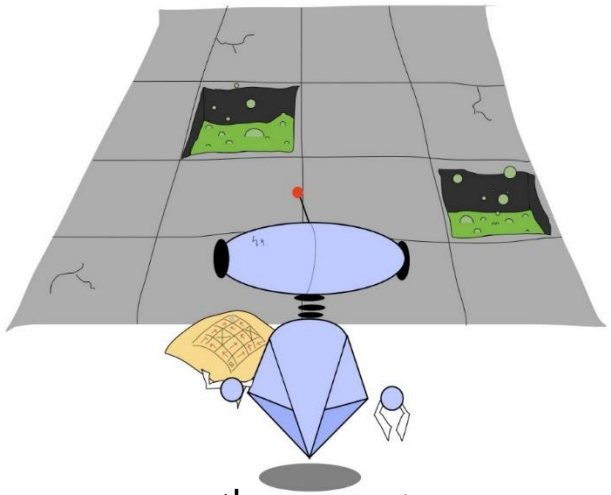
# مبانی و کاربردهای هوش مصنوعی

## فرایند تصمیم‌گیری مارکوف - 2 (فصل 17.1 الی 17.3)

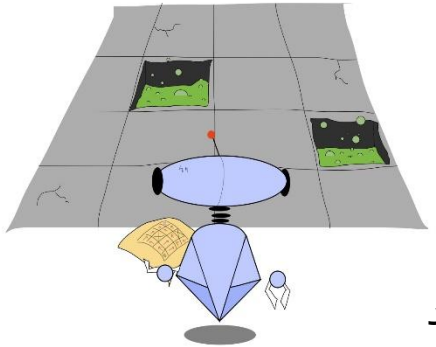


مدرس: مهدی جوانمردی

دانشکده مهندسی کامپیوتر دانشگاه صنعتی امیرکبیر

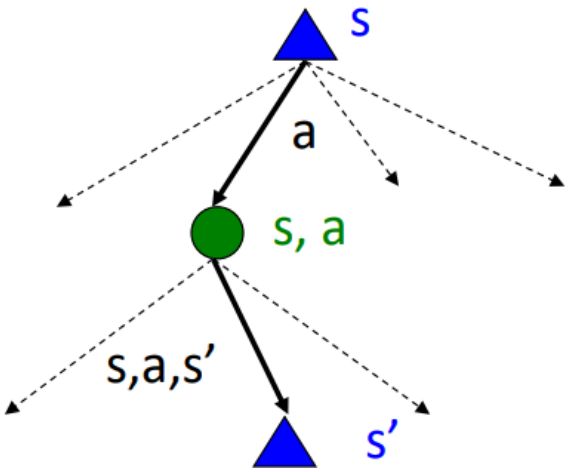
(الهام گرفته از محتوای درس هوش مصنوعی دانشگاه برکلی)

# مثال: جهان مشبک



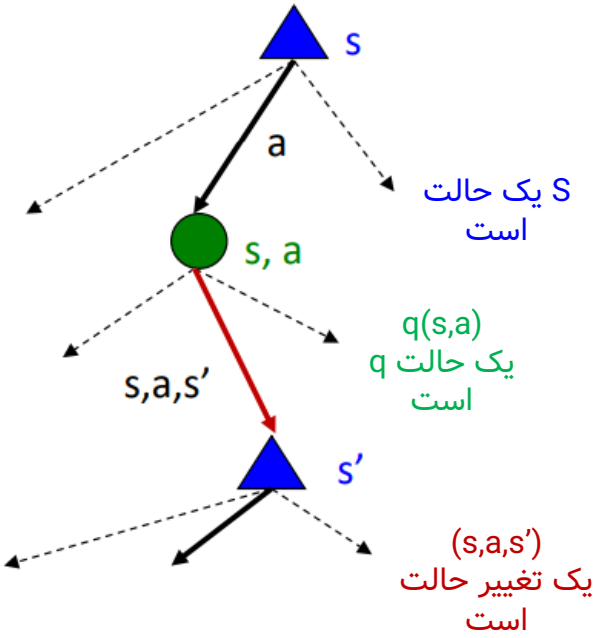
- یک مسئله ی هزارتو مانند
  - عامل در یک محیط مشبک زندگی می کند
  - دیوارها مسیر حرکت عامل را سد می کنند
- حرکت نویزدار: حرکات همیشه طبق نقشه پیش نمی روند
  - 80% مواقع ، حرکت شمال عامل را به شمال می برد(اگر دیوار نباشد)
  - 10% مواقع، شمال عامل را به غرب می برد; 10% شرق
  - اگر در جهتی که عامل انتخاب کرده بود دیوار باشد ، عامل سرجایش می ماند
- عامل هر گام زمانی پاداش می گیرد
  - پاداش کوچک "زنده بودن" در هر گام زمانی (می تواند منفی باشد)
  - پاداش بزرگ در آخر است (خوب یا بد)
- هدف: بیشینه کردن مجموع پاداش‌ها

# یادآوری: MDP ها




- فرآیندهای تصمیم گیری مارکوف:
  - حالت ها **S**
  - اعمال **A**
  - تغییر حالت‌ها $P(s' \mid s, a)$ (یا $T(s, a, s')$)
  - تابع پاداش $R(s, a, s')$ (و ضریب تخفیف $\gamma$)
  - حالت شروع $s_0$
- کمیت ها:
  - سیاست = نگاشت حالت‌ها به اعمال
  - سودمندی = مجموع (تخفیف یافته) پاداش‌ها
  - مقدارها = سودمندی مورد انتظار آینده از یک حالت (گره بیشینه)
  - مقدار Q = سودمندی مورد انتظار با شروع از یک حالت q (گره شانس)

# کمیت های بهینه




- مقدار (سودمندی) یک حالت s :

  V*(s) = سودمندی مورد انتظار با شروع از s و بهینه عمل کردن.

- مقدار (سودمندی) یک حالت q (s,a):

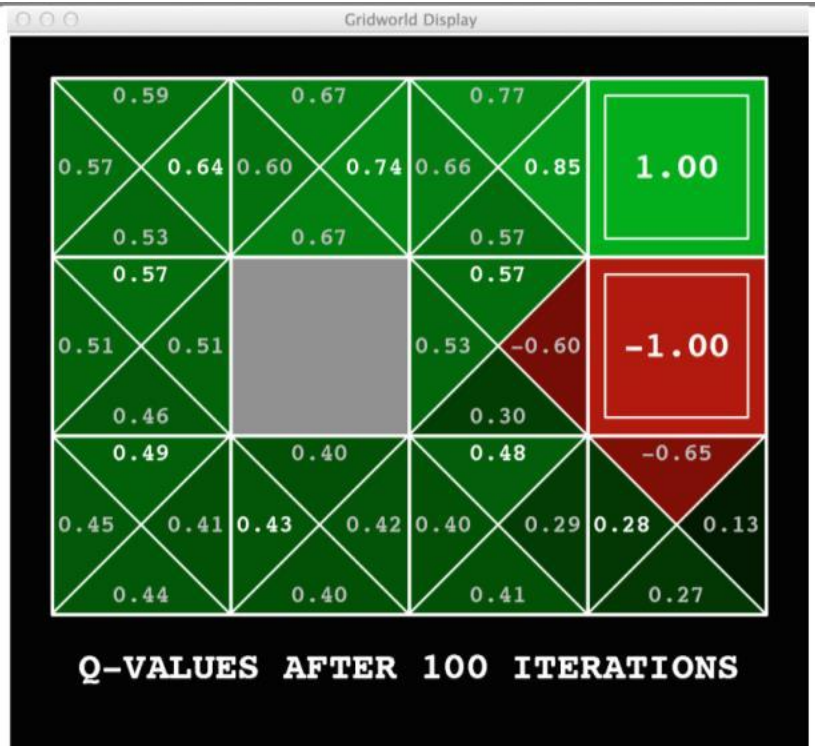
  Q*(s,a) = سودمندی مورد انتظاربا شروع از s و انتخاب a و (از آنجا به بعد) بهینه عمل کردن.

- سیاست بهینه:

  π *(s) = حرکت بهینه از حالت s

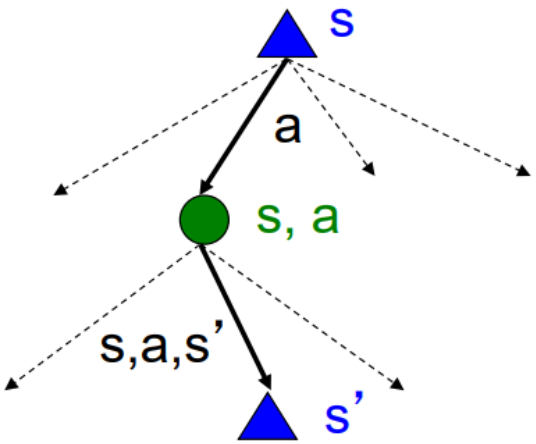
# مقادیر دنیای مشبک V*

# دنیای مشبک Q*


Gridworld Display
0.59
0.57
0.64
0.53
0.67
0.60
0.74
0.67
0.77
0.66
0.85
0.57
1.00
0.57
0.51
0.51
0.46
0.57
0.53
-0.60
0.30
-1.00
0.49
0.45
0.41
0.44
0.40
0.43
0.42
0.40
0.48
0.40
0.29
0.41
-0.65
0.28
0.13
0.27
Q-VALUES AFTER 100 ITERATIONS

# معادلات بلمن (The Bellman Equations)

# مقدار حالت‌ها




- عملیات پایه‌ای: محاسبه مقدار بهینه یک حالت $V^*(s)$
  - سودمندی مورد انتظار با حرکت بهینه
  - مجموع (تخفیف یافته) پاداش‌ها
  - این همان چیزیست که expectimax محاسبه می‌کند!

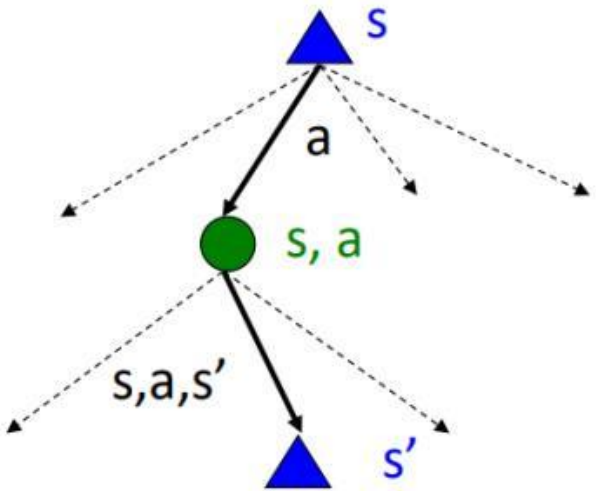
- تعریف بازگشتی مقدار

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

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

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

# معادلات بلمن




- تعریف "سودمندی بهینه" با استفاده از یک مرحله expectimax یک رابطه نگاه به جلوی ساده یک مرحله‌ای بین مقادیر بهینه می دهد.

