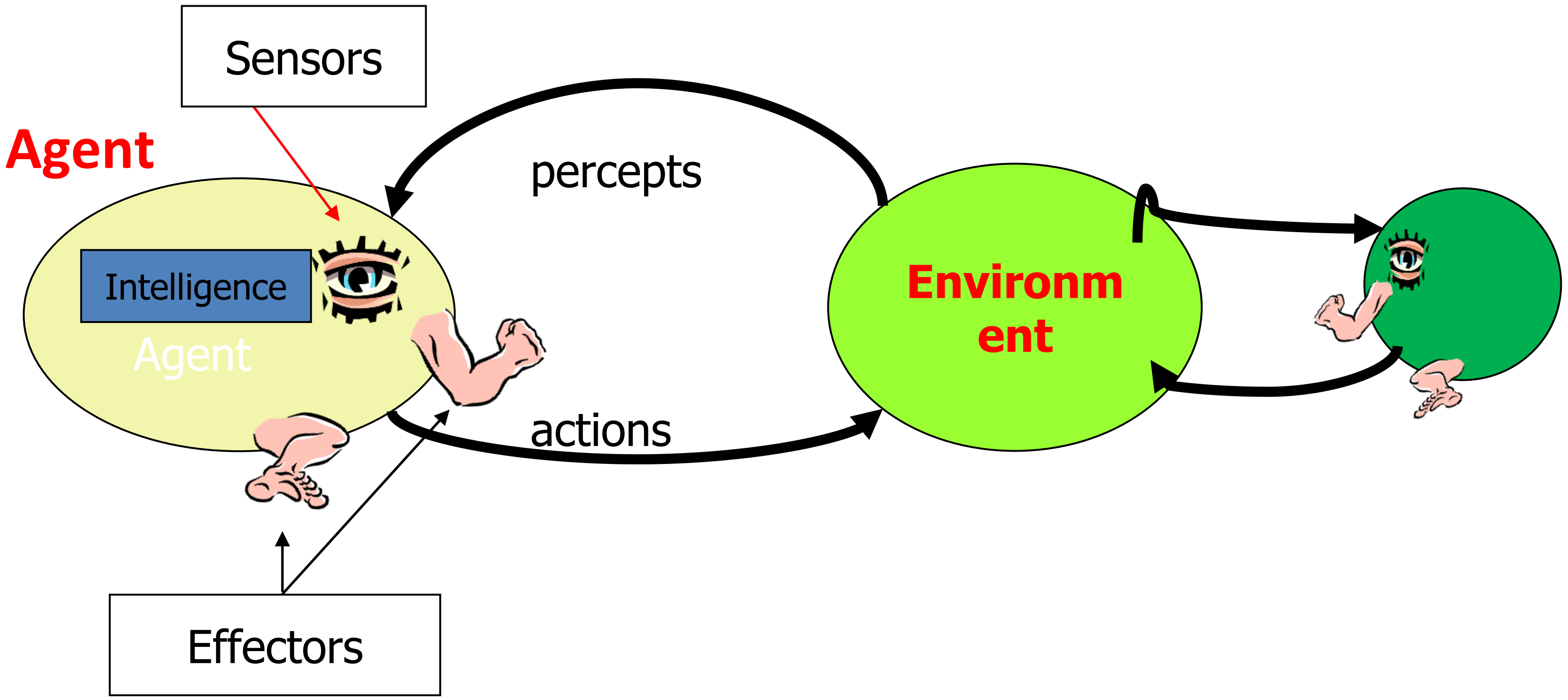
**f(n) =** $g(n)+h^*(n)$
**f(n) =** $g(n)+h(n)$

**Future Cost Values**
The true h*(n) is unknown
We can only estimate it by $h(n)$
Can we do better?

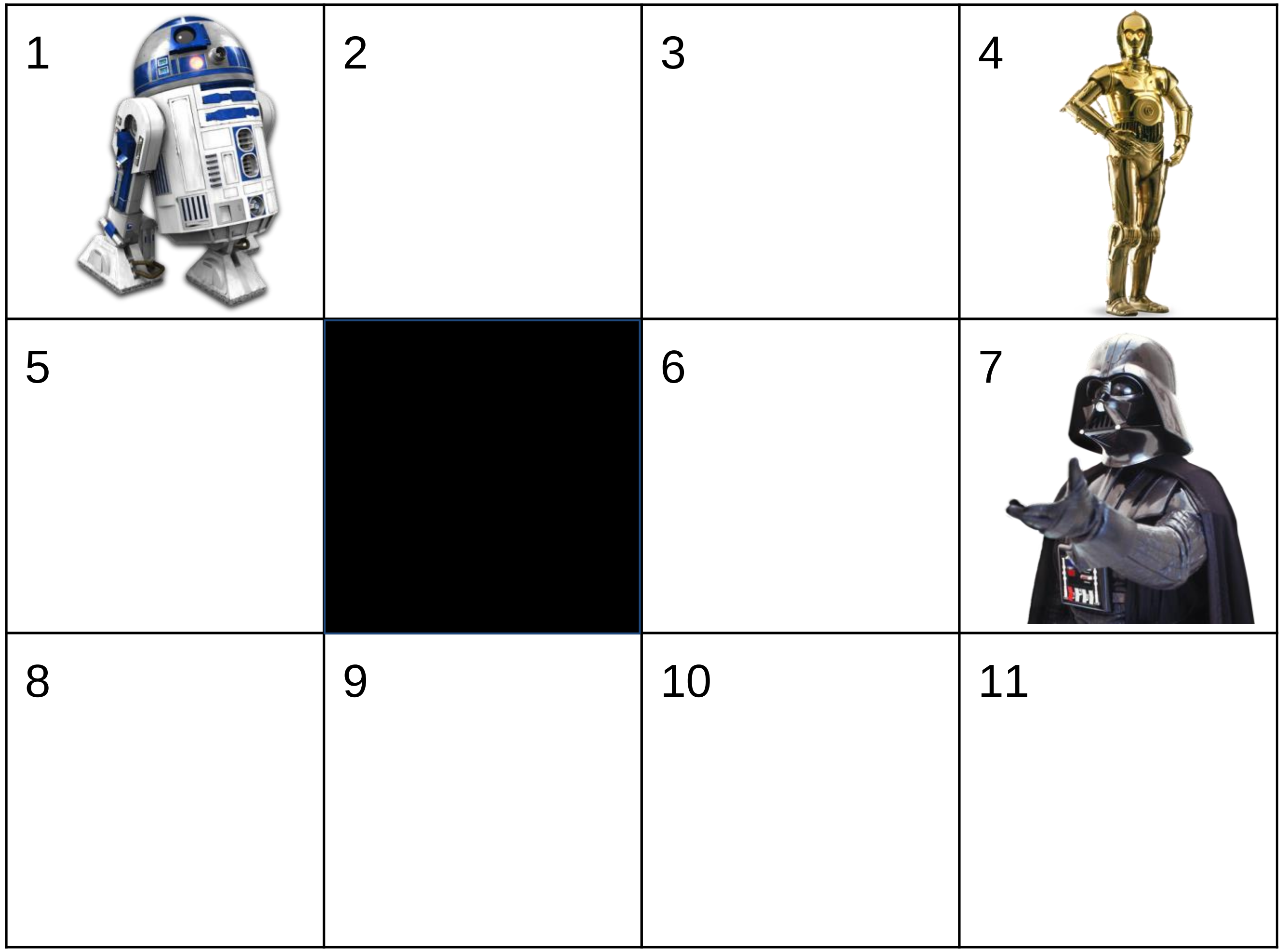
# Agent and Environment



# Agent, Environment, Game



# Agent and Environment (Star War)



Agents:

- R2D2
- 3PIO
- Darth Vader

States: 1,..,11

Actions: ^,v,>,<

Percepts: …

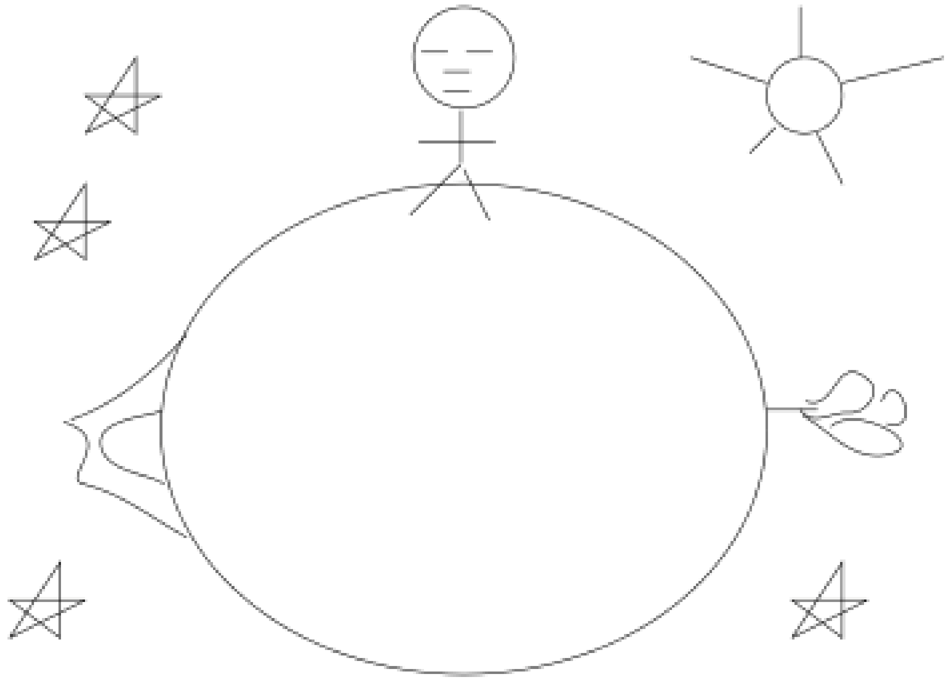
Goals:…

Rewards: …

State Utility Values: …
(your “crystal ball”)

# Agent and Environment

## The Little Prince's Planet



- Percepts: {rose, volcano, nothing}
- Actions: {forward, backward, turn-around}
- States: {north, south, east, west}

# The Little Prince Planet Environment

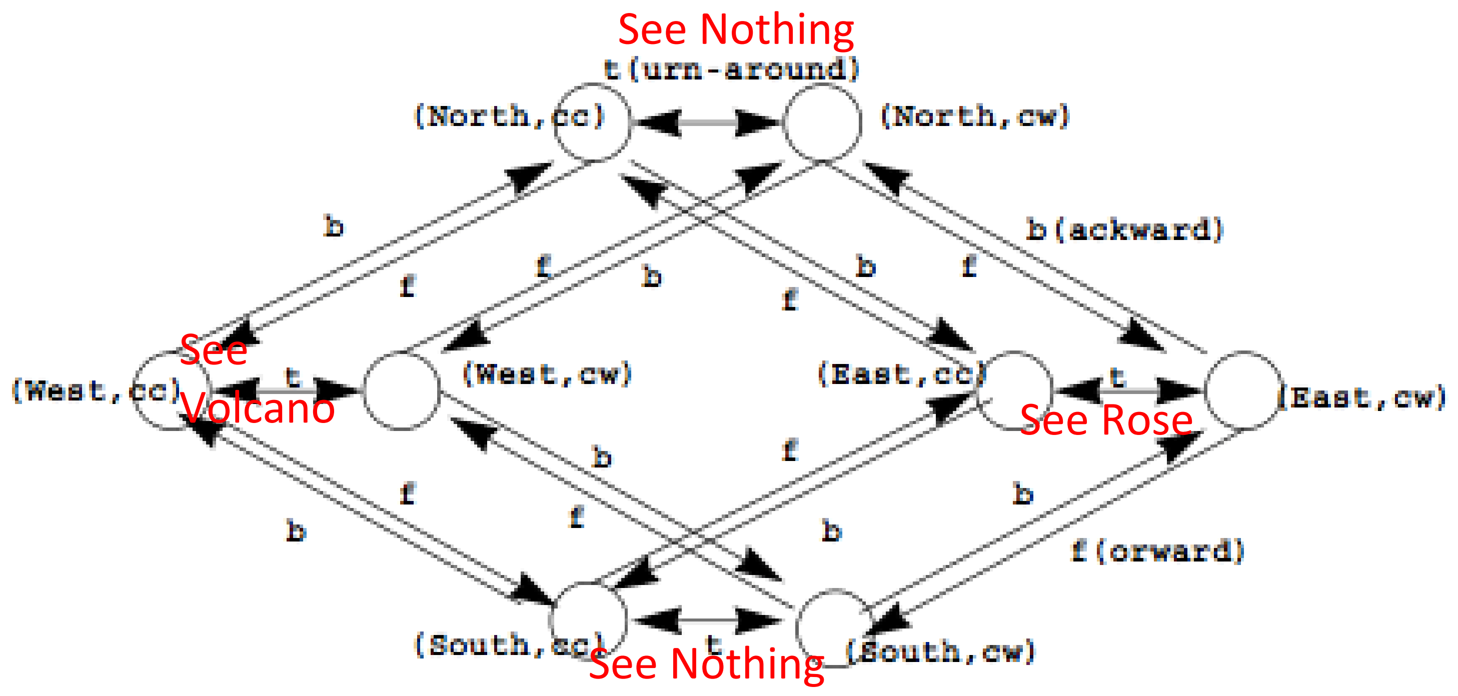


Figure 2.5: The little prince's actual environment.

# A Model for Little Prince's Planet

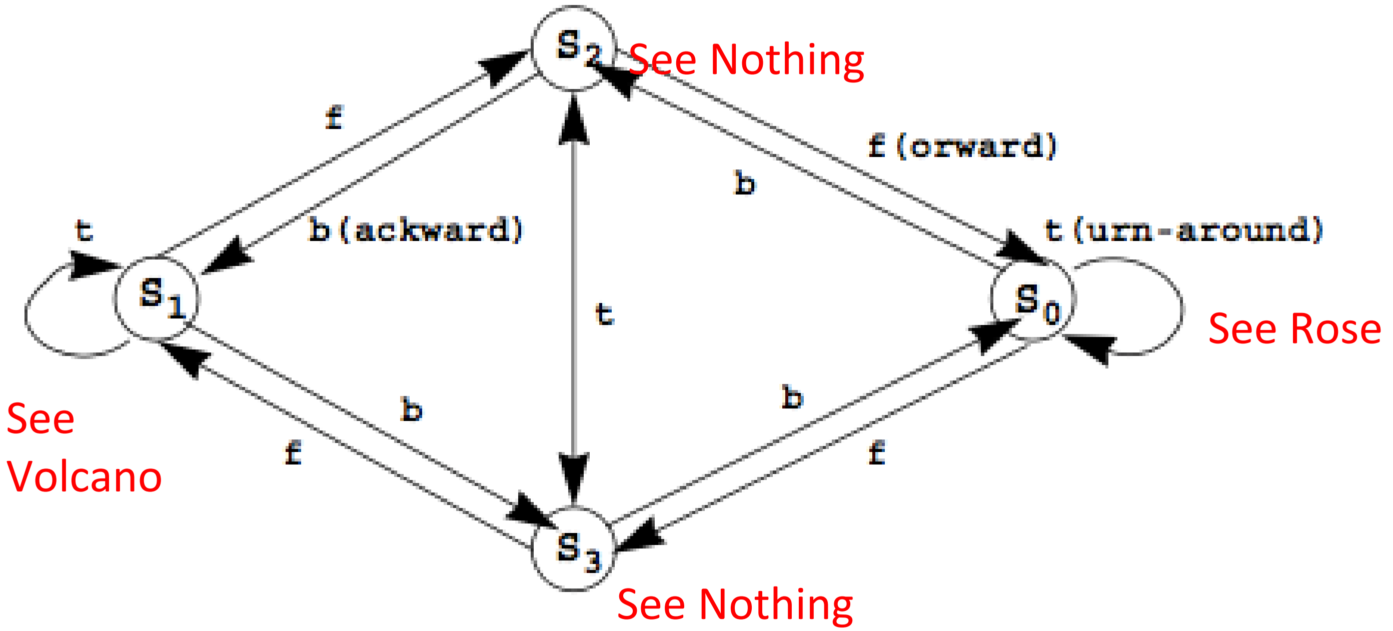
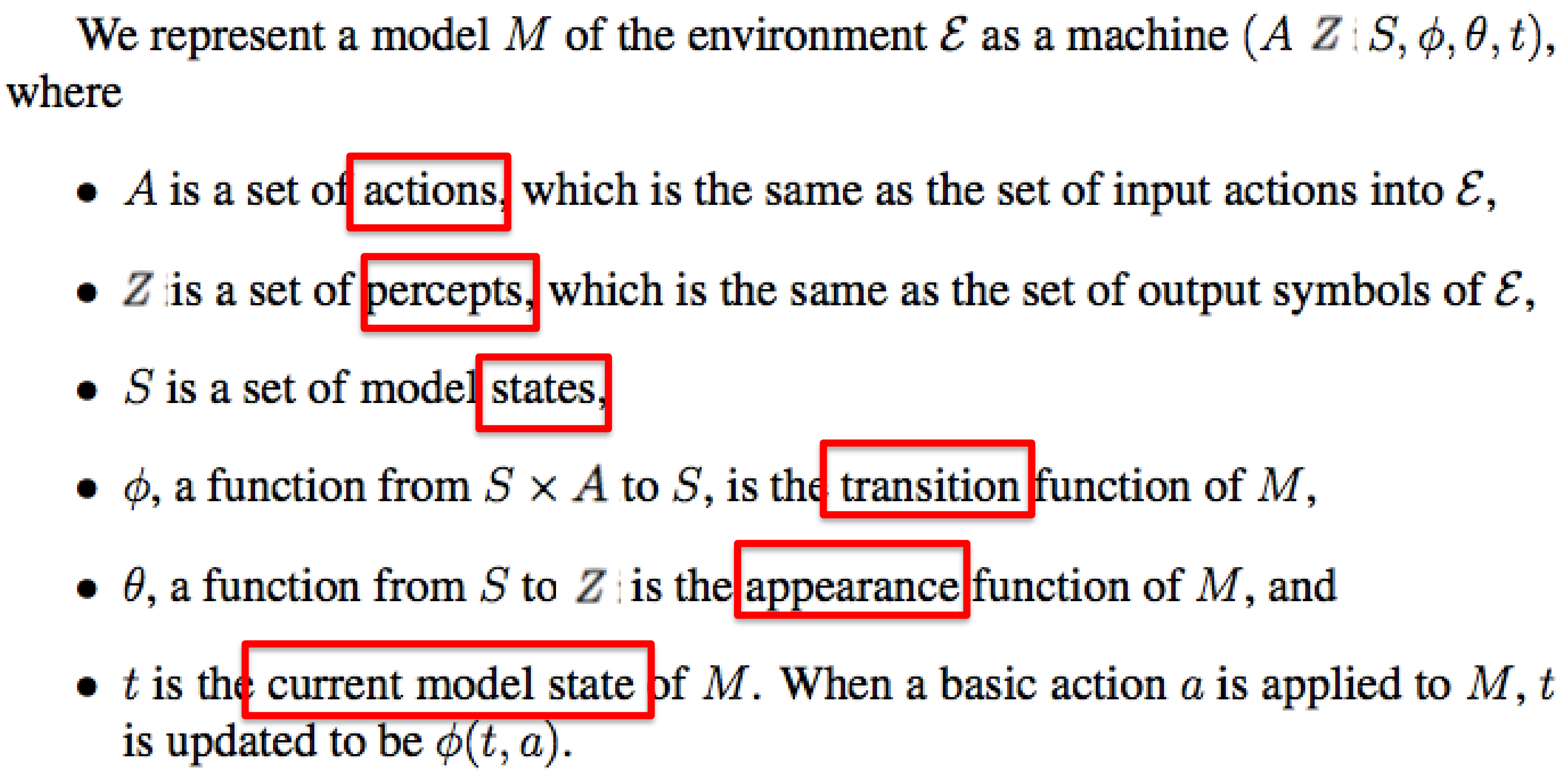


Figure 2.3: The little prince's model of his world.

His internal action model only needs 4 states! (why?)

# State, Action and Sensor Models



We represent a model $M$ of the environment $\mathcal{E}$ as a machine $(A\ Z\ S, \phi, \theta, t)$, where

- $A$ is a set of actions, which is the same as the set of input actions into $\mathcal{E}$,
- $Z$ is a set of percepts, which is the same as the set of output symbols of $\mathcal{E}$,
- $S$ is a set of model states,
- $\phi$, a function from $S \times A$ to $S$, is the transition function of $M$,
- $\theta$, a function from $S$ to $Z$ is the appearance function of $M$, and
- $t$ is the current model state of $M$. When a basic action $a$ is applied to $M$, $t$ is updated to be $\phi(t, a)$.

# Little Prince's Model

$$\begin{aligned}
A &\equiv \{\text{forward, backward, turn-around}\}\\
P &\equiv \{\text{rose, volcano, nothing}\}\\
\widehat{Z} &\equiv \{s_1, s_2, s_3, s_4\}\\
\phi &\equiv \phi(s_0, \text{forward}) = s_3, \phi(s_0, \text{backward}) = s_2, \ldots\\
\theta &\equiv \theta(s_1) = \{\text{volcano}\}, \theta(s_0) = \{\text{rose}\}, \theta(s_2) = \theta(s_3) = \{\}
\end{aligned}$$

Transition Model $\varphi$ can be deterministic or probabilistic
Sensor Model $\vartheta$ can be deterministic or probabilistic

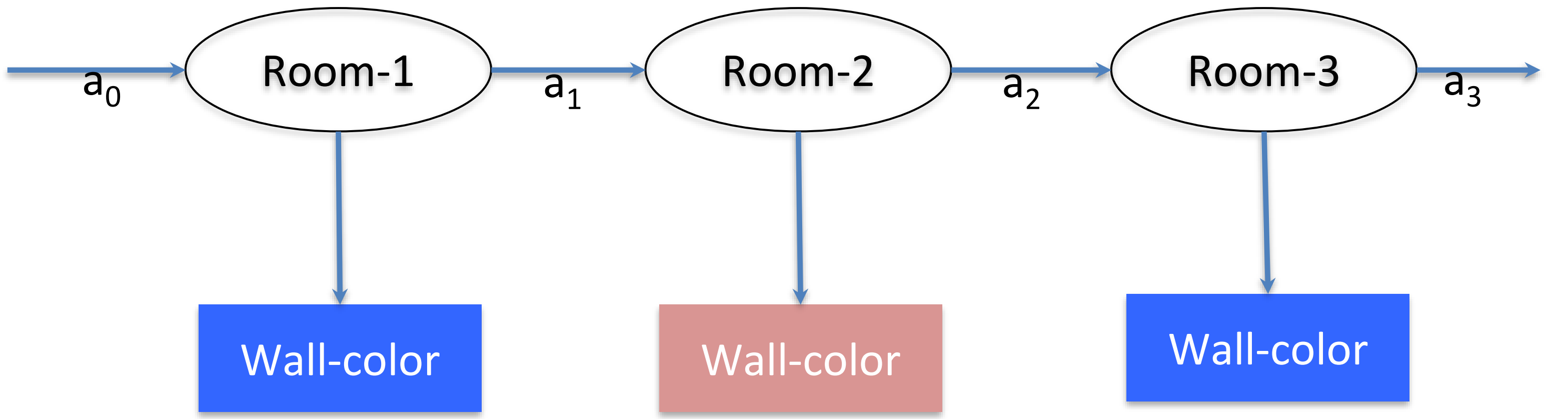
Are the transition model $\varphi$ here deterministic?
Are the sensor model $\vartheta$ deterministic?
Is there any hidden states in this example?

# The Environment may be <u>Uncertain</u>

- “Seeing may not always be believing”
  - States and observations are not always 1-1 mapping
    - One state may be seen in many different ways
    - The same observation may be seen in many different states
    - E.g., When the Prince sees nothing, where is he? Facing which way?
    - Can you make the little prince’s sensors non-deterministically?
- “Action may not always produce the same result”
  - Actions do not always take you to the same states
    - E.g., the Little Prince’s planet may have “planet-quakes” ☺
    - E.g., step forward on ice may not always succeed
  - Transitions between states by actions may be nondeterministic
    - E.g., Star War’s transitions are not deterministic
    - Can you make the Little Prince’s actions non-deterministically?

