



هوش مصنوعی

بنام حافظه ای از جان دو

فرایندهای تصمیم‌گیری مارکوف

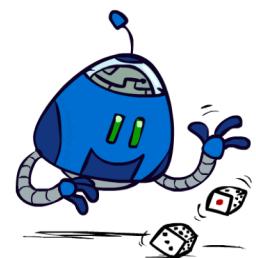
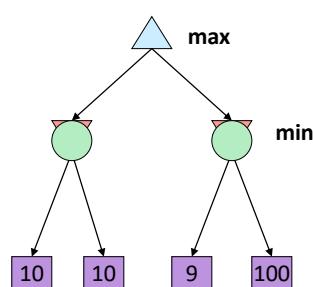
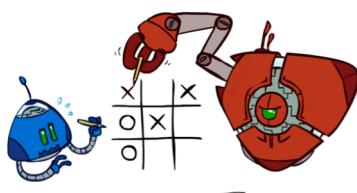
Markov Decision Processes



محمد طاهر پیلهور

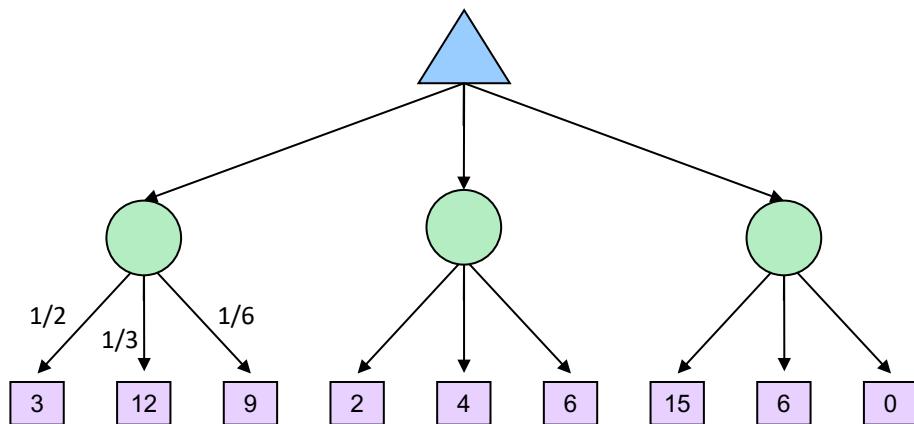
[Many of the slides were borrowed from Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley.]

جلسه‌ی قبل

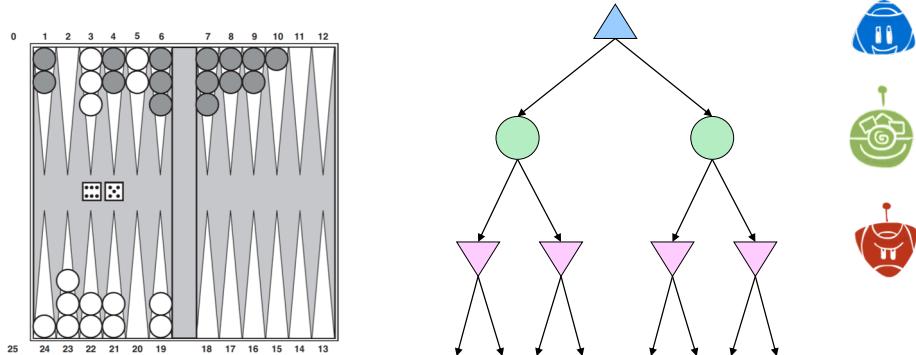


عدم قطعیت

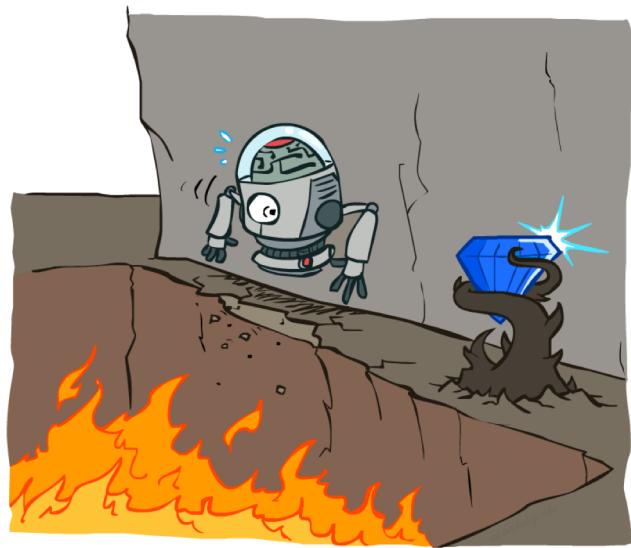
جلسه‌ی قبل



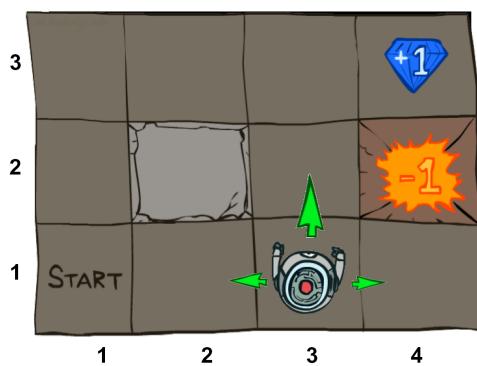
جلسه‌ی قبل



جستجوی غیر قطعی



مثال امروز



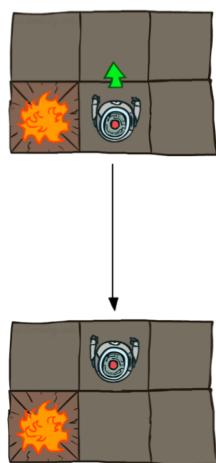
حرکات نامطمئن (نویزی)
مثال: اگر تصمیم به رفتن به طرف شمال کنیم
۸۰٪ اوقات درست می‌رویم (در صورتی که دیواری نباشد)
۱۰٪ به راست منحرف می‌شویم
۱۰٪ به چپ

جايزه برای هر حرکت
جايزه کوچک: برای زنده ماندن (با برعکس جریمه - منفی)
جايزه بزرگ: برای رسیدن به هدف (+1 یا -1)

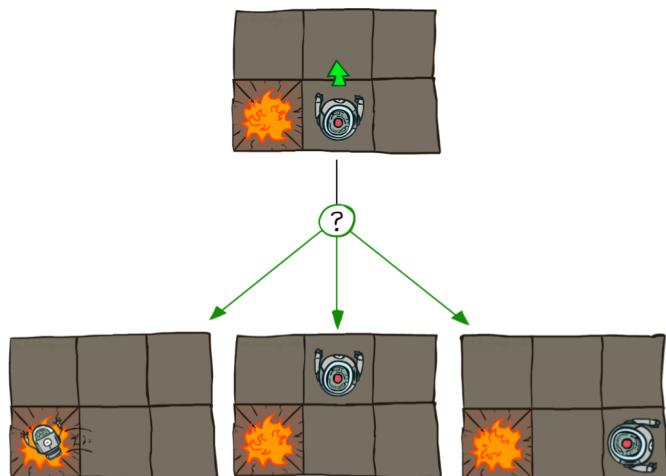
هدف: مجموع جائزه‌ها را ماکزیمم کن!

حالات ممکن

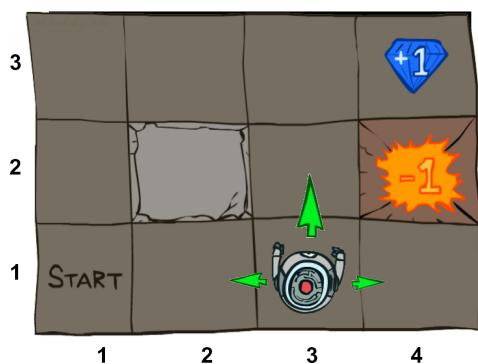
جهان قطعی



جهان تصادف



فرآیندهای تصمیمگیری مارکوف – MDP



مجموعه ای از حالت: $s \in S$

مجموعه ای از تصمیمات: $a \in A$

تابع انتقال: $T(s, a, s')$

احتمال اینکه از وضعیت s به وضعیت s' برویم - توزیع شرطی: $P(s' | s, a)$

تابع پاداش: $R(s, a, s')$

نقطه (حالت) شروع: start state

نقطه پایان (شاید): terminal state

مسائل جستجوی غیرقطعی

مشابه آنچه برای اکسپکتیمکس دیدیم

ولی روش‌های بهتری هم برای حل آنها وجود دارد! (که در این فصل خواهیم دید)

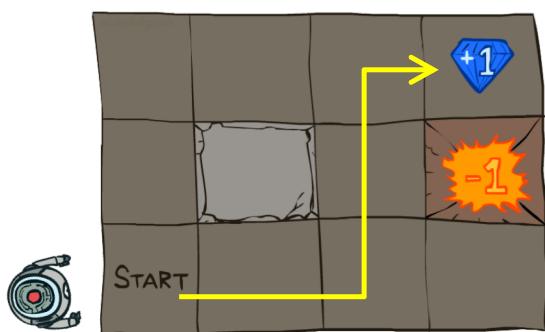
مثال



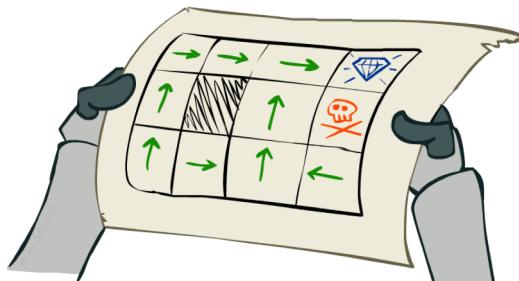
Nondeterministic.mp4 ویدئو

تا به اینجا – در جستجوی «نقشه»

- دنبال یافتن یک plan بودیم



نقشه یا راهبرد – plan vs. policy



در جستجوی تک عاملی قطعی:

نقشه بهینه (مجموعه‌ای از تصمیمات که ما را از شروع به هدف می‌رساند)

برای MDP:

راهبرد بهینه

policy $\pi^*: S \rightarrow A$

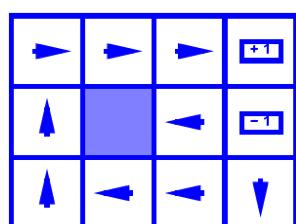
راهبردی که سودمندی مورد انتظار را بیشینه کند

راهبرد بهینه در جایی که

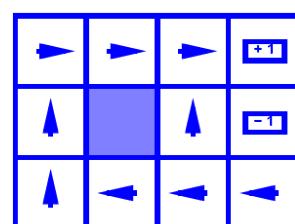
$$R(s, a, s') = -0.03$$

برای تمامی حالت (غیر از حالت ترمینال)

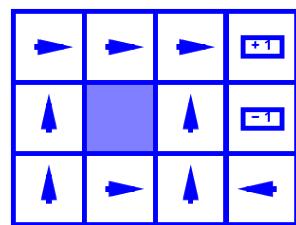
راهبرد بهینه



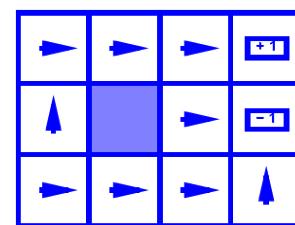
$$R(s) = -0.01$$



$$R(s) = -0.03$$



$$R(s) = -0.4$$



$$R(s) = -2.0$$

مارکوف؟

حالت آینده تنها به حالت فعلی وابسته است (بنابراین، از حالات گذشته مستقل است)

$$P(S_{t+1} = s' | S_t = s_t, A_t = a_t, S_{t-1} = s_{t-1}, A_{t-1}, \dots, S_0 = s_0)$$

$$= P(S_{t+1} = s' | S_t = s_t, A_t = a_t)$$

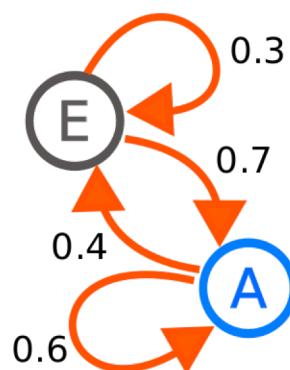


Andrey Markov
(1856-1922)

همانند مساله جستجو:

تابع پیشبرنده فقط با حالت فعلی کار داشت (ونه اینکه چگونه به آنجا رسیده ایم)

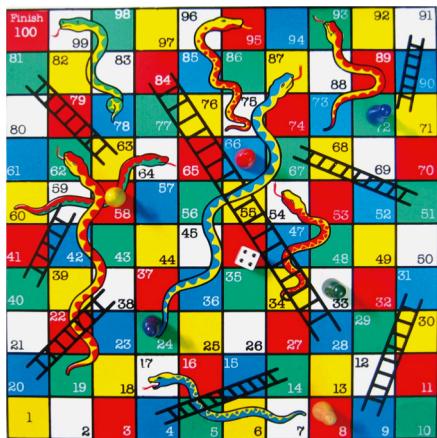
مثال: فرآیند مارکوفی دو حالتی



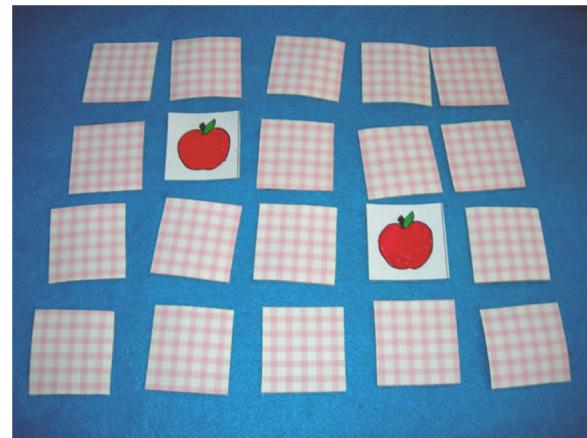
From Wikipedia: A diagram representing a two-state Markov process, with the states labelled E and A.

Each number represents the probability of the Markov process changing from one state to another state, with the direction indicated by the arrow. For example, if the Markov process is in state A, then the probability it changes to state E is 0.4, while the probability it remains in state A is 0.6.

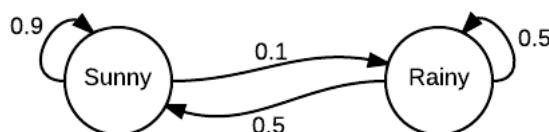
مثال: بازی‌های مارکوفی



VS.



مثال: فرآیند مارکوفی



$$P = \begin{bmatrix} 0.9 & 0.1 \\ 0.5 & 0.5 \end{bmatrix}$$

The weather on day 1 is known to be sunny. $\mathbf{x}^{(0)} = [1 \ 0]$

The weather on day 2 can be predicted by: $\mathbf{x}^{(1)} = \mathbf{x}^{(0)} P = [1 \ 0] \begin{bmatrix} 0.9 & 0.1 \\ 0.5 & 0.5 \end{bmatrix} = [0.9 \ 0.1]$

On day 3: $\mathbf{x}^{(2)} = \mathbf{x}^{(1)} P = [0.9 \ 0.1] \begin{bmatrix} 0.9 & 0.1 \\ 0.5 & 0.5 \end{bmatrix} = [0.86 \ 0.14]$

از ویکیپدیا

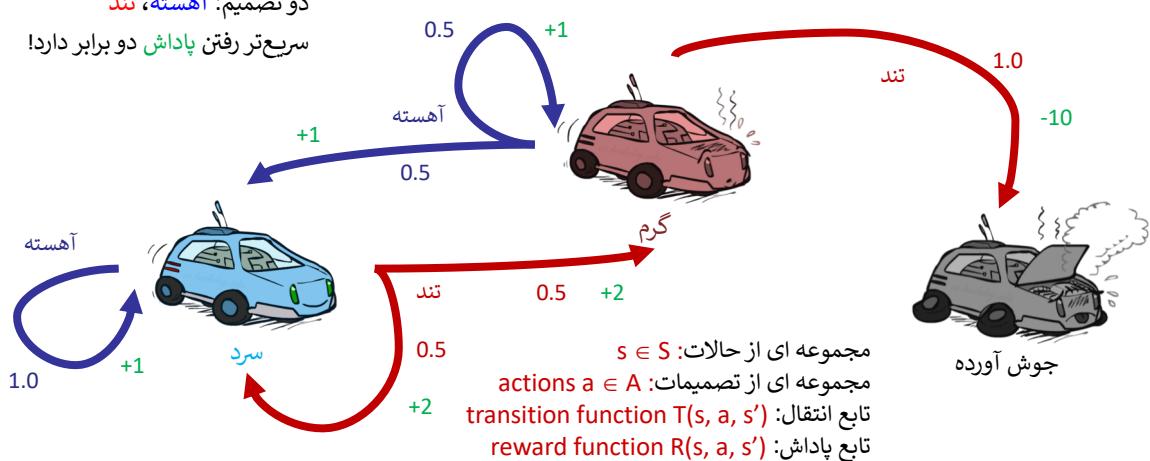
مثال: مسابقه ماشین رانی



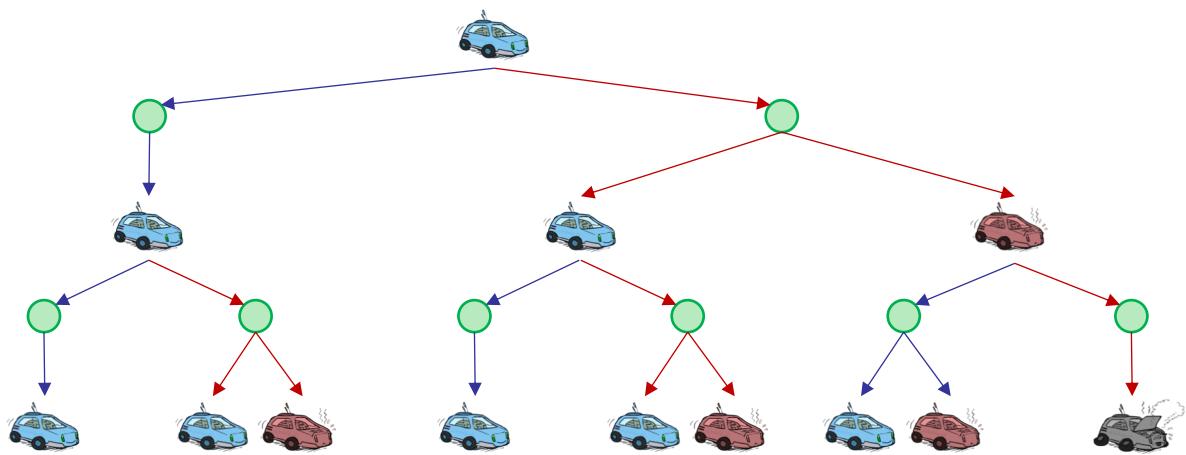
مثال: مسابقه ماشین رانی

هدف: سریع تر و طولانی تر رفتن
سه حالت: سرد، گرم، جوش آورده
دو تصمیم: آهسته، تند
سریع تر رفتن پاداش دو برابر دارد!