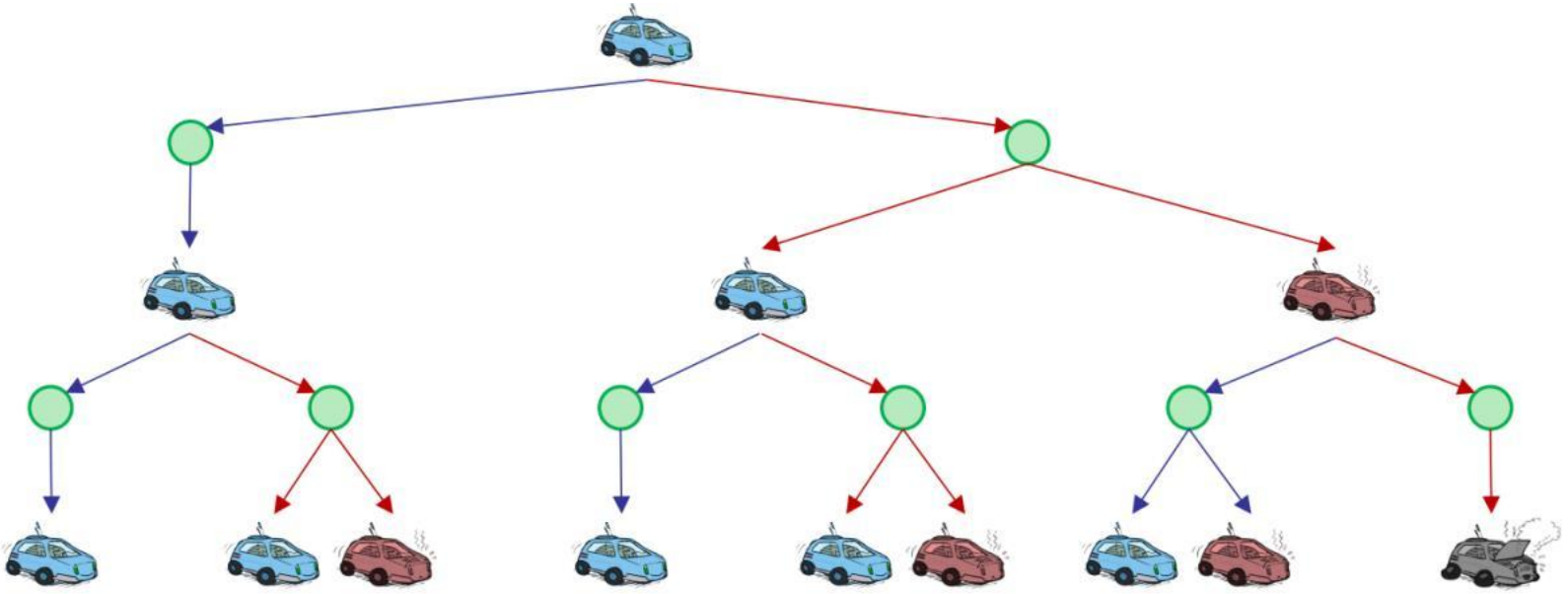
$$V^*(s) = \max_a Q^*(s,a)$$

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

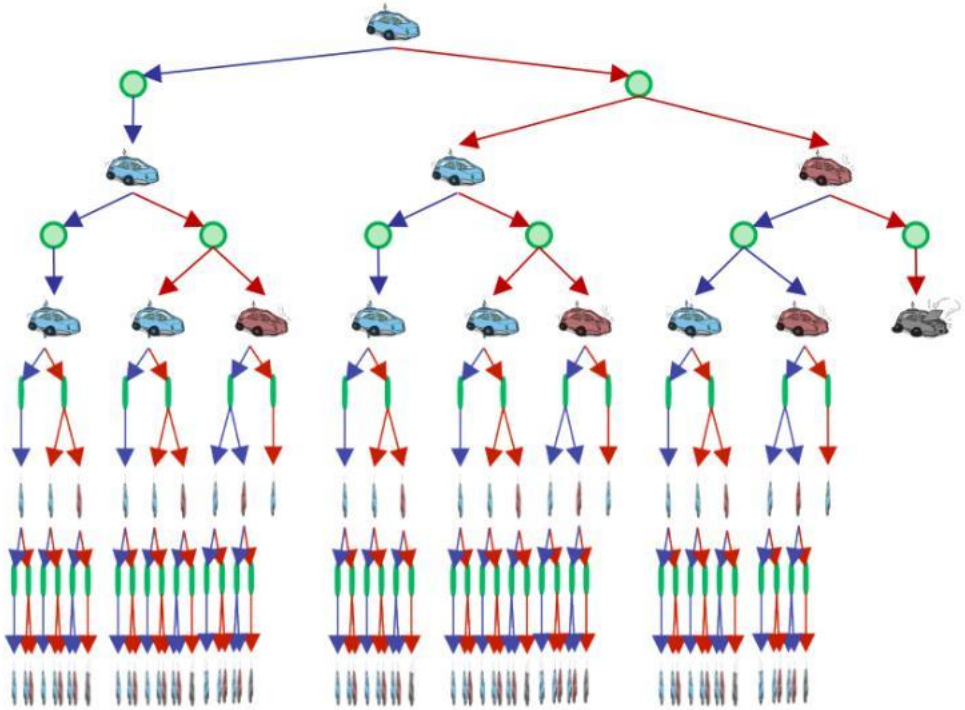
$$V^*(s) = \max_a \sum_{'} T(s,a,s') \left[R(s,a,s') + \gamma\, V^*(s')\right]$$

- این ها معادلات بلمن هستند، و مقادیر بهینه را به گونه ای توصیف می کنند که ما به وفور استفاده خواهیم کرد

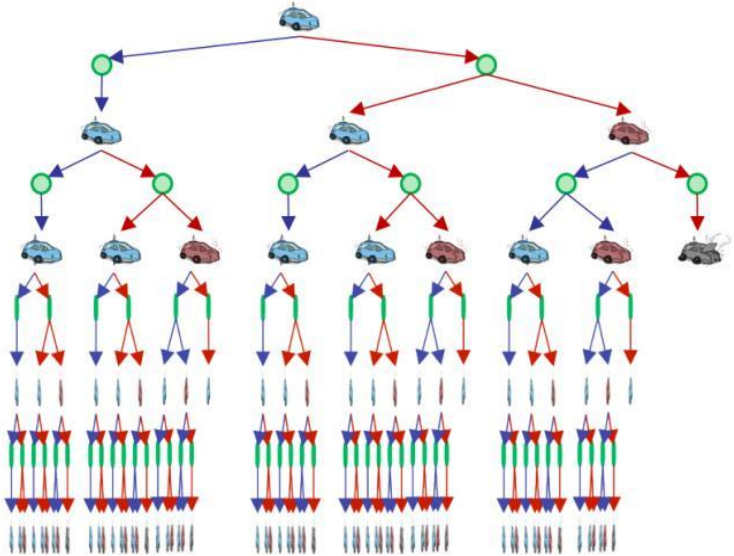
# درخت جستجوی مسابقه



# درخت جستجوی مسابقه



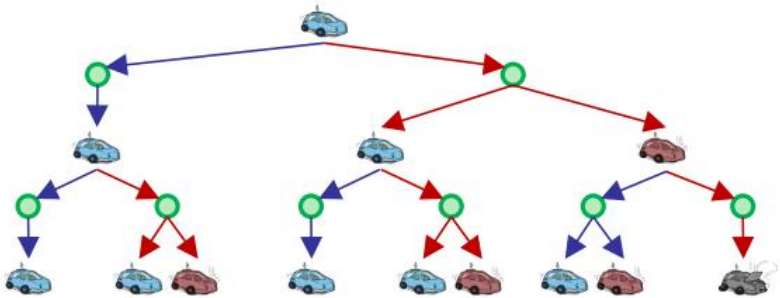
# درخت جستجوی مسابقه

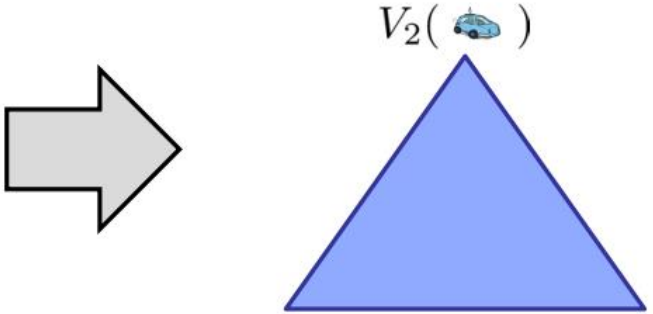


- با استفاده از expectimax ما بیش از اندازه کار می کنیم!
- مشکل: حالت‌ها تکرار میشوند
  - ایده : مقادیر مورد نیاز را فقط یکبار محاسبه کن
  - راه حل: برنامه نویسی پویا
- مشکل: درخت تا بینهایت ادامه دارد
  - ایده : از محاسبات با عمق محدود استفاده کن و آنقدر ادامه بده تا تغییرات کوچک شود
  - نکته : اگر $\gamma < 1$ قسمت‌های عمیق درخت در نهایت اهمیتی ندارند

# مقدار زمان محدود

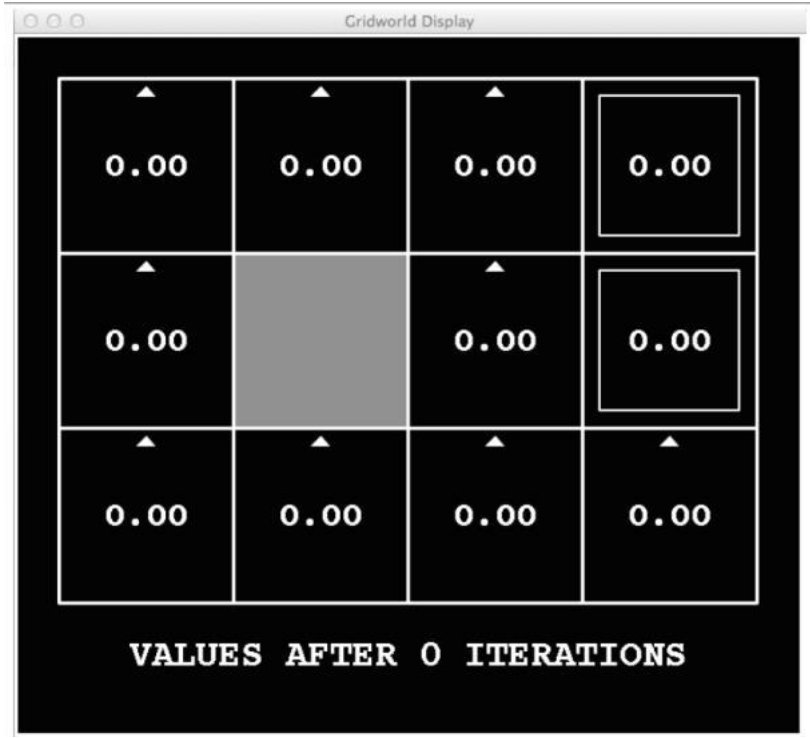
- ایده کلیدی: مقدار زمان محدود (time-limited values)
- تعریف $V_k(s)$ به عنوان مقدار بهینه s اگر بازی پس از k مرحله‌ی دیگر خاتمه یابد
  - معادل اجرای expectimax با عمق محدود k