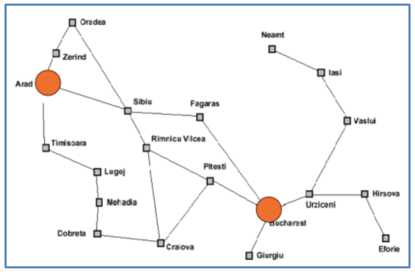
## Sensors can be uncertain (in real world)

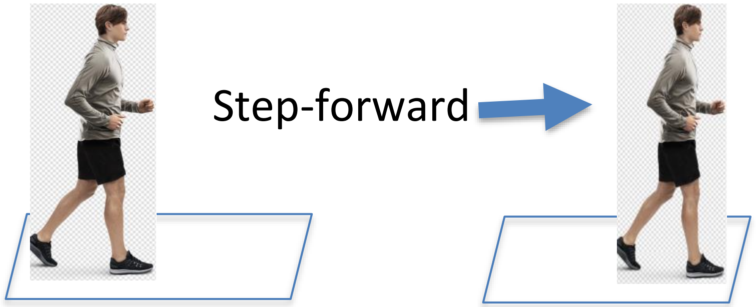
- Speech Recognition
  - “Listening is not always equal to hearing” ☺
- Traveling through rooms with colored walls
  - “Seeing does not always tell where you are” ☺

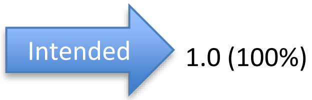


# Actions can be uncertain (in real world)

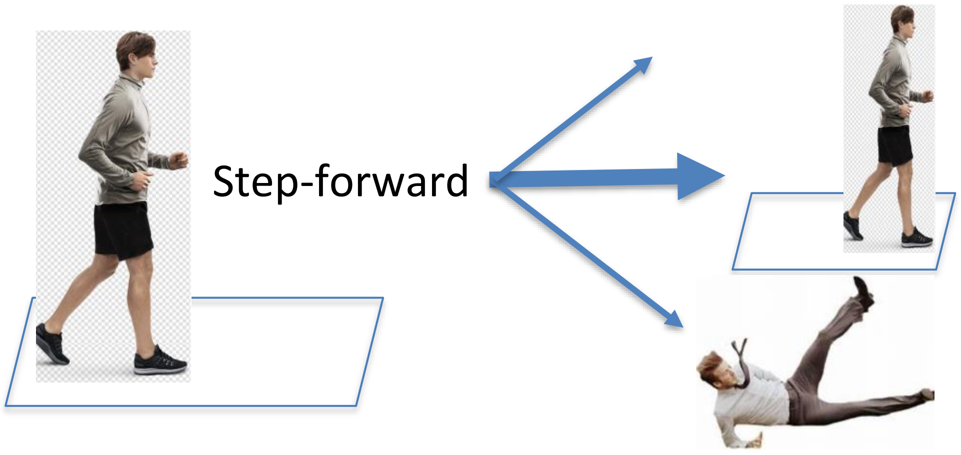
- Deterministic actions

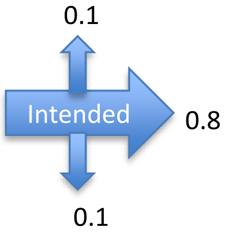






- Non-deterministic actions (e.g., walking on ice)





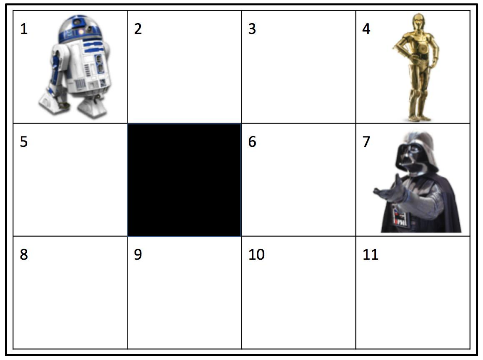
## Other Uncertainties in the Real World

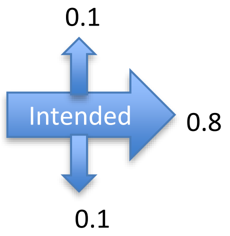
- Multiple Agents
  - The actions of other agents in the environment may be ”uncertain” from your point of view !
- Advanced Game Playing
  - Your opponent’s moves are definitely “uncertain” from your point of view !
- When you driving on a highway
  - What are uncertain?

# Star War's Probabilistic Transition Model

- Transitions can be probabilistic
  - $P(\text{state}_{t+1} \mid \text{state}_t, \text{action}_t)$

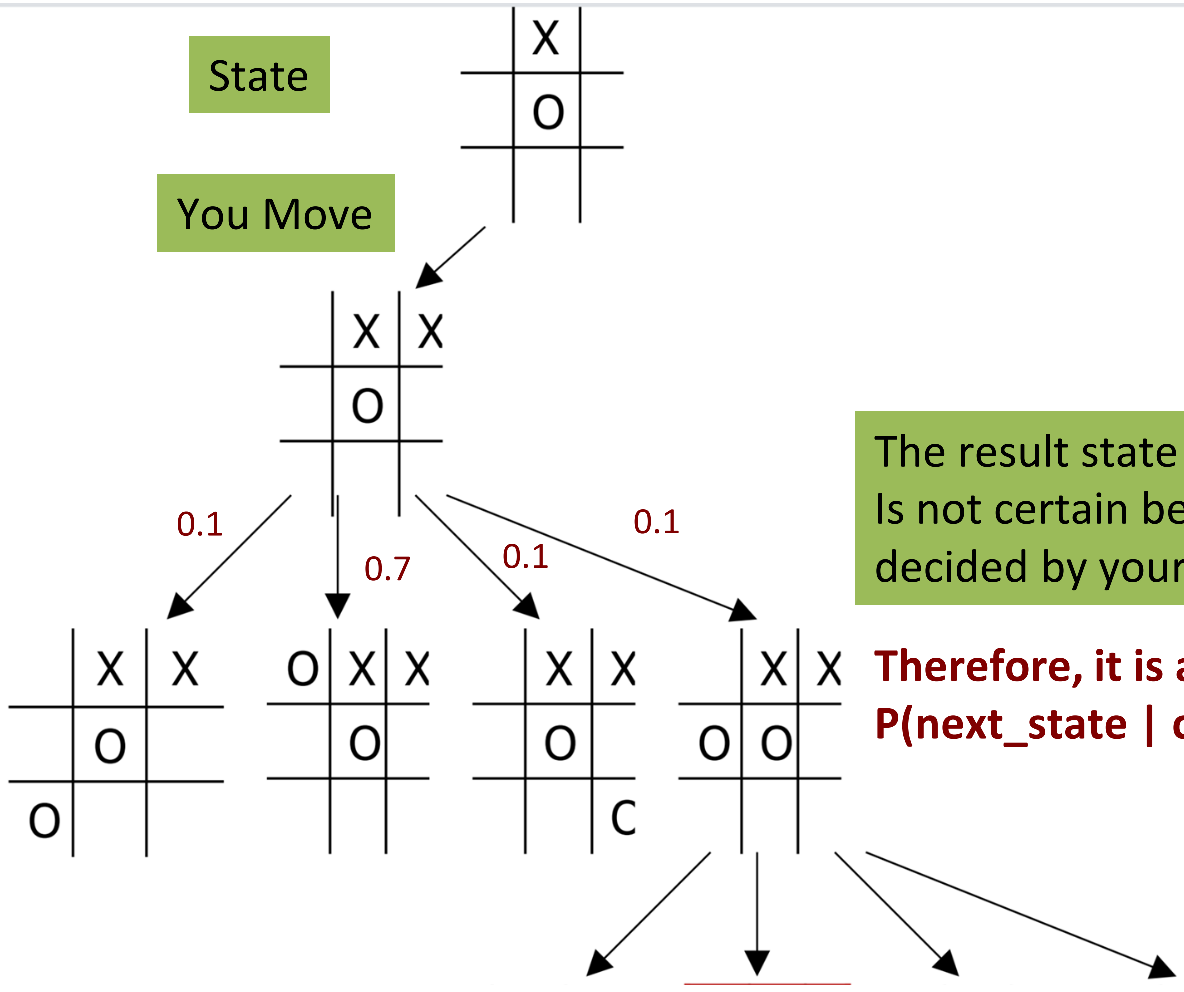
| $X_{t\text{-}1}$ | $A_{t\text{-}1}$ | $P(X_t{=}1)$ | $P(X_t{=}2)$ | $P(X_t{=}3)$ | $P(X_t{=}4)$ | $P(X_t{=}5)$ | ... |
|---|---|---|---|---|---|---|---|
| 1 | up | 0.8+0.1 | 0.1 | 0.0 | 0.0 | 0.0 | |
| 1 | down | 0.1 | 0.1 | 0.0 | 0.0 | 0.8 | |
| 1 | left | 0.8+0.1 | 0.0 | 0.0 | 0.0 | 0.1 | |
| 1 | right | 0.1 | 0.8 | 0.0 | 0.0 | 0.1 | |
| 2 | up | 0.1 | 0.8 | 0.1 | 0.0 | 0.0 | |
| ... | | | | | | | |





Bounce back
if hits a wall

# In Game Playing: Opponent's Actions are not certain



The result state of your move Is not certain because it will be decided by your opponent

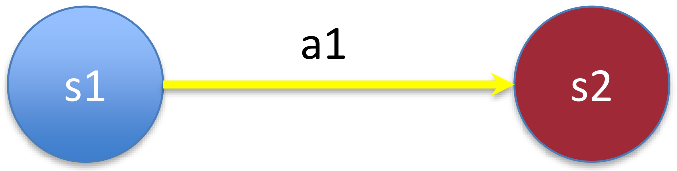
**Therefore, it is appropriate to use: P(next_state | current_state, your_move)**

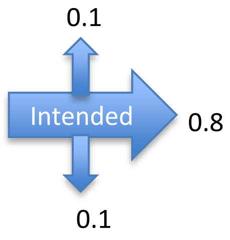
# The Key Representations

- Model the environment by States, Actions, Percepts, and
- Transition (action) model $\varphi = P(s_{t+1}|s_t,a_t)$ may be probabilistic
- Sensor Model $\vartheta = P(z|s)$ may be probabilistic
- States may have Utility Value U(s) or V(s)
- Agents may receive Reward ***r*** (implicit "goals") occasionally:
  – rewards may be given to agent in different format or time
- **Behaviors** may be represented as Policy: state -> action
- **Objective**: find the optimal policy based on utilities and rewards

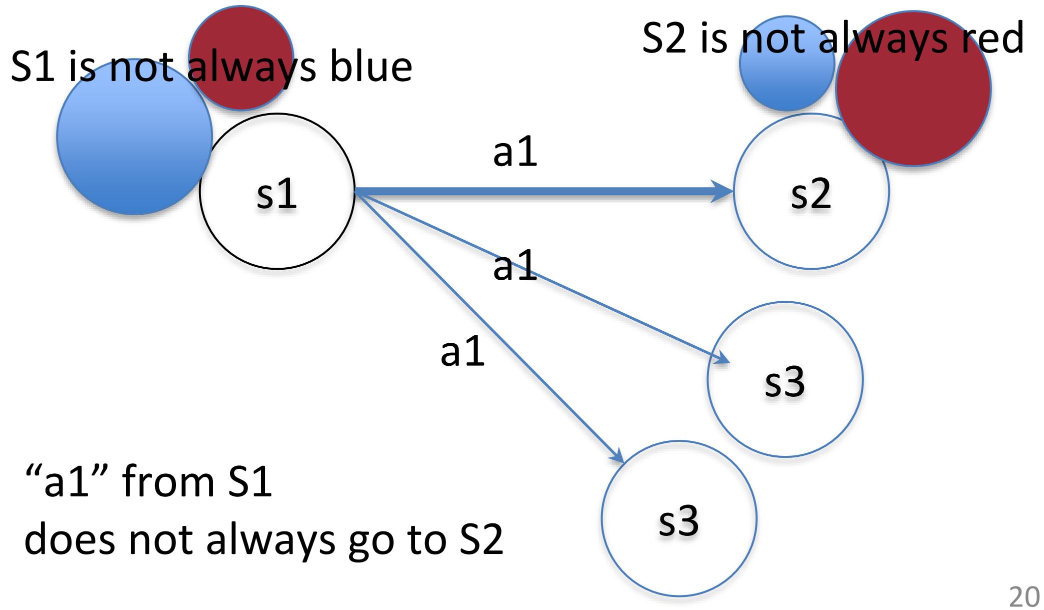
# Certainty vs. Uncertainty

## The certain way





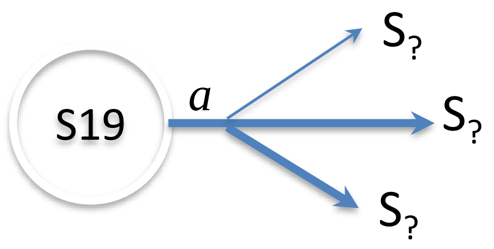
## The uncertain way



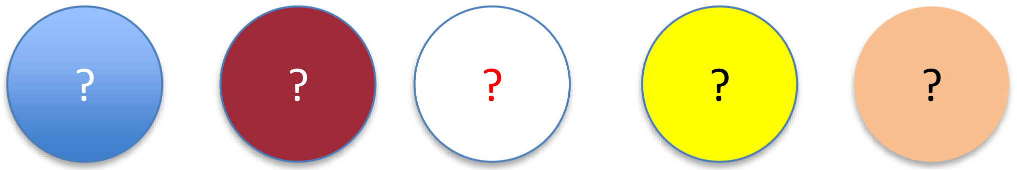
"a1" from S1
does not always go to S2

# Uncertainty in Sensor & Actions

State, action



Observations



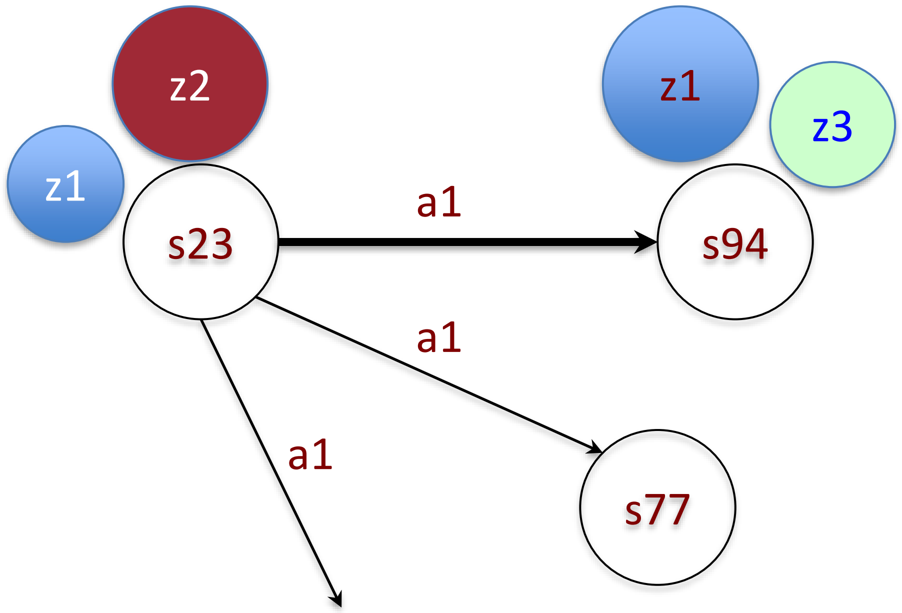
- Solution: let's use probabilities
  - Action/Transition Model (for states and actions)

    Use Probabilistic Transitions

    P( NextState | CurrentState, Action)
  - Sensor Model (for states and observations)

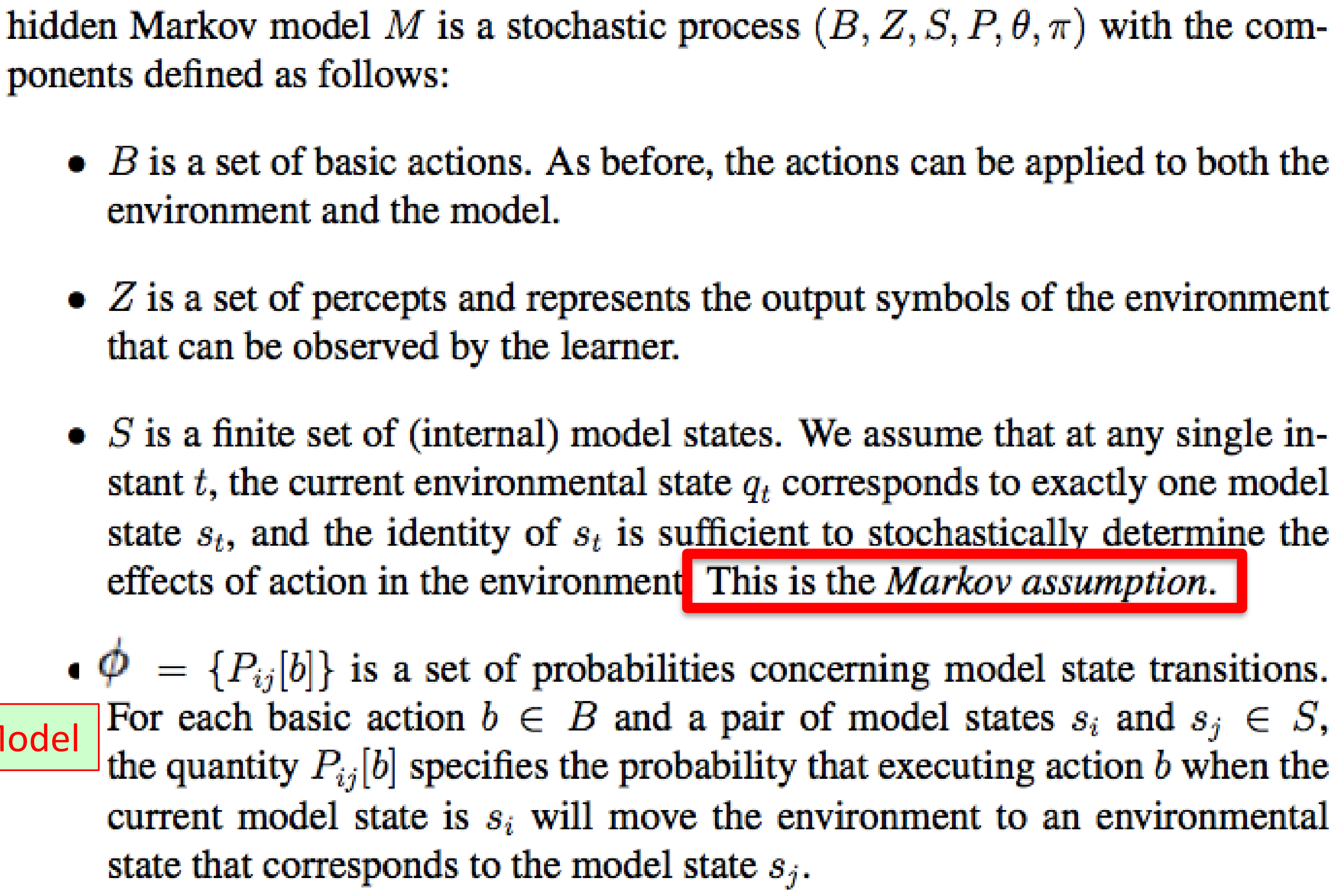
    P( Observations | State )

# Hidden Markov Model (with actions)



1. Actions
2. Percepts (observations)
3. States
4. Appearance: states -> observations
5. Transitions: (states, actions) -> states
6. Current State

# Hidden Markov Model (1/2)



hidden Markov model $M$ is a stochastic process $(B, Z, S, P, \theta, \pi)$ with the components defined as follows:

- $B$ is a set of basic actions. As before, the actions can be applied to both the environment and the model.
- $Z$ is a set of percepts and represents the output symbols of the environment that can be observed by the learner.
- $S$ is a finite set of (internal) model states. We assume that at any single instant $t$, the current environmental state $q_t$ corresponds to exactly one model state $s_t$, and the identity of $s_t$ is sufficient to stochastically determine the effects of action in the environment. This is the *Markov assumption*.
- $\phi = \{P_{ij}[b]\}$ is a set of probabilities concerning model state transitions. For each basic action $b \in B$ and a pair of model states $s_i$ and $s_j \in S$, the quantity $P_{ij}[b]$ specifies the probability that executing action $b$ when the current model state is $s_i$ will move the environment to an environmental state that corresponds to the model state $s_j$.

Action Model

## Hidden Markov Model (2/2)

Sensor Model

- $\theta = \{\theta_i(k)\}$, where $\theta_i(k) = p(z_k|s_i)$, $z_k \in Z$, and $s_i \in S$, is a set of probability distributions of observation symbols in each model state. (This corresponds to the appearance function for the deterministic models defined at the beginning of this chapter.) For each observation symbol $z_k$, the quantity $\theta_i(k)$ specifies the probability of observing $z_k$ if the current model state is $s_i$.

- $\pi(t) = \{\pi_i(t)\}$ is the probability distribution of the current model state at time $t$. That is, $\pi_i(t) = p(i_t = s_i)$, where $i_t$ denotes the current model state and $s_i \in S$ specifies the probability of $s_i$ being the current model state at time $t$.