نمودار انتقال حالت - State transition diagram

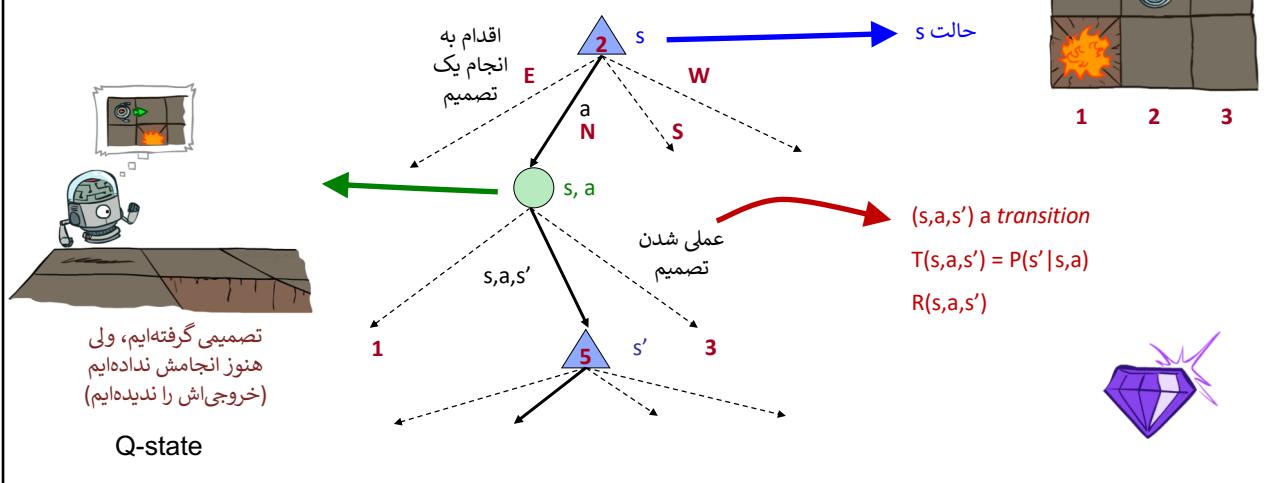


درخت جستجو



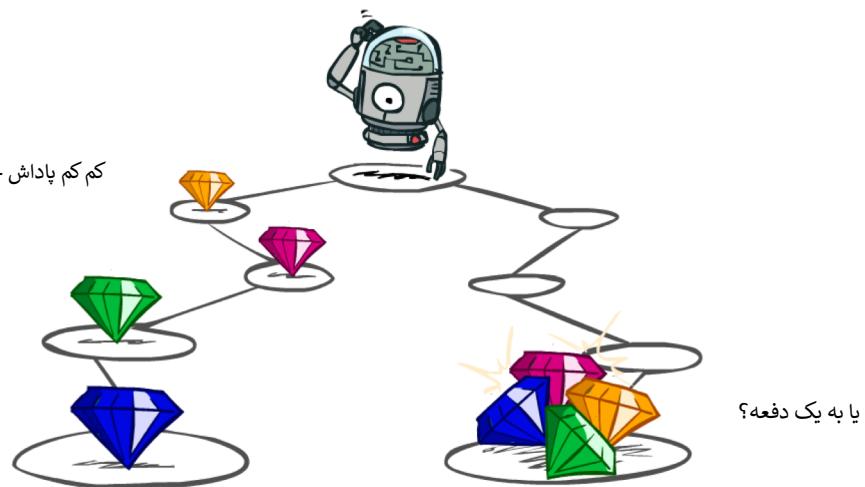
MDP جستجوی

وضعیت مشابه اکسپکتیمکس داریم:



سودمندی یک دنباله

کم کم پاداش جمع کنیم؟

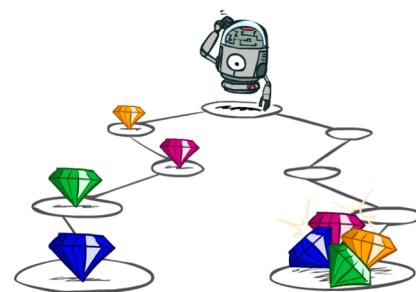


سودمندی یک دنباله

کدامیک را ترجیح دهیم؟

- کمتر یا بیشتر? [1, 2, 2] or [2, 3, 4]

- الان یا بعداً? [0, 0, 1] or [1, 0, 0]



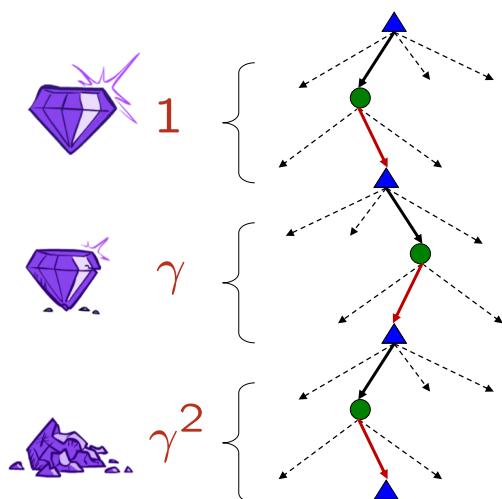
کاستن – Discounting

معقول است که دنبال پاداش بیشتری باشیم
همچنین، بهتر است که الان را به بعدا ترجیح دهیم!

یک راه حل: مقدار پاداش با گذشت زمان کاسته شود



کاستن



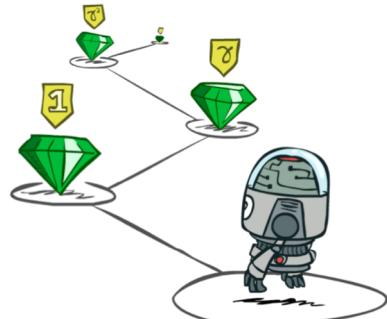
چطور؟
با هر قدم، ارزش پاداش کاسته شود (مثلاً با ضرب کردن در ضریب تخفیف)

چرا؟
زیرا که سودمندی الان بهتر از سودمندی آینده است
(به همگرایی الگوریتم هم کمک می کند)

مثال: ضریب تخفیف ۰/۵

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

اولویت‌های ایستا – Stationary preferences



در صورتی که یک دنباله به دیگری ترجیح داشته باشد، اضافه کردن یک پاداش ثابت به ابتدای هر دوی آنها ترجیح را تغییر نخواهد داد

$$[a_1, a_2, \dots] \succ [b_1, b_2, \dots]$$

\Updownarrow

$$[r, a_1, a_2, \dots] \succ [r, b_1, b_2, \dots]$$

اگر اولویت‌ها ایستا باشند، دو راه برای تعریف سودمندی داریم:

- Additive - سودمندی تجمیعی - $U([r_0, r_1, r_2, \dots]) = r_0 + r_1 + r_2 + \dots$
- Discounted - سودمندی تخفیفی - $U([r_0, r_1, r_2, \dots]) = r_0 + \gamma r_1 + \gamma^2 r_2 \dots$

کوییز!

(فقط در حالات East, West, Exit (a, e): تصمیمات ممکن
انتقالات قطعی

10					1
a	b	c	d	e	

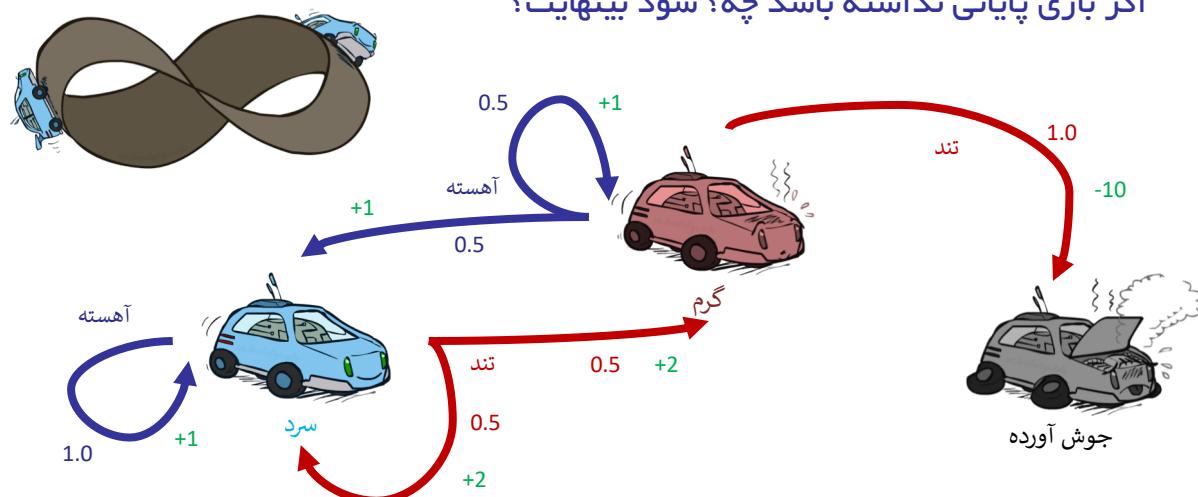
اگر در d باشیم، مقادیر گام‌ای زیر چه تاثیری در تصمیم گیری خواهند داشت؟

10				1
$\gamma = 1$				

10				1
$\gamma = 0.1$				

سودمندی بینهایت؟!

اگر بازی پایانی نداشته باشد چه؟ سود بینهایت؟



سودمندی بینهایت؟!

اگر بازی پایانی نداشته باشد چه؟ سود بینهایت؟

راه حل‌ها:

۱- افق محدود (همانند جستجوی با عمق محدود)

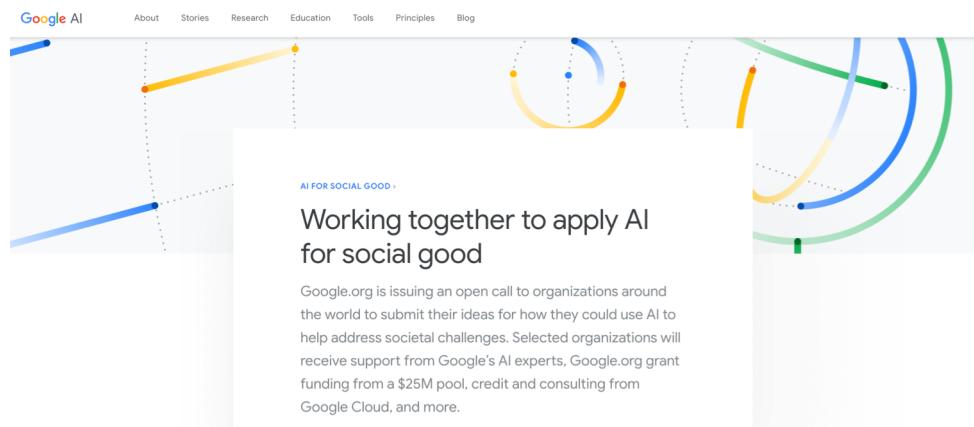
- بازی را پس از تعداد معینی حرکت پایان بده - عمر T

$$U([r_0, \dots, r_\infty]) = \sum_{t=0}^{\infty} \gamma^t r_t \leq R_{\max} / (1 - \gamma) \quad 0 < \gamma < 1$$

هر چه گاما کوچکتر، افق (عمر) کوتاه‌تر

۳- اضافه کردن این احتمال که ممکن است از هر حالتی به حالت ترمینال رفت.

AI for Social Good



The image shows the Google AI for Social Good landing page. At the top, there's a navigation bar with links for Google AI, About, Stories, Research, Education, Tools, Principles, and Blog. Below the navigation is a large graphic featuring abstract, colorful lines (blue, yellow, green) forming loops and curves against a white background. In the center of the page, the text "AI FOR SOCIAL GOOD >" is followed by "Working together to apply AI for social good". Below this, a detailed paragraph explains the initiative: "Google.org is issuing an open call to organizations around the world to submit their ideas for how they could use AI to help address societal challenges. Selected organizations will receive support from Google's AI experts, Google.org grant funding from a \$25M pool, credit and consulting from Google Cloud, and more."

<https://ai.google/social-good/impact-challenge>

AI for Social Good

The fight against illegal deforestation with TensorFlow

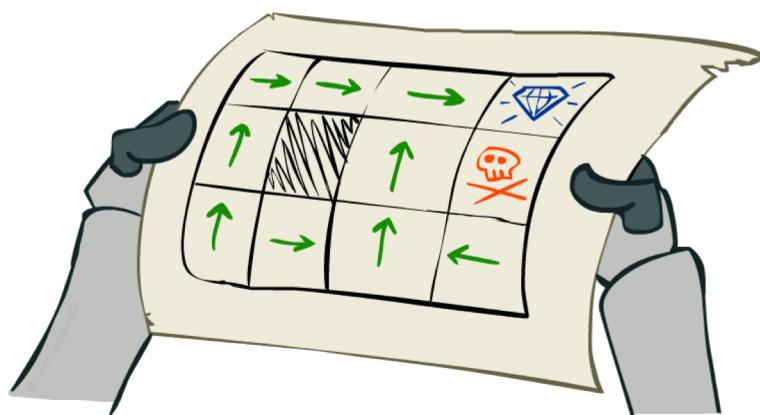


The fight against illegal deforestation with TensorFlow

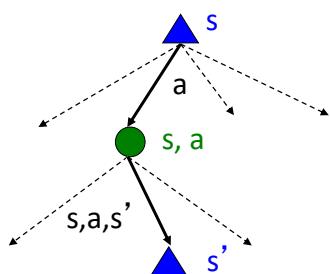
GoogleForGood-1.mp4 ۰:۰۰:۱۹



حل کردن MDP



فرآیند تصادفی مارکوفی - یادآوری



مجموعه ای از حالت: $s \in S$

مجموعه ای از تصمیمات: actions $a \in A$

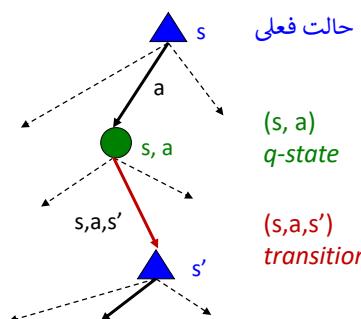
تابع انتقال: transition function $T(s, a, s')$

تابع پاداش: reward function $R(s, a, s')$

نقطه (حالت) شروع: start state

نقطه پایان (شاید): terminal state

فرآیند تصادفی مارکوفی



Policy – راهبرد •

نقشه‌ای که action مناسب در هر state را می‌دهد

Utility – سودمندی •

مجموع (تخفیف داده شده) پاداش‌ها

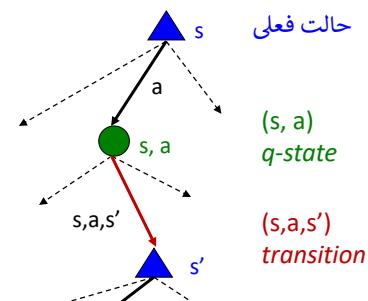
Values – ارزش‌ها •

سودمندی مورد انتظار از یک state (گرهی max)

Q-values – Q •

سودمندی مورد انتظار از یک q-state (گرهی احتمالاتی)

مقادیر بهینه



ارزش (سودمندی) حالت s

سودمندی متوسط اگر از حالت فعلی تمامی تصمیمات $V^*(s) =$
بهینه باشند

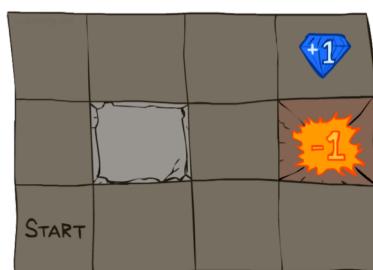
ارزش (سودمندی) Q-state

سودمندی متوسط پس از تصمیم در مرحله
فعلی و درصورتی که از این به بعد بهینه تصمیم گیری کنیم