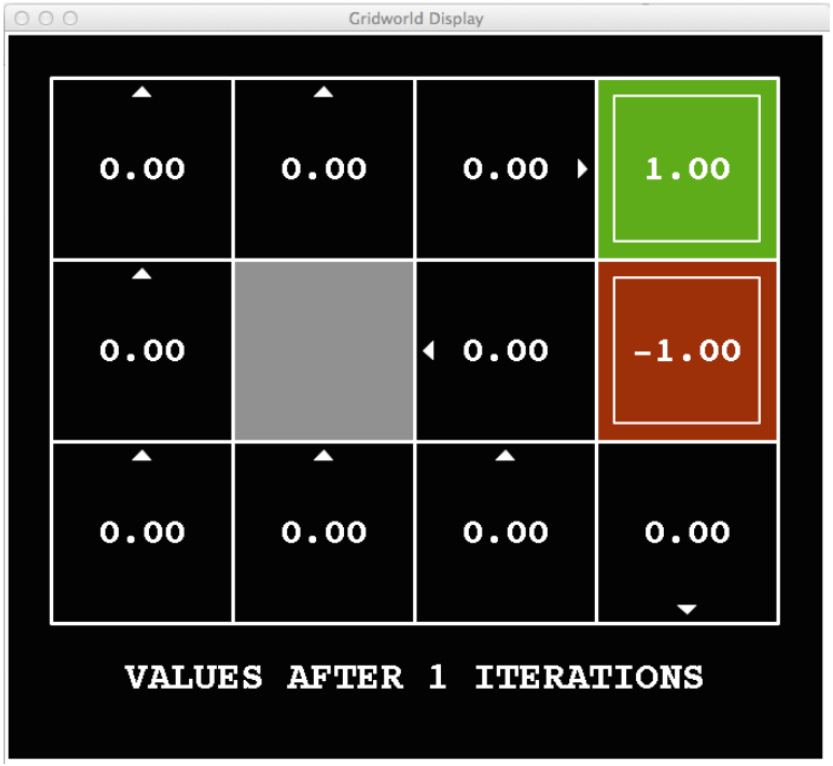

[Demo – time-limited values (L8D6)]

# k = 0




Noise = 0.2
Discount = 0.9
Living reward = 0

# k = 1




Noise = 0.2
Discount = 0.9
Living reward = 0

# k = 2


Noise = 0.2
Discount = 0.9
Living reward = 0

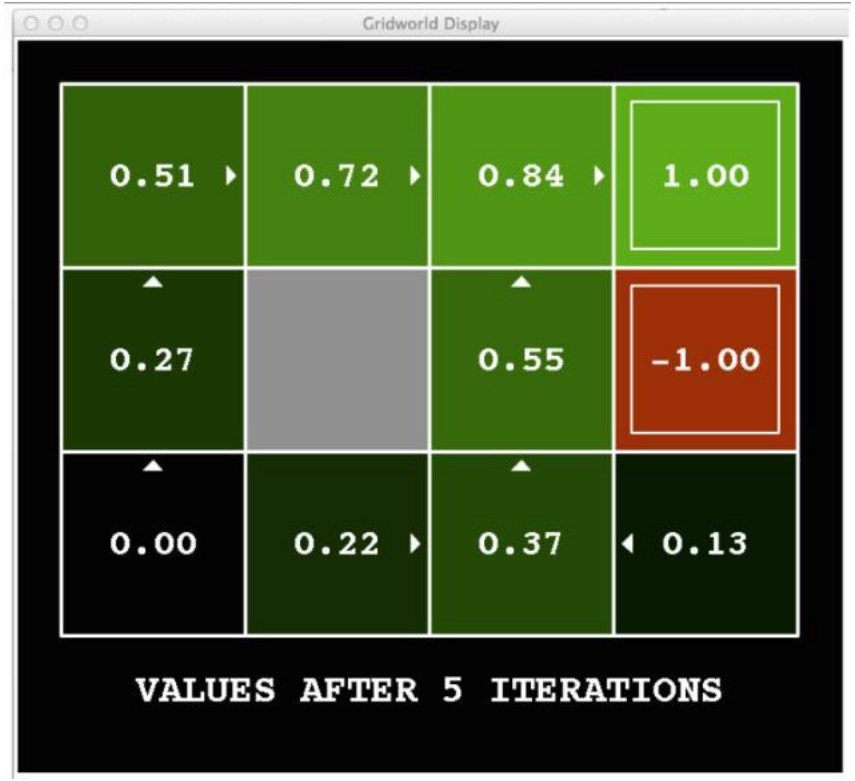
# k = 3


Noise = 0.2
Discount = 0.9
Living reward = 0

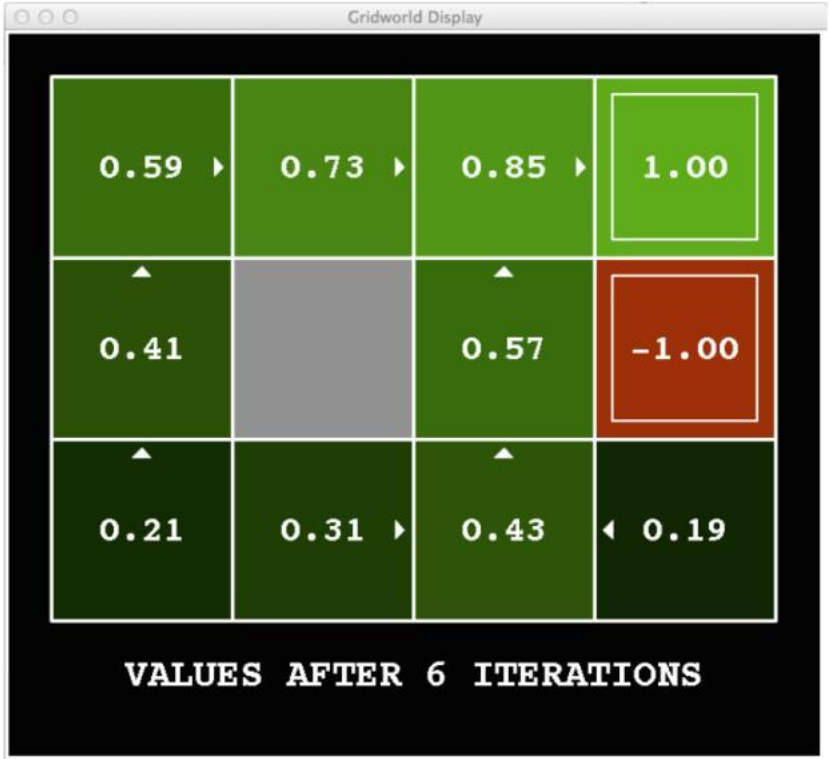
# k = 4


Gridworld Display
0.37
0.66
0.83
1.00
0.00
0.51
-1.00
0.00
0.00
0.31
0.00
VALUES AFTER 4 ITERATIONS

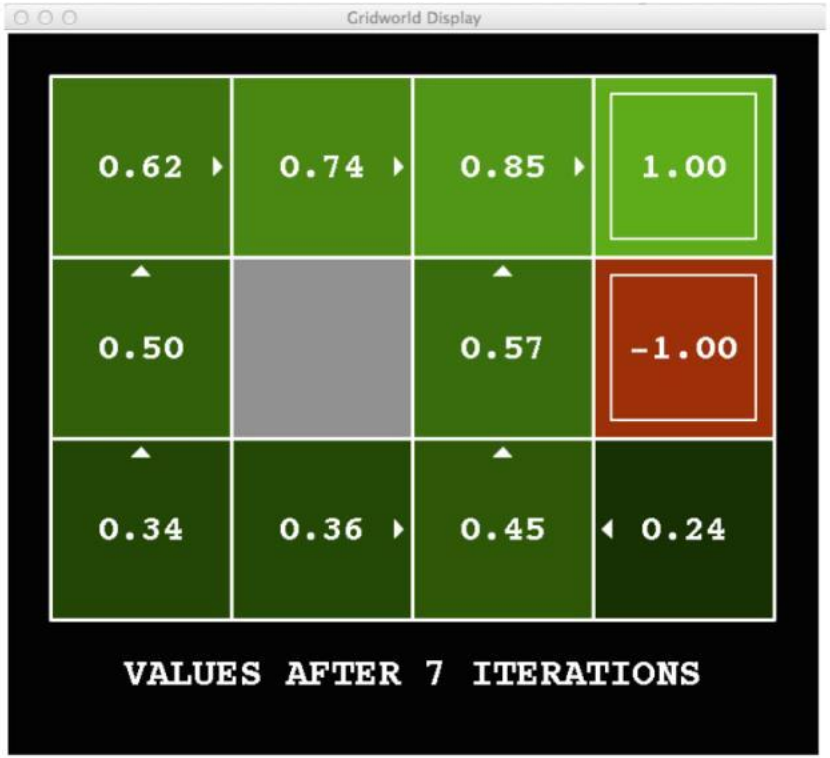
# k = 5


Gridworld Display
0.51
0.72
0.84
1.00
0.27
0.55
-1.00
0.00
0.22
0.37
0.13
VALUES AFTER 5 ITERATIONS