This is also called "localization"

# The HMM for Little Prince's Planet

(with uncertain actions and sensors)

$A \equiv \{\text{forward, backward, turn-around}\}$ $\{f, b, t\}$

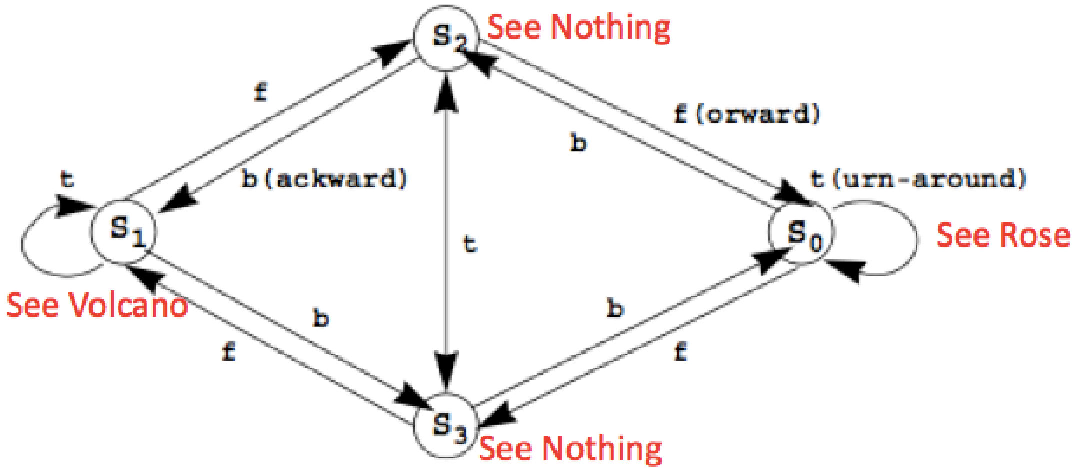
$Z \equiv \{\text{rose, volcano}$ , nothing$\}$

$S \equiv \{s_1, s_2, s_3, s_4\}$

$\phi \equiv$ {P($s_3$|$s_0$,f)=.51, P($s_2$|$s_1$,b)=.32, P($s_4$|$s_3$,t)=.89, …}

$\theta \equiv$ {P(rose|$s_0$)=.76, P(volcano|$s_1$)=.83, P(nothing|$s_3$)=.42, …}

$$\pi_1(0)=0.25, \pi_2(0)=0.25, \pi_3(0)=0.25, \pi_4(0)=0.25$$



# Little Prince Example (may add Rewards)

- (action) Transition Probabilities ϕ (Fwd, Back, Turn)

| F | S0 | S1 | S2 | S3 |
|---|---|---|---|---|
| S0 | 0.1 | 0.1 | 0.1 | 0.7 |
| S1 | 0.1 | 0.1 | 0.7 | 0.1 |
| S2 | 0.7 | 0.1 | 0.1 | 0.1 |
| S3 | 0.1 | 0.7 | 0.1 | 0.1 |

| B | S0 | S1 | S2 | S3 |
|---|---|---|---|---|
| S0 | 0.1 | 0.1 | 0.7 | 0.1 |
| S1 | 0.1 | 0.1 | 0.1 | 0.7 |
| S2 | 0.1 | 0.7 | 0.1 | 0.1 |
| S3 | 0.7 | 0.1 | 0.1 | 0.1 |

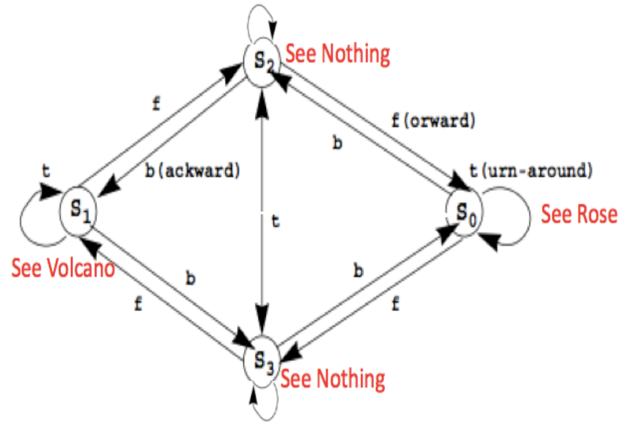
| T | S0 | S1 | S2 | S3 |
|---|---|---|---|---|
| S0 | 0.7 | 0.1 | 0.1 | 0.1 |
| S1 | 0.1 | 0.7 | 0.1 | 0.1 |
| S2 | 0.1 | 0.1 | 0.1 | 0.7 |
| S3 | 0.1 | 0.1 | 0.7 | 0.1 |

- (sensor) Appearance Probabilities θ

| θ | Rose | Volcano | Nothing |
|---|---|---|---|
| S0 | 0.8 | 0.1 | 0.1 |
| S1 | 0.1 | 0.8 | 0.1 |
| S2 | 0.1 | 0.1 | 0.8 |
| S3 | 0.1 | 0.1 | 0.8 |

- Initial State Probabilities π

| π | S0 | S1 | S2 | S3 |
|---|---|---|---|---|
| | .25 | .25 | .25 | .25 |



You may add rewards to states (e.g., prefer to see rose)

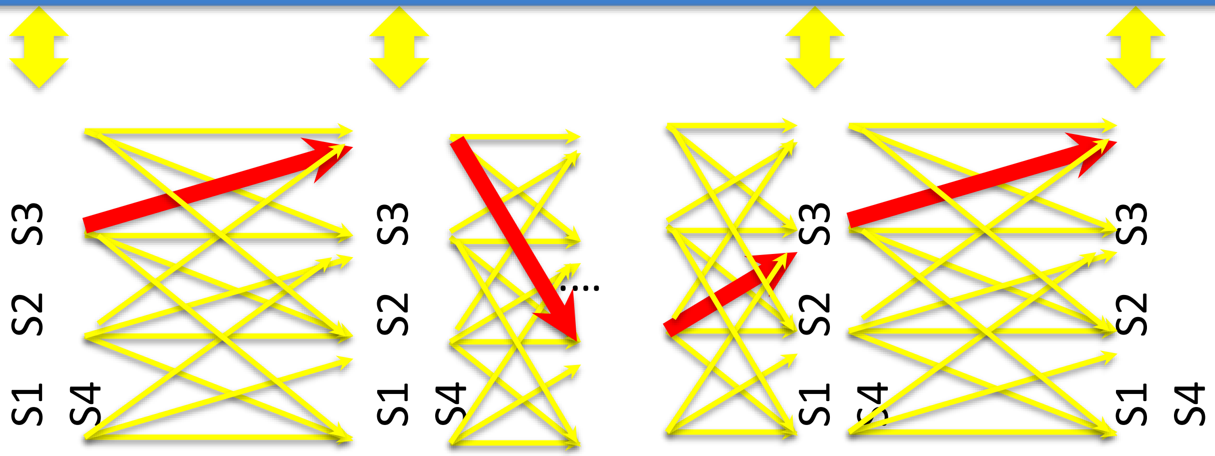
# Experience and State Sequence

- **$E_{1:T}$** is an experience which is a sequence of
  - {observe$_1$, act$_1$, observe$_2$, act$_2$, …., act$_{T-1}$, observe$_T$}
  - **$E_{1:T}$** $= \{o_1, a_1, o_2, a_2, o_3, \ldots, o_{T-1}, a_{T-1}, o_T\}$
- **$X_{1:T}$** is a sequence of states that may be hidden but correspond to the experience
  - $X_{1:T} = \{X_1, X_2, X_3, \ldots, X_{T-1}, X_T\}$

# Little Prince Example

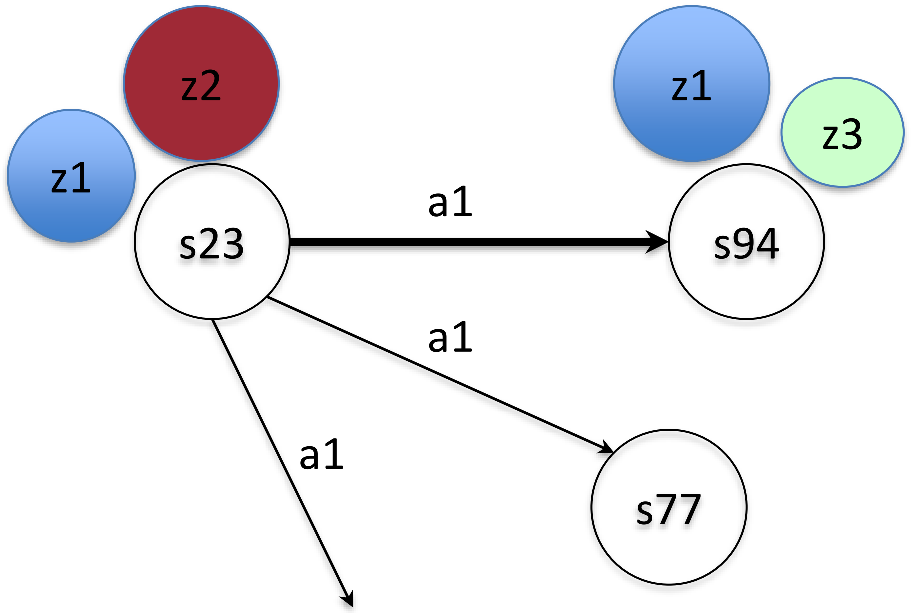
- From an "*experience*" (time$_1$ through time$_t$)
- Infer the most likely "*sequence of states*"

{rose}, forward, {.}, …, turn, …, {rose}, bwd, {volcano}



Find the "best sequence of (hidden) states" that support the experience!

# Action and Sensor Models (review)

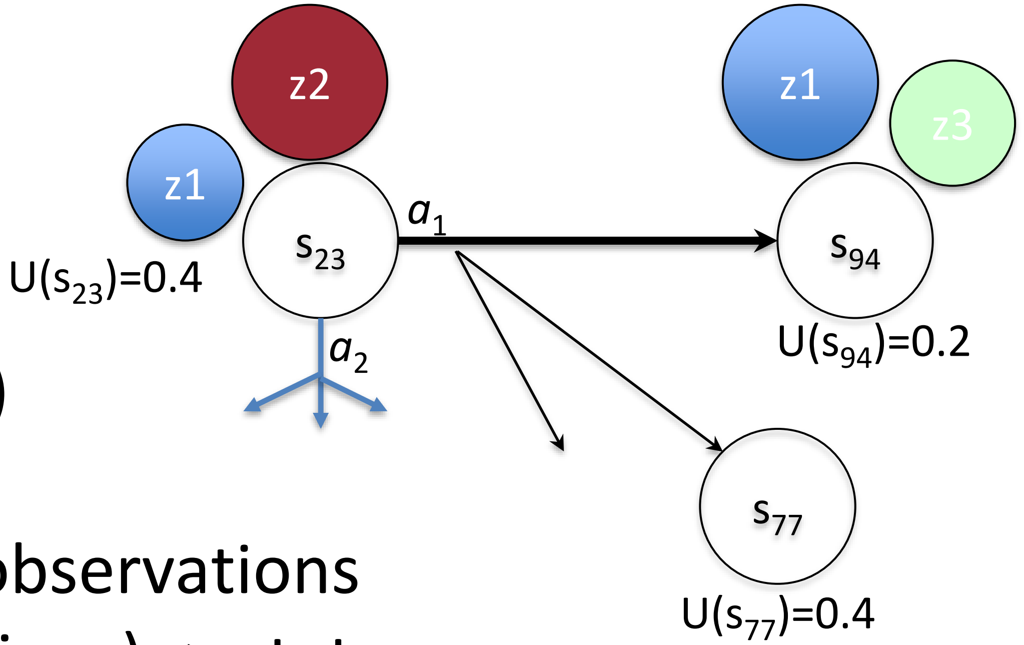


1. Actions
2. Percepts (observations)
3. States
4. Appearance: states -> observations
5. Transitions: (states, actions) -> states
6. Current State

What about the goals?

# Utility Value of States

## Utility Value ⇔ Goal Information

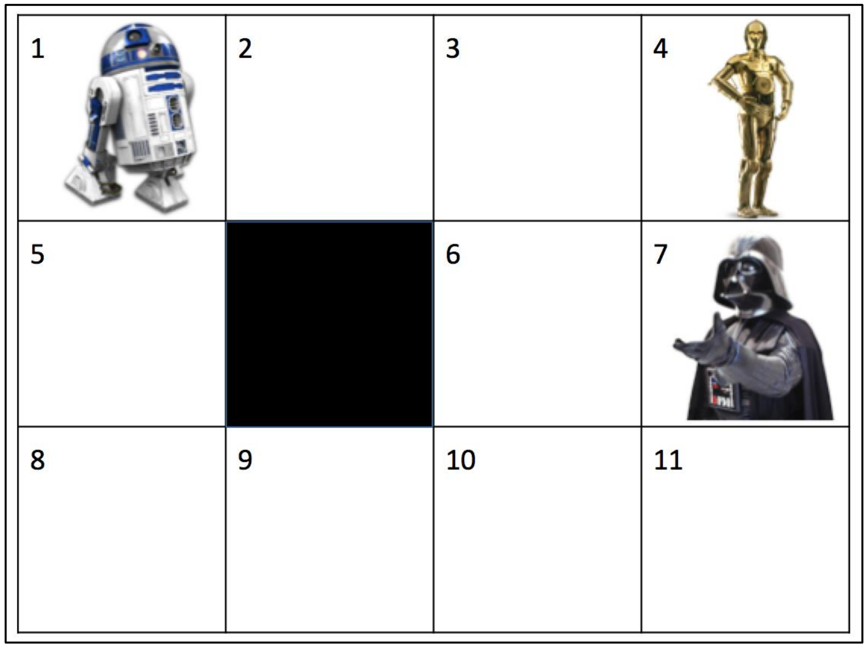


1. Actions
2. Percepts (observations)
3. States
4. Appearance: states -> observations
5. Transitions: (states, actions) -> states
6. Current State
7. **Rewards**: R(s) or R(s,a) (related to the goals, given to the agent)

(8) **Utility Value** of States: *U*(*s*) (how good for the goals, must compute)

# Rewards and Utilities

- Three types of rewards for agents
    1. Being at a state R(s), e.g., holding an ice cream
    2. Do an action on a state R(s, a), e.g., eating the ice cream
    3. Making a transition R(s, a, s'), e.g., feeling good after eating
- Every state may have a Utility Value U(s) or V(s)



Rewards and Utility for R2D2:

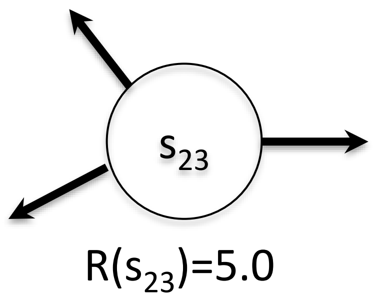
| $X_t$ | Reward | Utility |
|---|---|---|
| 4 | +1 | 0.0 |
| 7 | -1 | 0.0 |
| Else | -0.04 | 0.0 |

Given

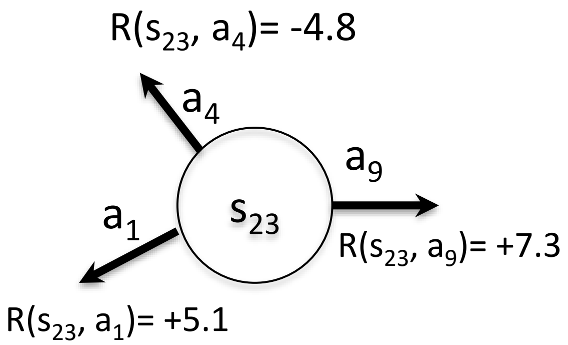
To be learned

# Rewards (caution, 3 types)

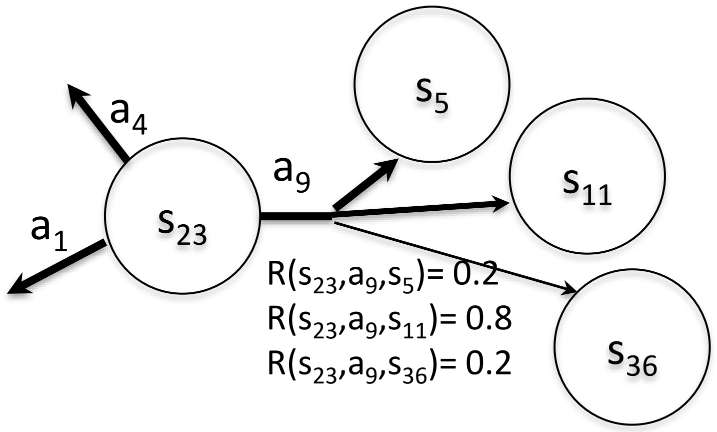
- Three types of rewards for agents
  1. Being at a state R(s), e.g., holding an ice cream
  2. Do an action on a state R(s, a), e.g., eating the ice cream
  3. Making a transition R(s, a, s'), e.g., feeling good after eating



Type I: R(s)



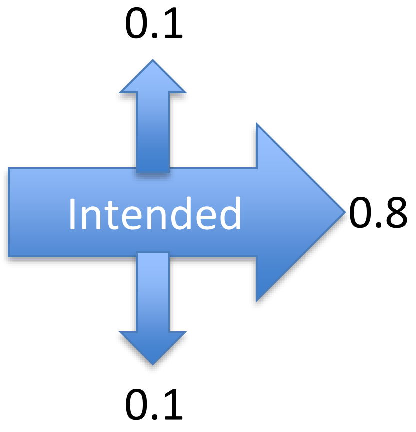
Type II: R(s,a)



Type III: R(s,a,s')

# Reward and Action Example

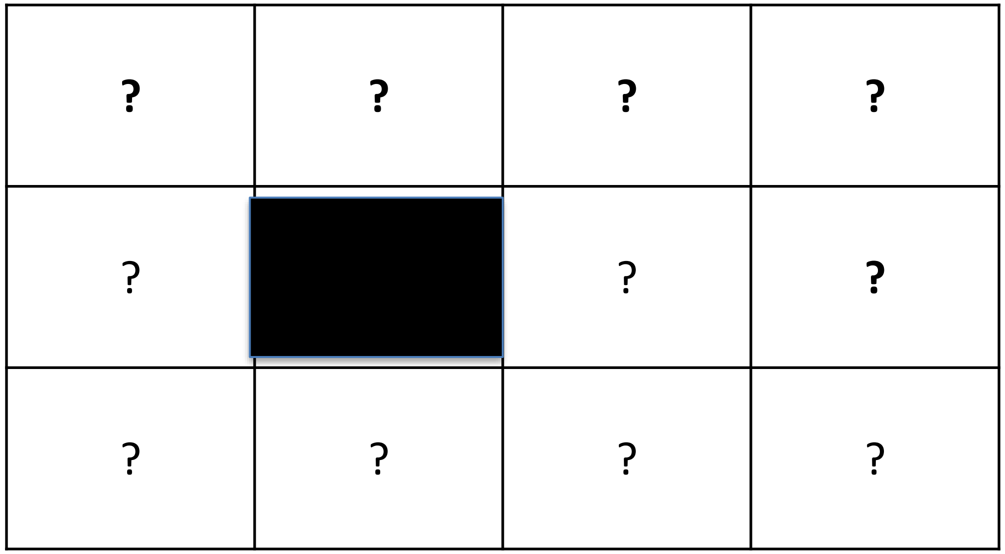
| | | | |
|---|---|---|---|
| -0.04 | -0.04 | -0.04 | **+1** |
| -0.04 | ■ | -0.04 | **-1** |
| -0.04 (start) | -0.04 | -0.04 | -0.04 |

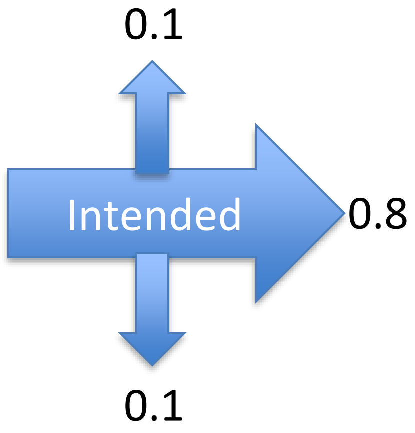


Rewards (not utility) as shown:
 Two terminal states: -1, +1 (goal)
-0.04 for all nonterminal states

Actions: >, <, ^, v,
 outcome probability:
 0.8 for the intended
 0.2 sideways
// bounce back if hits wall
// In a terminal state, the probability
// to go anywhere is 0.0

# State Utility Value Example



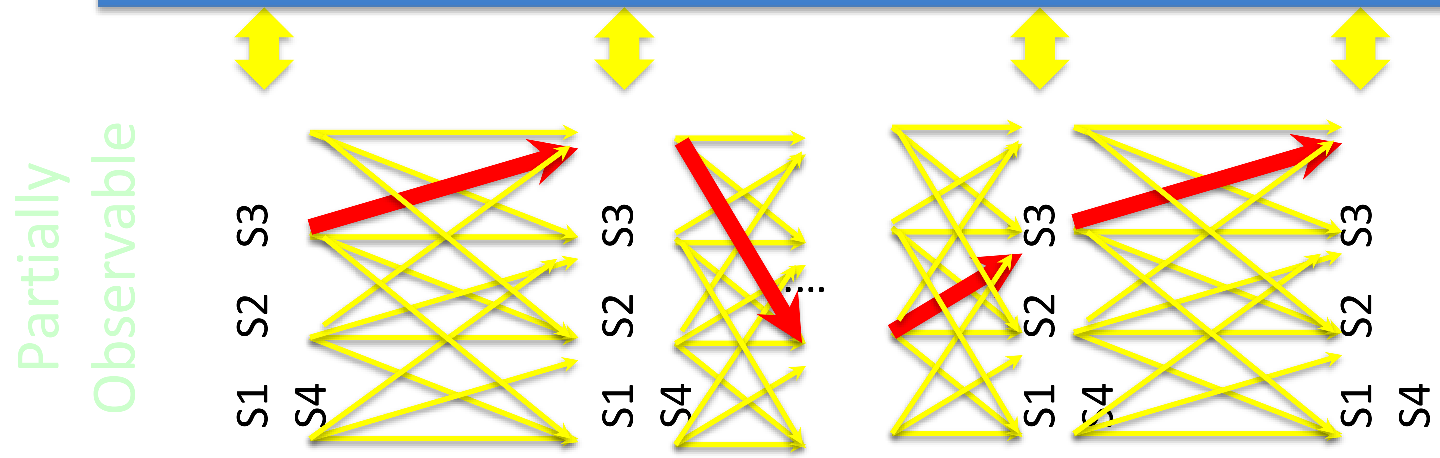


State Utility Values:
  Must be learned or computed by the agent
  Initially, could be all 0.0

# Little Prince in Action (POMDP)

$E_{1:T}$ {rose}, fward, {none}, ..., turn, {rose}, back, {volcano}

time



- Given: "*experience*" $\mathbf{E}_{1:T}$ (time 1 through T)
- Definitions:
  - State S and actions A (Forward, Back, Turn)
  - (Action model) Transition Probabilities Φ
  - (Sensor model) Appearance Probabilities θ (rose, volcano, none)
  - (localization) Initial/current State Probabilities π
- Partially Observable: agent sees only the percepts, not the states

## State Utility and Decision on Actions

- Combine the following together
  - Partially observable action/perception model
    - E.g., Little Prince Example
  - Compute state utility values from rewards (goals)
    - E.g., use Dynamic Programming (next slide)
    - Bellman equations, Bellman iterations (later slides)
  - Policy (decide actions based on state utility values)
    - To gain as much as rewards as you can
    - To go to the goals as close as you can

# Compute State Values by Dynamic Programming (review, from ALFE 6.1.1)

Backward recursion: compute utility/values by backing from the goal* stage by stage

$$\begin{tikzcd}
 & s_1 \arrow[r, "a_1[1]"] \arrow[dr, "a_2[3]"] & s_4 \arrow[r, "a_1[2]"] \arrow[dr, "a_2[1]"] & s_7^* \\
s_0 \arrow[ur, "a_0[1]"] \arrow[r, "a_1[2]"] \arrow[dr, "a_2[3]"] & s_2 \arrow[ur, "a_0[6]"] \arrow[r, "a_1[2]"] \arrow[dr, "a_2[5]"] & s_5 \arrow[ur, "a_0[1]"] \arrow[r, "a_1[1]"] \arrow[dr, "a_2[0]"] & s_8^* \\
 & s_3 \arrow[ur, "a_0[2]"] \arrow[r, "a_1[3]"] & s_6 \arrow[ur, "a_0[1]"] \arrow[r, "a_1[0]"] & s_9
\end{tikzcd}$$

Rewards are given $a_i[R] = R(s,a_i)$ from an action $a_i$ on a state $s$

are recorded. For example, the best value one can get from state $s_4$ to a goal state is 2, denoted as $V(s_4) = 2$. Similarly, $V(s_5) = 1$ and $V(s_6) = 1$. These

$$\begin{aligned}
V(s_1) &= \max\{R(s_1,a_1)+V(s_4), R(s_1,a_2)+V(s_5)\} = \max\{1+2, 3+1\} = 4 \\
V(s_2) &= \max\{R(s_2,a_0)+V(s_4), R(s_2,a_1)+V(s_5), R(s_2,a_2)+V(s_6)\} \\
&= \max\{6+2, 2+1, 5+1\} = 8 \\
V(s_3) &= \max\{R(s_3,a_0)+V(s_5), R(s_3,a_1)+V(s_6)\} = \max\{2+1, 3+1\} = 4
\end{aligned}$$

$$\begin{aligned}
V(s_0) &= \max\{R(s_0,a_0)+V(s_1), R(s_0,a_1)+V(s_2), R(s_0,a_2)+V(s_3)\} \\
&= \max\{1+4, 2+8, 3+4\} = 10
\end{aligned}$$

Backward recursion equation: $V(s_i) = \max_a\{R(s_i,a)+V(s_j)\}$ where $s_i \xrightarrow{a} s_j$

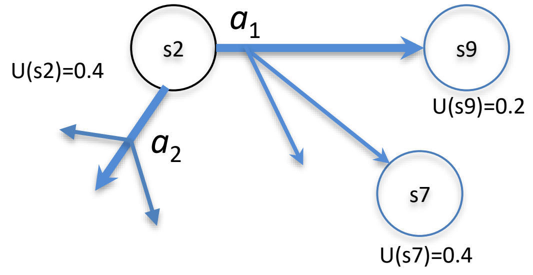
## Markov Decision Process (MDP)

- A MDP consists of
  - State S and actions A
  - Initial State $s_0$ Probability Distribution $\pi$
  - Transition Model $\Phi(s'|s,a)$
  - ~~Sensor Model $\theta(z|s)$~~
  - A reward function R(s)

Note: A typical MDP has no sensor model

# Maximum Expected Utility (MEU) and Rational Agents

- Every state has a utility value *U*(*s*)
- The expected utility of an action given the current evidence or observation *e*, is the average utility value of the outcomes, weighted by the probability that the outcome occurs:



$$EU(a|e) = \sum_{s'} P(result(a) = s' | a, e) U(s')$$

- The principle of maximum expected utility (MEU) is that a **rational** agent should choose the action that maximizes its expected utility:

$$action = \arg\max_{a} EU(a|e)$$

# POMDP: Sensor Model, Belief States

- A Partially Observable MDP (POMDP) is defined as follows:
  - A set of states S (with an initial state $s_0$)
  - A set of Actions: A(s) of actions in each state s
  - A transition model P(s'|s,a), or T(s,a,s')
  - A reward function R(s), or R(s,a)
  - A sensor model P(*e*|*s*) ⍰ ⍰
  - A belief of what the current state is *b*(*s*) ⍰ ⍰
- Belief States (where am I now? What is my current state?):
  - If *b(s*) was the previous belief state, and the robot does action "*a*" and then perceives a new evidence "*e*", then the new belief state:

$$b'(s') = \alpha P(e \mid s') \sum P(s' \mid s, a) b(s)$$

where α is a normalization constant making the belief states sum to 1

## Goals, Rewards, Utilities, Policies

- Goals
  - Given to the agent from the problem statements
- Rewards
  - Given to the agent, designed based on the goals
- Utility values for states
  - Computed by the agent, based on the rewards
- Policies
  - Computed or learned by the agent
  - Used by the agent to select its actions
  - The better a policy, the more rewards it collects