راهبرد بهینه

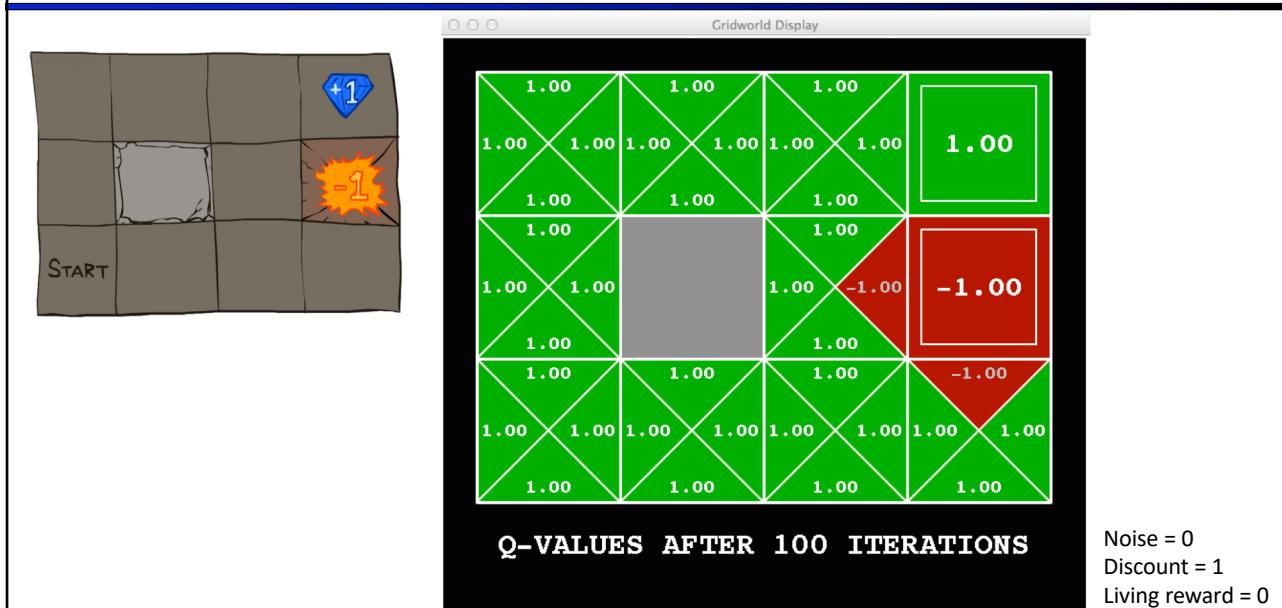
تصمیمات بهینه از حالت فعلی $\pi^*(s) =$

مثال – ارزش حالت: V

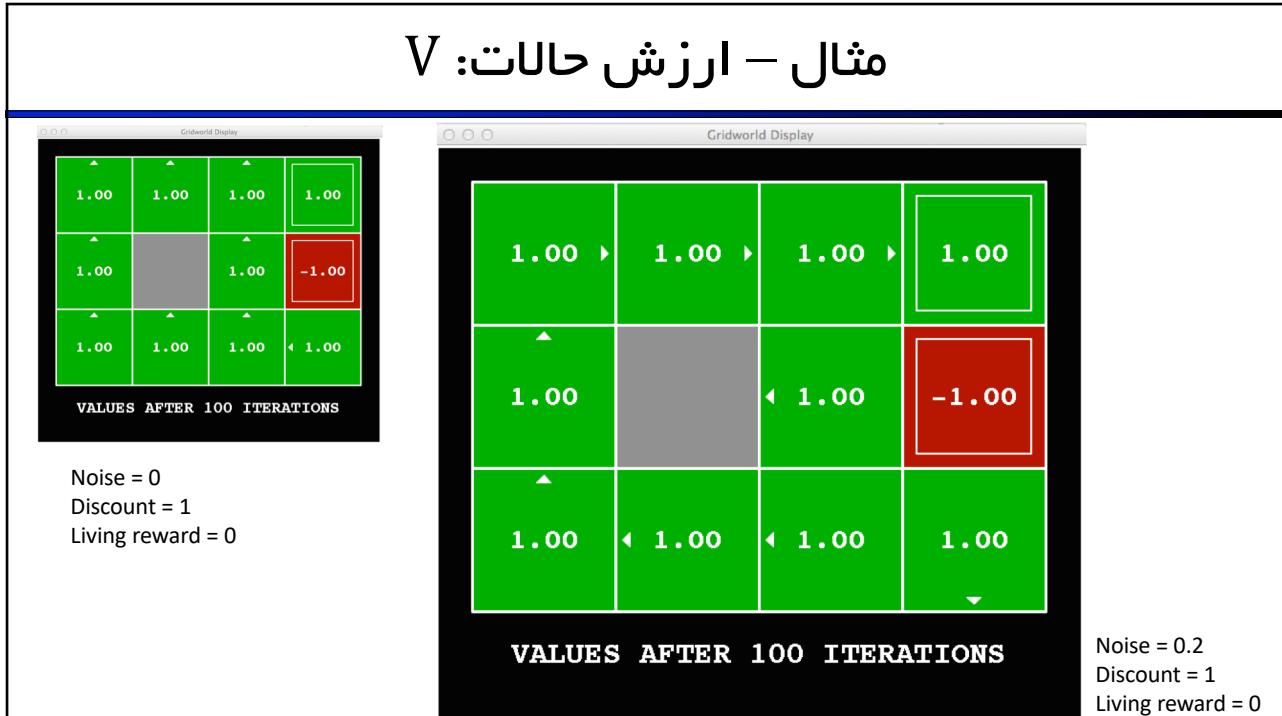


Noise = 0
Discount = 1
Living reward = 0

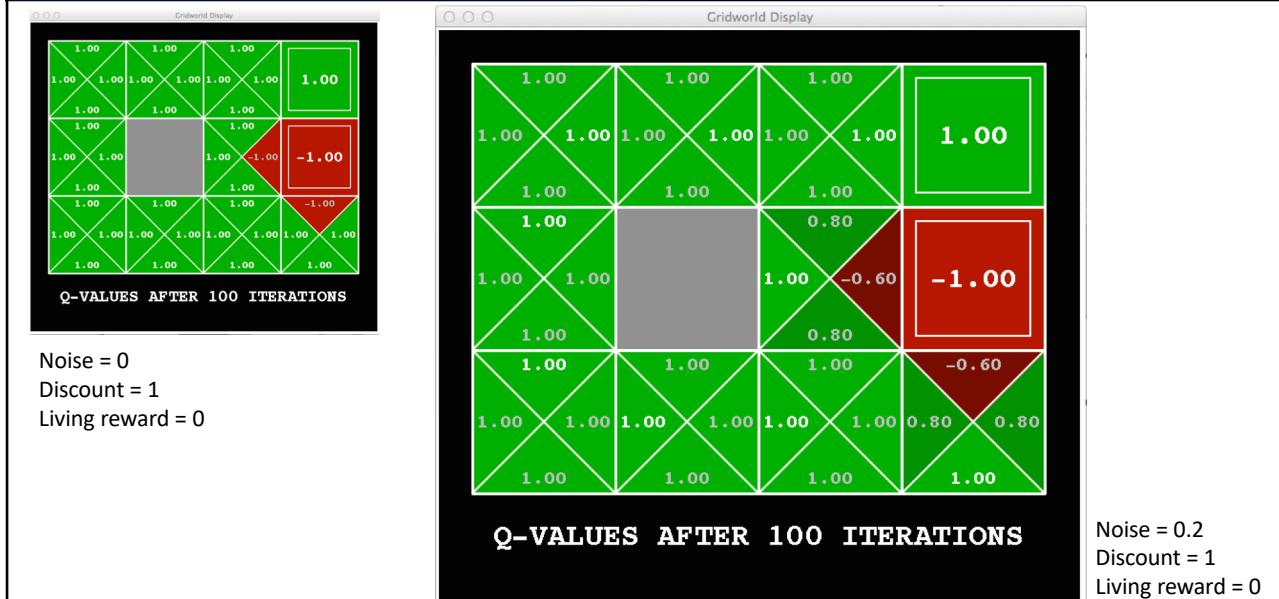
مثال – ارزش ایالت: Q-states



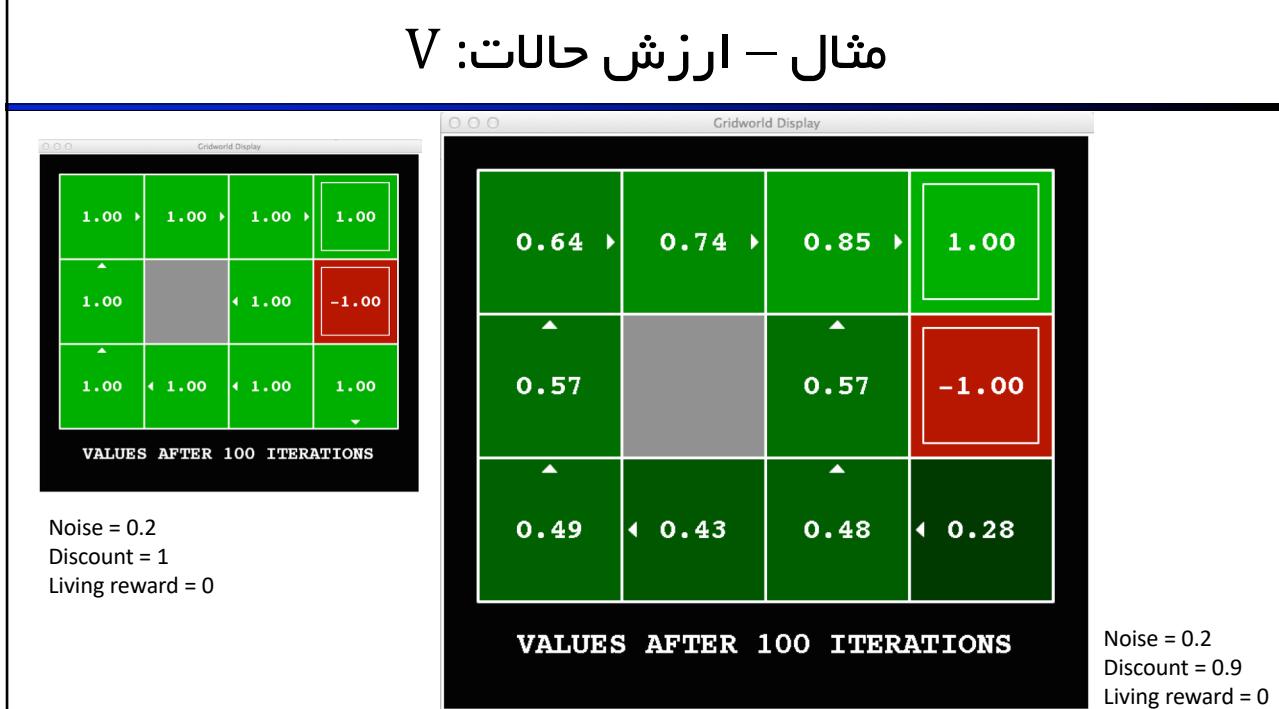
مثال – ارزش حالت: V



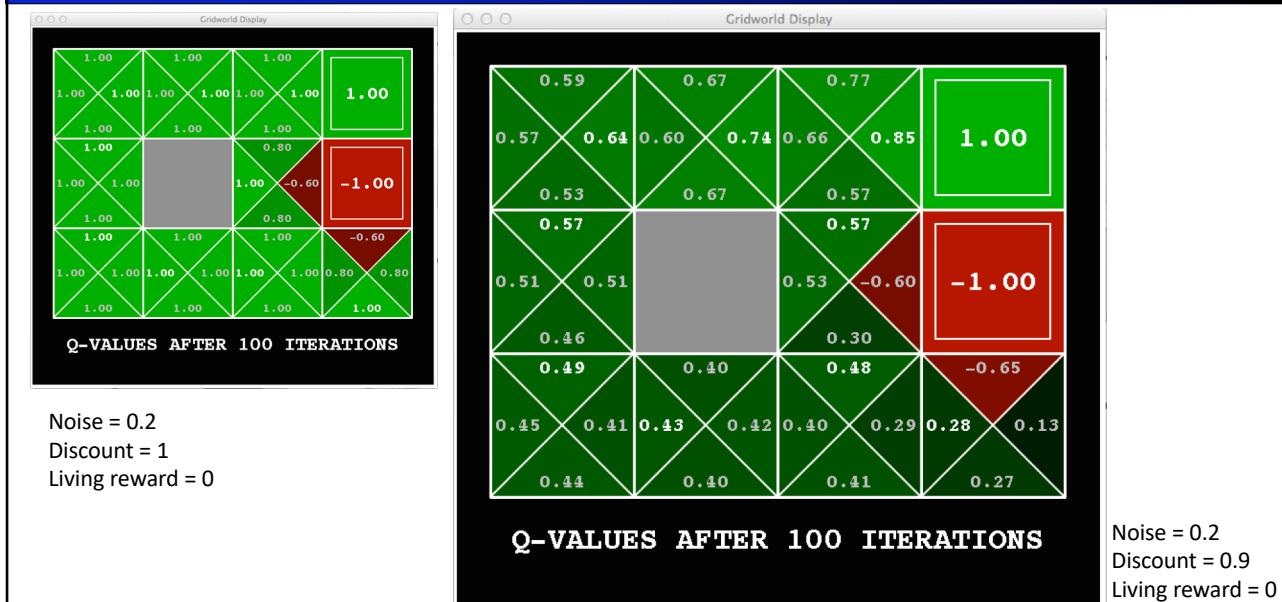
مثال – ارزش انتقالی Q-states



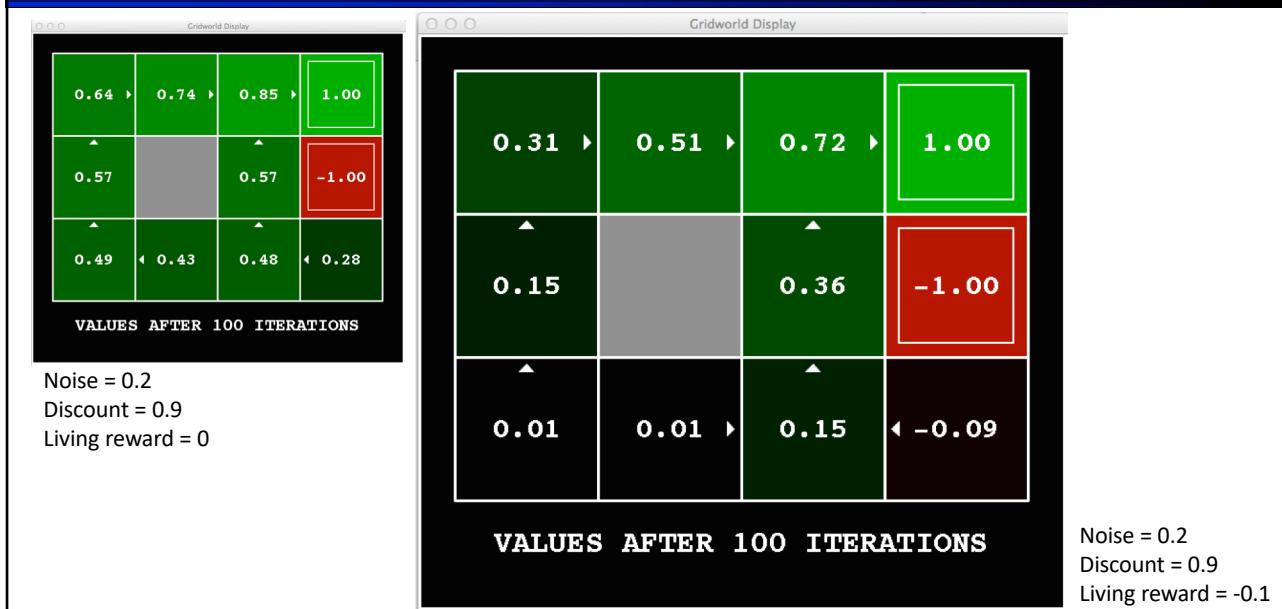
مثال – ارزش حالت: V



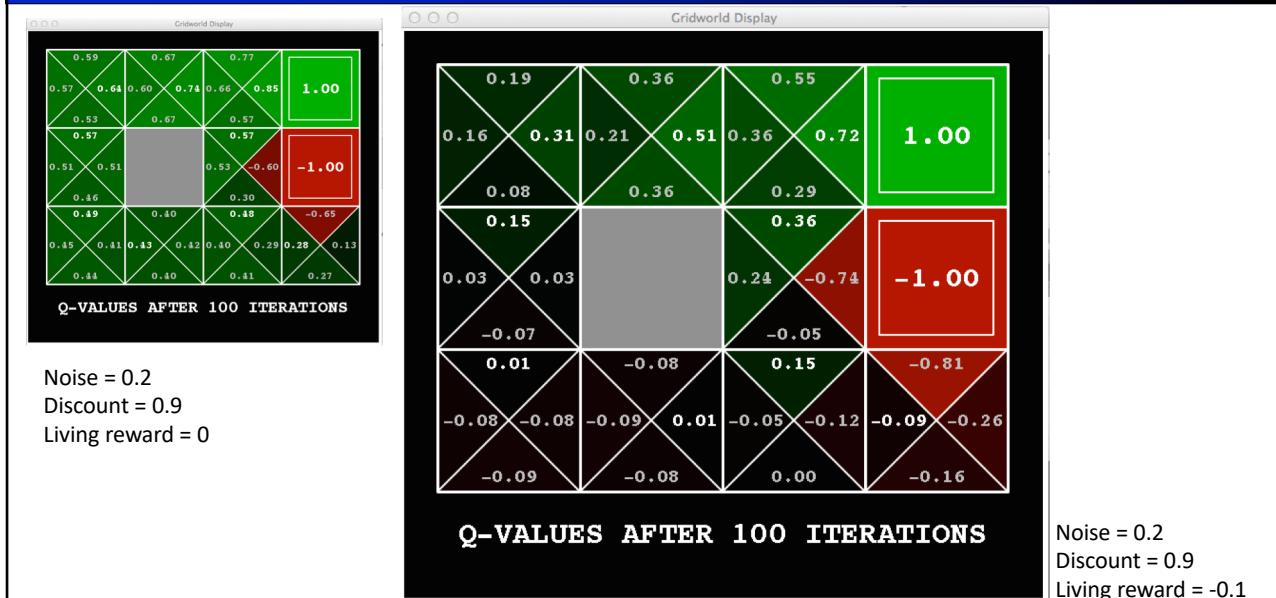
مثال – ارزش ایالت: Q-states



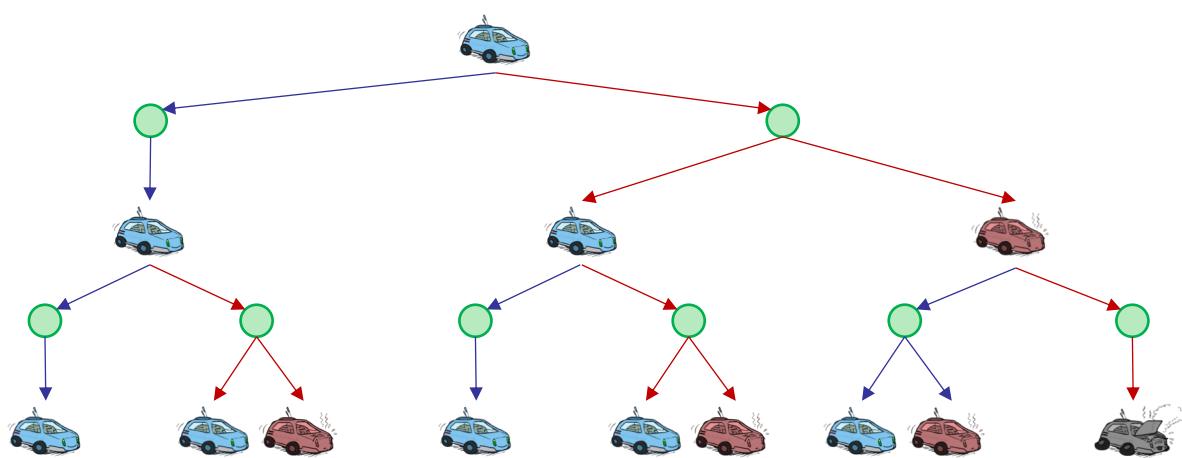
مثال – ارزش حالت: V



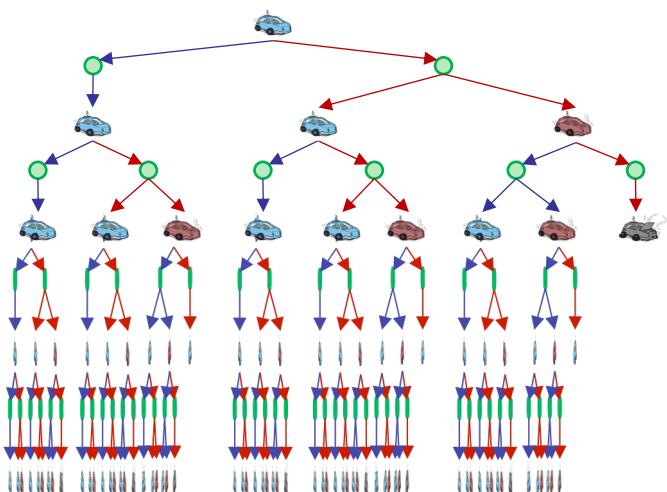
مثال – ارزش Q-states



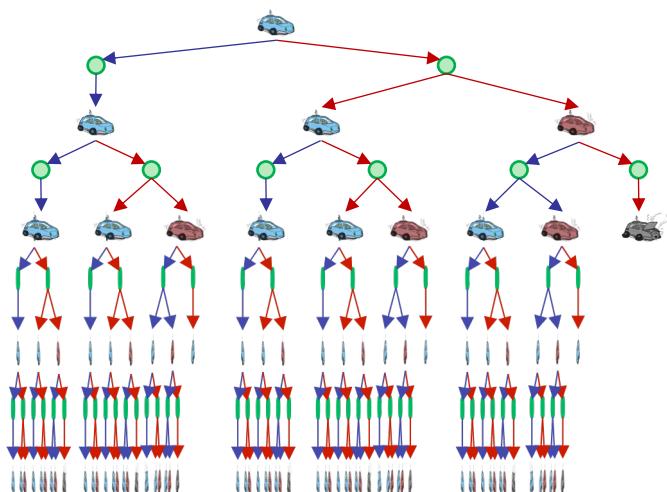
درخت جستجو



درخت جستجو - عمق بیشتر



درخت جستجو



۱- با اکسپکتیمکس، کلی کار تکراری
انجام می‌دهیم (حالت‌های تکراری)

فقط یک بار محاسبه کنیم

۲- درخت جستجو بی‌انتهاءست

عمر را محدود کنیم

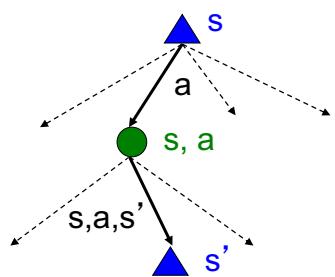
با توجه به ضریب تخفیف، خیلی با راه حل‌هایی
که در عمر زیاد هستند کاری نداریم

یافتن مسیر بهینه در MDP ها

▪ دنبال پیدا کردن ارزش یک حالت در صورت گرفتن تصمیم بهینه هستیم

- همانند اکسپکتیمکس عمل کنیم

- ولی برای جلوگیری از تکرار، از یک روش recursive استفاده کنیم



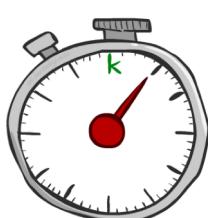
▪ معادلات Bellman

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

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

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

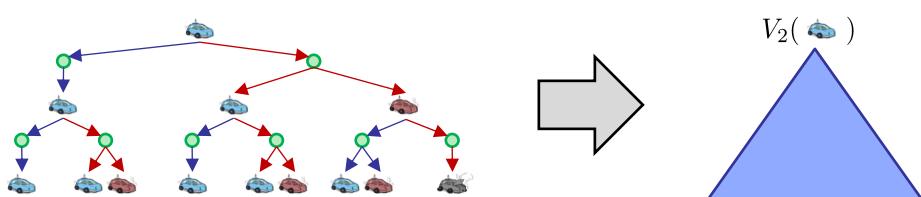
ارزش محدود شده در زمان



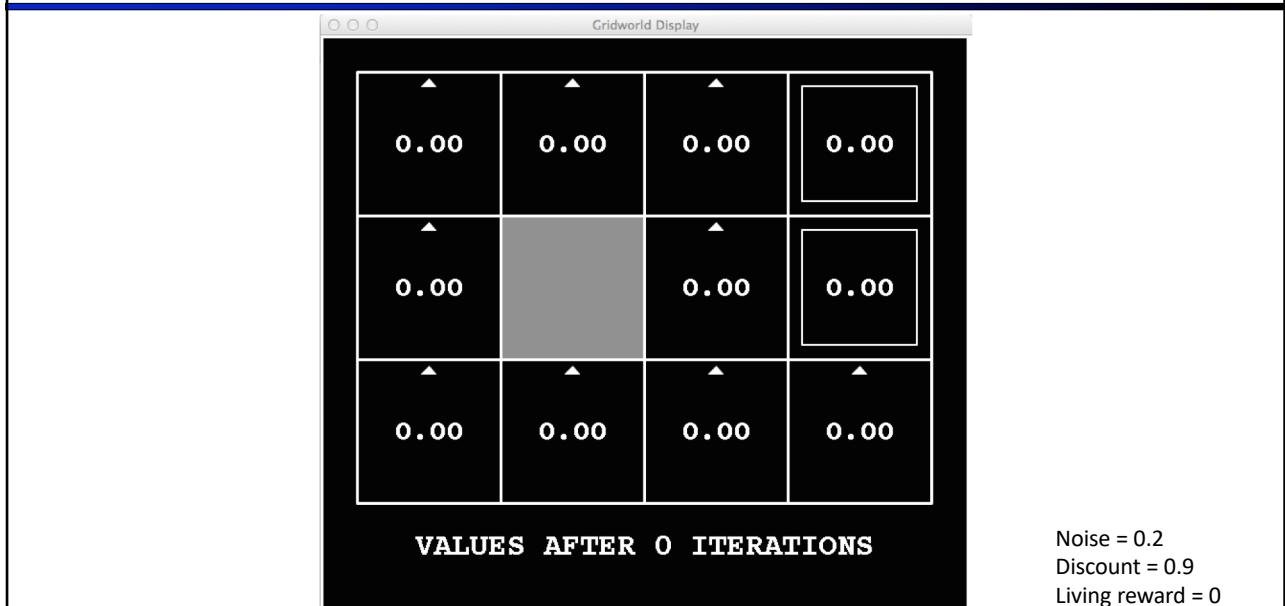
ارزش بهینه در صورتی که از حالت فعلی بهینه عمل کنیم و بازی در k حرکت تمام شود.