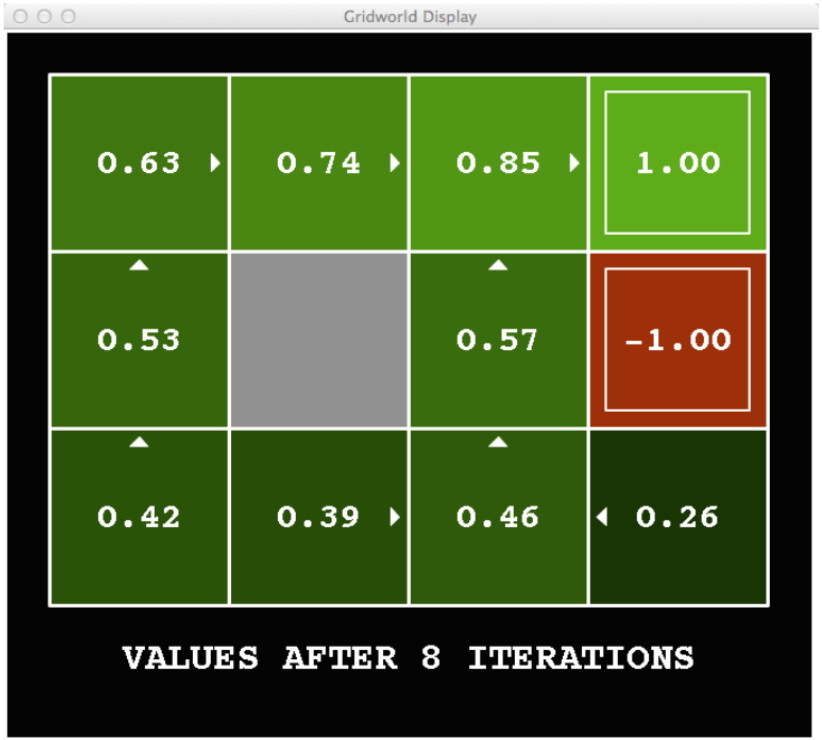
# k = 6


Gridworld Display
0.59
0.73
0.85
1.00
0.41
0.57
-1.00
0.21
0.31
0.43
0.19
VALUES AFTER 6 ITERATIONS

# k = 7


Gridworld Display
0.62
0.74
0.85
1.00
0.50
0.57
-1.00
0.34
0.36
0.45
0.24
VALUES AFTER 7 ITERATIONS

# k = 8


Gridworld Display
0.63
0.74
0.85
1.00
0.53
0.57
-1.00
0.42
0.39
0.46
0.26
VALUES AFTER 8 ITERATIONS

# k = 9


Gridworld Display
0.64
0.74
0.85
1.00
0.55
0.57
-1.00
0.46
0.40
0.47
0.27
VALUES AFTER 9 ITERATIONS

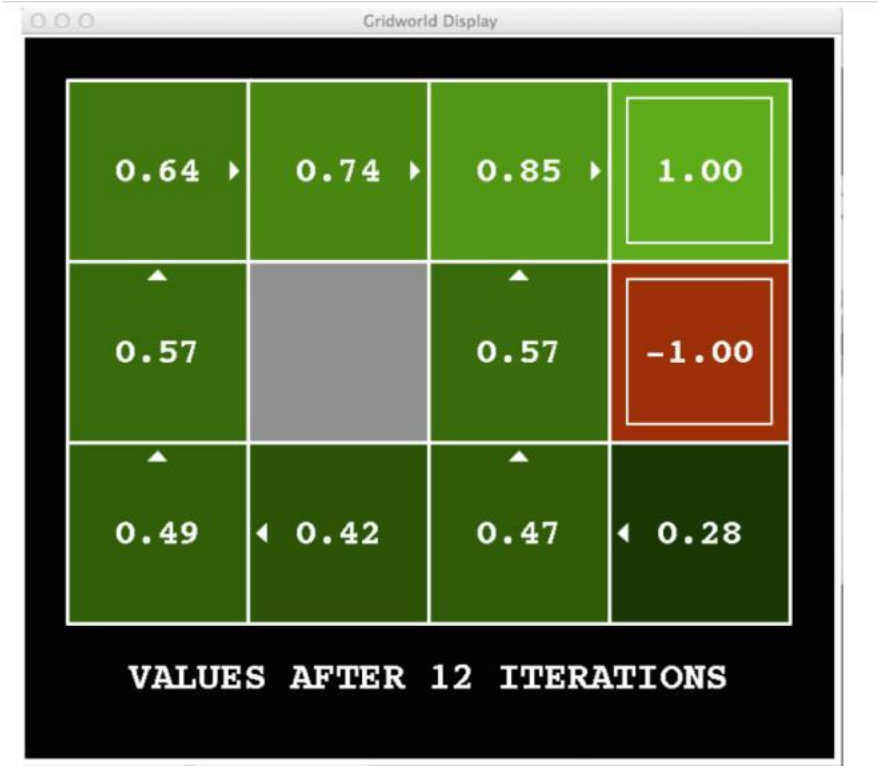
# k = 10


Gridworld Display
0.64
0.74
0.85
1.00
0.56
0.57
-1.00
0.48
0.41
0.47
0.27
VALUES AFTER 10 ITERATIONS

# k = 11


Gridworld Display
0.64
0.74
0.85
1.00
0.56
0.57
-1.00
0.48
0.42
0.47
0.27
VALUES AFTER 11 ITERATIONS

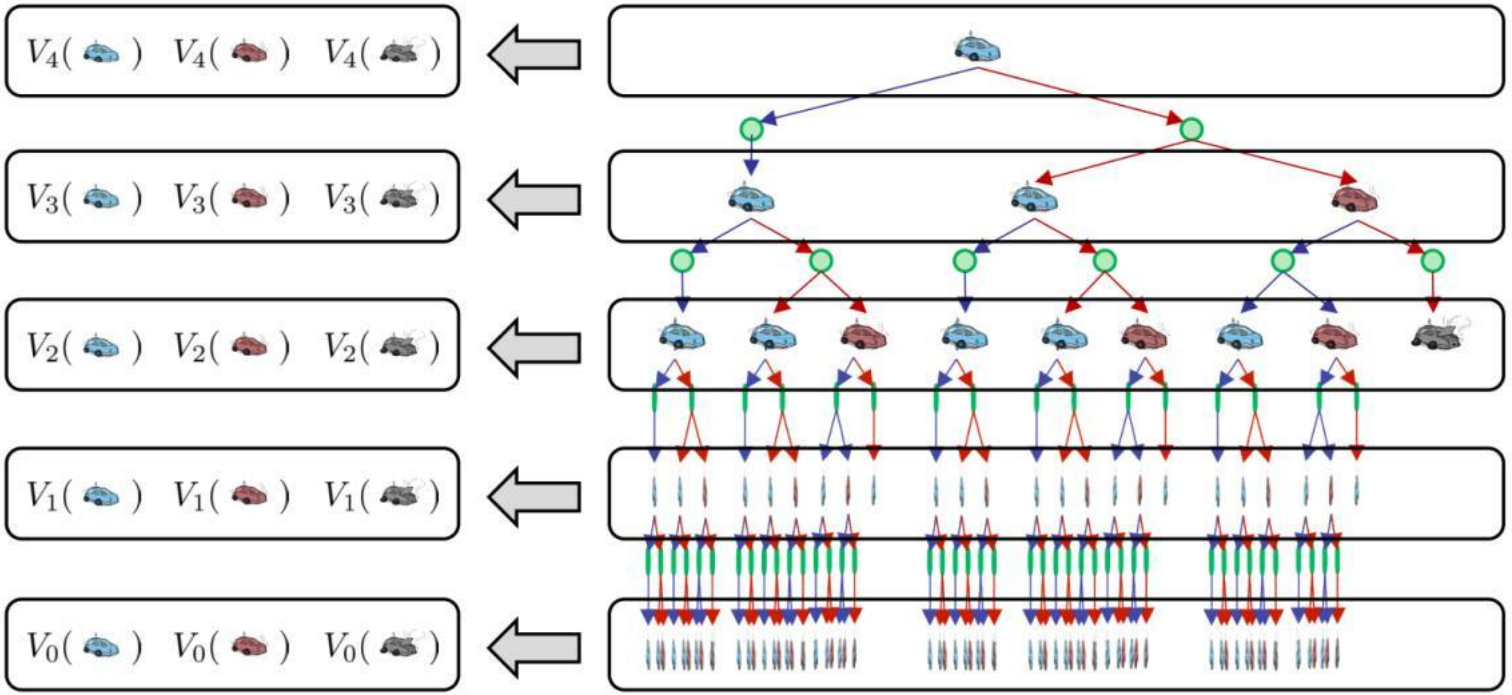
# k = 12


Gridworld Display
0.64
0.74
0.85
1.00
0.57
0.57
-1.00
0.49
0.42
0.47
0.28
VALUES AFTER 12 ITERATIONS

# k = 100

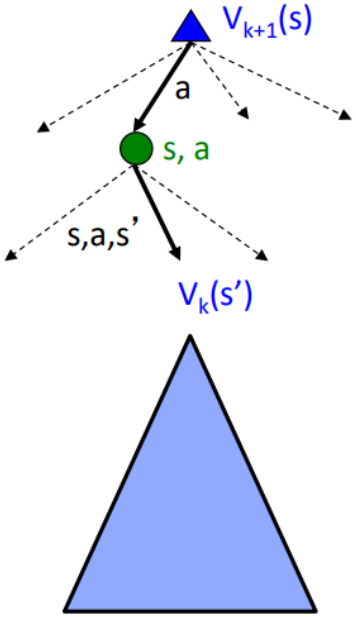

Gridworld Display
0.64
0.74
0.85
1.00
0.57
0.57
-1.00
0.49
0.43
0.48
0.28
VALUES AFTER 100 ITERATIONS

# محاسبه مقدارهای با زمان محدود



# Value Iteration تکرار مقدار

# تکرار مقدار Value Iteration

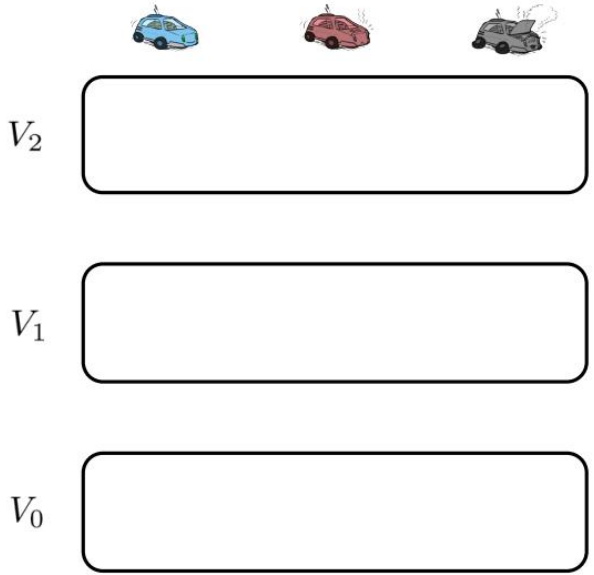


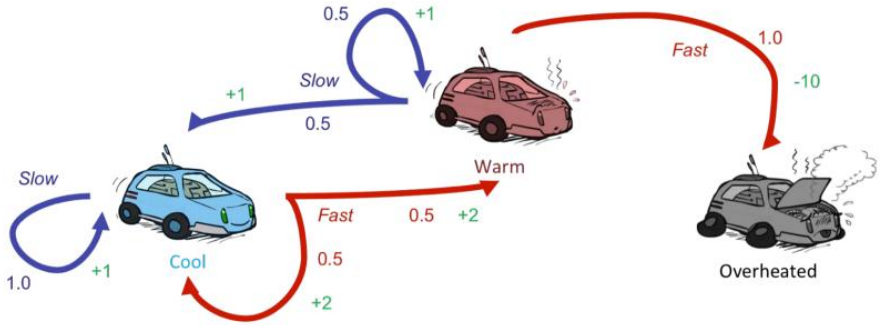

- شروع با $V_0(s) = 0$: باقی نماندن هیچ گام زمانی به معنای مجموع پاداش مورد انتظار صفر است
- با داشتن بردار $V_k(s)$ برای هر حالت یک لایه از expectimax را انجام بده:

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

- تا همگرایی تکرار کن
- پیچیدگی هر تکرار $O(S^2 A)$
- قضیه: به مقادیر بهینه یکتا همگرا خواهد شد
  - ایده پایه: تخمین‌ها به سمت مقادیر بهینه اصلاح می‌یابند
  - سیاست ممکن است خیلی زودتر از مقادیر همگرا شود

# مثال: تکرار مقدار Value Iteration

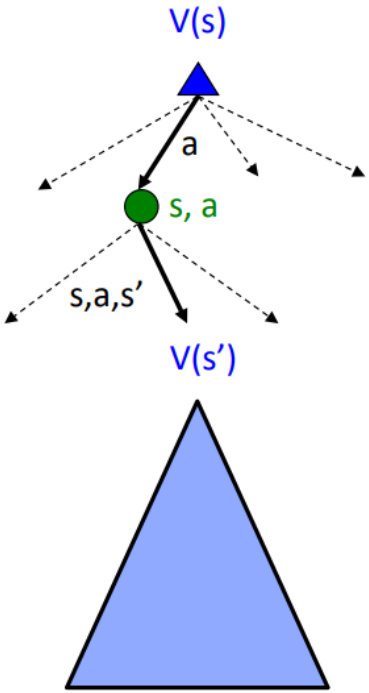



با فرض بدون تخفیف!

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

# تکرار مقدار (Value Iteration)




- معادلات بلمن مقادیر بهینه را تعریف می کنند:

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

- تکرار مقدار، آن ها را محاسبه می کند:

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

- تکرار مقدار یک روش تکرار ساده (نقطه ثابت) است.
  - با وجود اینکه بردار های $V_k$ به صورت زماندار نیز قابل تفسیر هستند.

# تکرار مقدار

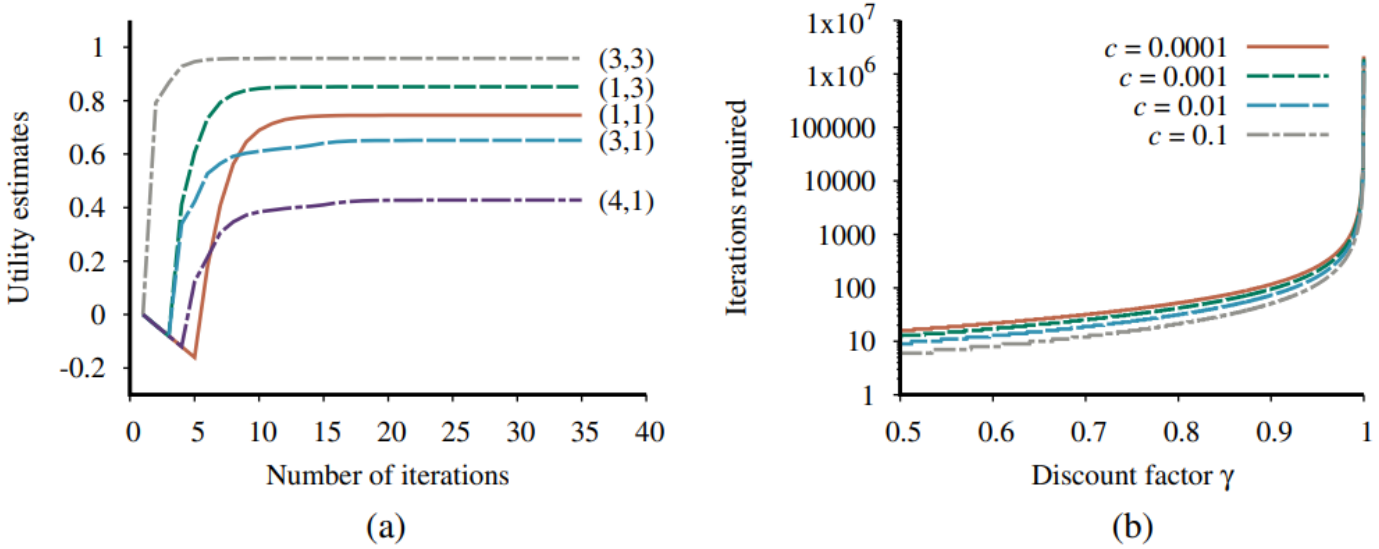
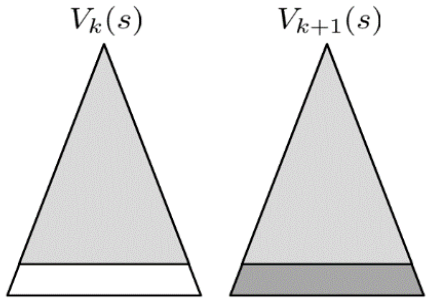


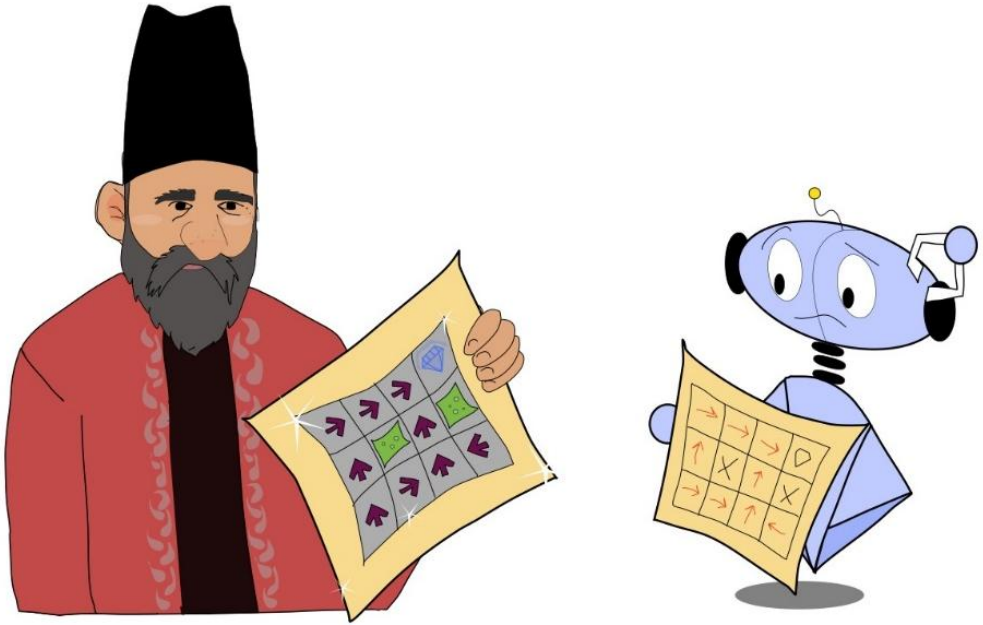

**Figure 16.7** (a) Graph showing the evolution of the utilities of selected states using value iteration. (b) The number of value iterations required to guarantee an error of at most $\epsilon = c \cdot R_{\max}$, for different values of $c$, as a function of the discount factor $\gamma$.

# تکرار مقدار

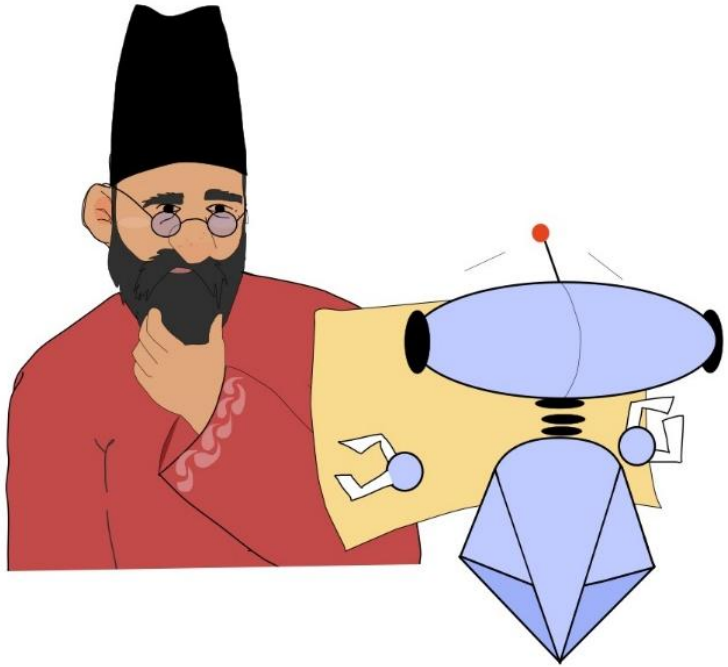
- از کجا می دانیم که بردارهای $V_k$ همگرا خواهند شد؟
- حالت 1: اگر درخت دارای بیشینه عمق M باشد ، آنگاه $V_m$شامل مقادیر واقعی است بدون آن که درخت برش خورده باشد.
- حالت 2: اگر کاهش از 1 کمتر باشد.
  - طرح کلی : به ازای هر حالت، مقادیر $V_k$ و $V_{k+1}$ را می توان به عنوان نتیجه ی یک جستجوی ، expectimaxبا محدوده ی عمقی k + 1در دو درخت جستجوی تقریباً یکسان در نظر گرفت
  - تفاوت در این است که در آخرین لایه $V_{k+1}$شامل پاداش های واقعی است اما $V_k$ شامل پاداش های مساوی صفر است.
  - پاداش ها در این لایه همگی حداکثر $R_{max}$ هستند.
  - و حداقل برابر با $R_{min}$هستند.
  - اما همه چیز با ضریب $\gamma^k$کاهش یافته است.
  - بنابراین اخلاف $V_k$ و $V_{k+1}$ حداکثر برابر با $\gamma^k max|R|$ است.
  - در نتیجه با افزایش k ، مقادیر همگرا می شوند



# روش‌های مبتنی بر سیاست

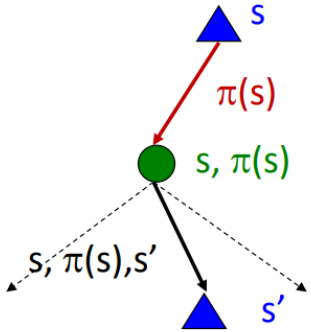


# ارزیابی سیاست

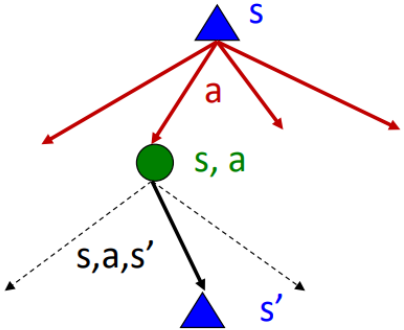


# سیاست‌های ثابت

کاری که π می گوید را انجام بده.