## Compute Utilities from Rewards over Time

- Utility Value = sum of all future rewards
  - Add the rewards as they are

$$U_h = ([s_0, s_1, s_2, \ldots]) = R(s_0) + R(s_1) + R(s_2) + \ldots$$

  - Discount the far-away rewards in the future

$$U_h = ([s_0, s_1, s_2, \ldots]) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \ldots$$

- The expected future utility value $U^\pi(s)$ obtained by executing a policy *π* starting from *s*

$$U^\pi(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t)\right]$$

The Optimal Policy

$$\pi_s^* = \underset{\pi}{\arg\max}\, U^\pi(s)$$

## Compute Utilities (example)

| | | | |
|---|---|---|---|
| -0.04 | -0.04 | -0.04 | **+1** |
| -0.04 | | -0.04 | **-1** |
| -0.04 (start) | -0.04 | -0.04 | -0.04 |

Rewards (given)

| | | | |
|---|---|---|---|
| **0.812** | **0.868** | **0.918** | **+1** |
| **0.762** | | **0.660** | **-1** |
| **0.705** | **0.655** | **0.611** | **0.388** |

Utilities (computed)

Given the rewards in all nonterminal state are R(s) = - 0.04,
Compute utilities using a future discount γ = 1
(we will see how these utilities are computed)

## State Utility Value Iteration
(improving *U*(*s*) every step)

- *U*(*s*): the expected sum of maximum rewards achievable starting from a particular state *s*
- Bellman equations:
  - For n states, there are n equations must be solved *simultaneously*

$$U^*(s) = R(s) + \gamma \max_a \sum_{s'} T(s,a,s') U^*(s') \qquad (17.5)$$

- Bellman iteration:
  - Converge to *U**(*s*) step by step

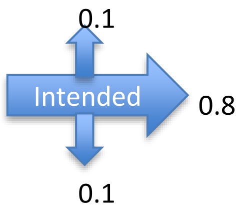
$$U_{i+1}(s) \leftarrow R(s) + \gamma \max_a \sum_{s'} T(s,a,s') U_i(s') \qquad (17.6)$$

# Bellman Iteration Example

Reward *R(s)*

|  |  |  |  |
|---|---|---|---|
| -0.04 | -0.04 | -0.04 | **+1** |
| -0.04 | ▨ | -0.04 | **-1** |
| -0.04 | -0.04 | -0.04 | -0.04 |

Transaction probability
T(s, a, s') = 0.8 for intended,
0.2 for sideways
Discount Gamma $\gamma$ = 1.0



The Initial Utility Values

|  |  |  |  |
|---|---|---|---|
| 0.0 | 0.0 | 0.0 | 0.0 |
| 0.0 | ▨ | 0.0 | 0.0 |
| 0.0 | 0.0 | 0.0 | 0.0 |

$$U_{i+1}(s) \leftarrow R(s) + \gamma \max_a \sum_{s'} T(s,a,s') U_i(s')$$

|  |  |  |  |
|---|---|---|---|
| 0.812 | 0.868 | 0.918 | +1 |
| 0.762 | ▨ | 0.660 | -1 |
| 0.705 | 0.655 | 0.611 | 0.388 |

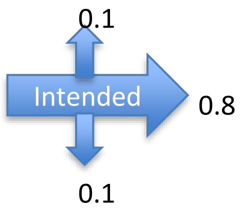
The Final Utility Values

# Bellman Iteration Example (1st iteration)

Reward *R(s)*

| | | | |
|---|---|---|---|
| -0.04 | -0.04 | -0.04 | **+1** |
| -0.04 | ▨ | -0.04 | **-1** |
| -0.04 | -0.04 | -0.04 | -0.04 |

Transaction probability
T(s, a, s') = 0.8 for intended,
0.2 for sideways
Discount Gamma $\gamma$ = 1.0



The Initial Utilities $U_0$

| | | | |
|---|---|---|---|
| 0.0 | 0.0 | 0.0 | 0.0 |
| 0.0 | ▨ | 0.0 | 0.0 |
| 0.0 | 0.0 | 0.0 | 0.0 |

$U_1(s_1)$ = -0.04 + 1.0*max(0.0*0.0)

$U_1(s_4)$ = +1.0 + 1.0*max(0.0*0.0)
$U_1(s_7)$ = -1.0 + 1.0*max(0.0*0.0)

| | | | |
|---|---|---|---|
| -0.04 | -0.04 | -0.04 | +1.0 |
| -0.04 | ▨ | -0.04 | -1.0 |
| -0.04 | -0.04 | -0.04 | -0.04 |

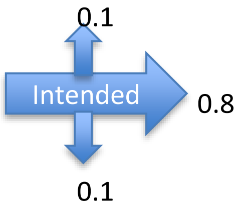
The Utility Values $U_1$ after 1st iteration

# Bellman Iteration Example (2nd iteration)

Reward *R(s)*

| | | | |
|---|---|---|---|
| -0.04 | -0.04 | -0.04 | **+1** |
| -0.04 | | -0.04 | **-1** |
| -0.04 | -0.04 | -0.04 | -0.04 |

Transaction probability
T(s, a, s') = 0.8 for intended,
0.2 for sideways
Discount Gamma $\gamma$ = 1.0



$U_1$

| | | | |
|---|---|---|---|
| -0.04 | -0.04 | -0.04 | +1.0 |
| -0.04 | | -0.04 | -1.0 |
| -0.04 | -0.04 | -0.04 | -0.04 |

$U_2(s_4)$ = +1.0 + 1.0*max(0, 0, 0)
$U_2(s_7)$ = -1.0 + 1.0*max(0, 0, 0)
$U_2(s_3)$ = -.04 + 1.0*max(.8*1-.1*.04-.1*.04, …)
$U_2(s_6)$ = -.04 + 1.0*max(-.8*.04-.1*.04-.1*.04,…)

$U_2$

| | | | |
|---|---|---|---|
| ? | ? | 0.752 | +1.0 |
| ? | | -0.08 | -1.0 |
| ? | ? | ? | ? |

The Utility Values $U_2$ after 2nd iteration
Please fill in the values for "?"

# Bellman Iteration Example

Reward *R(s)*

| | | | |
|---|---|---|---|
| -0.04 | -0.04 | -0.04 | **+1** |
| -0.04 | | -0.04 | **-1** |
| -0.04 | -0.04 | -0.04 | -0.04 |

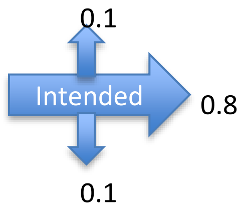
The Initial Utilities

| | | | |
|---|---|---|---|
| 0.0 | 0.0 | 0.0 | 0.0 |
| 0.0 | | 0.0 | 0.0 |
| 0.0 | 0.0 | 0.0 | 0.0 |

$$U_{i+1}(s) \leftarrow R(s) + \gamma \max_a \sum_{s'} T(s,a,s') U_i(s')$$

| | | | |
|---|---|---|---|
| 0.812 | 0.868 | 0.918 | +1 |
| 0.762 | | 0.660 | -1 |
| 0.705 | 0.655 | 0.611 | 0.388 |

Transaction probability
T(s, a, s') = 0.8 for intended,
0.2 for sideways
Discount Gamma $\gamma$ = 1.0



Please verify these final Utility Values by yourself!!

# Utility Value Iteration (3 types of rewards)

- When rewards are given as R(s)
  - $U_{t+1} \leftarrow R(s) + \gamma \max_{a} \sum_{s'} T(s,a,s') U_t(s')$
- When rewards are given as R(s,a)
  - $U_{t+1} \leftarrow R(s,a) + \gamma \max_a \sum_{s'} T(s,a,s') U_t(s')$
- When rewards are given as R(s,a,s')
  - $U_{t+1} \leftarrow \gamma \max_{a} \sum_{s'} T(s,a,s') [R(s,a,s') + U_t(s')]$
  - $U_{t+1} \leftarrow \max_{a} \sum_{s'} T(s,a,s') [R(s,a,s') + \gamma U_t(s')]$

# Utility Value Iteration for POMDP

- In MDP, utility value iteration, when sensors are certain, we know the next state *s'*

$$U_{i+1}(s) \leftarrow R(s) + \gamma \max_{a} \sum_{s'} T(s,a,s') U_i(s')$$

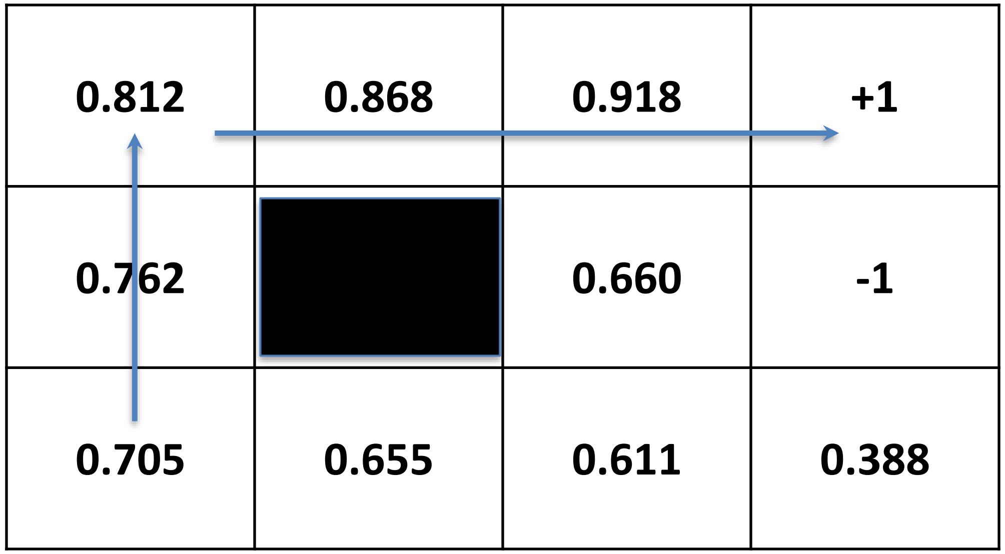
- In POMDP, where sensors are uncertain, we must average over all possible evidences for the next state *s'* (see the last term below):

$$\alpha_p(s) = R(s) + \gamma \left( \sum_{s'} T(s,a,s') \sum_{e} P(e \mid s') \alpha_{p.e}(s') \right)$$

## Policy and Optimal Policy

- A solution of (PO)MDP can be represented as
  - A Policy $\pi(s)=a$
  - Each execution of policy from $s_0$ may yield a different history or path due to the probabilistic nature of the transition model
  - An optimal policy $\pi^*(s)$ is a policy that yields the highest expected utility

# Policy Based on Known Utility Values



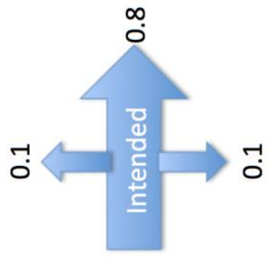
| | | | |
|---|---|---|---|
| 0.812 | 0.868 | 0.918 | +1 |
| 0.762 | ■ | 0.660 | -1 |
| 0.705 | 0.655 | 0.611 | 0.388 |

Suppose we know the utility values of the states as above,
What path would an optimal policy would choose?

**Rewards -> Utility Values -> Policies**

# Optimal Policy Examples
## Depending on **Reward** distribution R(s)

If the **rewards** for the nonterminal states are evenly distributed (e.g., -0.04), then the path chosen will depend on the probabilities of the transition model



| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 3 | -> | -> | -> | **+1** |
| 2 | ^ | | ^ | **-1** |
| 1 | ^ | <- | ^ | <- |

Nonterminal R(s)= - 0.04

Caution: because the nondeterministic actions, from state (3,2) or (4,1), you may "accidentally" go to (4,2). So there is a "risk" in this policy.

# Optimal Policy Examples

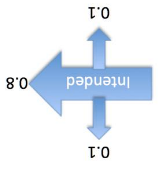
## Depending on the reward distribution R(s)

Nonterminal rewards R(s)=-0.04

| | | | |
|---|---|---|---|
| > | > | > | +1 |
| ^ | | ^ | -1 |
| ^ | < | < ? | < |

Why not go up?

Hint: Consider the nature of actions, deterministic or not
Go up is "risky"



| | | | |
|---|---|---|---|
| > | > | > | +1 |
| ^ | | > suicide | -1 |
| ^ | > | > | ^ suicide |

R(s)<-1.6284
(life is painful, death is better)

| | | | |
|---|---|---|---|
| > | > | > | +1 |
| ^ | | ^ risky | -1 |
| ^ | > | ^ | < |

-0.4278 < R(s)<-0.0850
(life is OK, willing to risk)

| | | | |
|---|---|---|---|
| > | > | > end nicely | +1 |
| ^ | | < no risk | -1 |
| ^ | < | < | < risky |

-0.0221<R(s)<0
(life is good, minimize risks, willing to end nicely)

| | | | |
|---|---|---|---|
| ❖ | ❖ | < don't end | +1 |
| ❖ | | < no risk | -1 |
| ❖ | ❖ | ❖ | V no risk |

R(s)>0
(life is rewarding, I don't want to end it)

## Optimal Policy

- You can compute the optimal policy once after all *U**(*s*) are known

$$\pi_s^* = \operatorname*{argmax}_a \sum T(s,a,s')U^*(s')$$

- Or, you can compute it incrementally every iteration when $U_i(s)$ is updated
  - This is called "Policy Iteration" (see the next slide)

# Policy Iteration
## (improving π(s) every step)

- Start with a randomly chosen initial policy $\pi_0$
- Iterate until no change in utility values of the state:
- Policy evaluation: given a policy $\pi_i$, calculate the utility $U_i(s)$ of every state *s* using policy $\pi_i$ by solving the system of equations:

$$U_\pi(s) = R(s) + \gamma \sum_{s'} T(s, \pi(s), s') U_\pi(s')$$

- Policy improvement: calculate the new policy $\pi_{i+1}$ using one-step look-ahead based on the current $U_i(s)$:

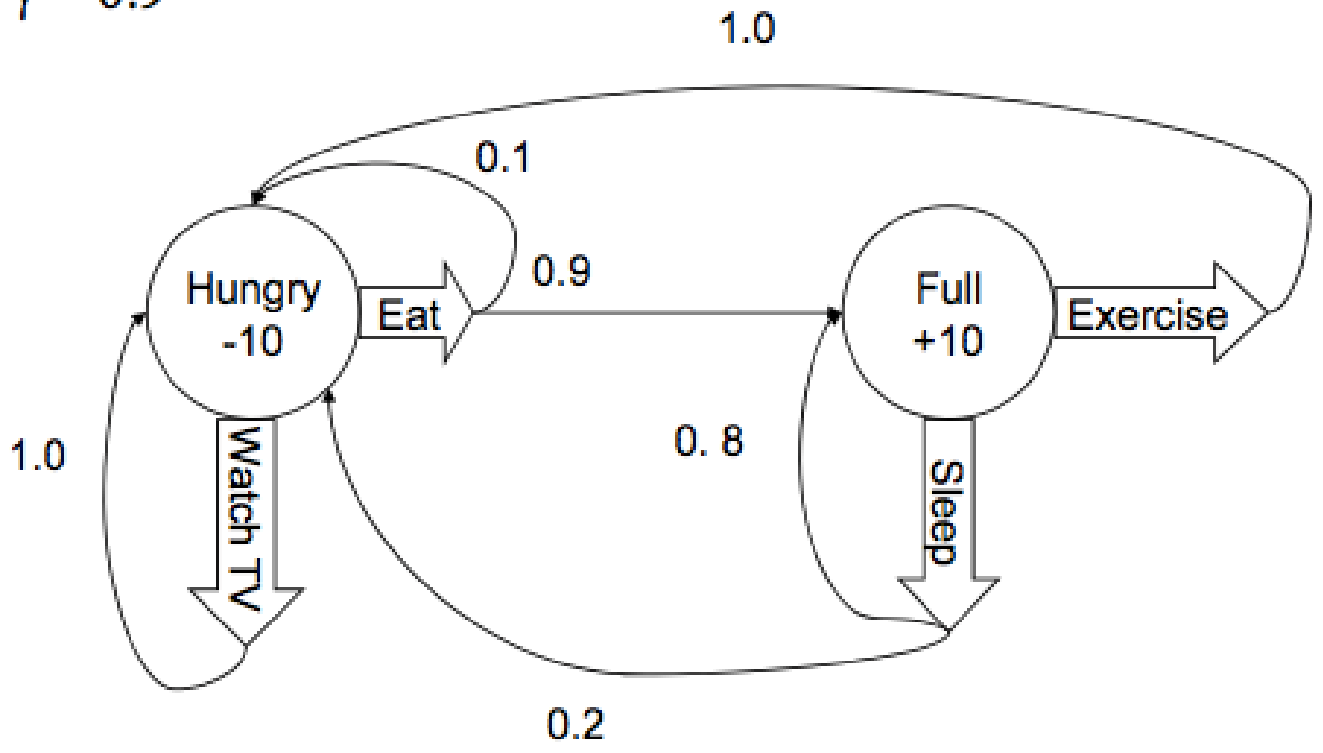
$$\pi_{i+1}(s) = \underset{a}{\operatorname{argmax}} \sum_{s'} T(s, a, s') U^*(s')$$

# Policy Iteration Comments

- Each step of policy iteration is guaranteed to strictly improve the policy at some state when improvement of state utility value is possible
- Converge to the optimal policy
- Gives the exact value of the optimal policy

# Policy Iteration Example

Do one iteration of policy iteration on the MDP below. Assume an initial policy of $\pi_1$(Hungry) = Eat and $\pi_1$(Full) = Sleep.
Let $\gamma = 0.9$



## Policy Iteration Example

Use initial policy for Hungry: $\pi_1(\text{Hungry}) = \text{Eat}$

$U_1(\text{Hungry}) = -10 + (0.9)[(0.1)U_1(\text{Hungry})+(0.9)U_1(\text{Full})]$

$\Rightarrow U_1(\text{Hungry}) = -10 + (0.09)U_1(\text{Hungry})+(0.81)U_1(\text{Full})$

$\Rightarrow (0.91)U_1(\text{Hungry})-(0.81)U_1(\text{Full}) = -10$

Use initial policy for Full: $\pi_1(\text{Full}) = \text{Sleep}$.

$U_1(\text{Full}) = 10 + (0.9)[(0.8)U_1(\text{Full}) + (0.2)U_1(\text{Hungry})]$

$\Rightarrow U_1(\text{Full}) = 10 + (0.72)U_1(\text{Full}) + (0.18)U_1(\text{Hungry})]$

# Policy Iteration Example

$$\left.\begin{array}{l}(0.91)U_1(\text{Hungry})-(0.81)U_1(\text{Full}) = -10 \text{ ....(Equation 1)}\\(0.28)U_1(\text{Full}) - (0.18)U_1(\text{Hungry})=10 \text{ ...(Equation 2)}\end{array}\right\}$$

Solve for $U_1$(Hungry) and $U_1$(Full)

From Equation 1:

$(0.91)U_1(\text{Hungry}) = -10+(0.81)U_1(\text{Full})$

$\Rightarrow U_1(\text{Hungry}) = (-10/0.91)+(0.81/0.91)U_1(\text{Full})$

$\Rightarrow U_1(\text{Hungry})=-10.9+(0.89)U_1(\text{Full})$

# Policy Iteration Example

$$\left.\begin{array}{l}(0.91)U_1(\text{Hungry})-(0.81)U_1(\text{Full}) = -10 \text{ ....(Equation 1)} \\ (0.28)U_1(\text{Full}) - (0.18)U_1(\text{Hungry})=10 \text{ ...(Equation 2)}\end{array}\right\} \text{Solve for } U_1(\text{Hungry}) \text{ and } U_1(\text{Full})$$

Substitute $U_1(\text{Hungry})=-10.9+(0.89)U_1(\text{Full})$ into Equation 2

$(0.28)U_1(\text{Full}) - (0.18)[-10.9+(0.89)U_1(\text{Full})]=10$

$\Rightarrow(0.28)U_1(\text{Full}) + 1.96-(0.16)U_1(\text{Full})=10$

$\Rightarrow(0.12)U_1(\text{Full})=8.04$

$\Rightarrow U_1(\text{Full})=67$

$\Rightarrow U_1(\text{Hungry})=-10.9+(0.89)(67)=-10.9+59.63=48.7$

# Policy Iteration Example

$$
\begin{aligned}
&\pi_2(\text{Hungry}) \\
&= \underset{\{\text{Eat},\text{WatchTV}\}}{\operatorname{argmax}} \begin{Bmatrix} \text{T(Hungry, Eat, Full)}U_1(\text{Full}) + & \\ \quad \text{T(Hungry, Eat, Hungry)}U_1(\text{Hungry}) & [\text{Eat}] \\ \text{T(Hungry, WatchTV, Hungry)}U_1(\text{Hungry}) & [\text{WatchTV}] \end{Bmatrix} \\
&= \underset{\{\text{Eat},\text{WatchTV}\}}{\operatorname{argmax}} \begin{Bmatrix} (0.9)\text{U1(Full)} + (0.1)\text{U1(Hungry)} & [\text{Eat}] \\ (1.0)\text{U1(Hungry)} & [\text{WatchTV}] \end{Bmatrix} \\
&= \underset{\{\text{Eat},\text{WatchTV}\}}{\operatorname{argmax}} \begin{Bmatrix} (0.9)(67) + (0.1)(48.7) & [\text{Eat}] \\ (1.0)(48.7) & [\text{WatchTV}] \end{Bmatrix} \\
&= \underset{\{\text{Eat},\text{WatchTV}\}}{\operatorname{argmax}} \begin{Bmatrix} 65.2 & [\text{Eat}] \\ 48.7 & [\text{Watch}] \end{Bmatrix} \\
&= \text{Eat}
\end{aligned}
$$

# Policy Iteration Example

$$
\begin{aligned}
&\pi_2(\text{Full}) \\
&= \underset{\{\text{Exercise},\text{Sleep}\}}{\text{argmax}} \left\{ \begin{array}{ll} \text{T(Full, Exercise, Hungry)U}_1\text{(Hungry)} & \text{[Exercise]} \\ \text{T(Full, Sleep, Full)U}_1\text{(Full)} + & \\ \quad \text{T(Full, Sleep, Hungry)U}_1\text{(Hungry)} & \text{[Sleep]} \end{array} \right\} \\
&= \underset{\{\text{Exercise},\text{Sleep}\}}{\text{argmax}} \left\{ \begin{array}{ll} (1.0)\text{U}_1\text{(Hungry)} & \text{[Exercise]} \\ (0.8)\text{U}_1\text{(Full)} + (0.2)\text{U}_1\text{(Hungry)} & \text{[Sleep]} \end{array} \right\} \\
&= \underset{\{\text{Exercise},\text{Sleep}\}}{\text{argmax}} \left\{ \begin{array}{ll} (1.0)(48.7) & \text{[Exercise]} \\ (0.8)(67) + (0.2)(48.7) & \text{[Sleep]} \end{array} \right\} \\
&= \underset{\{\text{Exercise},\text{Sleep}\}}{\text{argmax}} \left\{ \begin{array}{ll} 48.7 & \text{[Exercise]} \\ 63.34 & \text{[Sleep]} \end{array} \right\} \\
&= \text{Sleep}
\end{aligned}
$$

## Policy Iteration Example

- The new policy $\pi_2$ after an iteration from $\pi_1$
  - $\pi_2$(Hungry)->Eat
  - $\pi_2$(Full)->Sleep

## Summary for Uncertain Environments

- POMDP is very (the most) general model
  - Deal with uncertainty by
    - action models and sensor models
- Incorporate goals -> rewards -> utilities -> policy
  - Utilities and policies can be computed from rewards
    - Systematically  (Bellman equations)
    - Iteratively (Bellman’s iteration algorithm)
  - Solve a problem by following a good policy
    - One policy for one problem/goal
    - Different policies are needed for different goals