$$V_k(s)$$

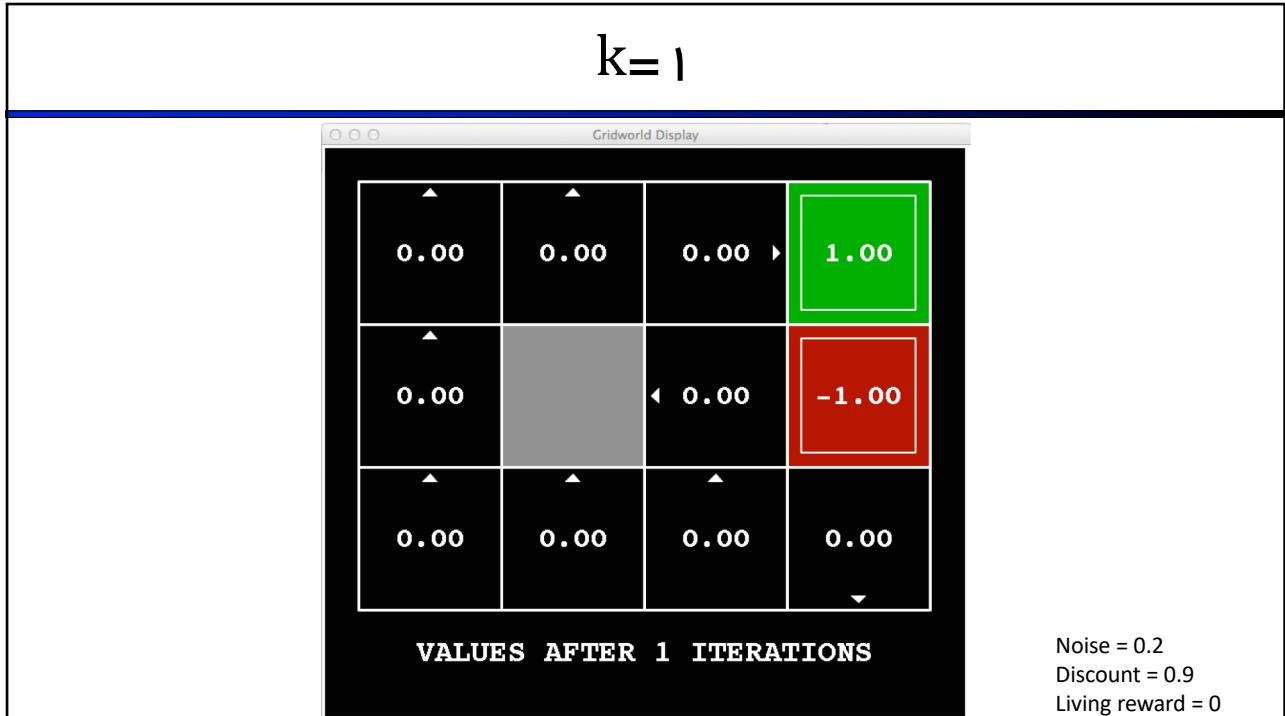
مشابه اکسپکتیمکس عمق-محدود



$k=0$



$k=1$



$k=\mu$



$k=\omega$



$k=\gamma$



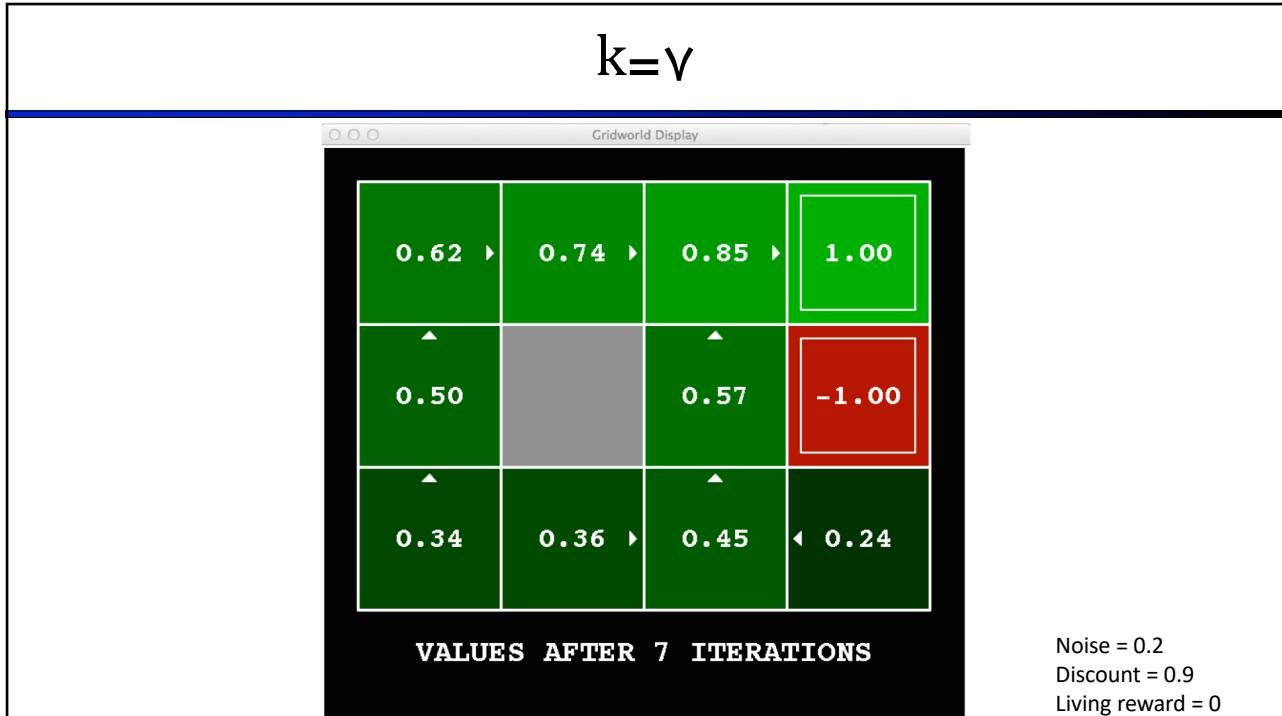
$k=\omega$



$k = \varsigma$



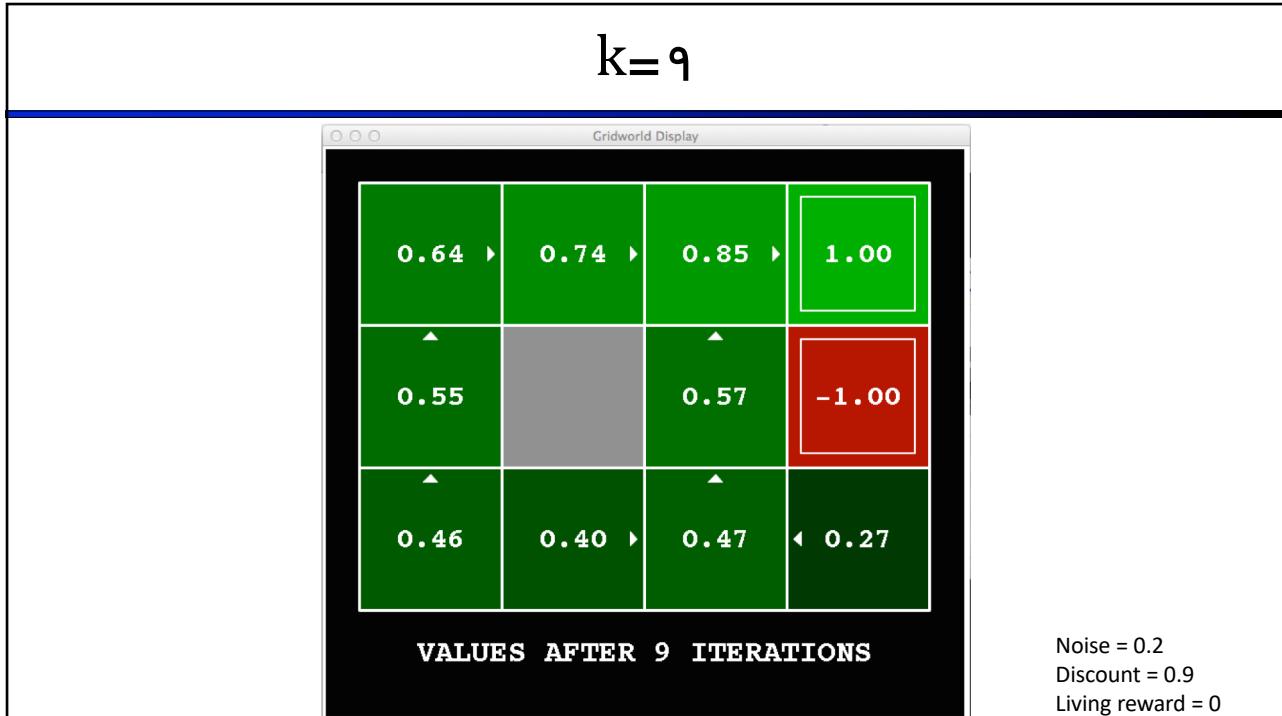
$k = \gamma$



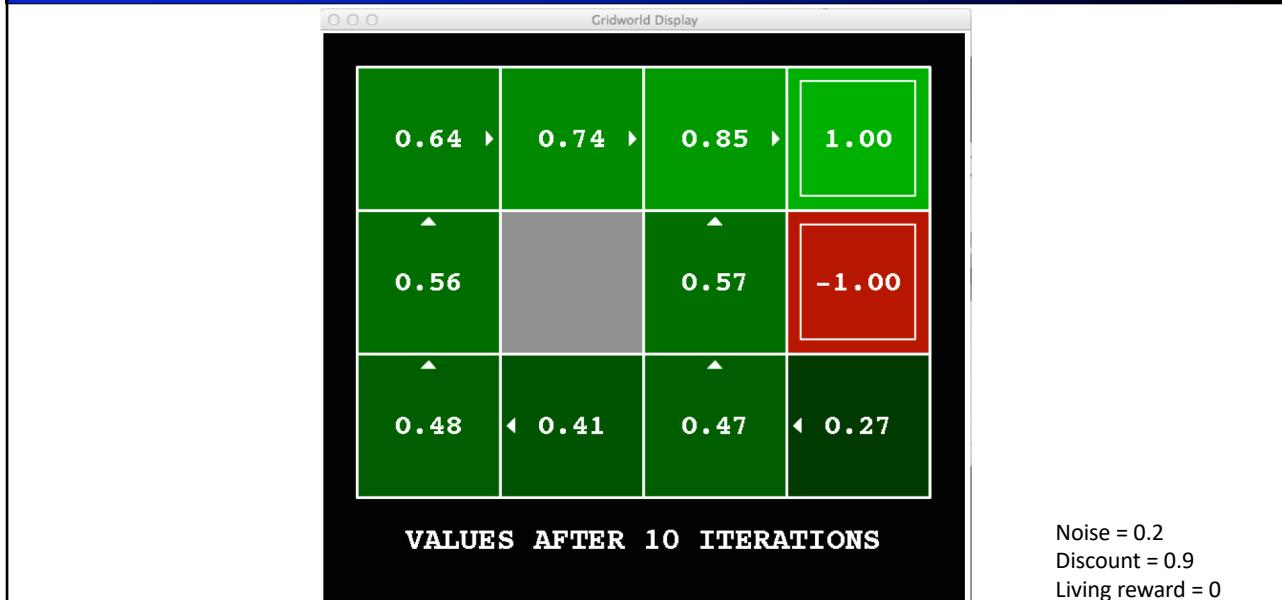
$k = \lambda$



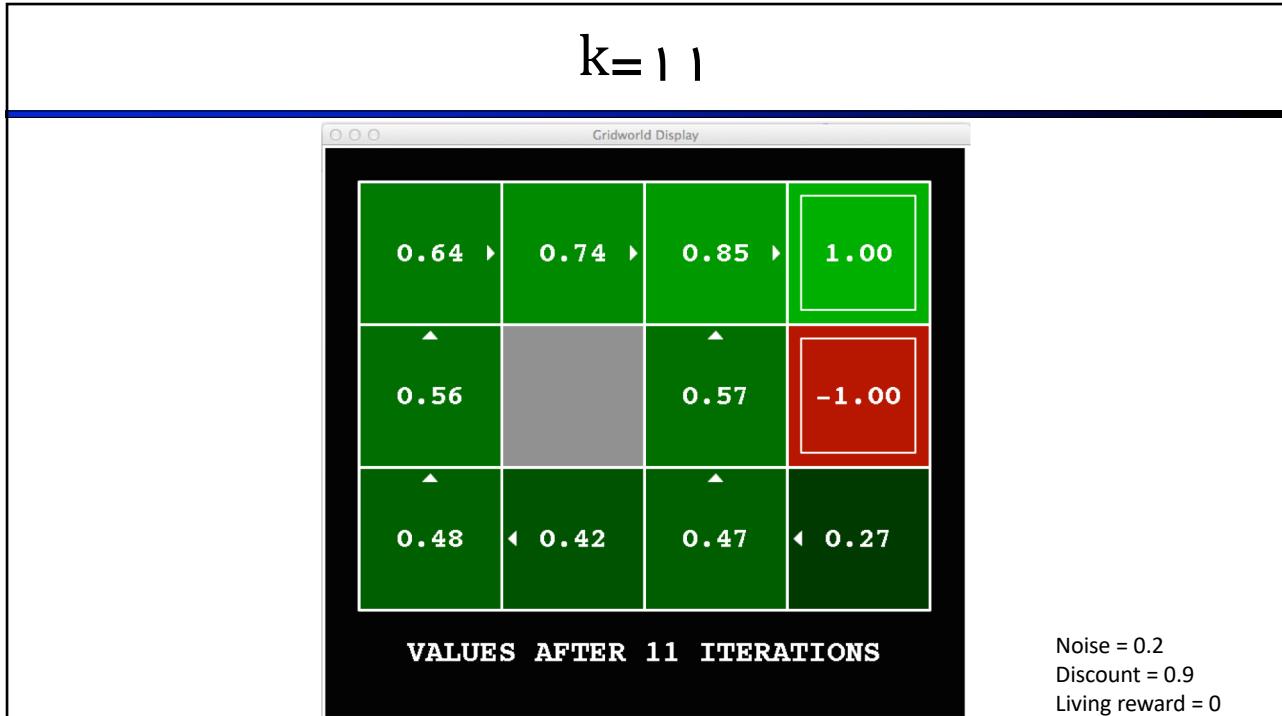
$k = \varphi$



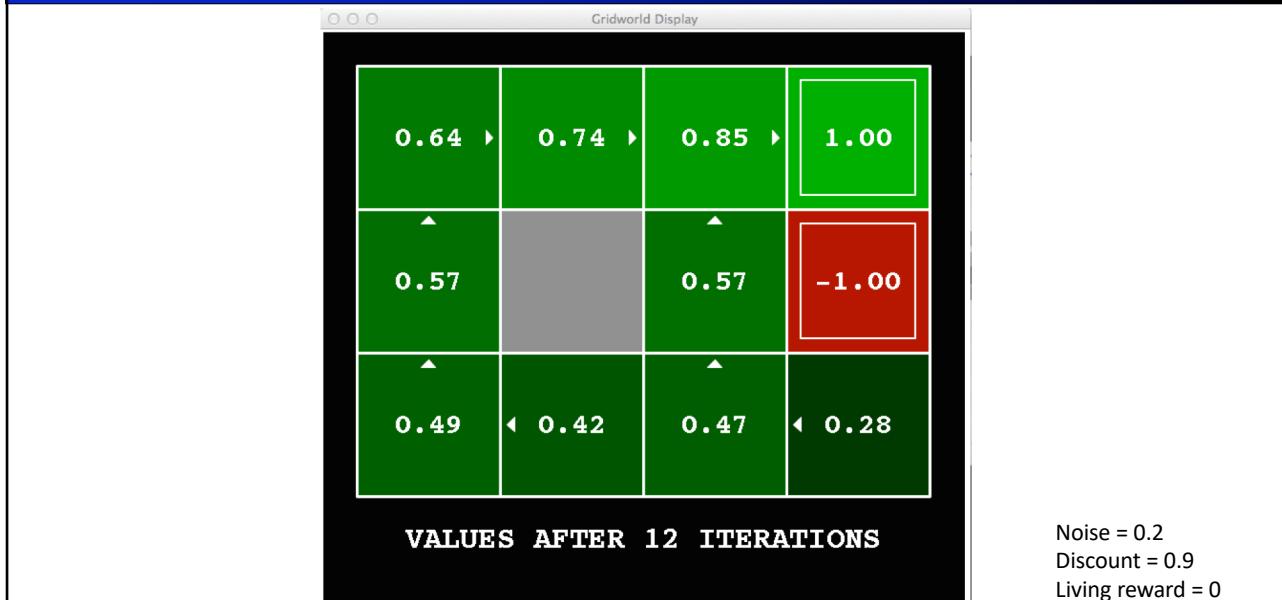
$k=1 \circ$



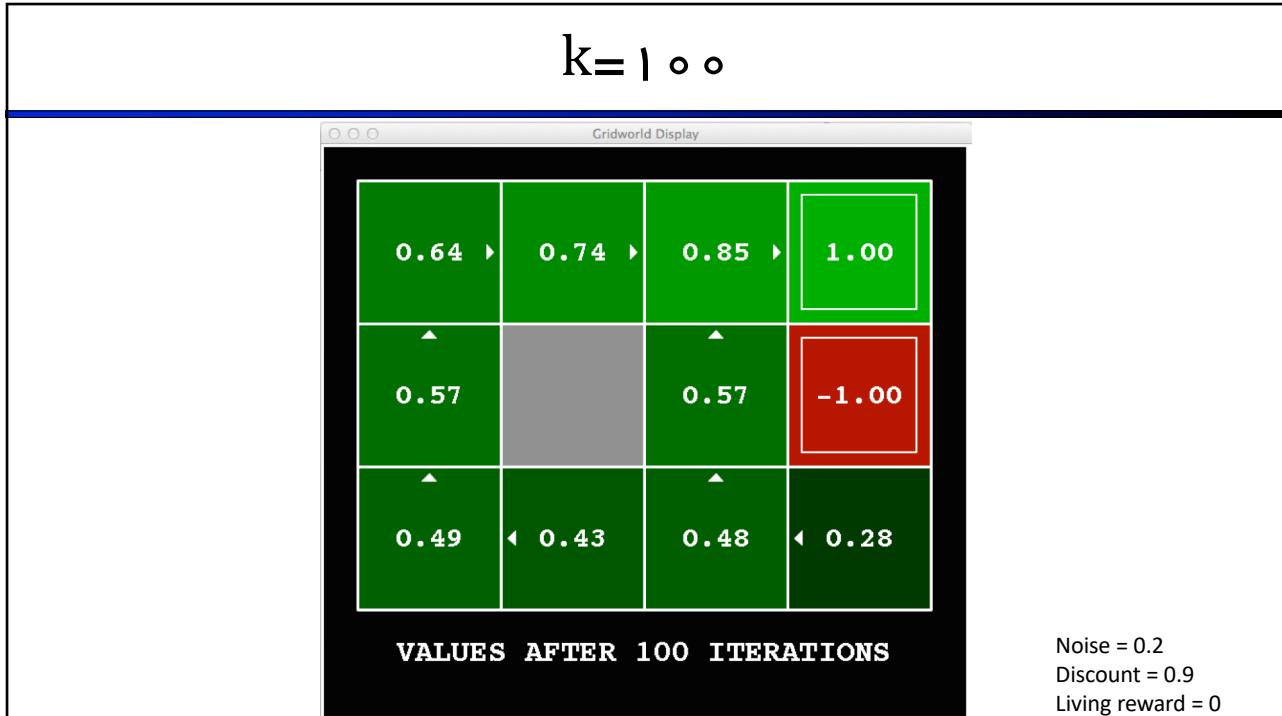
$k=1 \downarrow$



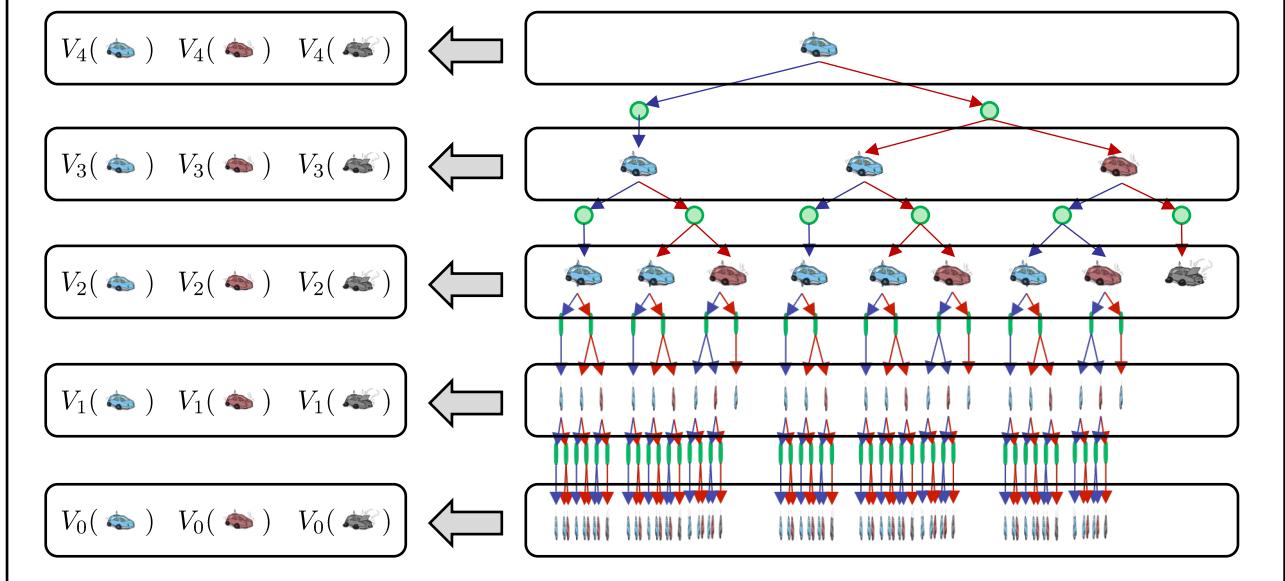
$k=1 \uparrow$



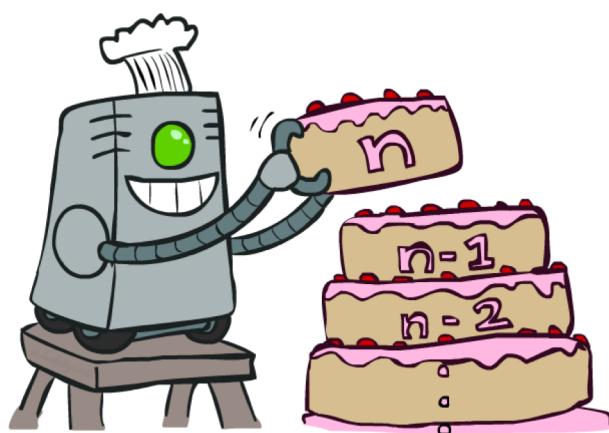
$k=1 \circ \circ$



محاسبه ارزش‌های زمان-محدود



روش Value Iteration



روش Value Iteration

شروع از: $V_0(s) = 0$

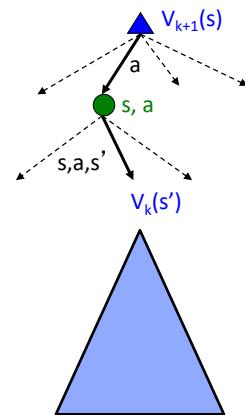
هیچ فرصتی برای حرکت نیست، پس مجموع ارزش‌ها/امتیازها صفر است

با داشتن بردار $V_k(s)$
اکسپکتیمکس را در عمق یک (یک حرکت) اجرا کن

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

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V_k(s')]$$

تا همگرایی ادامه بده

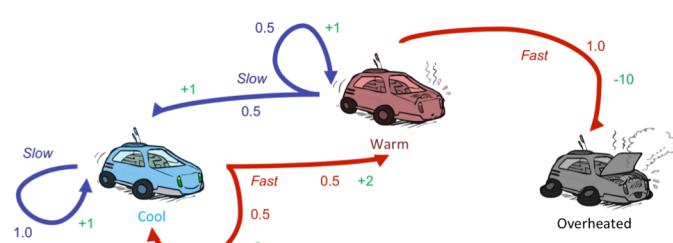


تمرین: Value Iteration

V_2	3.5	2.5	0

V_1	2	1	0

V_0	0	0	0

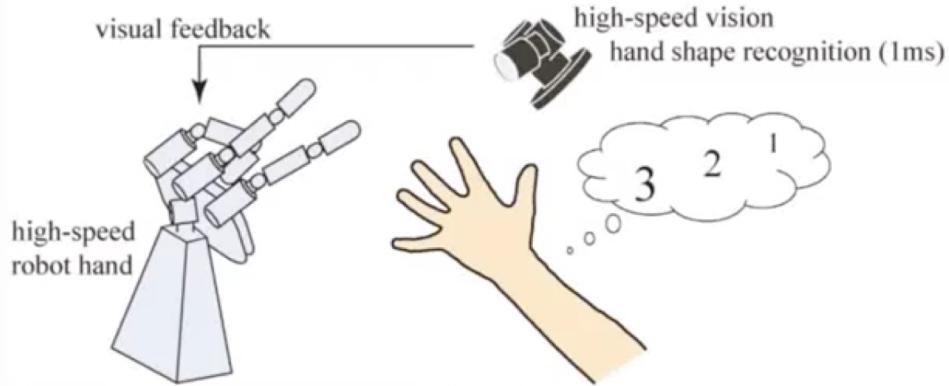


بدون در نظر گرفتن تخفیف

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V_k(s')]$$

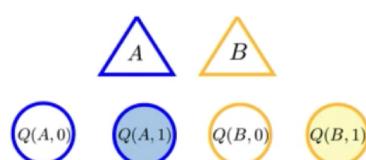
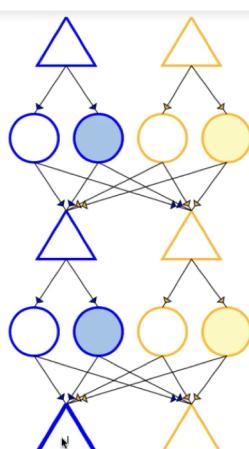
سنگ کاغذ قیچی!