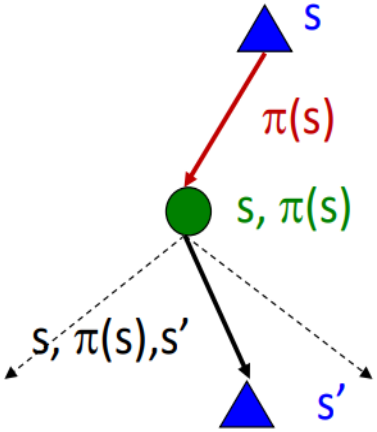

عمل بهینه را انجام بده.




- درخت expectimax برای محاسبه مقادیر بهینه، بر روی تمام عمل‌ها بیشینه را حساب می‌کنند
- اگر از سیاست ثابت π(s) استفاده کنیم، درخت ساده تر می شود، فقط یک عمل به ازای هر حالت.
  - ... اگرچه مقدار حالت‌های درخت به سیاست انتخاب شده بستگی خواهد داشت.

# سودمندی برای یک سیاست ثابت

- یک عملیات پایه دیگر: محاسبه مقدار حالت s تحت یک سیاست ثابت (معمولا غیربهینه)
- تعریف مقدار حالت s تحت سیاست ثابت $\pi$:



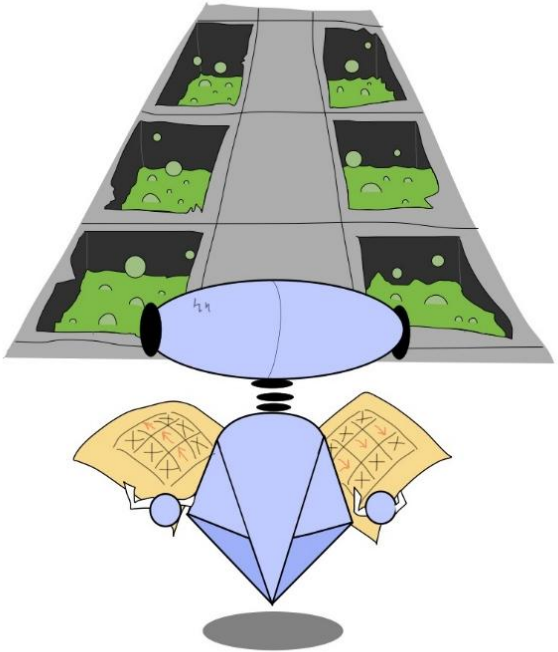

$V^{\pi}(s)$ = مجموع پاداش‌های تخفیف یافته موردانتظار با شروع از s و دنبال کردن $\pi$

- رابطه بازگشتی (نگاه به جلوی تک گامی / معادله بلمن):

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

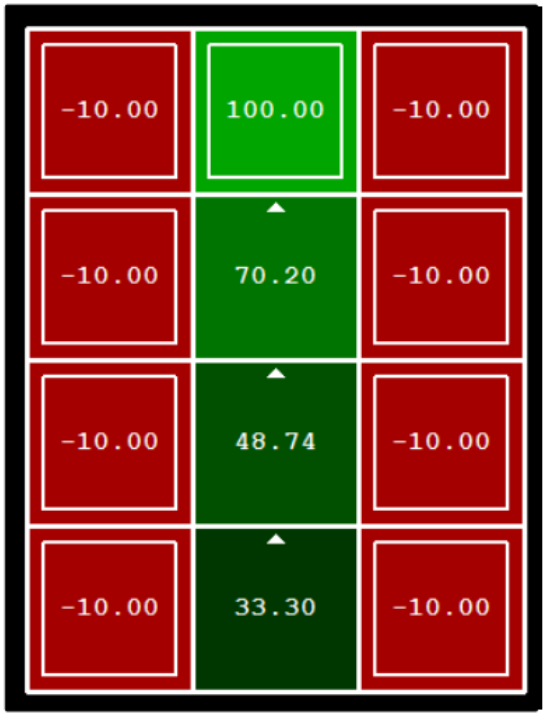
# مثال: ارزیابی سیاست

همیشه به راست برو

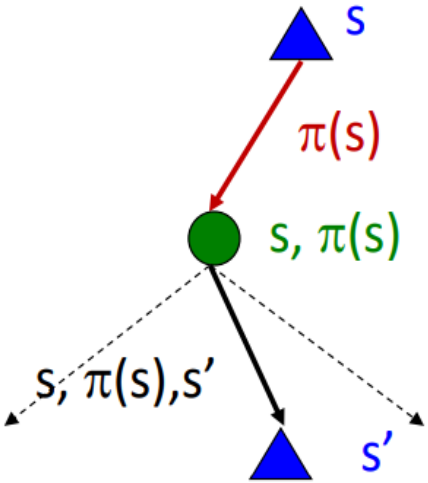


همیشه به جلو برو

# مثال: ارزیابی سیاست

# ارزیابی سیاست



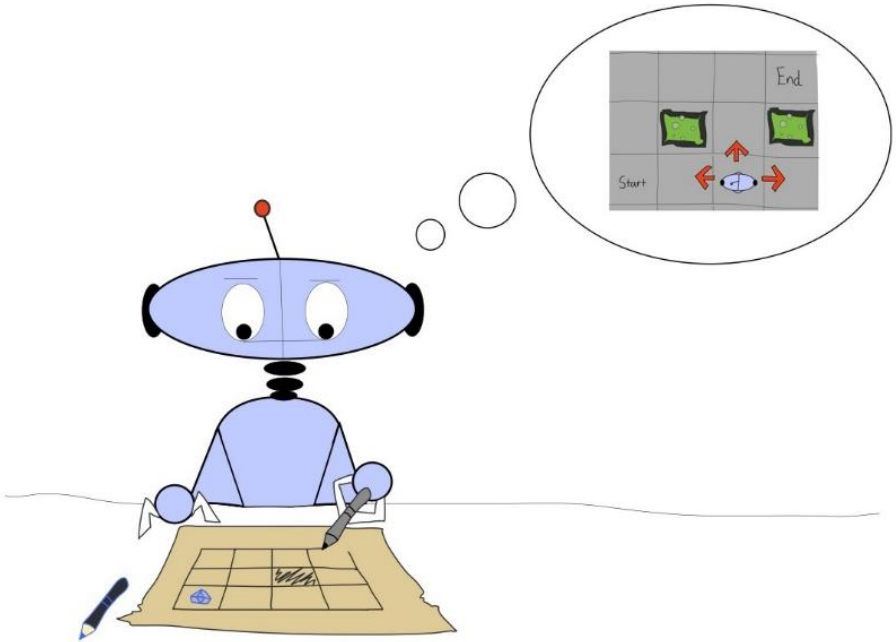

- چگونه مقادیر V را برای سیاست ثابت π محاسبه می کنیم؟
- ایده 1: تبدیل معادلات بازگشتی بلمن به به روز رسانی‌ها (شبیه تکرار مقدار)

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

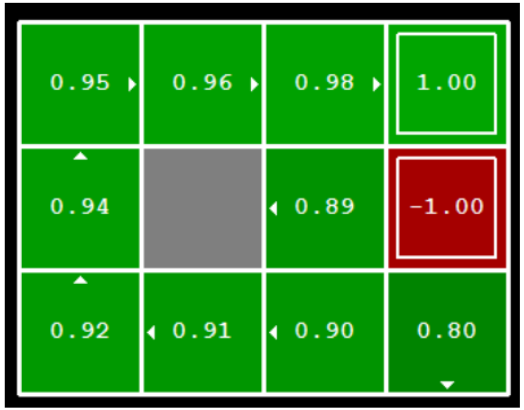
$$V_{k+1}^\pi(s) \leftarrow \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V_k^\pi(s')]$$

- کارآیی: $O(s^2)$ به ازای هر تکرار
- ایده 2: بدون بیشینه‌ها، معادلات بلمن تنها یک دستگاه معادلات خطی هستند
  - حل با متلب (یا حل کننده ی سیستم های خطی مورد علاقه خود)

# استخراج سیاست



# محاسبه اعمال از مقادیر

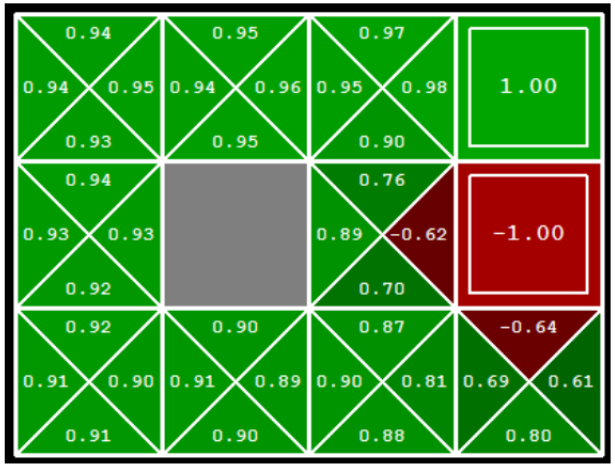



- بیایید فرض کنیم که ما مقادیر بهینه V*(s) را داریم.
- باید چگونه عمل کنیم؟
  - واضح نیست!
- باید یک mini-expectimax انجام دهیم (یک گام).

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

- اینکار استخراج سیاست نام دارد، زیرا سیاستی را به دست می آورد که از مقادیر برداشت می شود.

# محاسبه ی اعمال از روی مقادیر Q



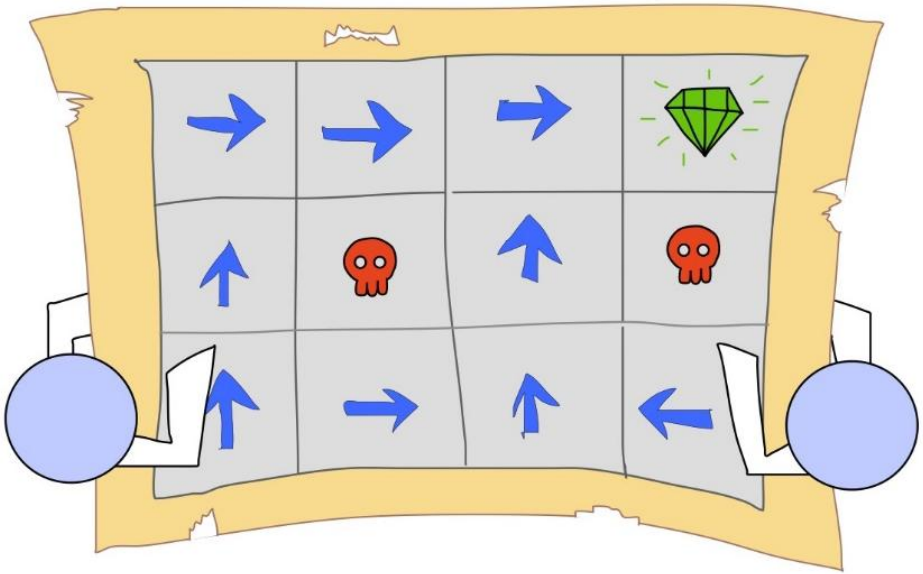

- بیایید فرض کنیم که ما مقادیر بهینه q را داریم.