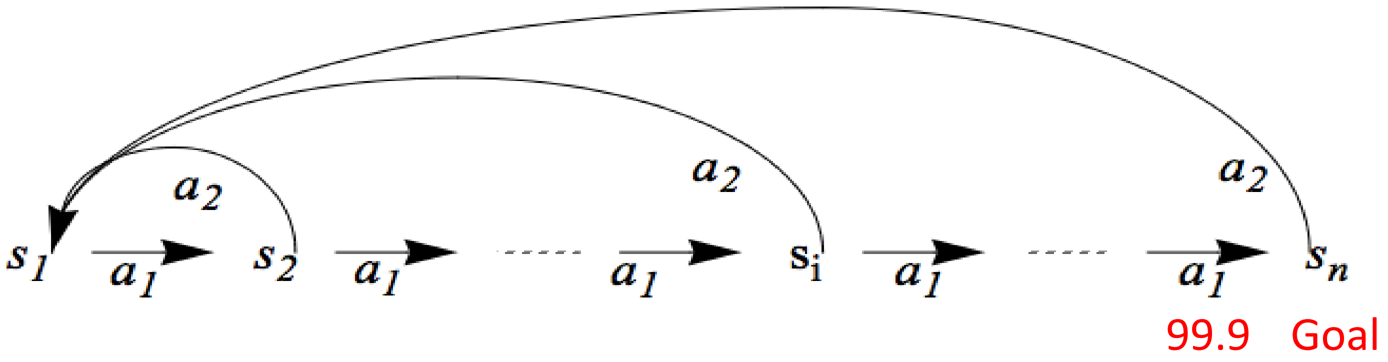
## REINFORCEMENT LEARNING

# Markov Decision Process

- **Markov Decision Process**
  - We learned that a Markov Decision Process (MDP) consists of
    - A set of states S (with an initial state $s_0$)
    - A set of actions A in each state,
    - A set of percepts that can be sensed
    - A transition model P(s'|s,a), or T(s,a,s')
    - ~~A sensor model θ = P(z|s)~~
    - The current state distribution
    - A reward function R(s,a,s')     // careful here
  - The MDP's solution identifies the best action to take in each state
    - Optimal policy π(s)=a obtained by value iteration or policy iteration (aka dynamic programming)

# An Example for MDP

- Given
  - States: $s_1, \ldots, s_{n,}$ Actions: $a_1, a_2$
  - Transition Probability Distribution: (assume $s_{n+1}=s_1$)
    - $P(s_i,a_1,s_{i+1})=0.8$, $P(s_i,a_1,s_1)=0.2$, $P(s_i,a_2,s_1)=0.9$, $P(s_i,a_2,s_{i+1})=0.1$.
  - Reward: all $R(s_i, a_i, s_j) = 0.0$, except $R(s_{n-1}, a_1, s_n) = 99.9$
  - We define the goal state to be $s_n$
  - The Future discount factor: gamma $\gamma = 0.7$
- State utility values: $U(s_{n-1})$, $U(s_{n-2})$, $U(s_{n-3})$, … and $U(s_2)$, $U(s_1)$
- Optimal Policy: $\pi(s_i)=a_1$



# Reinforcement Learning

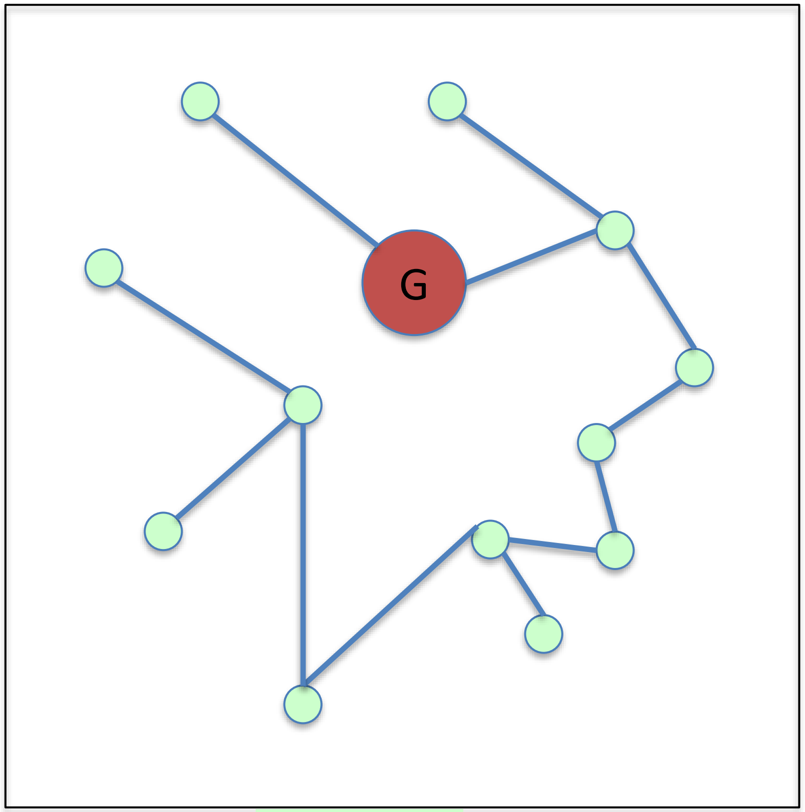
- **Given a Markov Decision Process (environment)**
  - A set of states S ✓ (known)
  - A set of actions A in each state ✓ (known)
  - A set of percepts that can be sensed ✓ (known)
  - The current state ✓ (known)
  - Transitions P(*s'*| *s*, *a*) ✕ (only known by the env)
  - Rewards R(*s*, *a*, *s'*) ✕ (only known by the env)
- **Could the agent still learn the optimal policy?**

## Goals, Rewards, Utilities, Policies

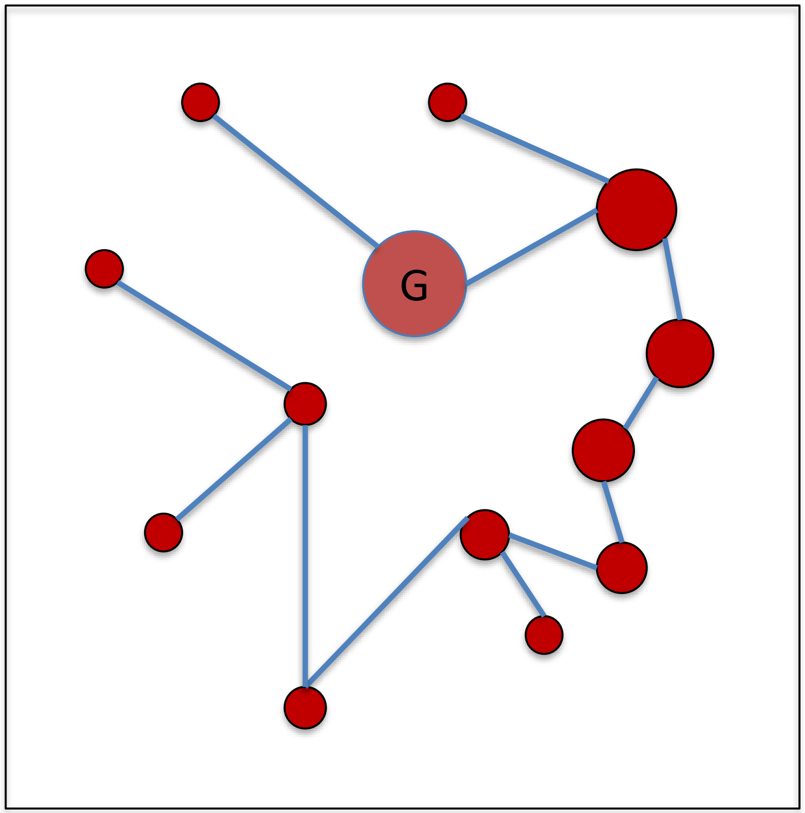
- Goals
  - Given to the agent from the problem statements
- Rewards
  - Given to the agent, designed based on the goals
- Utility values for states
  - Computed by the agent, based on the rewards
- Policies
  - Computed or learned by the agent
  - Used by the agent to select its actions
  - The better a policy, the more rewards it collects

# Reinforcement Learning

## Propagating the Delayed Rewards



Before RL



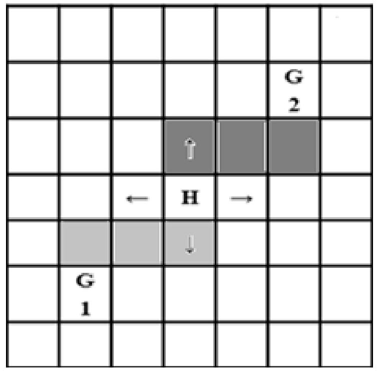
After RL

After RL, no matter where you are, you know which way to go to the Goal!
This is because all states now have utility values that will lead your agent to the goal.

# Reinforcement Learning (Key Ideas)

- Can the agent still learn the optimal policy?
  - When it knows only the states S and actions A, but not the transition model P(s,a,s’) and/or reward function R(s,a,s’)
- Try an action on a state of the environment, and get a “sample” that includes (s, a, s’, r) and maybe U:
  - A state transition (s,a)->s’
  - A reward received for the action r = R(s, a, s’)
  - And maybe the utility value of the next state U(s’)
- Objective: Try different actions in states to discover an optimal policy π(s)=a and eventually tells the agent which action will lead to the most rewarding states

# How to Explore the State Space



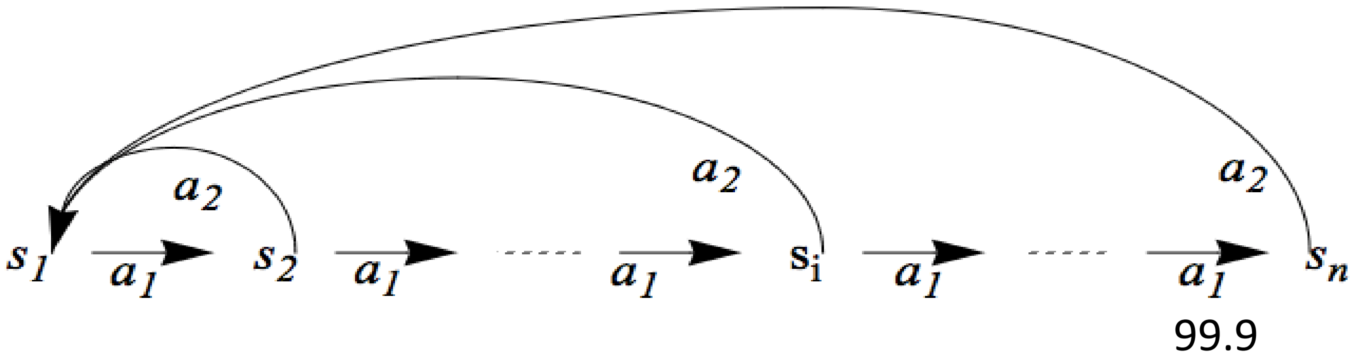
- Visit different states to discover a policy and improve it
  - Numerous strategies
  - A simple strategy is executing actions randomly
    - Initially there is no policy, but random actions eventually discover a policy
    - At every time step, act randomly with a probability (for exploration), otherwise, follow the current policy π(s)=a
      - The random action may causes unnecessary action execution when the optimal policy is already discovered
      - One solution is to lower the probability of selecting a random action over time (i.e., reduce the “temperature”)

# Reinforcement Learning

- Objective: Learn the optimal policy $\pi^*(s)=a$
- Two general classes of algorithms:
  - **Model-based RL**
    - First learn the transition model and the utility values of states, then learn the policy (e.g., policy iteration)
  - Model-free RL
    - Learn the policy without learning an explicit transition model, but by receiving “samples” from the environment
    - Three algorithms we will learn:
      - Monte Carlo
      - Temporal Difference
      - Q-Learning

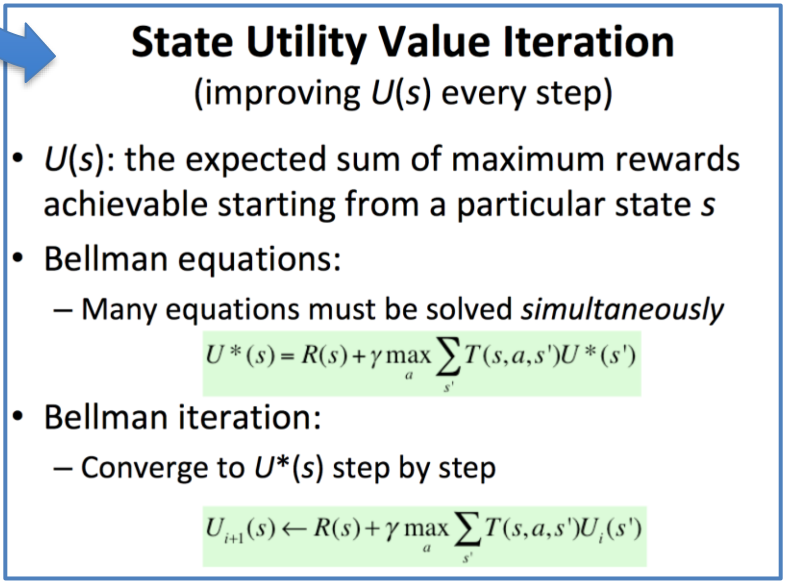
# An Example for Model-Based RL

- Given
  - States: $s_1, \ldots, s_{n,}$
  - Actions: $a_1, a_2$
  - Probabilistic transition model: (assume $s_{n+1}=s_1$)
    - $P(s_i,a_1,s_{i+1})=0.8$, $P(s_i,a_1,s_1)=0.2$, $P(s_i,a_2,s_1)=0.9$, $P(s_i,a_2,s_{i+1})=0.1$.
  - Reward: all $R(s_i, a_i, s_j) = 0.0$, except $R(s_{n-1}, a_1, s_n) = 99.9$
  - Future discount factor: $\gamma = 0.7$
- Try to learn: $U(s_{n-1})$, $U(s_{n-2})$, $U(s_{n-3})$, … and $U(s_2)$, $U(s_1)$



# Model-Based RL

- Learn an approximate model based on random actions
  - From a state s, count outcomes of s' for each (s, a)
  - Normalize to give an estimate of P(s, a, s')
  - Learn state utility U(s) from rewards R(s, a, s') when encountered
  - Learn a policy (e.g., policy iteration) and solve the MDP



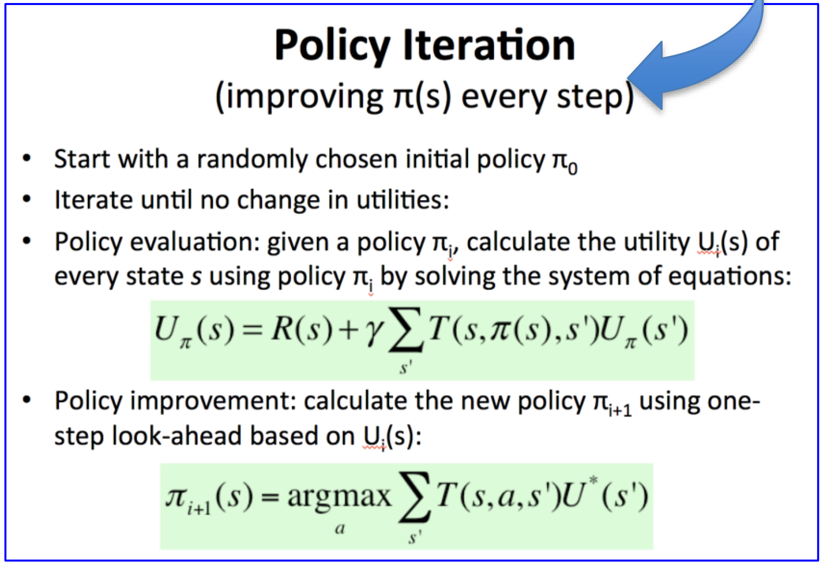
**State Utility Value Iteration**
(improving *U*(*s*) every step)

- *U*(*s*): the expected sum of maximum rewards achievable starting from a particular state *s*
- Bellman equations:
  - Many equations must be solved *simultaneously*

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

- Bellman iteration:
  - Converge to *U**(*s*) step by step

$$U_{i+1}(s) \leftarrow R(s) + \gamma \max_a \sum_{s'} T(s,a,s') U_i(s')$$



**Policy Iteration**
(improving π(s) every step)

- Start with a randomly chosen initial policy $\pi_0$
- Iterate until no change in utilities:
- Policy evaluation: given a policy $\pi_i$, calculate the utility $U_i$(s) of every state *s* using policy $\pi_i$ by solving the system of equations:

$$U_\pi(s) = R(s) + \gamma \sum_{s'} T(s,\pi(s),s') U_\pi(s')$$

- Policy improvement: calculate the new policy $\pi_{i+1}$ using one-step look-ahead based on $U_i$(s):

$$\pi_{i+1}(s) = \operatorname{argmax}_a \sum_{s'} T(s,a,s') U^*(s')$$

# State Utility Value Iteration (review)

(improving *U*(*s*) every step)

- *U*(*s*): the expected sum of maximum rewards achievable starting from a particular state *s*
- Bellman equations:
  - Many equations must be solved *simultaneously*

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

- Bellman iteration:
  - Converge to *U**(*s*) step by step

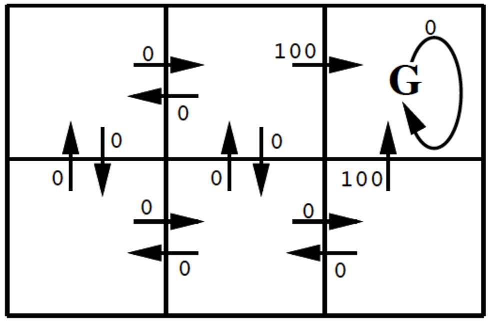
$$U_{i+1}(s) \leftarrow R(s,a) + \gamma \max_a \sum_{s'} T(s,a,s') U_i(s')$$

# Utility Value Iteration (example)

Find your way to the goal

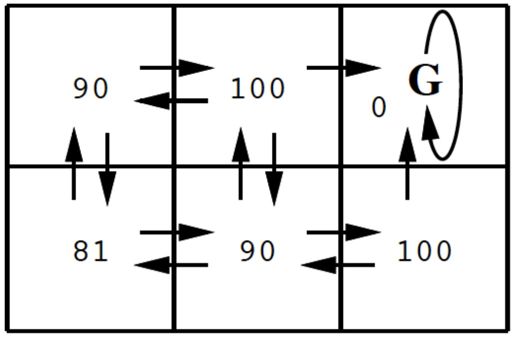
Use Bellman Iteration to compute state values from rewards

$$U_{i+1}(s) \leftarrow R(s,a) + \gamma \max_{a} \sum_{s'} T(s,a,s') U_i(s')$$

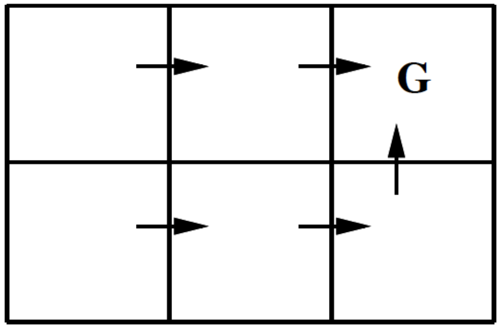


$r(s,a)$ (immediate reward) values

γ=0.9



$V^*(s)$ values



One optimal policy

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

$$= \sum_{i=0}^{\infty} \gamma^i r_{t+i} \qquad 0 \le \gamma < 1$$

# Policy Iteration (review)

(improving π(s) every step)

- Start with a randomly chosen initial policy $\pi_0$
- Iterate until no change in the utility values of the states:
- Policy evaluation: given a policy $\pi_i$, calculate the utility value $U_i(s)$ of every state *s* using policy $\pi_i$ by solving the system of equations:

$$U_\pi(s) = R(s,\pi(s),s') + \gamma \sum_{s'} T(s,\pi(s),s') U_\pi(s')$$

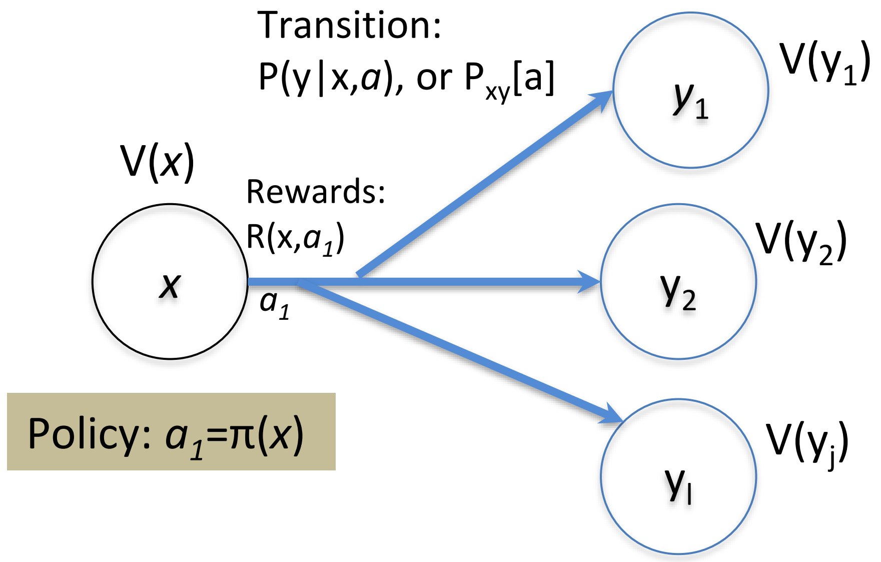
- Policy improvement: calculate the new policy $\pi_{i+1}$ using one-step look-ahead based on $U_i(s)$:

$$\pi_{i+1}(s) = \underset{a}{\operatorname{argmax}} \sum_{s'} T(s,a,s') U^*(s')$$

# Compute Utility from Rewards and Policy

to determine an optimal policy, one that maximizes the total discounted expected reward. By *discounted reward*, we mean that rewards received $m$ steps later are worth less than rewards received now by a factor of $\gamma^m (0 < \gamma < 1)$. Under a policy $\pi$, the value of state $x$ (again with respect to $R(x, a)$) is

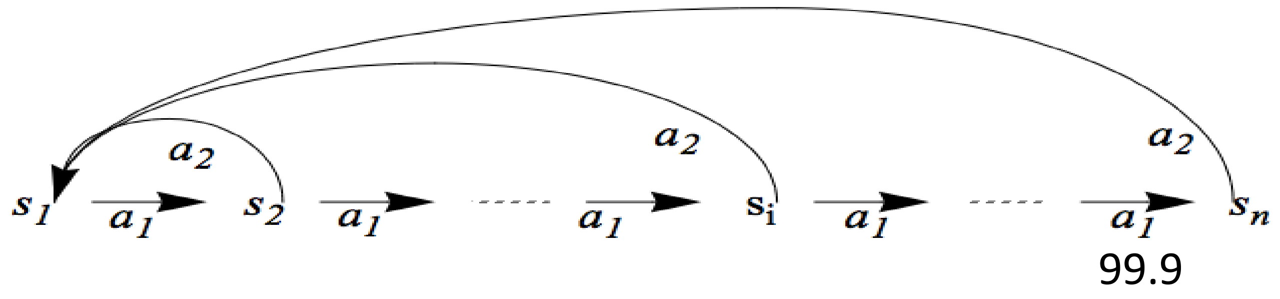
$$V^\pi(x) \equiv R(x, \pi(x)) + \gamma \sum_y P_{xy}[\pi(x)] V^\pi(y)$$



Assume the transitions are known

# An Example for Model-Based RL

- Given
  - States: $s_1, \ldots, s_n$ Actions: $a_1, a_2$
  - Probabilistic transition model: (assume $s_{n+1}=s_1$)
  - $P(s_i,a_1,s_{i+1})=0.8$, $P(s_i,a_1,s_1)=0.2$, $P(s_i,a_2,s_1)=0.9$, $P(s_i,a_2,s_{i+1})=0.1$.
  - Reward: all $R(s_i, a_i, s_j) = 0.0$, except $R(s_{n-1}, a_1, s_n) = 99.9$
  - Future discount factor: $\gamma = 0.7$, all Initial utility values are: $U(s_{i-})=0$
- 1st iteration
  - $U(s_{n-1})=0.7\max\{0.8(99.9+0)+0.2(0+0), \ldots, \ldots\} = 55.944$; or
  - $U(s_{n-1})=\max\{0.8(99.9+0.7*0)+0.2(0+0), \ldots, \ldots\} = 79.92$;
  - $U(s_{n-2})=0$, $U(s_{n-3})=0$, …, $U(s_1)=0$
- After many iterations: $U(s_1)$ = ? Is it $99.9*(0.7*0.8)^{n-1}$ ?