Janken (rock-paper-scissors) Robot with 100% winning rate



Ishikawa Oku Laboratory
The University of Tokyo

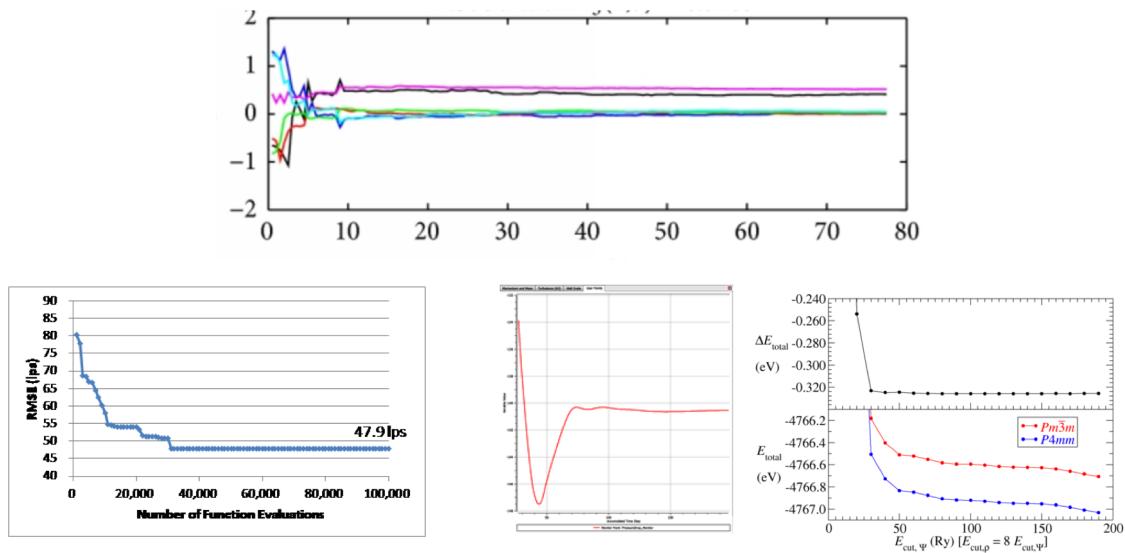
تمرين



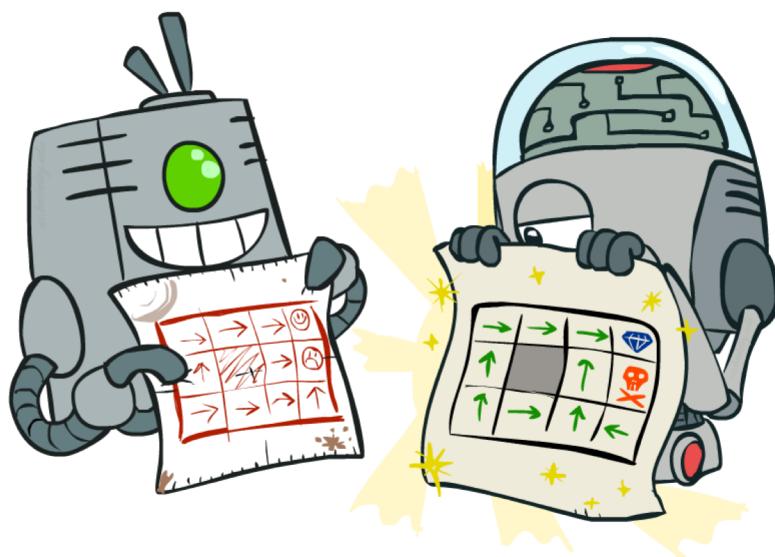
s	a	s'	$T(s, a, s')$	$R(s, a, s')$
A	0	A	0.50	2.0
A	0	B	0.50	-1.0
A	1	A	0.50	1.0
A	1	B	0.50	2.0
B	0	A	0.0	-2.0
B	0	B	1.0	-1.0
B	1	A	0.10	-3.0
B	1	B	0.90	-1.0

<https://www.youtube.com/watch?v=A5oqQDNntYc>

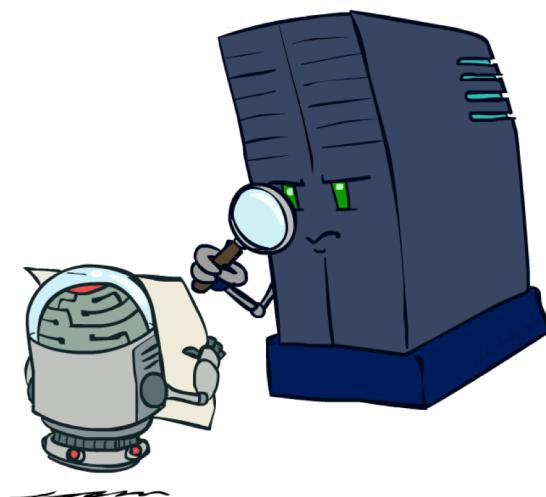
توضیح: همگرایی - Convergence



انتخاب راهبرد

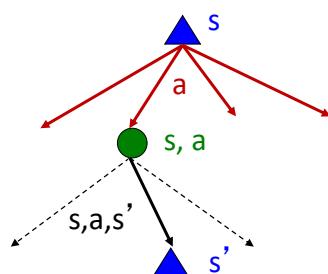


ارزیابی راهبرد

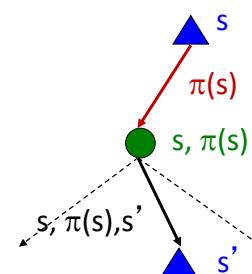


راهبرد مشخص – Fixed policy

Do the optimal action



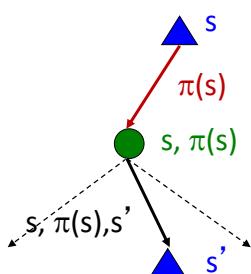
Do what π says to do



در درخت اکسپتیمیکس برای محاسبه
ارزش بهینه s ، ما کزیم تمامی اکشن‌ها را
پیدا می‌کردیم

در صورتی که یک راهبرد $\pi(s)$ انتخاب
کنیم: درخت ساده‌تر با فقط یک
اکشن

سودمندی برای یک راهبرد مشخص



ارزش s را چگونه محاسبه کنیم؟ ▪

سودمندی حالت s با داشتن راهبرد π : ▪

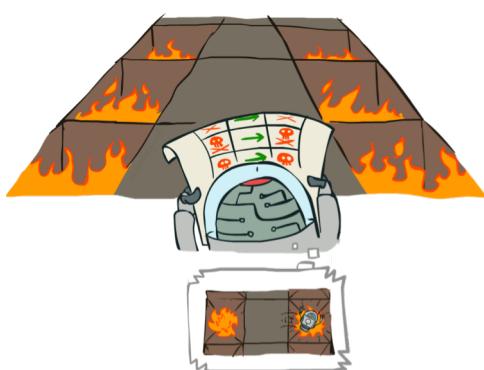
$V^\pi(s)$ = expected total discounted rewards starting in s and following π

▪ Recursive relation (Bellman equation):

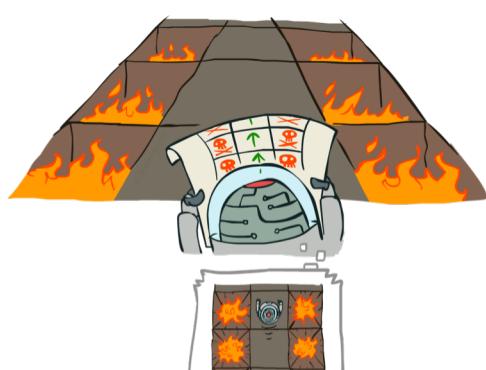
$$V^\pi(s) = \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V^\pi(s')]$$

مثال: ارزیابی راهبرد

Always Go Right



Always Go Forward



مثال: ارزیابی راهبرد

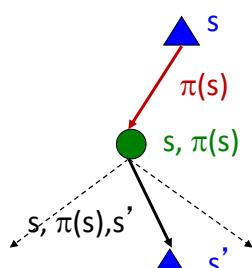
Always Go Right

-10.00	100.00	-10.00
-10.00	1.09	→ -10.00
-10.00	-7.88	→ -10.00
-10.00	-8.69	→ -10.00

Always Go Forward

-10.00	100.00	-10.00
-10.00	70.20	↑ -10.00
-10.00	48.74	↑ -10.00
-10.00	33.30	↑ -10.00

ارزیابی یک راهبرد



▪ مقادیر ۷ را چطور برای یک راهبرد معین محاسبه کنیم؟

▪ روشن اول: مشابه : value iteration

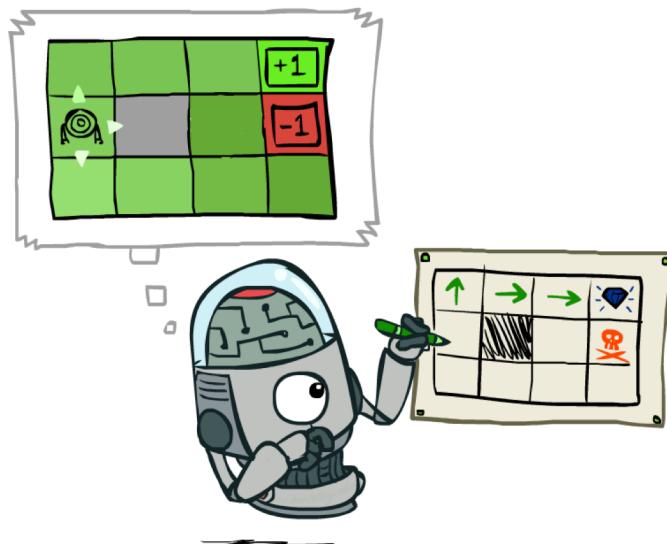
$$V_0^\pi(s) = 0$$

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

▪ Efficiency: $O(S^2)$ per iteration

▪ روشن دوم: بدون \max ها، می‌توان با استفاده از معادلات خطی حل کرد:
▪ مثلاً با متلب

پیدا کردن یک راهبرد



محاسبه action ها بر مبنای ارزشها

0.95	0.96	0.98	1.00
0.94		0.89	-1.00
0.92	0.91	0.90	0.80

فرض کنید ارزش‌های یهینه را برای تمامی حالت داریم: $V^*(s)$

حال، چطور عمل کنیم؟

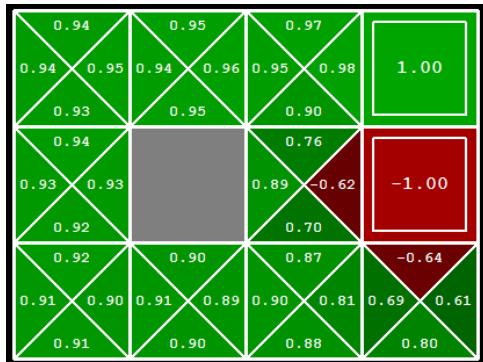
▪ باید برای هر قدم یک اکسپکتیمکس اجرا کنیم:

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

به این روش **policy extraction** می‌گویند: استخراج یک راهبرد از ارزش‌های یهینه

محاسبه action مبنای مقادیر Q

- فرض کنید مقادیر بهینه Q را برای تمامی حالت داریم



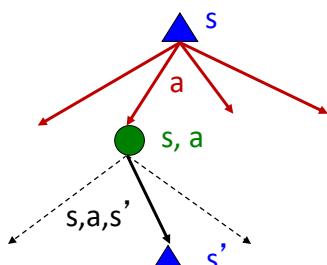
در این صورت چه کنیم؟

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

- محاسبه راهبرد از مقادیر Q ساده‌تر است!

مشکلات Value Iteration

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V_k(s')]$$

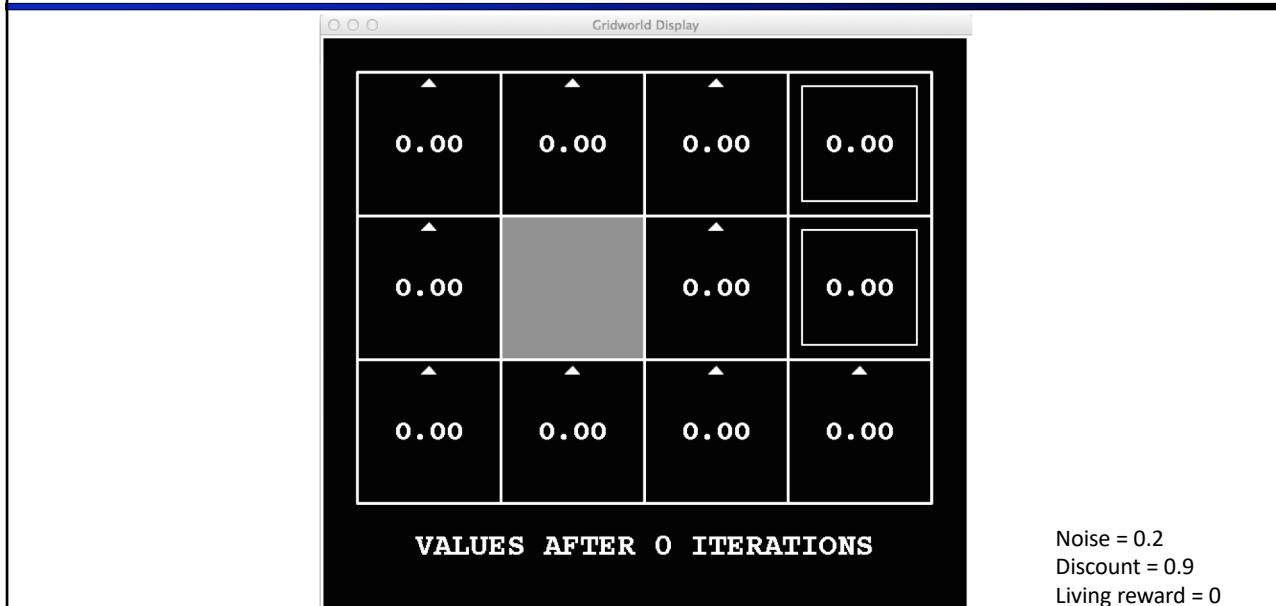


- مشکل ۱: کند است!