- باید چگونه عمل کنیم؟
  - ساده است!

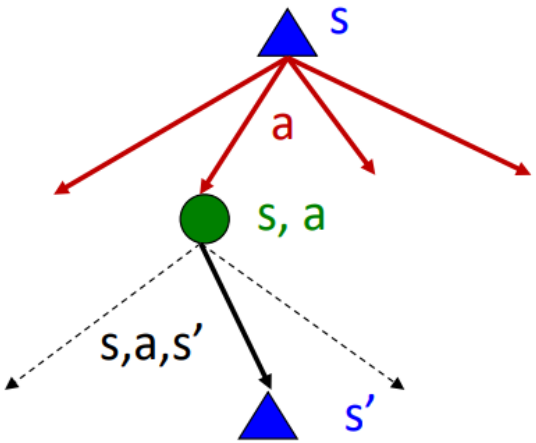
$$\pi^*(s) = \arg\max_a Q^*(s,a)$$

- درس مهم: انتخاب اعمال از روی مقادیر q ساده تر از مقادیر V است!

# تکرار سیاست (Policy Iteration)



# مشکلات تکرار مقدار



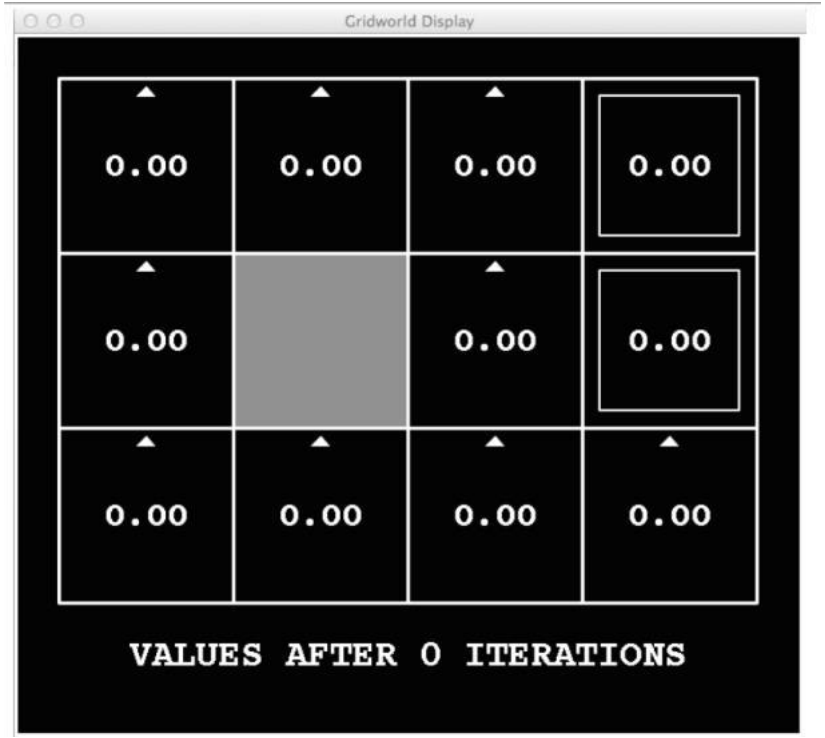

- تکرار مقدار به روز رسانی های بلمن را تکرار می کند:

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

- مشکل 1: کند است - $O(S^2 A)$ به ازای هر تکرار
- مشکل 2: "بیشینه" در هر حالت به ندرت تغییر می کند
- مشکل 3: در بیشتر موارد سیاست بسیار زودتر از مقادیر همگرا می شود

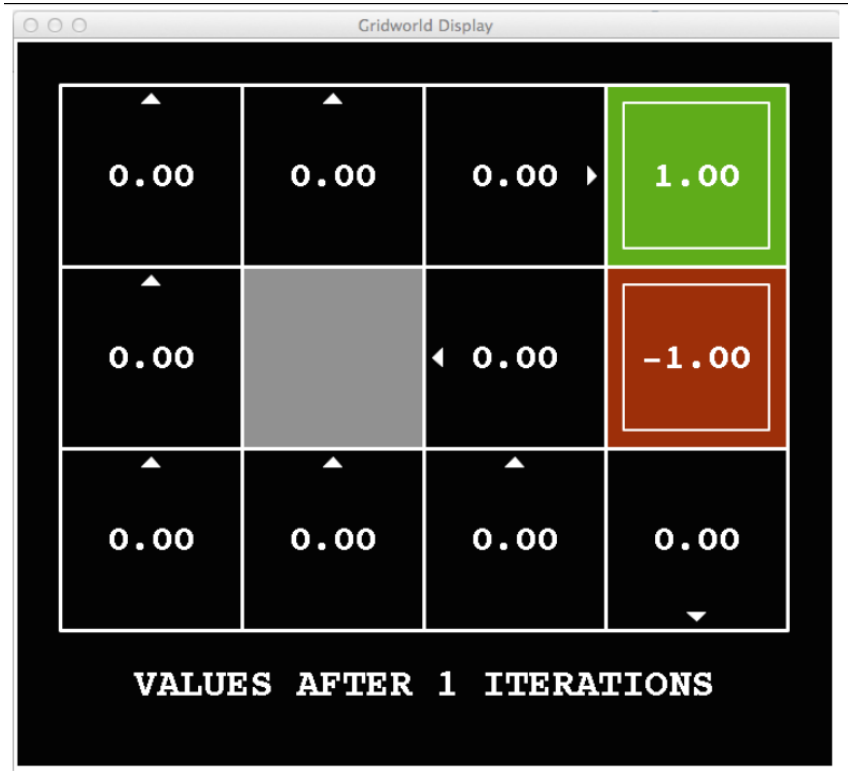
[Demo: value iteration (L9D2)]