$$U_{t+1}(s) \leftarrow \gamma \max_a \sum_{s'} P(s,a,s')[R(s,a,s') + U_t(s')]$$

$$U_{t+1}(s) \leftarrow \max_a \sum_{s'} P(s,a,s')[R(s,a,s') + \gamma U_t(s')]$$

## Model-Based RL

- Learn an approximate model based on random actions
  - From a state s, count outcomes of s’ for each (s, a)
  - Normalize to give an estimate of P(s, a, s’)
  - Learn state utility U(s) from rewards R(s, a, s’) when encountered
  - Solve the learned MDP and obtain a policy (e.g. policy iteration)

Pros:

If and when the goal changes, one can use the learned model P(s, a, s’) to recalculate U(s) and compute a new policy offline

Cons:

Larger representation than model-free RL

# Model-Free Reinforcement Learning

- Objective: Learn the optimal policy $\pi^*(s)=a$
- Model-based RL
  - First learn the transition model and the utility values of states, then learn the policy (e.g., policy iteration)
- **Model-free RL**
  - Learn the policy without learning an explicit model, but by receiving "samples" from the environment
  - Three algorithms we will learn:
    - Monte Carlo
    - Temporal Difference
    - Q-Learning

# Monte Carlo RL (model free)

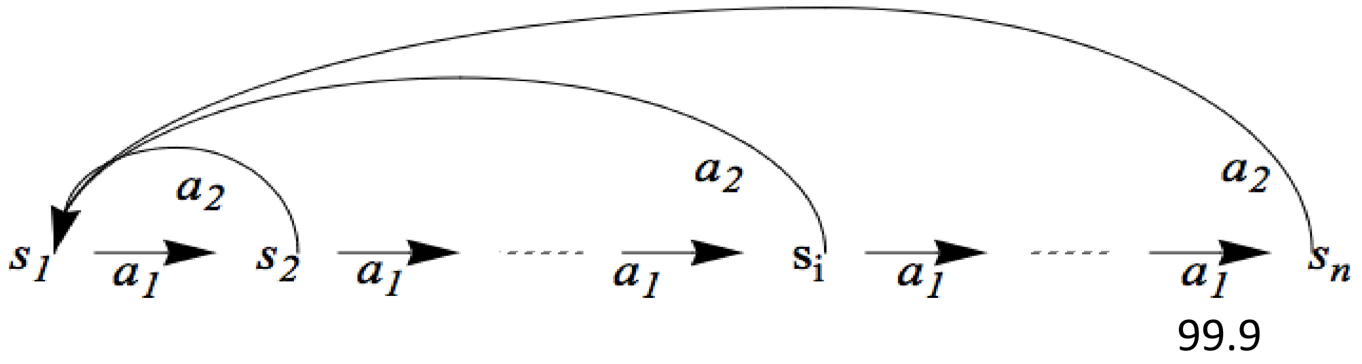
- Generate a fixed policy $\pi(s) = a$ based on $U(s)$
    - In each episode, the learner follows the policy starting at a random state
    - Average the sampled returns originating from each state
    - Generate a policy based on U(s) as in "policy iteration"
- Cons:
    - The utility value of states U(s) is given and fixed (not learned)
    - Works only in episodic problems
    - Takes a very long time to converge as learning is from complete sample returns
    - Wastes information as it figures out state values in isolation from other states

# Temporal Difference RL (model free)

- Improve Monte Carlo, still using a fixed policy
  - Update U(s) using each sample (s, a, s', r) received from the environment and each sample includes:
    1. A state transition (s,a)->s' done by the environment
    2. A reward received for the action r = R(s, a, s')
    3. And the value of the next state U(s')
  - That is, Sample = r + γU(s'), where γ = future discount
- **TD Algorithm:**
  - U(s) ← (1-α)U(s) + α*Sample = (1-α)U(s) + α[r + γU(s')]
  - Where α is the "learning rate" (how much you trust the sample)
  - You might decrease α over time when converges to the average
  - The parameter α also represents "forgetting long past values"
- Cons:
  - Only provides the utility values of states, not improve policy directly

# An Example for TD-RL

- Given
  - States: $s_1, \ldots, s_{n,}$
  - Actions: $a_1, a_2$
  - Probabilistic transition model: (assume $s_{n+1}=s_1$)
    - $P(s_i,a_1,s_{i+1})=0.8$, $P(s_i,a_1,s_1)=0.2$, $P(s_i,a_2,s_1)=0.9$, $P(s_i,a_2,s_{i+1})=0.1$.
  - Reward: all $R(s_i, a_i, s_j) = 0.0$, except $R(s_{n-1}, a_1, s_n) = 99.9$
  - We define the goal state to be $s_n$
  - The learning rate $\alpha = 0.4$
  - The Future discount factor: $\gamma = 0.7$
- Use TD algorithm to learn: $U(s_{n-1})$, $U(s_{n-2})$, $U(s_{n-3})$, … and $U(s_2)$, $U(s_1)$



## Q-Learning (Definition)

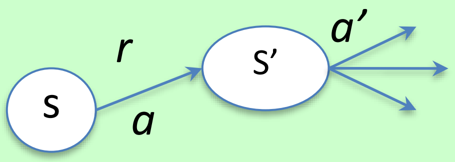
- The key idea is to use *Q*(*s*,*a*) to represent the value of taking an action *a* in state s, rather than only the value of state U(s):
  - Define: $Q(s,a) \equiv \sum_{s'} P(s,a,s')[R(s,a,s') + \gamma U(s')]$
- Optimal π(s) = argmax $U^{\pi}$ (s) = argmax$_a$ Q(s, a)

$$Q_{k+1}(s,a) \leftarrow \sum_{s'} P(s,a,s')[R(s,a,s') + \gamma \max_{a'} Q_k(s',a')]$$

  - The last term using Q to replace U

# Q-Learning (Algorithm)

- Initially, let *Q*(s, *a*)=0, given α and γ
- Sample a transition and reward: (s,a,s',r)
  - Sample = r + γ $max_{a'}$ Q(s',*a*')
  - Q(s,a) = (1-α)Q(s,a) + α*Sample



$$Q_{t+1}(s,a) = (1-\alpha)Q_t(s,a) + \alpha[R(s,a,s') + \gamma \max_{a'} Q_t(s',a')]$$

Advantages:

1. No need for a fixed policy, can do off-policy learning, or directly learns a policy
2. The optimal policy π(s)=a is to select the action *a* that has the best *Q(s, a)*
3. Provably convergent to the optimal policy when t -> infinite