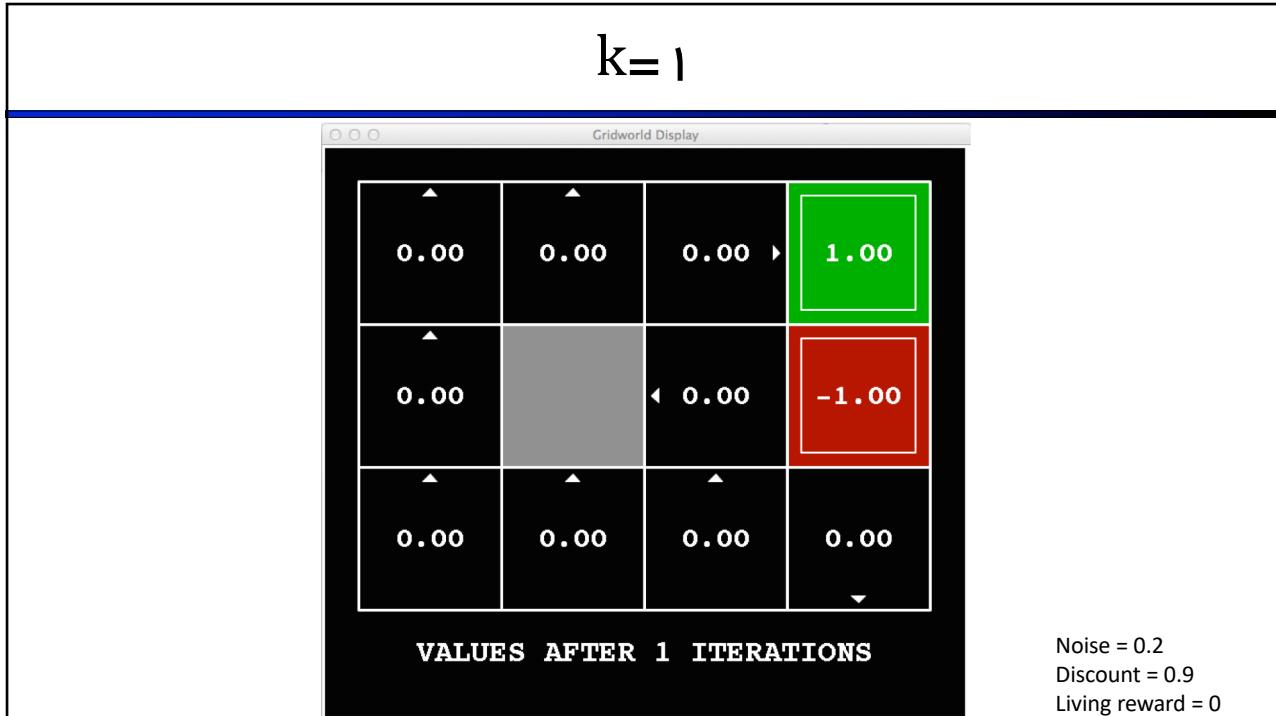
- مشکل ۲: مقدار "max" برای هر حالت به ندرت عوض می‌شود

- مشکل ۳: راهبرد خیلی قبل تراز مقادیر به همگرایی می‌رسد (converge می‌کند)

$k=0$



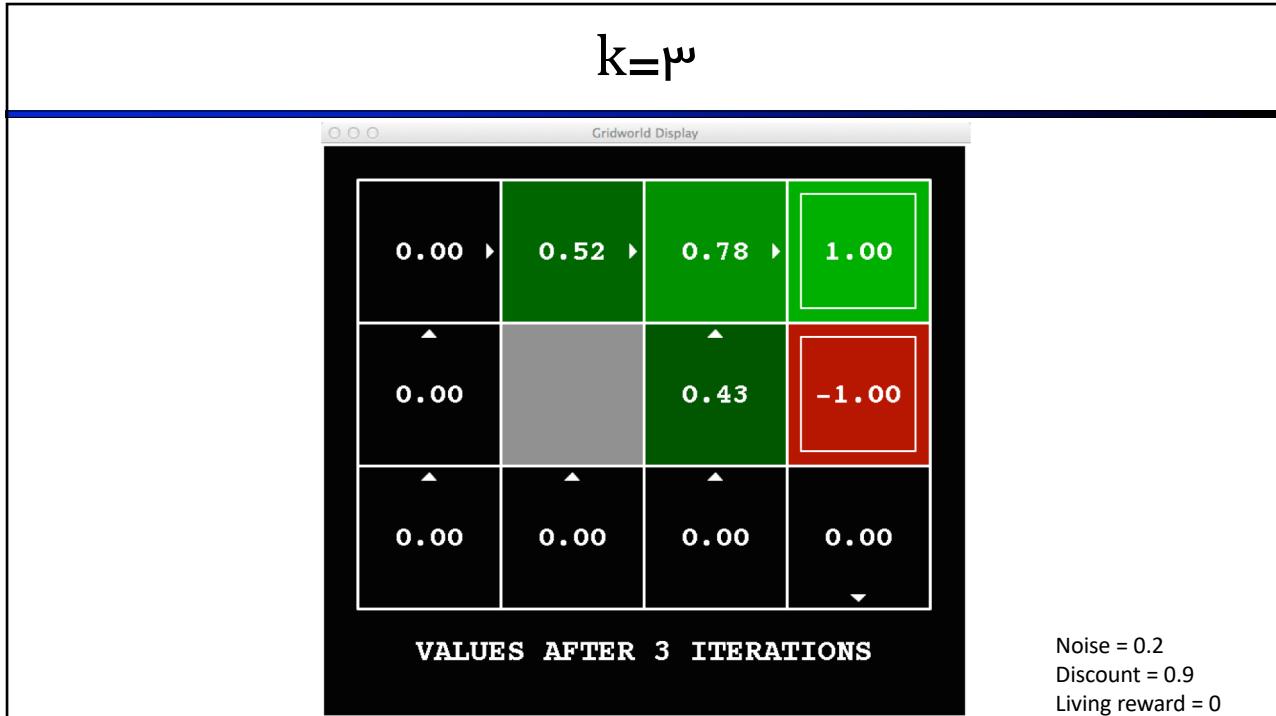
$k=1$



$k=\mu$



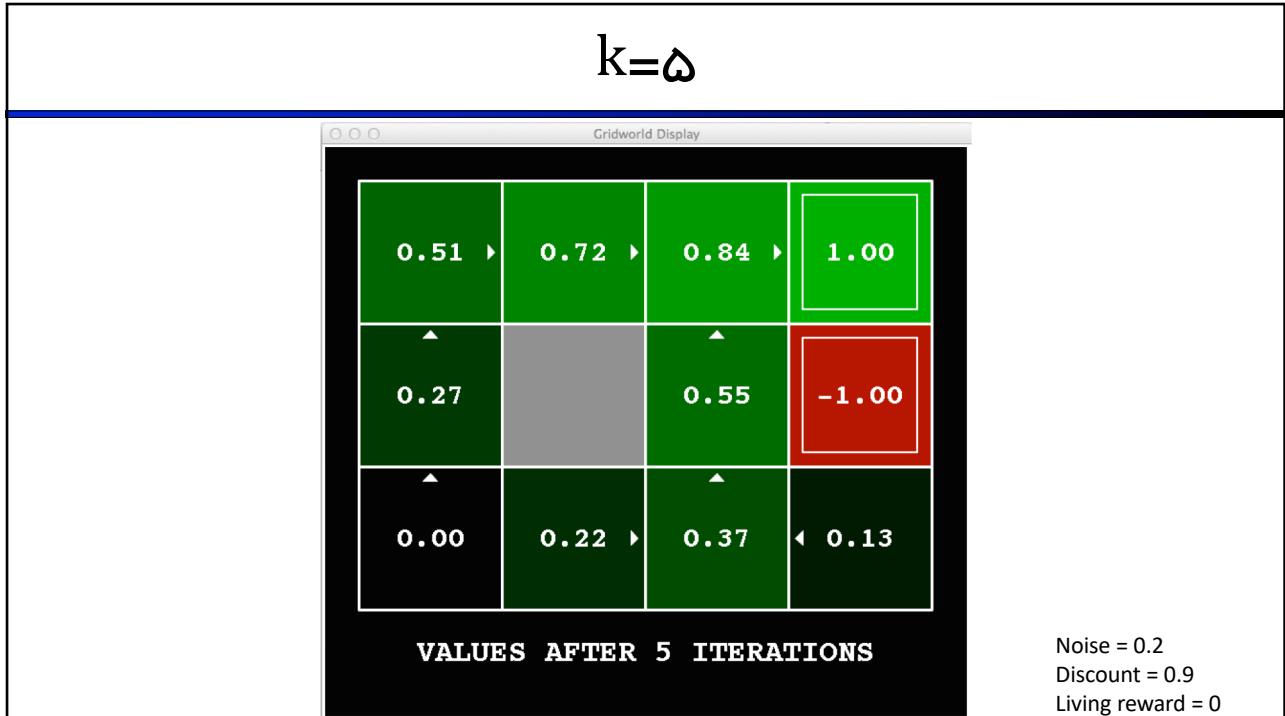
$k=\omega$



$k=\gamma$



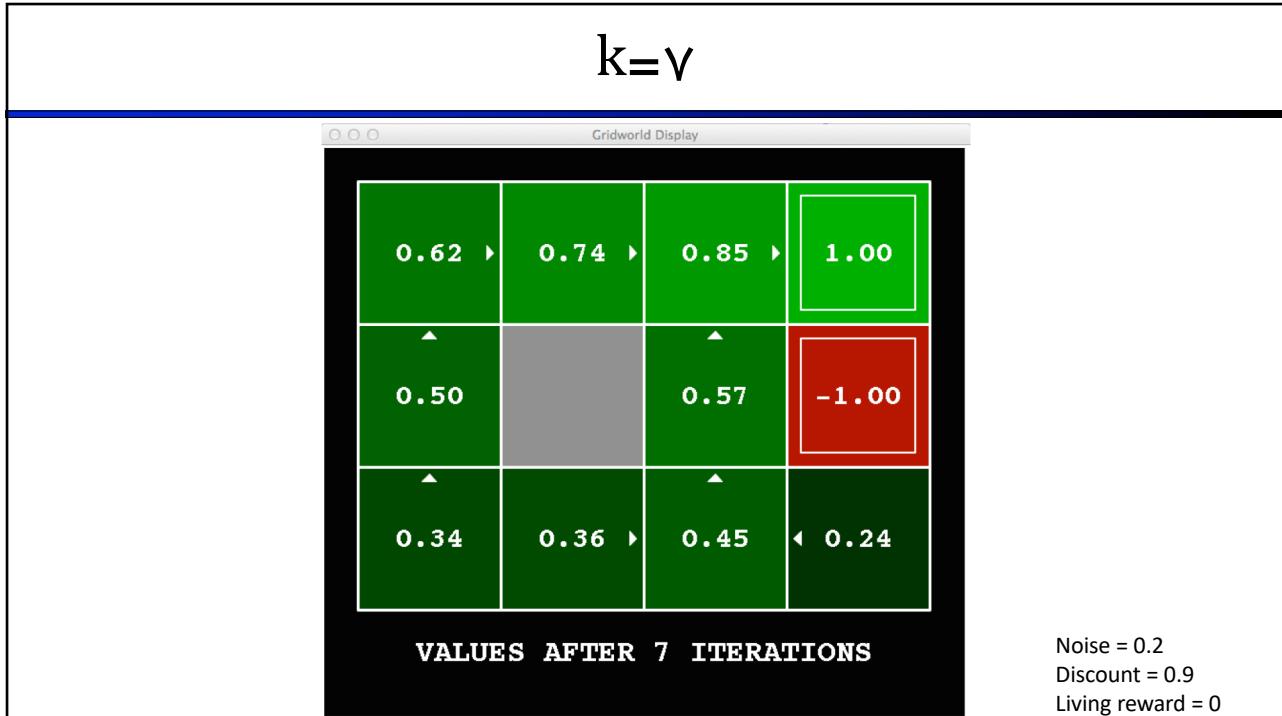
$k=\omega$



$k = \varsigma$



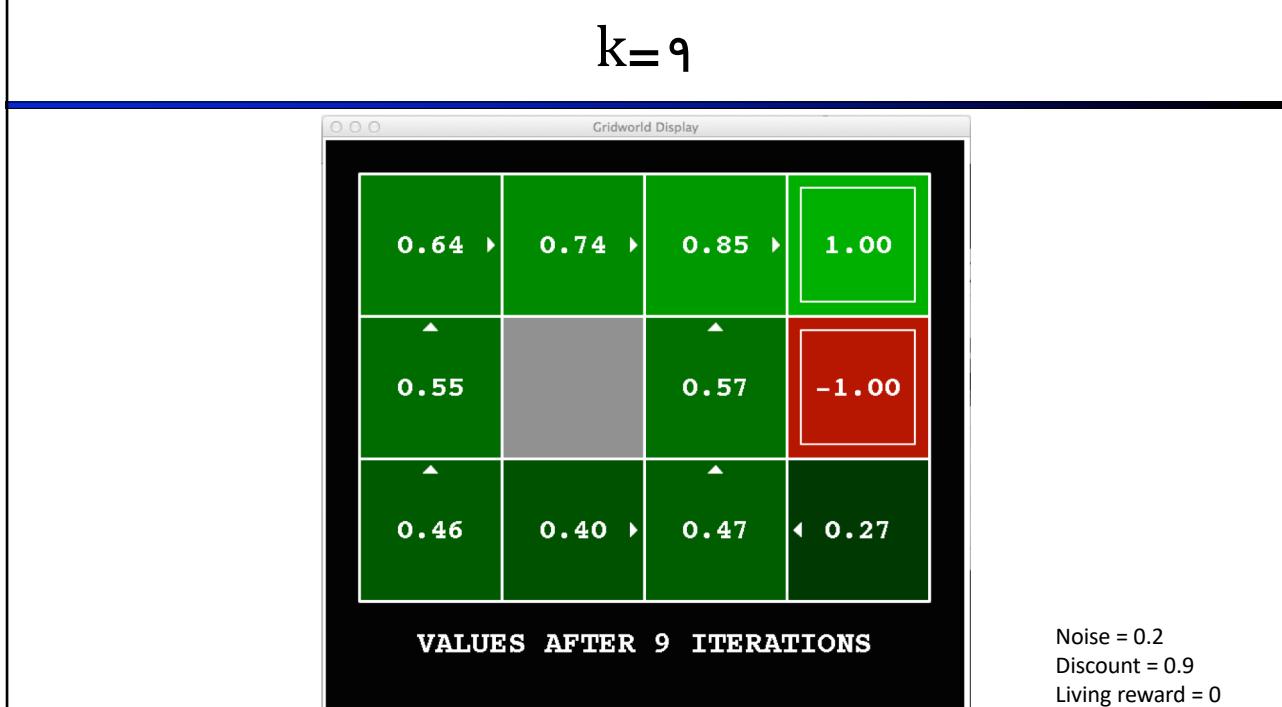
$k = \gamma$



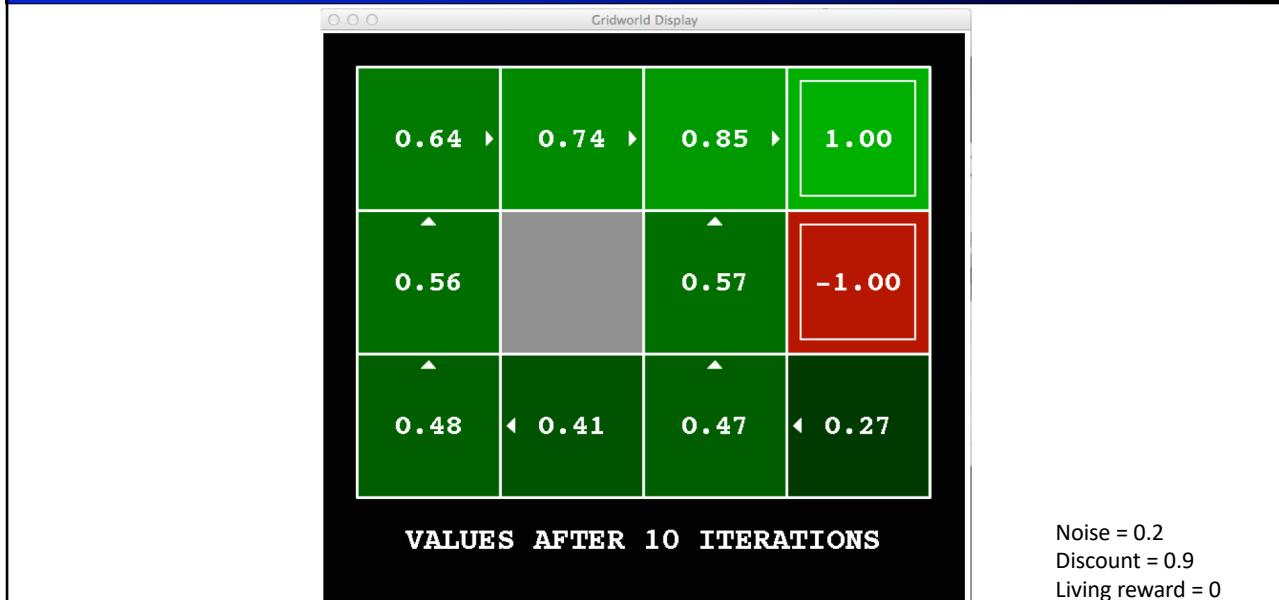
$k = \Lambda$



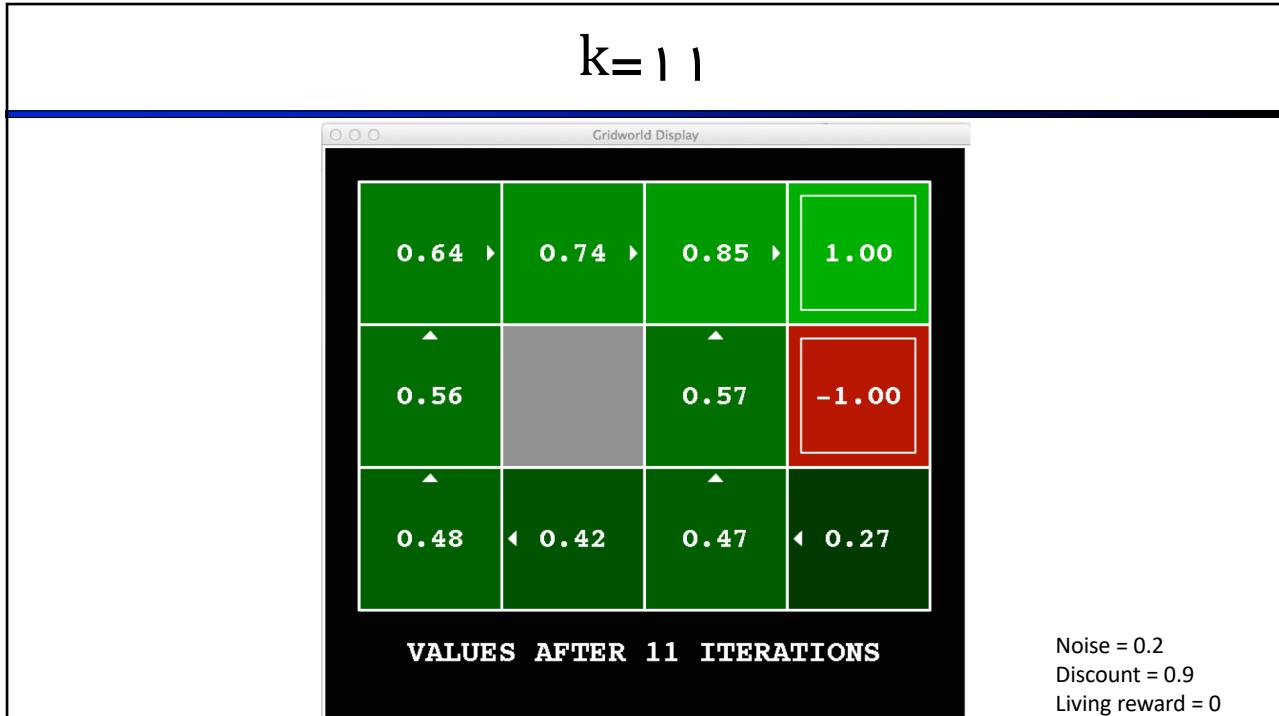
$k = \vartheta$



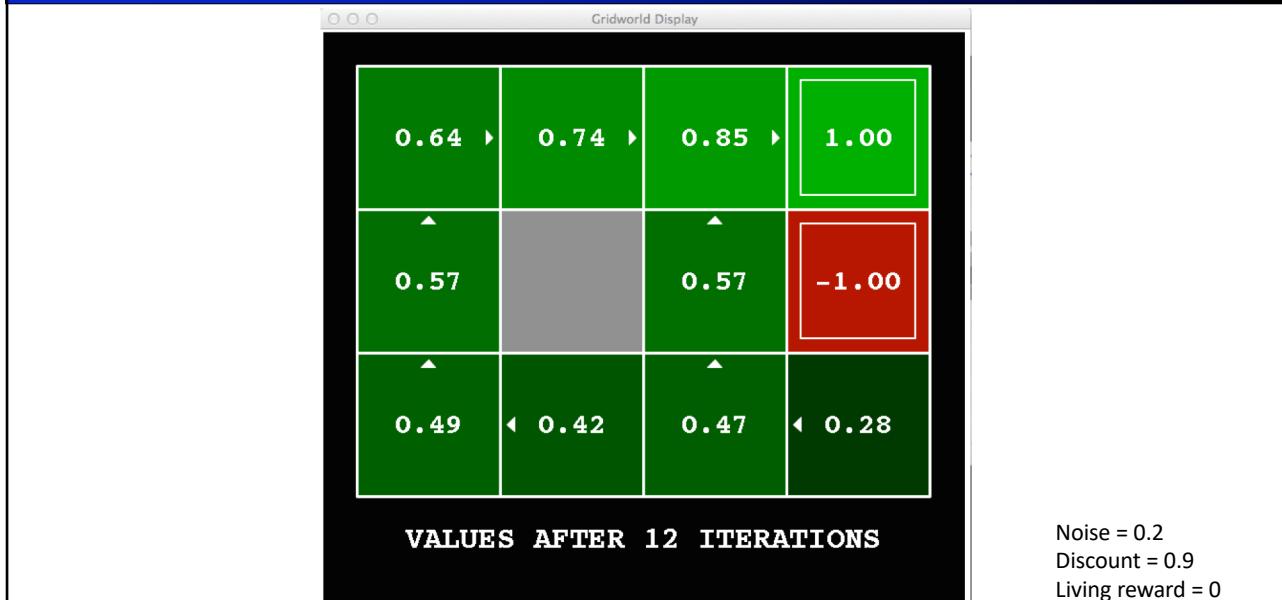
$k=1 \circ$



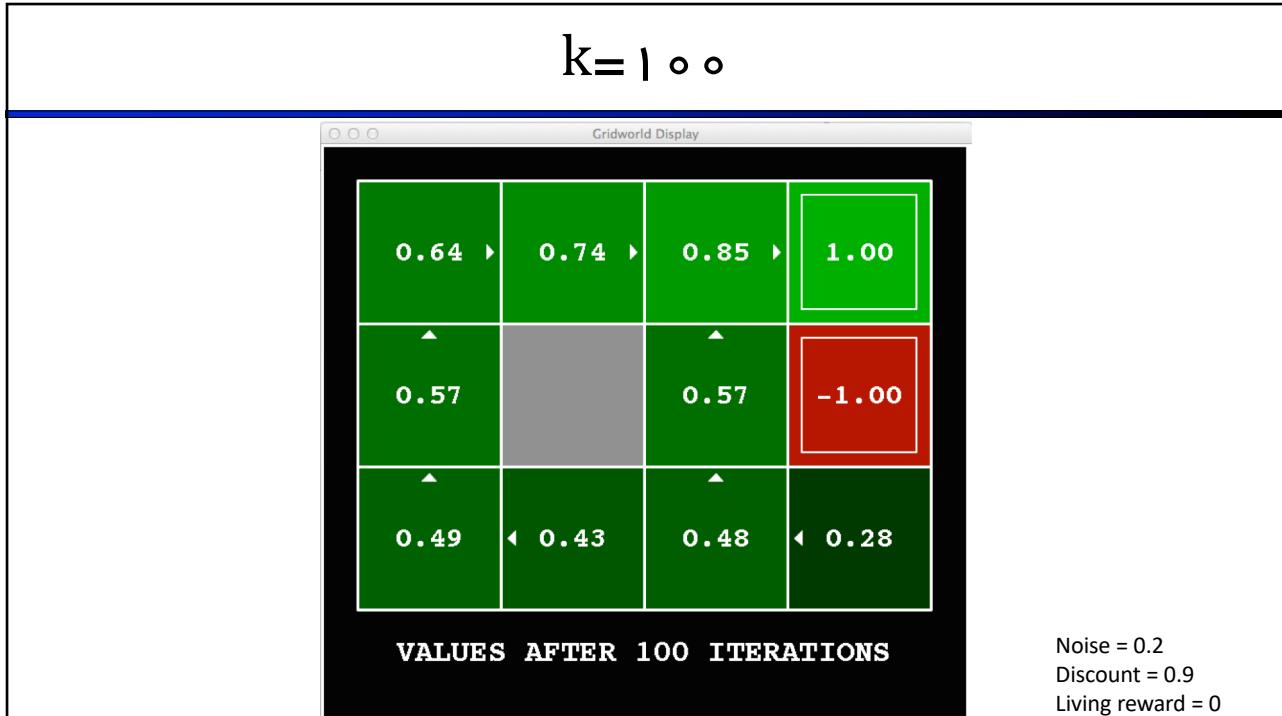
$k=1 \downarrow$



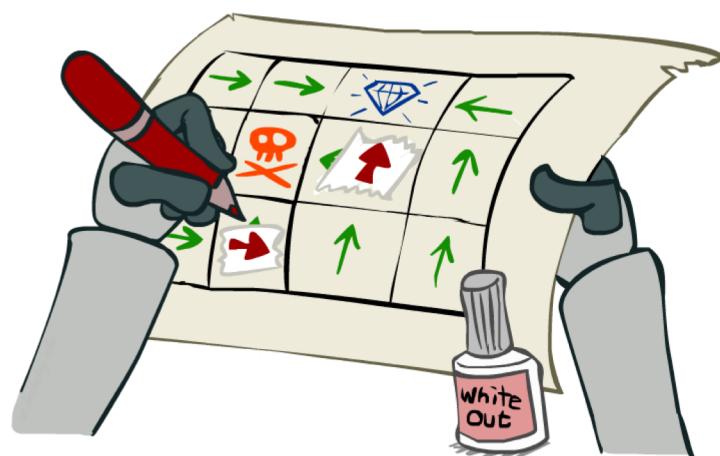
$k=1 \uparrow$



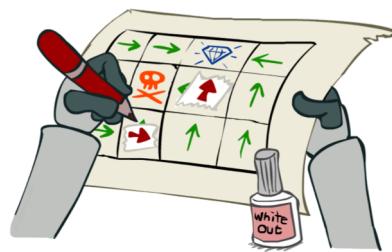
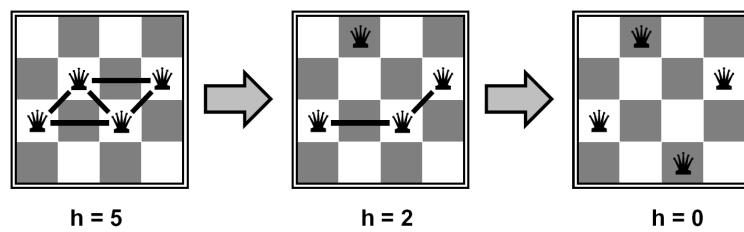
$k=1 \circ \circ$



Policy Iteration



Policy Iteration



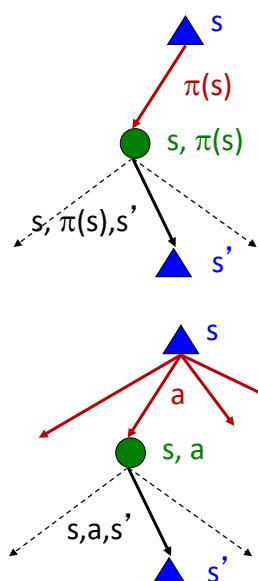
Policy Iteration

یک الگوریتم دیگر برای محاسبه مقادیر بهینه:

- **گام اول - ارزیابی راهبرد:** یک راهبرد ثابت (نه الزاماً بهینه) را در نظر بگیر. ارزش حالات را برای آن محاسبه کن
- **گام دوم - بهبود راهبرد:** راهبرد را یک قدم پیش ببر و تخمین بهتری از مقادیر بدست آور.
- این کار را تا همگرایی ادامه بده

روش بهینه است و در عین حال می‌تواند به مراتب سریع‌تر باشد

Policy Iteration



▪ **گام اول - ارزیابی راهبرد:** یک راهبرد ثابت (نه الزاماً بهینه) را در نظر بگیر. سودمندی‌ها را برای آن محاسبه کن

$$V_{k+1}^{\pi_i}(s) \leftarrow \sum_{s'} T(s, \pi_i(s), s') [R(s, \pi_i(s), s') + \gamma V_k^{\pi_i}(s')]$$

▪ **گام دوم - بهبود راهبرد:** با استفاده از ارزش‌های بالا، راهبرد جدیدی به دست آور.

▪ سپس، ارزش‌ها را برای این راهبرد محاسبه کن

$$\pi_{i+1}(s) = \arg \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^{\pi_i}(s')]$$

Value Iteration vs. Policy Iteration

هر دو روش یک چیز را محاسبه می‌کنند (ارزش‌های بهینه)

- در value iteration
 - در هر تکرار، ارزش‌ها آپدیت می‌شوند (و در نتیجه راهبرد)