# The Q-Learning Algorithm

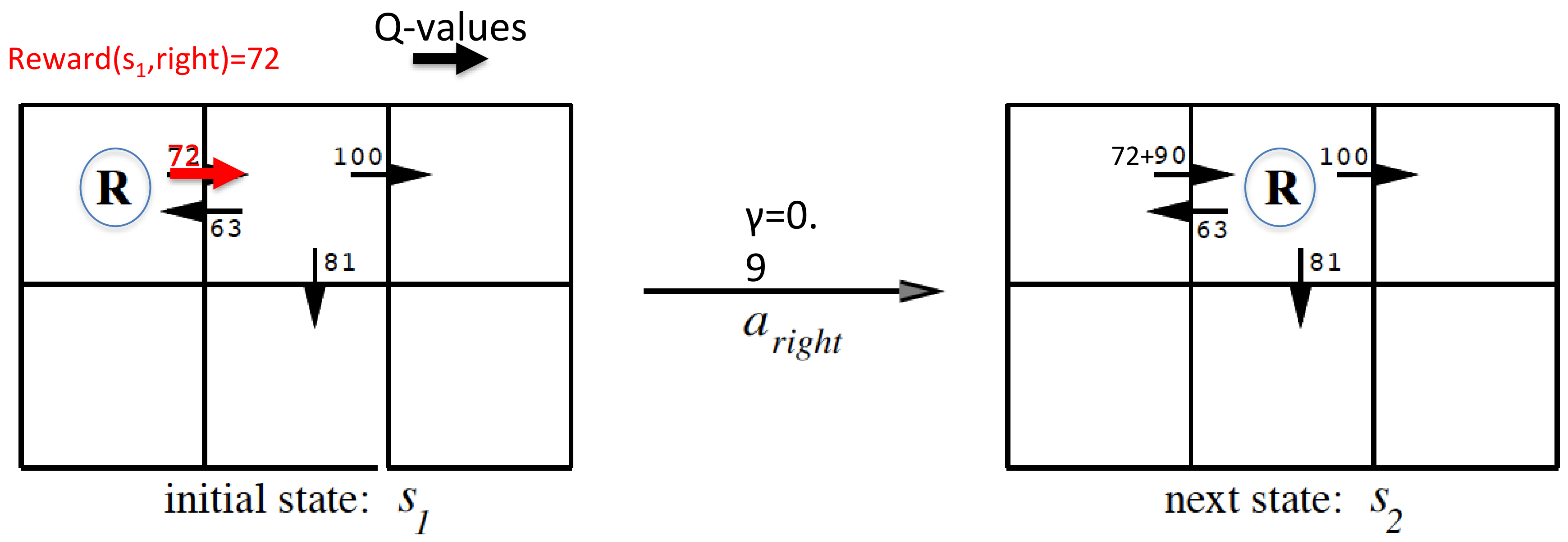
```
Initialize all $Q(s,a)$ arbitrarily
For all episodes
    Initalize $s$
    Repeat
        Choose $a$ using policy derived from $Q$, e.g., $\epsilon$-greedy
        Take action $a$, observe $r$ and $s'$
        Update $Q(s,a)$:
            $Q(s,a) \leftarrow Q(s,a) + \eta(r + \gamma \boxed{\max_{a'}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

# Q-Learning (in One Formula)

$$Q(s_t,a_t) \leftarrow (1-\alpha)\cdot \underbrace{Q(s_t,a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \left( \overbrace{\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1},a)}_{\text{estimate of optimal future value}}}^{\text{learned value}} \right)$$

$\boxed{r(s)\ \text{ or }\ r(s,a)}$

# Q-Learning example from reward r(s,a)



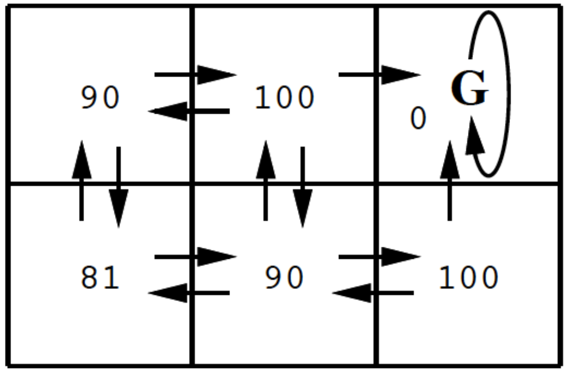
$$\hat{Q}(s_1, a_{right}) \leftarrow r + \gamma \max_{a'} \hat{Q}(s_2, a')$$ // when α = 1

72+ 0.9 max{63,81,100}

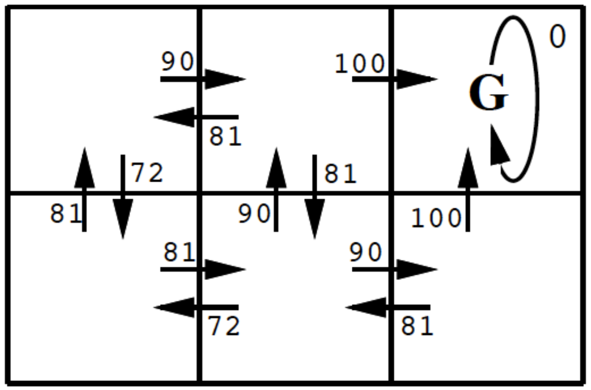
72+ 90

Back-propagate Q-values 1-step backwards

# Q-Learning from state utility values



$V^*(s)$ values

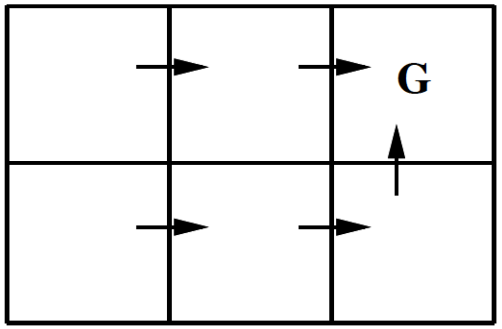


$Q(s,a)$ values

Notice these

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

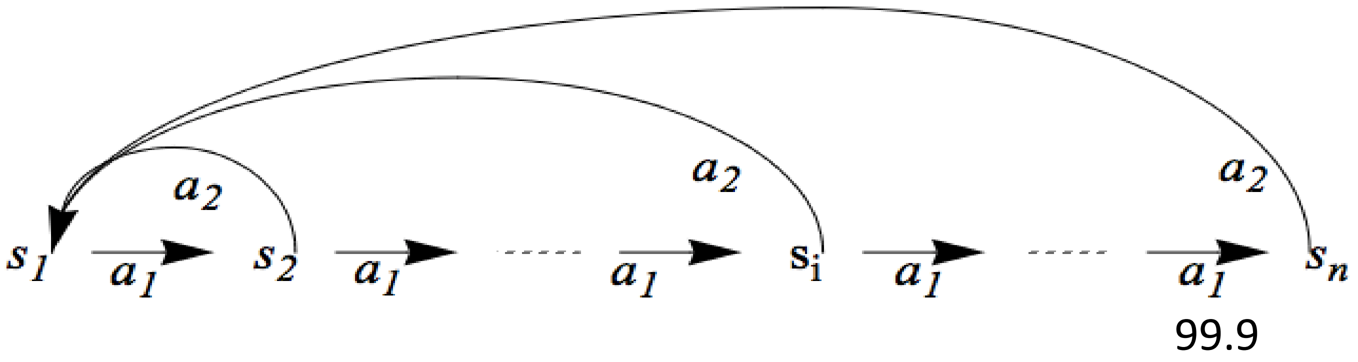
$$V^*(s) = \max_a Q(s,a)$$



One optimal policy

# An Example for Q-Learning

- Given
  - States: $s_1, \ldots, s_{n,}$
  - Actions: $a_1, a_2$
  - Probabilistic transition model: (assume $s_{n+1}=s_1$)
    - $P(s_i,a_1,s_{i+1})=0.8$, $P(s_i,a_1,s_1)=0.2$, $P(s_i,a_2,s_1)=0.9$, $P(s_i,a_2,s_{i+1})=0.1$.
  - Reward: all $R(s_i, a_i, s_j) = 0.0$, except $R(s_{n-1}, a_1, s_n) = 99.9$
  - We define the goal state to be $s_n$
  - The learning rate $\alpha = 0.4$
  - The Future discount factor: $\gamma = 0.7$
- Try to learn: $Q(s_{n-1},a_1)$, $Q(s_{n-1},a_2)$, $Q(s_{n-2},a_1)$, and $Q(s_{n-2},a_2)$=?



# An Example for Q-Learning

Initially: all Q($s_i$, $a_j$) = 0.0

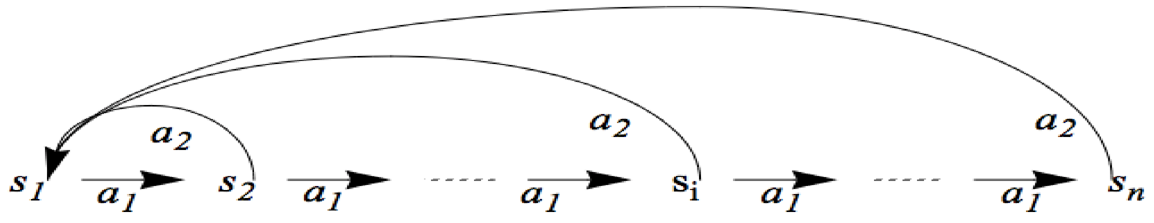
Assume you got two samples at the state $s_{n-1}$ for action $a_1$

- $s_{n-1}$,$a_1$, $s_n$, r=R($s_{n-1}$,$a_1$, $s_n$)=99.9, // P($s_{n-1}$,$a_1$)->? $s_n$
- $s_{n-1}$,$a_1$, $s_1$, r=R($s_{n-1}$,$a_1$, $s_1$)=0.0, // P($s_{n-1}$,$a_1$)->? $s_1$

Updated Q value, if P(s,a) are known:

- – Q($s_{n-1}$,$a_1$)= 0.8*[99.9 + 0.7*0.0] + 0.2[0.0+0.7*0.0] = 79.9

- If P(s,a) are unknown, assume $s_{n-1}$ was visited 2 times so far:
  - – sample$_1$=99.9+.7*0; Q($s_{n-1}$,$a_1$)= (1-0.4)*0.0 + 0.4*99.9 = 39.96
  - – sample$_2$=0+.7*0; Q($s_{n-1}$,$a_1$)= (1-0.4)*39.96 + 0.4*0.0 = 23.976



99.9

# Discussions of RL (1)

- Pros:
  - Easy to use in problems where the data can be mapped to states easily
  - Guaranteed to find the optimal policy given enough time even with suboptimal actions
- Cons:
  - States of the environment are not always known
  - Computation time is intractable for large or continuous state spaces
    - E.g. if each cell in a grid world is a state, then state space grows exponentially with number of rows and columns (states = map size = m*n)
  - Cannot handle raw data, must use an approximation/reduction function
    - Designing approximation functions to disambiguate similar states requires human intelligence or an alternate learning technique
      - E.g. use the relative distance (states = m*n) between two agents in a hunter-prey problem as opposed to their cell coordinates (states = m*n*m*n)
  - Model-free RL cannot transfer the learned knowledge when the goal changes
    - Forgetting a learned policy is much more difficult (hysteresis), quicker to start from scratch

# Discussions of RL (2)

Utility function is unknown at the beginning

- When agent visits each state, it receives a reward
    - Possibly negative

What function should it learn?

- $R(s)$: Utility-based agent
    - If it already knows the transition model
    - Then use MDP algorithm to solve for MEU actions
- $U(s,a)$: Q-learning agent
    - If it doesn’t already know the transition model
    - Then pick action that has highest $U$ in current state
- $\pi^*(s)$: reflex agent
    - Learn a policy directly, then pick the action that the policy says

## Q Learning Summary

Modify Bellman equation to learn *Q* values of actions

- $U(s) = R(s)+\gamma \max_a \sum_{s1}(P(s_1|s,a)U(s_1))$
    - We don't know *R* or *P*
    - But when we perform *a'* in *s*, we move to *s*' and receive *R*(*s*)
    - We want *Q*(*s*,*a*) = Expected utility of performing *a* in state *s*

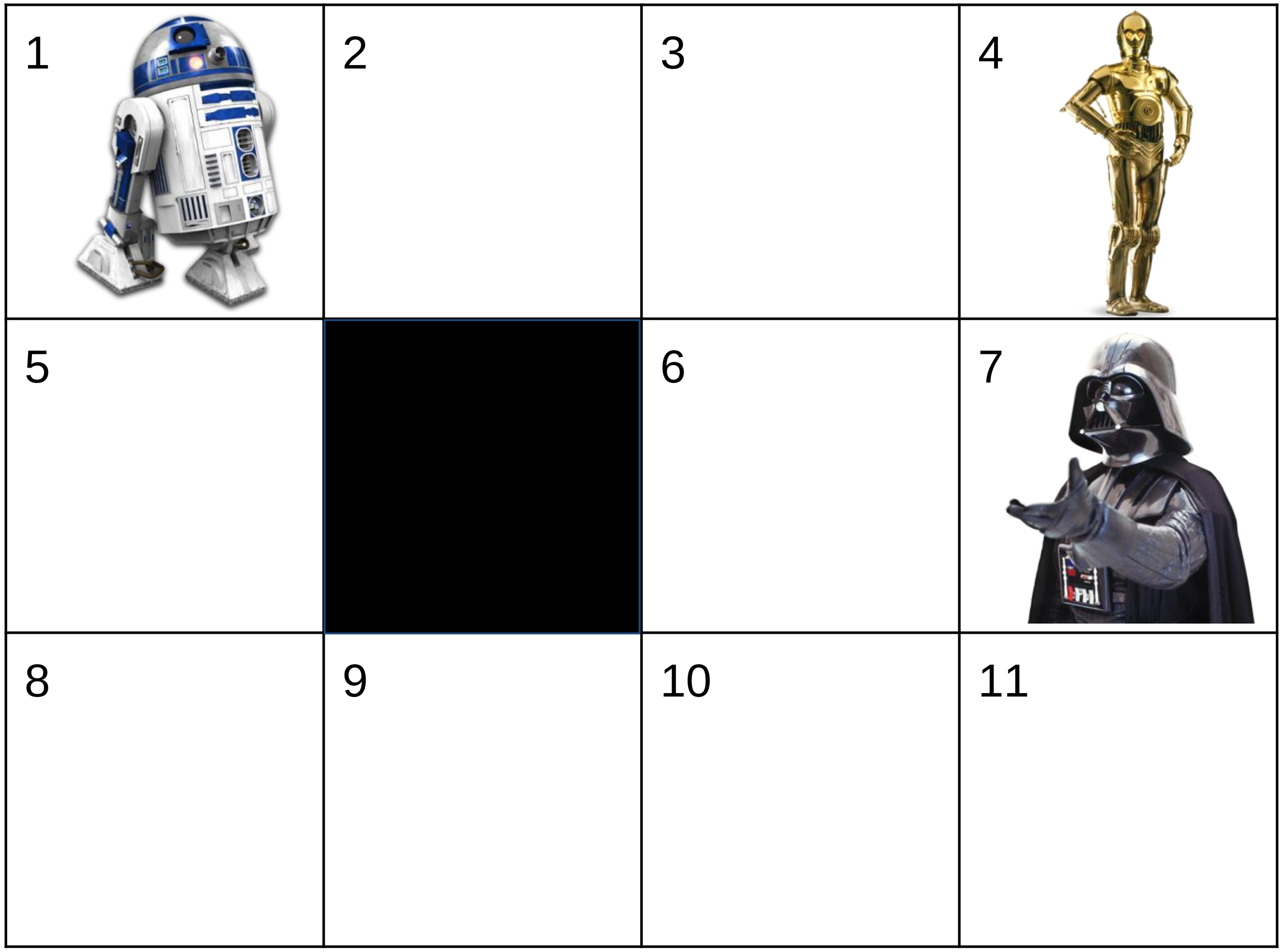
Update *Q* after each step

- If we get a good reward now, then increase *Q*
- If we later get to a state with high *Q*, then increase *Q* here too
- $Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha(R(s)+\gamma \max_{a'} Q(s',a'))$
    - $\alpha$ is the learning rate, $\gamma$ is the discount factor

Converges to correct values if $\alpha$ decays over time

- Similar to the "temperature" in simulated annealing

# Q-Learning in Star War Environment



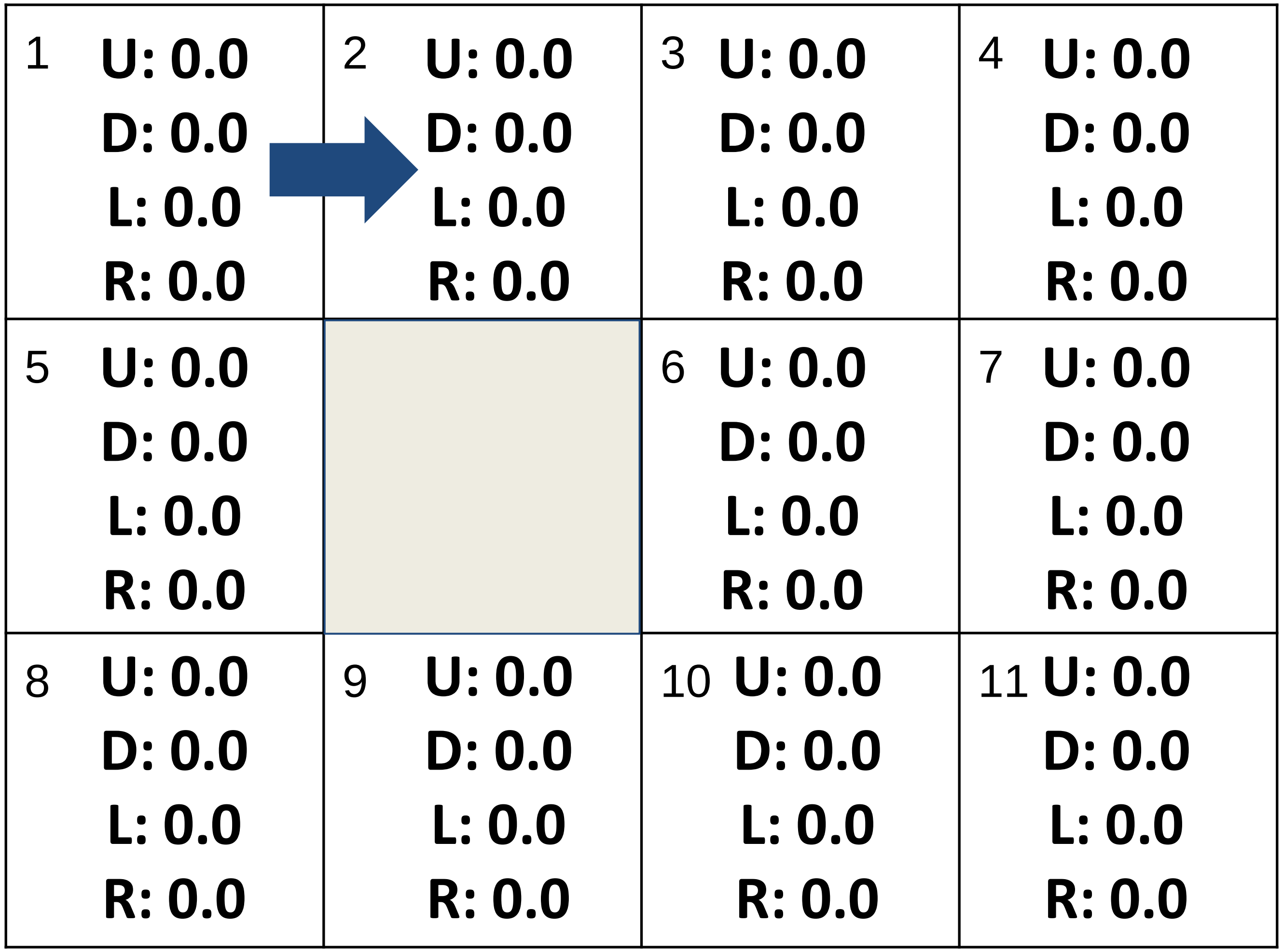
## Initial *Q* Values

$\alpha = 0.1$ $\gamma = 1$

| | | | |
|---|---|---|---|
| 1 **U: 0.0 D: 0.0 L: 0.0 R: 0.0** | 2 **U: 0.0 D: 0.0 L: 0.0 R: 0.0** | 3 **U: 0.0 D: 0.0 L: 0.0 R: 0.0** | 4 **U: 0.0 D: 0.0 L: 0.0 R: 0.0** |
| 5 **U: 0.0 D: 0.0 L: 0.0 R: 0.0** | ▨ | 6 **U: 0.0 D: 0.0 L: 0.0 R: 0.0** | 7 **U: 0.0 D: 0.0 L: 0.0 R: 0.0** |
| 8 **U: 0.0 D: 0.0 L: 0.0 R: 0.0** | 9 **U: 0.0 D: 0.0 L: 0.0 R: 0.0** | 10 **U: 0.0 D: 0.0 L: 0.0 R: 0.0** | 11 **U: 0.0 D: 0.0 L: 0.0 R: 0.0** |

## Step 1

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha(R(s)+\gamma \max_{a'} Q(s',a'))$$



| | | | |
|---|---|---|---|
| 1 U: 0.0 D: 0.0 L: 0.0 R: 0.0 | 2 U: 0.0 D: 0.0 L: 0.0 R: 0.0 | 3 U: 0.0 D: 0.0 L: 0.0 R: 0.0 | 4 U: 0.0 D: 0.0 L: 0.0 R: 0.0 |
| 5 U: 0.0 D: 0.0 L: 0.0 R: 0.0 | | 6 U: 0.0 D: 0.0 L: 0.0 R: 0.0 | 7 U: 0.0 D: 0.0 L: 0.0 R: 0.0 |
| 8 U: 0.0 D: 0.0 L: 0.0 R: 0.0 | 9 U: 0.0 D: 0.0 L: 0.0 R: 0.0 | 10 U: 0.0 D: 0.0 L: 0.0 R: 0.0 | 11 U: 0.0 D: 0.0 L: 0.0 R: 0.0 |

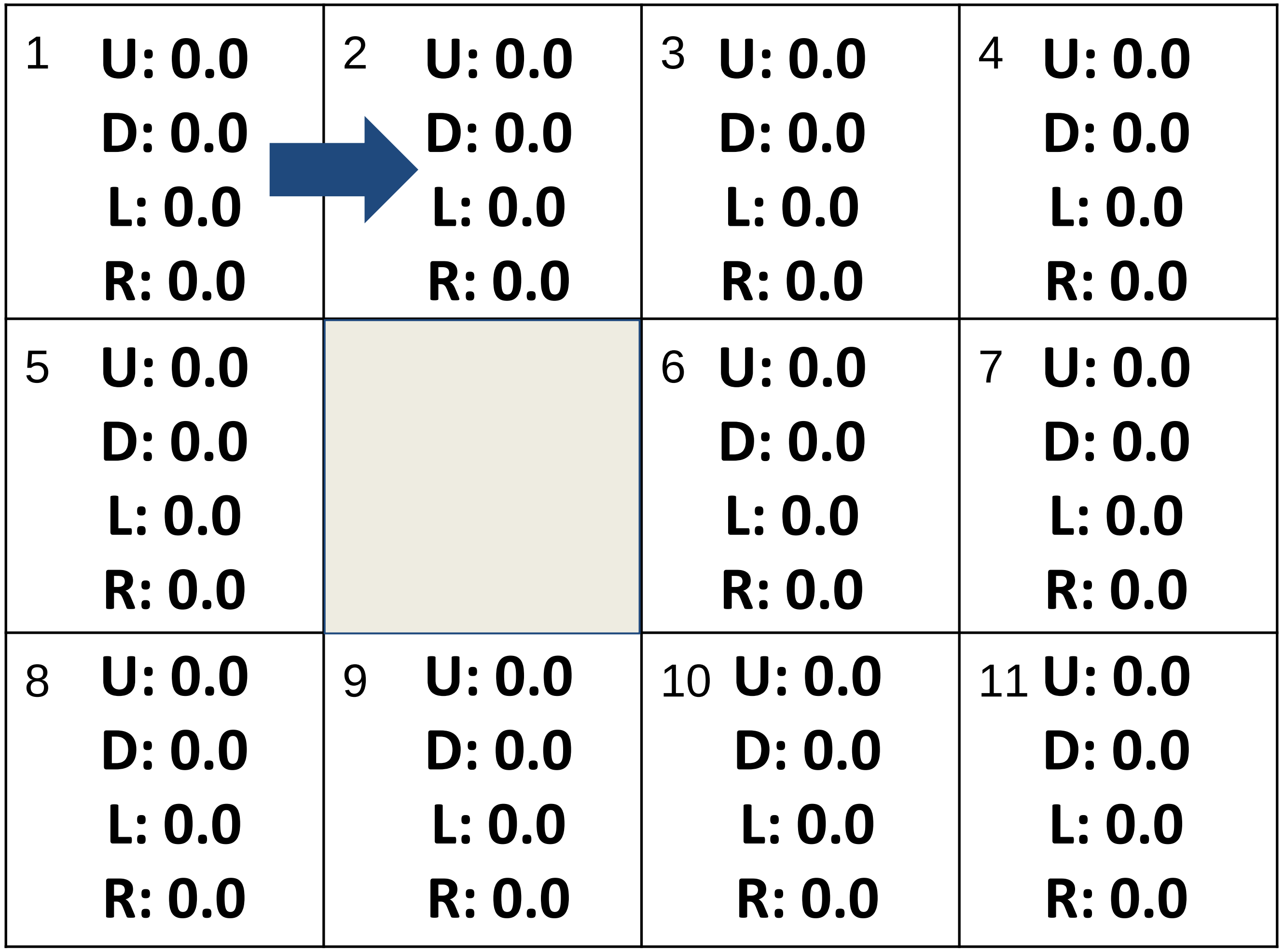
## Step 1

$$Q(1,\mathrm{R}) \leftarrow (1-\alpha)Q(1,\mathrm{R}) + \alpha(R(1)+\gamma \max_{a'} Q(2,a'))$$

| | | | |
|---|---|---|---|
| 1 U: 0.0<br>D: 0.0<br>L: 0.0<br>R: 0.0 | 2 U: 0.0<br>D: 0.0<br>L: 0.0<br>R: 0.0 | 3 U: 0.0<br>D: 0.0<br>L: 0.0<br>R: 0.0 | 4 U: 0.0<br>D: 0.0<br>L: 0.0<br>R: 0.0 |
| 5 U: 0.0<br>D: 0.0<br>L: 0.0<br>R: 0.0 | ▨ | 6 U: 0.0<br>D: 0.0<br>L: 0.0<br>R: 0.0 | 7 U: 0.0<br>D: 0.0<br>L: 0.0<br>R: 0.0 |
| 8 U: 0.0<br>D: 0.0<br>L: 0.0<br>R: 0.0 | 9 U: 0.0<br>D: 0.0<br>L: 0.0<br>R: 0.0 | 10 U: 0.0<br>D: 0.0<br>L: 0.0<br>R: 0.0 | 11 U: 0.0<br>D: 0.0<br>L: 0.0<br>R: 0.0 |

## Step 1

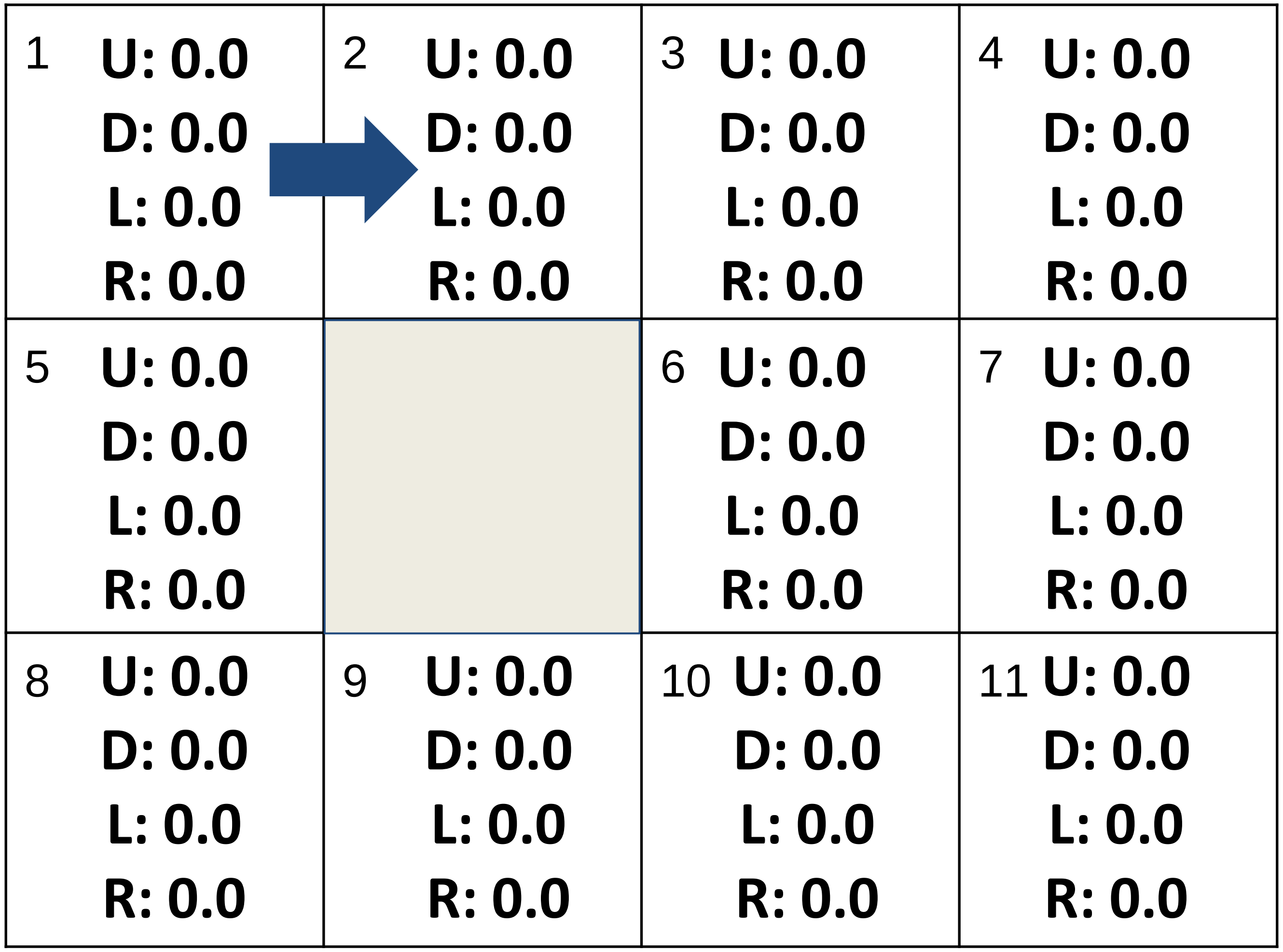
$$Q(1,R) \leftarrow (1-\alpha)Q(1,R) + \alpha(-0.04+\gamma \max_{a'} Q(2,a'))$$



| | | | |
|---|---|---|---|
| 1 U: 0.0<br>D: 0.0<br>L: 0.0<br>R: 0.0 | 2 U: 0.0<br>D: 0.0<br>L: 0.0<br>R: 0.0 | 3 U: 0.0<br>D: 0.0<br>L: 0.0<br>R: 0.0 | 4 U: 0.0<br>D: 0.0<br>L: 0.0<br>R: 0.0 |
| 5 U: 0.0<br>D: 0.0<br>L: 0.0<br>R: 0.0 | ▨ | 6 U: 0.0<br>D: 0.0<br>L: 0.0<br>R: 0.0 | 7 U: 0.0<br>D: 0.0<br>L: 0.0<br>R: 0.0 |
| 8 U: 0.0<br>D: 0.0<br>L: 0.0<br>R: 0.0 | 9 U: 0.0<br>D: 0.0<br>L: 0.0<br>R: 0.0 | 10 U: 0.0<br>D: 0.0<br>L: 0.0<br>R: 0.0 | 11 U: 0.0<br>D: 0.0<br>L: 0.0<br>R: 0.0 |

## Step 1

$$Q(1,\mathrm{R}) \leftarrow (1-\alpha)Q(1,\mathrm{R}) + \alpha(-0.04+\gamma\ 0.0)$$



| | | | |
|---|---|---|---|
| 1 U: 0.0 D: 0.0 L: 0.0 R: 0.0 | 2 U: 0.0 D: 0.0 L: 0.0 R: 0.0 | 3 U: 0.0 D: 0.0 L: 0.0 R: 0.0 | 4 U: 0.0 D: 0.0 L: 0.0 R: 0.0 |
| 5 U: 0.0 D: 0.0 L: 0.0 R: 0.0 | ▨ | 6 U: 0.0 D: 0.0 L: 0.0 R: 0.0 | 7 U: 0.0 D: 0.0 L: 0.0 R: 0.0 |
| 8 U: 0.0 D: 0.0 L: 0.0 R: 0.0 | 9 U: 0.0 D: 0.0 L: 0.0 R: 0.0 | 10 U: 0.0 D: 0.0 L: 0.0 R: 0.0 | 11 U: 0.0 D: 0.0 L: 0.0 R: 0.0 |

## Step 1

$$Q(1,R) \leftarrow (1-\alpha)0.0 + \alpha(-0.04+\gamma\ 0.0)$$

| | | | |
|---|---|---|---|
| 1 **U: 0.0**<br>**D: 0.0**<br>**L: 0.0**<br>**R: 0.0** | 2 **U: 0.0**<br>**D: 0.0**<br>**L: 0.0**<br>**R: 0.0** | 3 **U: 0.0**<br>**D: 0.0**<br>**L: 0.0**<br>**R: 0.0** | 4 **U: 0.0**<br>**D: 0.0**<br>**L: 0.0**<br>**R: 0.0** |
| 5 **U: 0.0**<br>**D: 0.0**<br>**L: 0.0**<br>**R: 0.0** | | 6 **U: 0.0**<br>**D: 0.0**<br>**L: 0.0**<br>**R: 0.0** | 7 **U: 0.0**<br>**D: 0.0**<br>**L: 0.0**<br>**R: 0.0** |
| 8 **U: 0.0**<br>**D: 0.0**<br>**L: 0.0**<br>**R: 0.0** | 9 **U: 0.0**<br>**D: 0.0**<br>**L: 0.0**<br>**R: 0.0** | 10 **U: 0.0**<br>**D: 0.0**<br>**L: 0.0**<br>**R: 0.0** | 11 **U: 0.0**<br>**D: 0.0**<br>**L: 0.0**<br>**R: 0.0** |

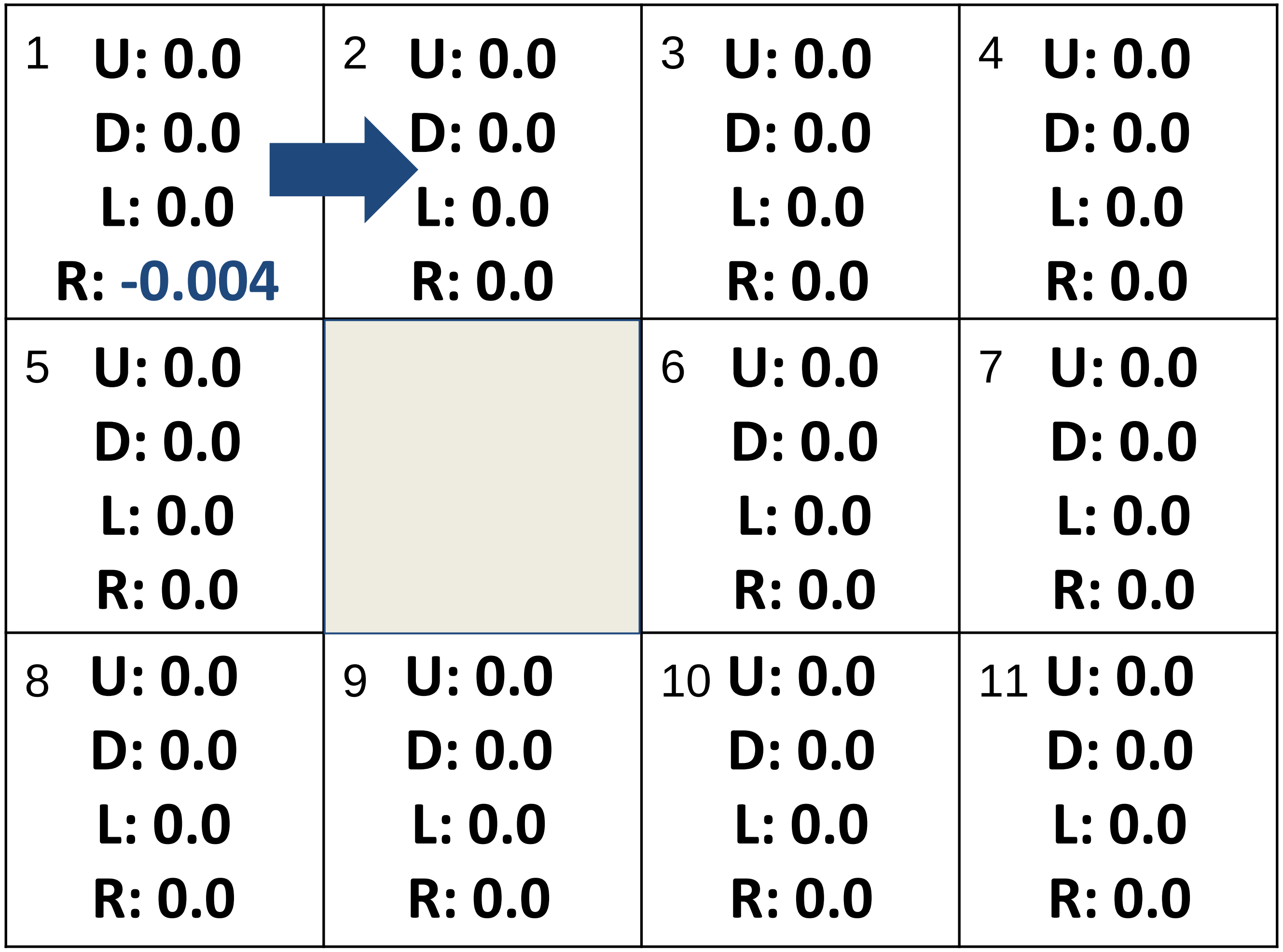
## Step 1

$Q(1,R) \leftarrow 0.1(-0.04+0.0)$

| | | | |
|---|---|---|---|
| 1 U: 0.0<br>D: 0.0<br>L: 0.0<br>R: 0.0 | 2 U: 0.0<br>D: 0.0<br>L: 0.0<br>R: 0.0 | 3 U: 0.0<br>D: 0.0<br>L: 0.0<br>R: 0.0 | 4 U: 0.0<br>D: 0.0<br>L: 0.0<br>R: 0.0 |
| 5 U: 0.0<br>D: 0.0<br>L: 0.0<br>R: 0.0 | ▨ | 6 U: 0.0<br>D: 0.0<br>L: 0.0<br>R: 0.0 | 7 U: 0.0<br>D: 0.0<br>L: 0.0<br>R: 0.0 |