- در policy iteration
 - چندین تکرار انجام می‌دهیم تا ارزش‌ها را برای یک راهبرد ثابت بیابیم
 - راهبرد را تنها در مرحله‌ی بهبود راهبرد تغییر می‌دهیم

هر دو روش dynamic programming برای حل MDP‌ها

خلاصه: الگوریتم‌های MDP

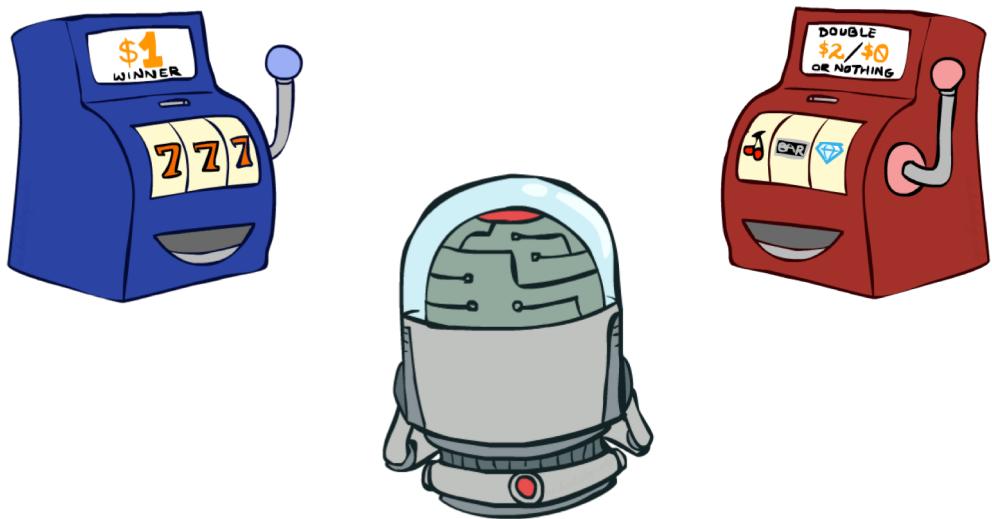
اگر بخواهیم

- ارزش‌های بهینه را محاسبه کنیم: value iteration or policy iteration
- ارزش‌ها را برای یک راهبرد خاص محاسبه کنیم: policy evaluation
- یک راهبرد بدست آوریم (با استفاده از ارزش‌ها): policy extraction

در حقیقت همه‌ی اینها یک چیز هستند!

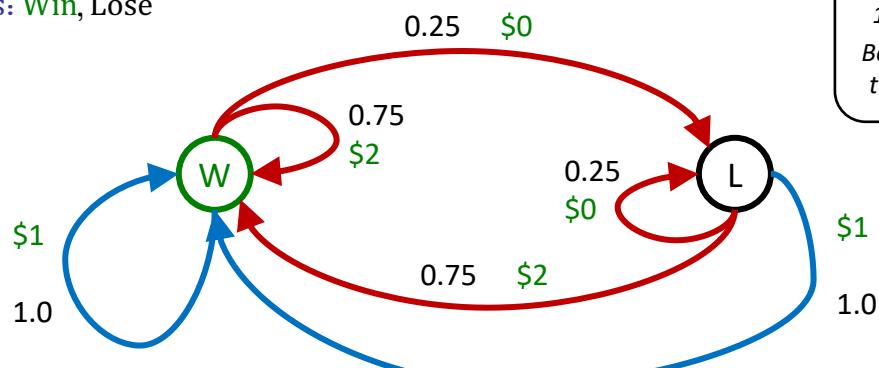
- همگی معادلات Bellman هستند که با استفاده از روابط بازگشتی محاسبه می‌کنند
- همگی از اکسپکتیمکس‌های کوچک استفاده می‌کنند
- تنها تفاوت: راهبرد ثابت یا ماقزیم کردن روی تمام action‌ها

میان پرده!



دیاگرام MDP

- Actions: Blue, Red
- States: Win, Lose

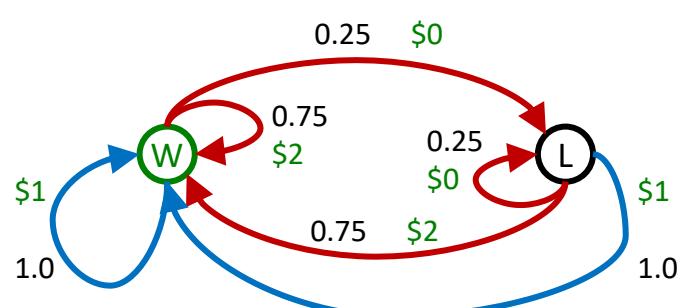


No discount
100 time steps
Both states have the same value

نقشه کشی پیش از بازی – Offline planning

No discount
100 time steps
Both states have
the same value

	Value
Play Red	150
Play Blue	100



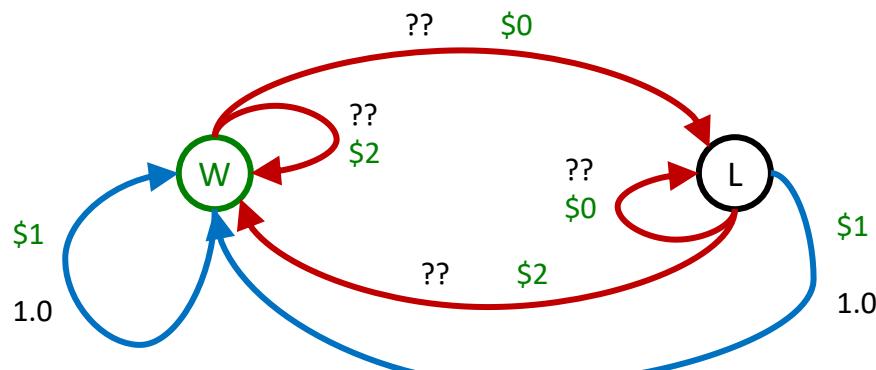
بازی کنیم!



\$2 \$2 \$0 \$2 \$2
\$2 \$2 \$0 \$0 \$0

نقشه کشی برخط – Online planning

- اگر جزئیات MDP را نداشتیم چطور؟ (بازی را به خوبی نمی‌شنایدیم)



بازی کنیم!



\$0 \$0 \$0 \$2 \$0
\$2 \$0 \$0 \$0 \$0

یادگیری تقویتی

▪ در شرایط جدید، دیگر نقشه کشی نمی‌کنیم، «یادگیری» داریم!



▪ Reinforcement learning

- دیگر نمی‌توان از پیش راهبرد بهینه را انتخاب کرد و نقشه کشید
- باید وارد بازی شد و اطلاعات جمع کرد

▪ ایده‌های مهم در یادگیری تقویتی:

- اکتشاف: باید action‌های مختلف را امتحان کنیم تا اطلاعات بدست آوریم
- بهره‌برداری: از آنچه تجربه کردی بهره ببر!
- حتی اگر یادگیری خوبی انجام دهی، ممکن است اشتباه کنی
- یادگیری به مراتب دشوارتر از نقشه کشیست

سوال؟