| 8 U: 0.0<br>D: 0.0<br>L: 0.0<br>R: 0.0 | 9 U: 0.0<br>D: 0.0<br>L: 0.0<br>R: 0.0 | 10 U: 0.0<br>D: 0.0<br>L: 0.0<br>R: 0.0 | 11 U: 0.0<br>D: 0.0<br>L: 0.0<br>R: 0.0 |

## Step 1

$Q(1,\text{R}) \leftarrow -0.004$



| | | | |
|---|---|---|---|
| 1 U: 0.0 D: 0.0 L: 0.0 R: -0.004 | 2 U: 0.0 D: 0.0 L: 0.0 R: 0.0 | 3 U: 0.0 D: 0.0 L: 0.0 R: 0.0 | 4 U: 0.0 D: 0.0 L: 0.0 R: 0.0 |
| 5 U: 0.0 D: 0.0 L: 0.0 R: 0.0 | ▨ | 6 U: 0.0 D: 0.0 L: 0.0 R: 0.0 | 7 U: 0.0 D: 0.0 L: 0.0 R: 0.0 |
| 8 U: 0.0 D: 0.0 L: 0.0 R: 0.0 | 9 U: 0.0 D: 0.0 L: 0.0 R: 0.0 | 10 U: 0.0 D: 0.0 L: 0.0 R: 0.0 | 11 U: 0.0 D: 0.0 L: 0.0 R: 0.0 |

## Step 2

$$Q(2,\mathrm{R}) \leftarrow (1-\alpha)Q(2,\mathrm{R}) + \alpha(R(2)+\gamma \max_{a'} Q(3,a'))$$

| | | | |
|---|---|---|---|
| 1 U: 0.0 D: 0.0 L: 0.0 R: -0.004 | 2 U: 0.0 D: 0.0 L: 0.0 R: -0.004 | 3 U: 0.0 D: 0.0 L: 0.0 R: 0.0 | 4 U: 0.0 D: 0.0 L: 0.0 R: 0.0 |
| 5 U: 0.0 D: 0.0 L: 0.0 R: 0.0 | | 6 U: 0.0 D: 0.0 L: 0.0 R: 0.0 | 7 U: 0.0 D: 0.0 L: 0.0 R: 0.0 |
| 8 U: 0.0 D: 0.0 L: 0.0 R: 0.0 | 9 U: 0.0 D: 0.0 L: 0.0 R: 0.0 | 10 U: 0.0 D: 0.0 L: 0.0 R: 0.0 | 11 U: 0.0 D: 0.0 L: 0.0 R: 0.0 |

## Step 3

$$Q(3,R) \leftarrow (1-\alpha)Q(3,R) + \alpha(R(3)+\gamma \max_{a'} Q(4,a'))$$

| | | | |
|---|---|---|---|
| 1 U: 0.0<br>D: 0.0<br>L: 0.0<br>R: -0.004 | 2 U: 0.0<br>D: 0.0<br>L: 0.0<br>R: -0.004 | 3 U: 0.0<br>D: 0.0<br>L: 0.0<br>R: -0.004 | 4 U: 0.0<br>D: 0.0<br>L: 0.0<br>R: 0.0 |
| 5 U: 0.0<br>D: 0.0<br>L: 0.0<br>R: 0.0 | | 6 U: 0.0<br>D: 0.0<br>L: 0.0<br>R: 0.0 | 7 U: 0.0<br>D: 0.0<br>L: 0.0<br>R: 0.0 |
| 8 U: 0.0<br>D: 0.0<br>L: 0.0<br>R: 0.0 | 9 U: 0.0<br>D: 0.0<br>L: 0.0<br>R: 0.0 | 10 U: 0.0<br>D: 0.0<br>L: 0.0<br>R: 0.0 | 11 U: 0.0<br>D: 0.0<br>L: 0.0<br>R: 0.0 |

## Step 4

$Q(4,*) \leftarrow 1$

| | | | |
|---|---|---|---|
| 1 U: 0.0<br>D: 0.0<br>L: 0.0<br>R: -0.004 | 2 U: 0.0<br>D: 0.0<br>L: 0.0<br>R: -0.004 | 3 U: 0.0<br>D: 0.0<br>L: 0.0<br>R: -0.004 | 4<br>**+1** |
| 5 U: 0.0<br>D: 0.0<br>L: 0.0<br>R: 0.0 | | 6 U: 0.0<br>D: 0.0<br>L: 0.0<br>R: 0.0 | 7 U: 0.0<br>D: 0.0<br>L: 0.0<br>R: 0.0 |
| 8 U: 0.0<br>D: 0.0<br>L: 0.0<br>R: 0.0 | 9 U: 0.0<br>D: 0.0<br>L: 0.0<br>R: 0.0 | 10 U: 0.0<br>D: 0.0<br>L: 0.0<br>R: 0.0 | 11 U: 0.0<br>D: 0.0<br>L: 0.0<br>R: 0.0 |

## Step 5

$$Q(1,\mathrm{R}) \leftarrow (1-\alpha)Q(1,\mathrm{R}) + \alpha(R(s)+\gamma \max_{a'} Q(2,a'))$$

| | | | |
|---|---|---|---|
| 1 **U: 0.0**<br>**D: 0.0**<br>**L: 0.0**<br>**R: -0.004** | 2 **U: 0.0**<br>**D: 0.0**<br>**L: 0.0**<br>**R: -0.004** | 3 **U: 0.0**<br>**D: 0.0**<br>**L: 0.0**<br>**R: -0.004** | 4<br>**+1** |
| 5 **U: 0.0**<br>**D: 0.0**<br>**L: 0.0**<br>**R: 0.0** | | 6 **U: 0.0**<br>**D: 0.0**<br>**L: 0.0**<br>**R: 0.0** | 7 **U: 0.0**<br>**D: 0.0**<br>**L: 0.0**<br>**R: 0.0** |
| 8 **U: 0.0**<br>**D: 0.0**<br>**L: 0.0**<br>**R: 0.0** | 9 **U: 0.0**<br>**D: 0.0**<br>**L: 0.0**<br>**R: 0.0** | 10 **U: 0.0**<br>**D: 0.0**<br>**L: 0.0**<br>**R: 0.0** | 11 **U: 0.0**<br>**D: 0.0**<br>**L: 0.0**<br>**R: 0.0** |

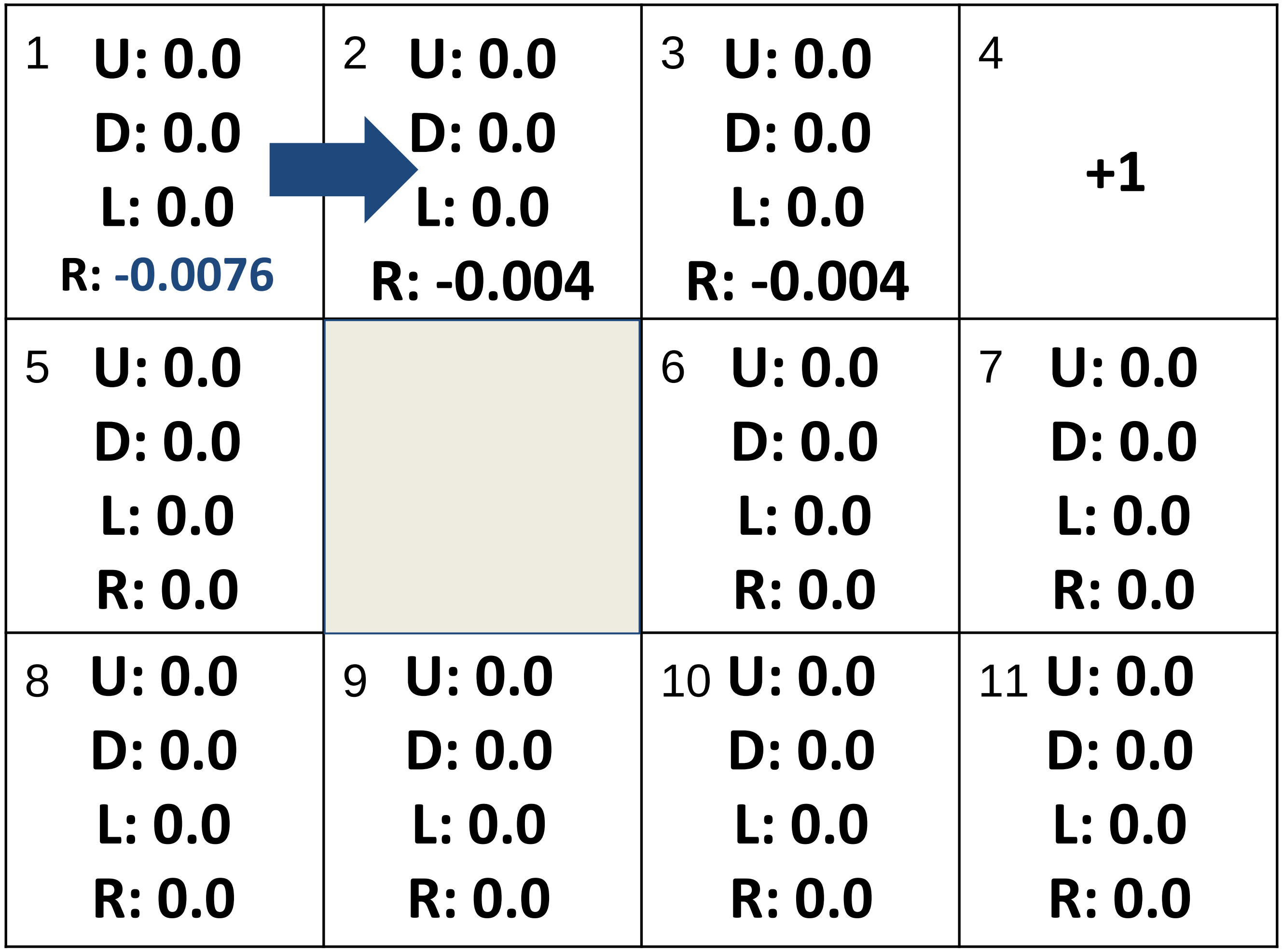
## Step 5

$$Q(1,\mathrm{R}) \leftarrow (1-\alpha)Q(1,\mathrm{R}) + \alpha(R(s)+\gamma\ 0.0)$$

| | | | |
|---|---|---|---|
| 1 **U: 0.0**<br>**D: 0.0**<br>**L: 0.0**<br>**R: -0.004** | 2 **U: 0.0**<br>**D: 0.0**<br>**L: 0.0**<br>**R: -0.004** | 3 **U: 0.0**<br>**D: 0.0**<br>**L: 0.0**<br>**R: -0.004** | 4<br>**+1** |
| 5 **U: 0.0**<br>**D: 0.0**<br>**L: 0.0**<br>**R: 0.0** | | 6 **U: 0.0**<br>**D: 0.0**<br>**L: 0.0**<br>**R: 0.0** | 7 **U: 0.0**<br>**D: 0.0**<br>**L: 0.0**<br>**R: 0.0** |
| 8 **U: 0.0**<br>**D: 0.0**<br>**L: 0.0**<br>**R: 0.0** | 9 **U: 0.0**<br>**D: 0.0**<br>**L: 0.0**<br>**R: 0.0** | 10 **U: 0.0**<br>**D: 0.0**<br>**L: 0.0**<br>**R: 0.0** | 11 **U: 0.0**<br>**D: 0.0**<br>**L: 0.0**<br>**R: 0.0** |

## Step 5

$Q(1,\mathrm{R}) \leftarrow -0.0076$



| | | | |
|---|---|---|---|
| 1 U: 0.0<br>D: 0.0<br>L: 0.0<br>R: -0.0076 | 2 U: 0.0<br>D: 0.0<br>L: 0.0<br>R: -0.004 | 3 U: 0.0<br>D: 0.0<br>L: 0.0<br>R: -0.004 | 4<br>+1 |
| 5 U: 0.0<br>D: 0.0<br>L: 0.0<br>R: 0.0 | | 6 U: 0.0<br>D: 0.0<br>L: 0.0<br>R: 0.0 | 7 U: 0.0<br>D: 0.0<br>L: 0.0<br>R: 0.0 |
| 8 U: 0.0<br>D: 0.0<br>L: 0.0<br>R: 0.0 | 9 U: 0.0<br>D: 0.0<br>L: 0.0<br>R: 0.0 | 10 U: 0.0<br>D: 0.0<br>L: 0.0<br>R: 0.0 | 11 U: 0.0<br>D: 0.0<br>L: 0.0<br>R: 0.0 |

# SARSA: State-Action-Reward-State-Action

Modify Q learning to use chosen action, *a'*, not max

- $Q_{t+1}(s,a) \leftarrow (1-\alpha)Q_t(s,a) + \alpha\,[R(s,a,s') + \gamma\, \sout{\max_{a'} Q_t(s',a')}]$
- $Q_{t+1}(s,a) \leftarrow (1-\alpha)Q_t(s,a) + \alpha\,[R(s,a,s') + \gamma\, Q_t(s',a')]$

On-policy, instead of off-policy

- SARSA learns based on real behavior, not optimal behavior
  - Good if real world provides obstacles to optimal behavior
- Q learning learns optimal behavior, beyond real behavior
  - Good if training phase has obstacles that won't persist

# SARSA: State-Action-Reward-State-Action

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

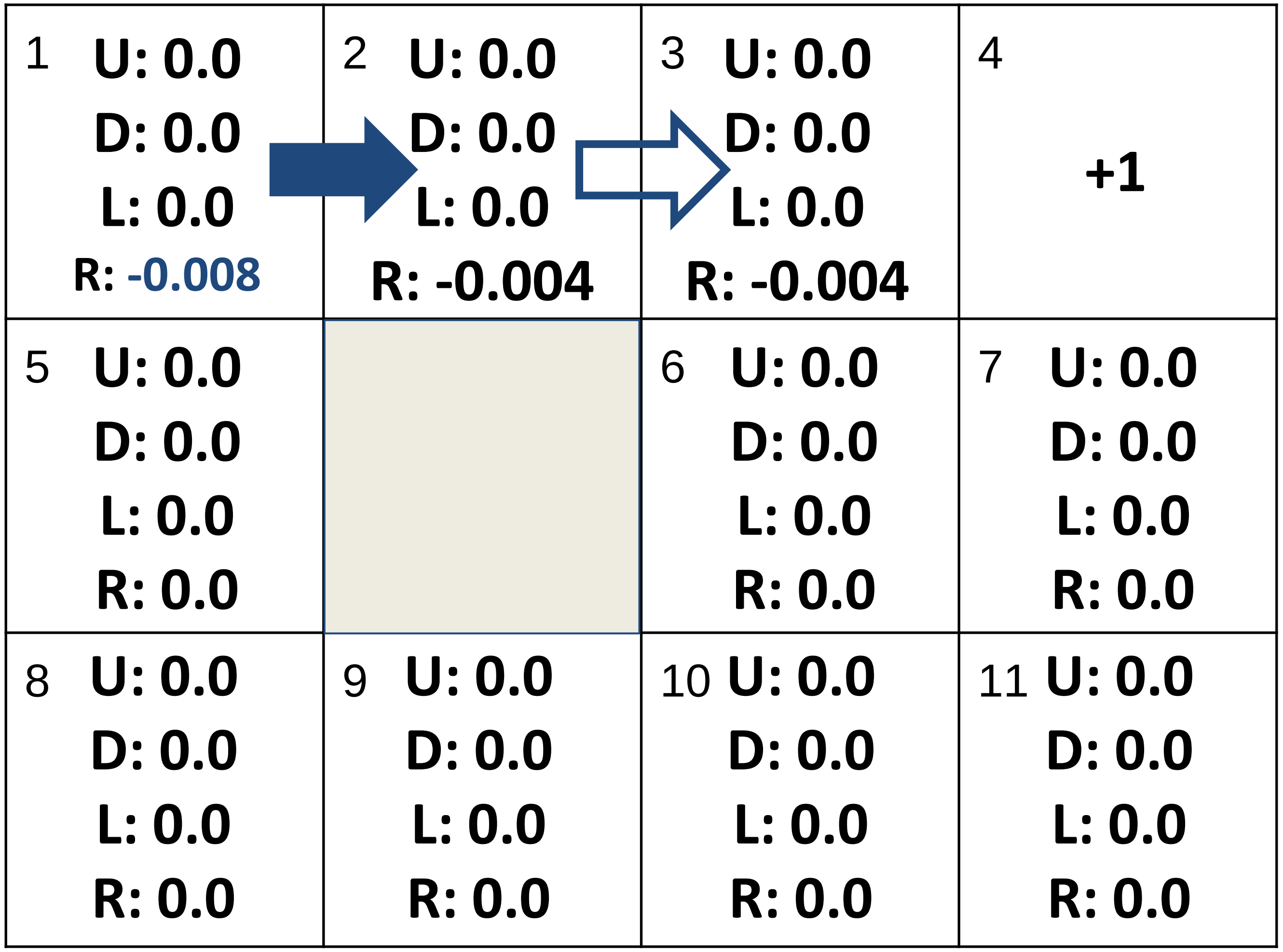
## Step 5: SARSA

$$Q(1,R) \leftarrow (1-\alpha)Q(1,R) + \alpha(R(s)+\gamma\, Q(2,R))$$

| | | | |
|---|---|---|---|
| 1 **U: 0.0**<br>**D: 0.0**<br>**L: 0.0**<br>**R: -0.004** | 2 **U: 0.0**<br>**D: 0.0**<br>**L: 0.0**<br>**R: -0.004** | 3 **U: 0.0**<br>**D: 0.0**<br>**L: 0.0**<br>**R: -0.004** | 4<br>**+1** |
| 5 **U: 0.0**<br>**D: 0.0**<br>**L: 0.0**<br>**R: 0.0** | ▨ | 6 **U: 0.0**<br>**D: 0.0**<br>**L: 0.0**<br>**R: 0.0** | 7 **U: 0.0**<br>**D: 0.0**<br>**L: 0.0**<br>**R: 0.0** |
| 8 **U: 0.0**<br>**D: 0.0**<br>**L: 0.0**<br>**R: 0.0** | 9 **U: 0.0**<br>**D: 0.0**<br>**L: 0.0**<br>**R: 0.0** | 10 **U: 0.0**<br>**D: 0.0**<br>**L: 0.0**<br>**R: 0.0** | 11 **U: 0.0**<br>**D: 0.0**<br>**L: 0.0**<br>**R: 0.0** |

## Step 5: SARSA

$Q(1,\mathrm{R}) \leftarrow -0.008$ which is closer to $-0.04$ than $-0.076$



| | | | |
|---|---|---|---|
| 1 U: 0.0 D: 0.0 L: 0.0 R: -0.008 | 2 U: 0.0 D: 0.0 L: 0.0 R: -0.004 | 3 U: 0.0 D: 0.0 L: 0.0 R: -0.004 | 4 +1 |
| 5 U: 0.0 D: 0.0 L: 0.0 R: 0.0 | ▨ | 6 U: 0.0 D: 0.0 L: 0.0 R: 0.0 | 7 U: 0.0 D: 0.0 L: 0.0 R: 0.0 |
| 8 U: 0.0 D: 0.0 L: 0.0 R: 0.0 | 9 U: 0.0 D: 0.0 L: 0.0 R: 0.0 | 10 U: 0.0 D: 0.0 L: 0.0 R: 0.0 | 11 U: 0.0 D: 0.0 L: 0.0 R: 0.0 |

## Q Learning & SARSA

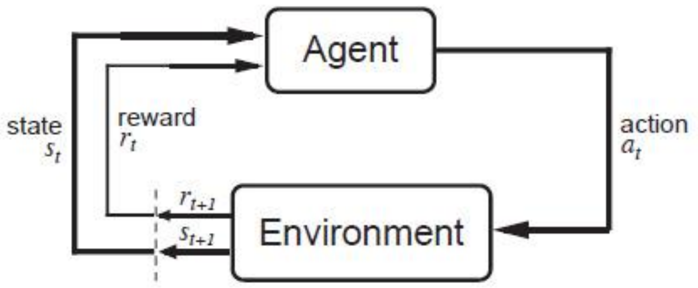
Converge to correct values

- Assuming agent tries all actions, in all states, many times
    - Otherwise, there won't be enough experience to learn *Q*
- And if $\alpha$ decays over time
    - Similar to simulated annealing

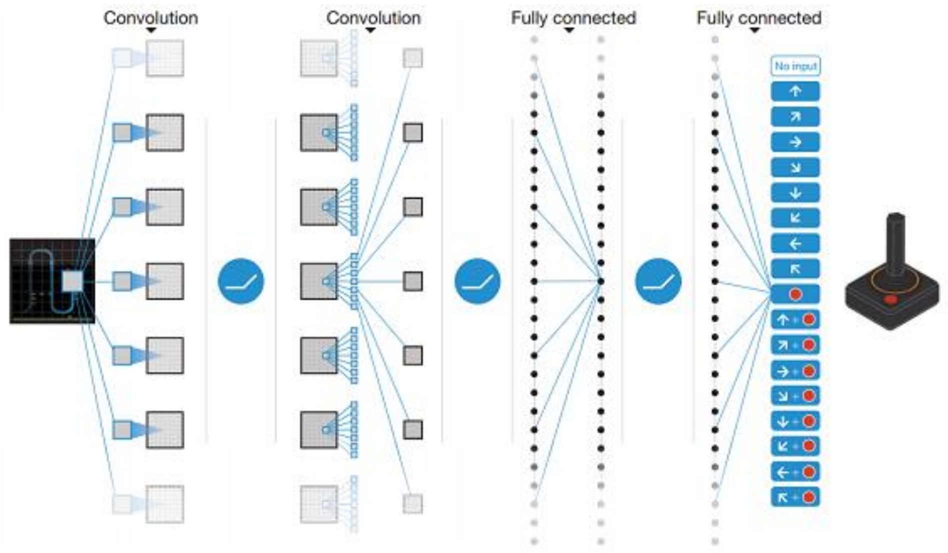
Avoids the complexity of solving an MDP

- MDP requires solving for the optimal policy before starting
- An RL agent can start right away and then learn as it goes
    - Although this might be wasteful if you already know *P* and *R*

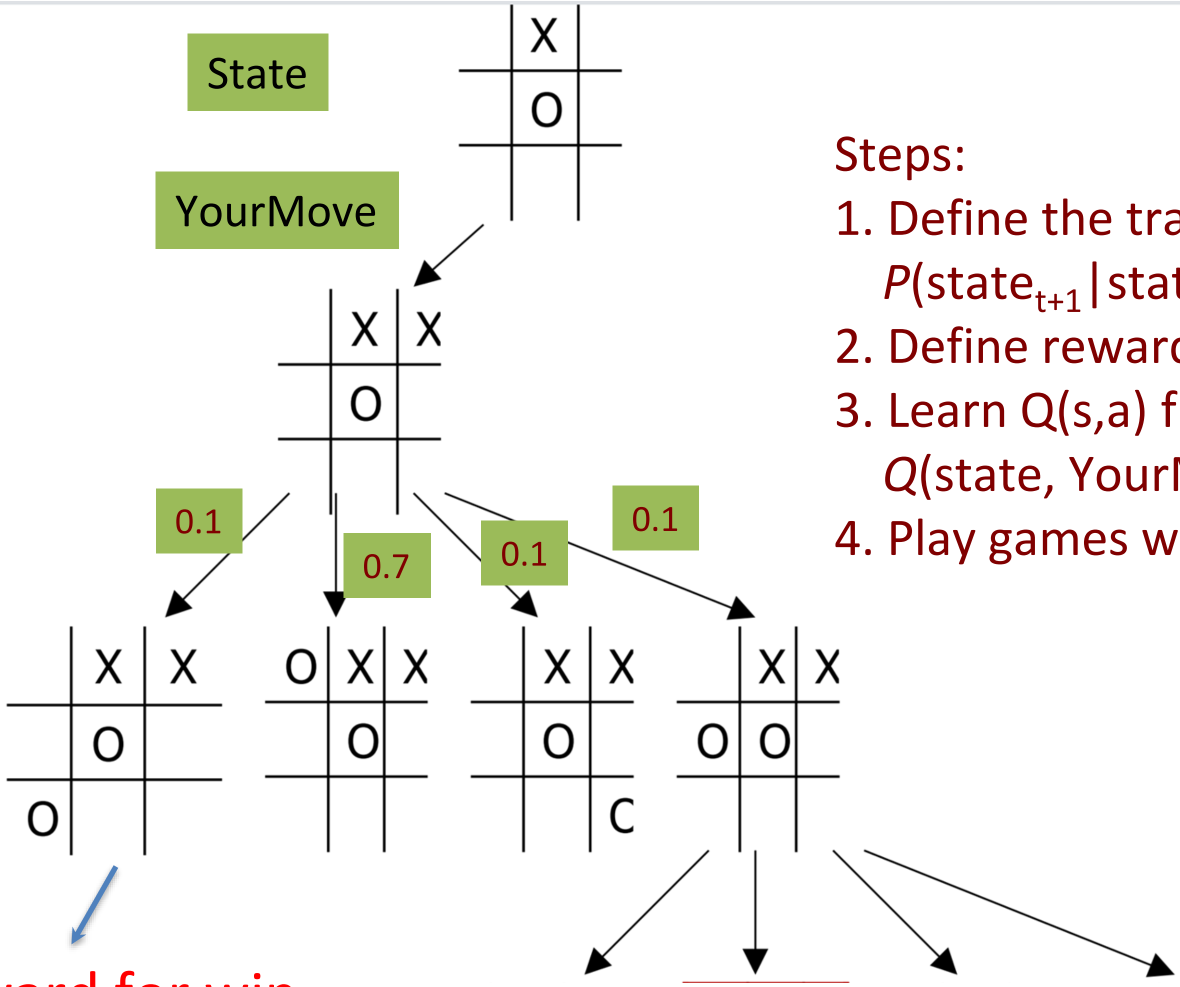
# Q-Learning for Game Playing



AlphaGo



# Q-Learning for Game Playing



Reward for win

Steps:

1. Define the transition model
   $P(\text{state}_{t+1} | \text{state}_t, \text{YourMove})$
2. Define rewards
3. Learn Q(s,a) from rewards
   $Q(\text{state}, \text{YourMove})$
4. Play games with learned Q

# A Key Question

- In all search and game playing problems, what is the key information that you wish to have for finding the optimal solution?
  - You wish to have but you may not always have

- The answer: Play a lot of games and
  - Learn the state utility values for winning the games
  - Learn the Q values for winning the games

# Exploitation vs. Exploration

- RL algorithms do not specify how actions are chosen by the agent
- One strategy would be in state *s* to select the action *a* that maximizes *Q*(*s*,*a*), thereby *exploiting* its current approximation *Q*.
  - Using this strategy the agent runs the risk that it will overcommit to actions that are found during early training to have high *Q* values, while failing to *explore* other actions that have even higher values.
  - It is common in Q learning to use a probabilistic approach to selecting actions.
  - Actions with higher Q values are assigned higher probabilities, but every action is assigned a nonzero probability. One way to assign such probabilities is where $P(a_i \mid s)$ is the probability *T* of selecting action $a_i$, given that the agent is in state *s*, and where $T > 0$ is a constant that determines how strongly the selection favors actions with high *Q* values.

$$P(a_i \mid s) = \frac{e^{\hat{Q}(s,a_i)/T}}{\sum_j e^{\hat{Q}(s,a_j)/T}}$$

# Exploitation vs. Exploration

- Hence it is good to try new things now and then
  - If *T* is large, then do more exploration,
  - If *T* is small, then follow or exploit the current policy
- One can decrease *T* over time to first explore, and then converge and exploit
  - For example *T= c/k+d* where *k* is iteration of the algorithm

$$P(a_i \mid s) = \frac{e^{\hat{Q}(s,a_i)/T}}{\sum_j e^{\hat{Q}(s,a_j)/T}}$$

- Decreasing *T* over time is also known as *simulated annealing*, which is inspired by annealing process in nature. *T* is also known as *Temperature*