# k = 0




نویز = 0.2
ضریب کاهش = 0.9
پاداش زنده بودن = 0

# k = 1




نویز = 0.2
ضریب کاهش = 0.9
پاداش زنده بودن = 0

# k = 2


نویز = 0.2
ضریب کاهش = 0.9
پاداش زنده بودن = 0

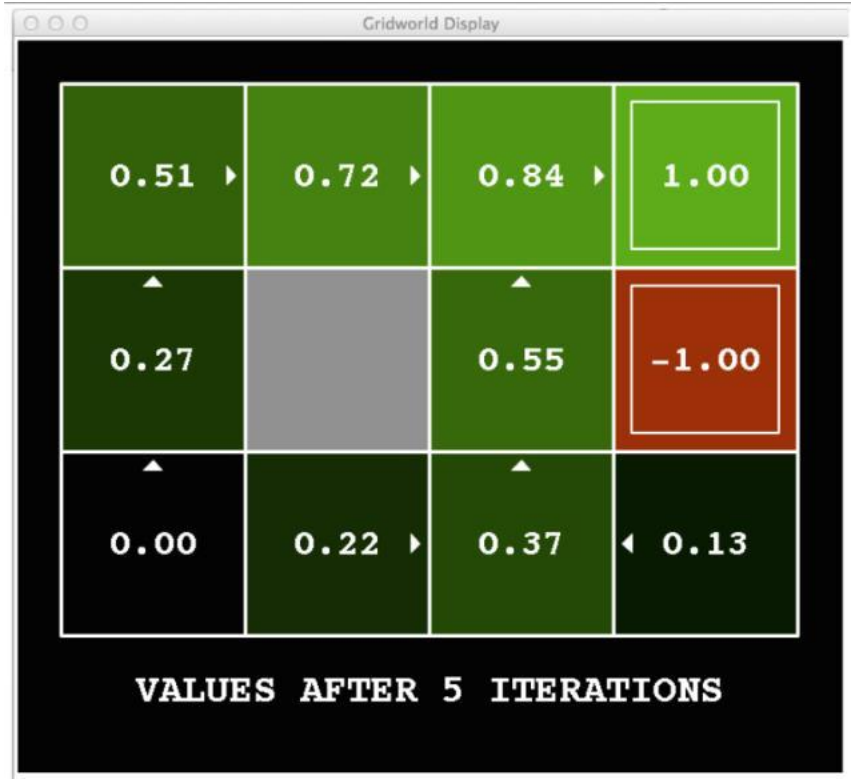
# k = 3


نویز = 0.2
ضریب کاهش = 0.9
پاداش زنده بودن = 0

# k = 4

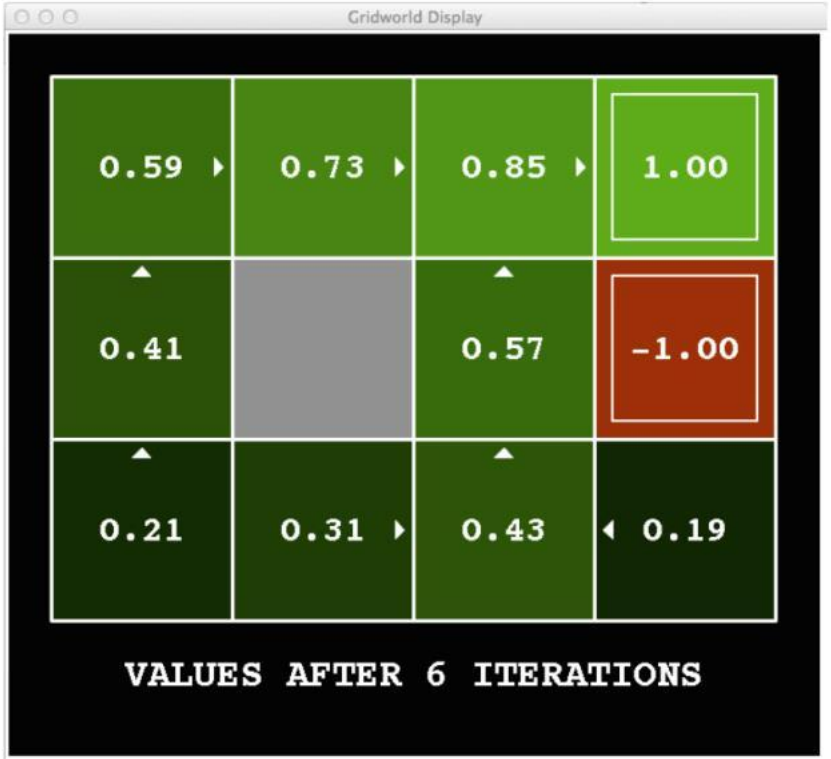

نویز = 0.2
ضریب کاهش = 0.9
پاداش زنده بودن = 0

# k = 5



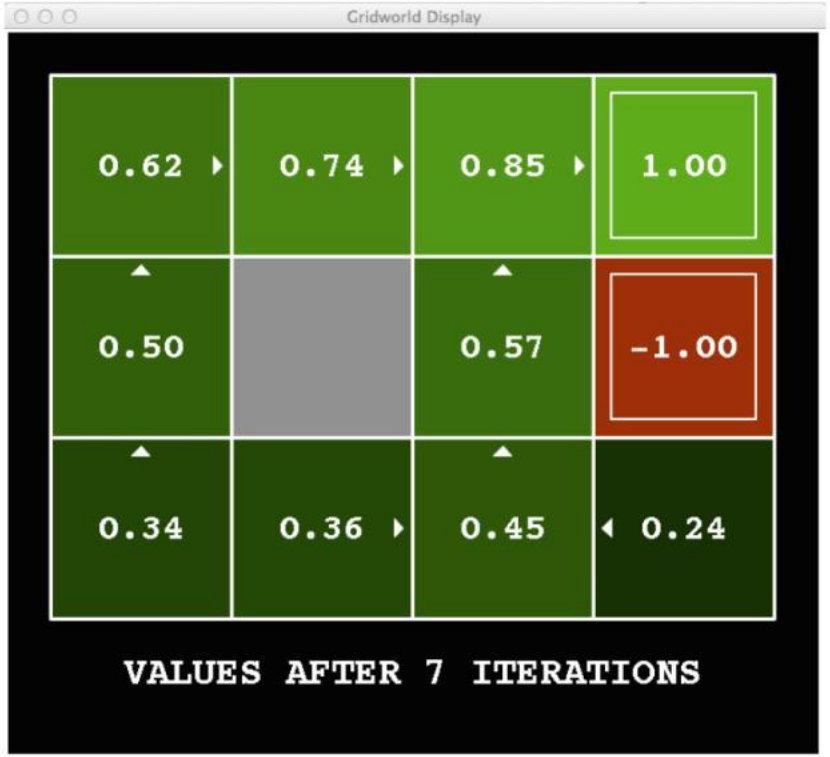

نویز = 0.2
ضریب کاهش = 0.9
پاداش زنده بودن = 0

# k = 6



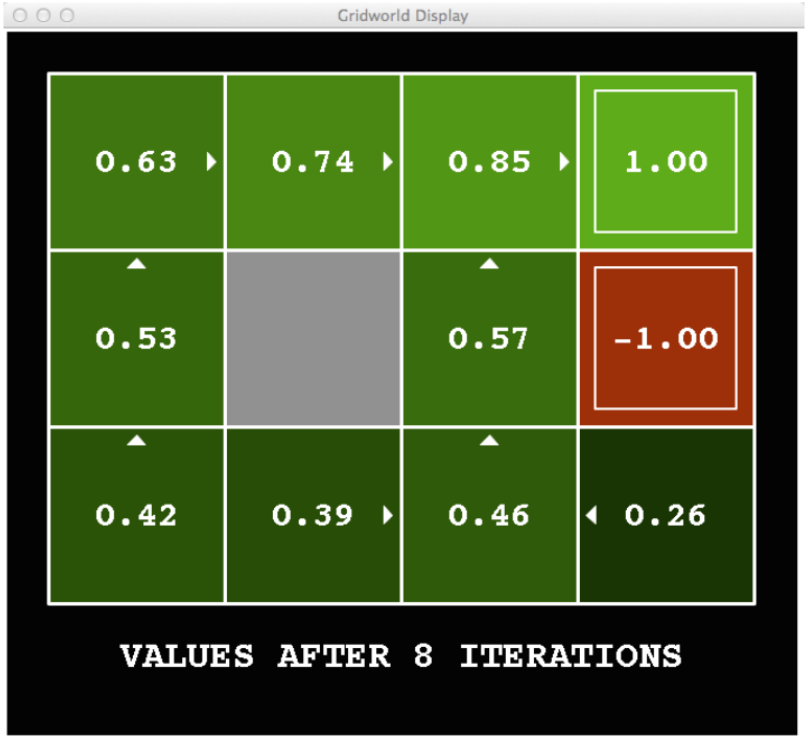

نویز = 0.2
ضریب کاهش = 0.9
پاداش زنده بودن = 0

# k = 7



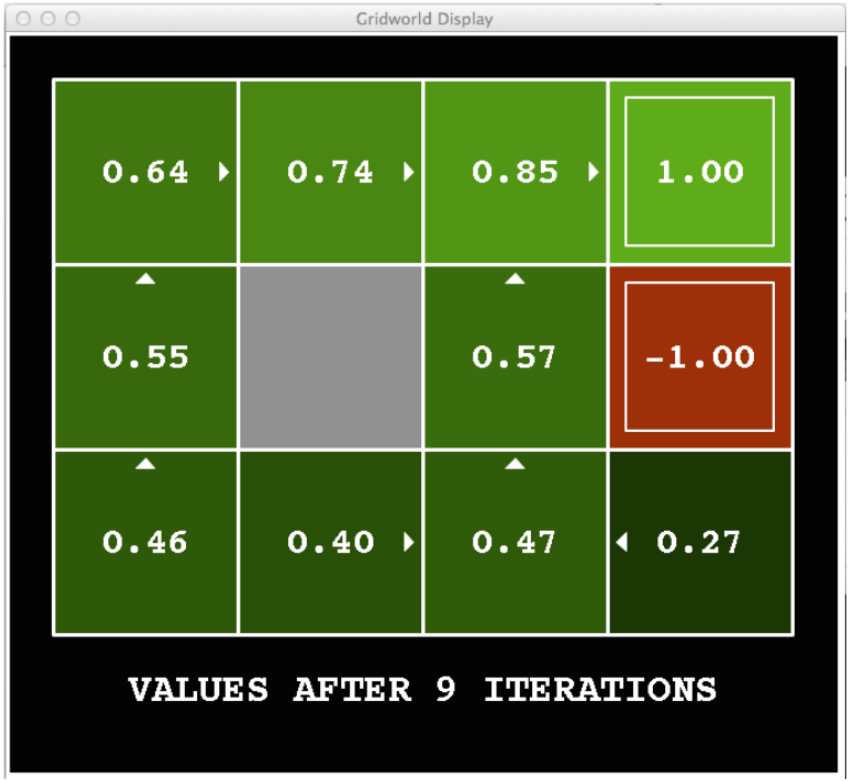

نویز = 0.2
ضریب کاهش = 0.9
پاداش زنده بودن = 0

# k = 8



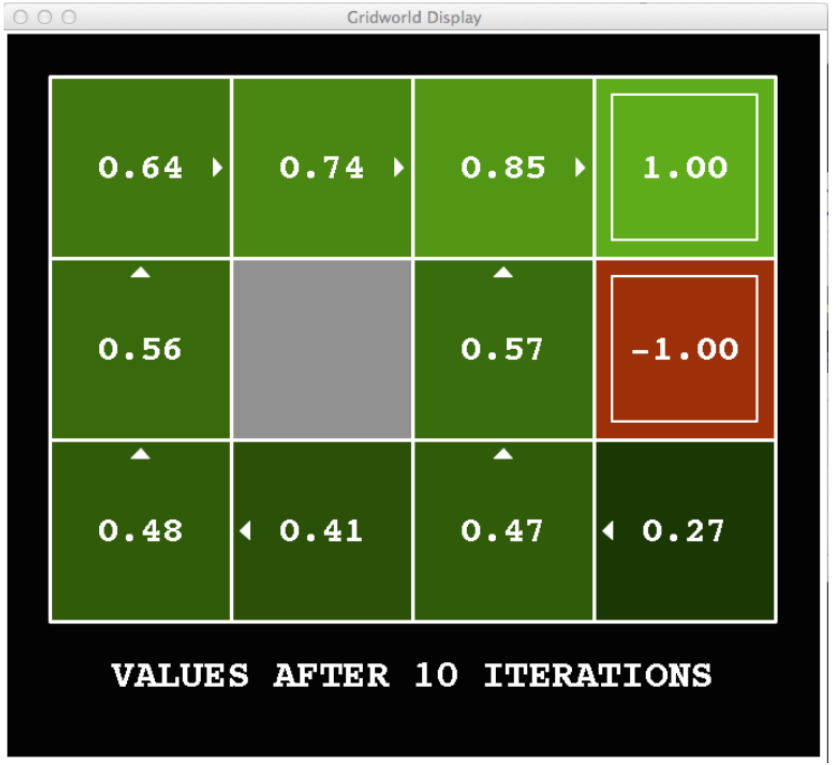

نویز = 0.2
ضریب کاهش = 0.9
پاداش زنده بودن = 0

# k = 9



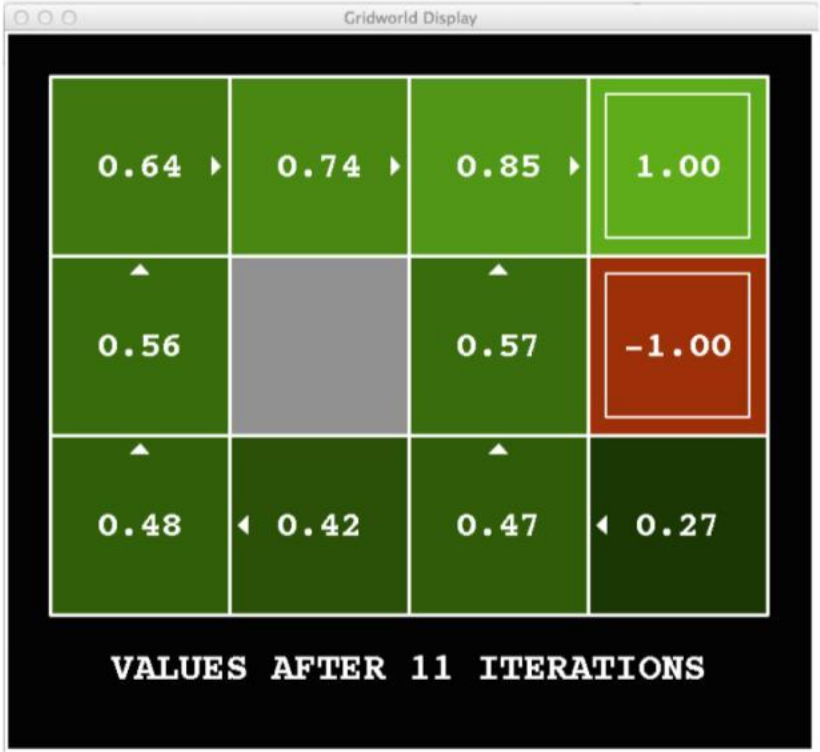

نویز = 0.2
ضریب کاهش = 0.9
پاداش زنده بودن = 0

# k = 10



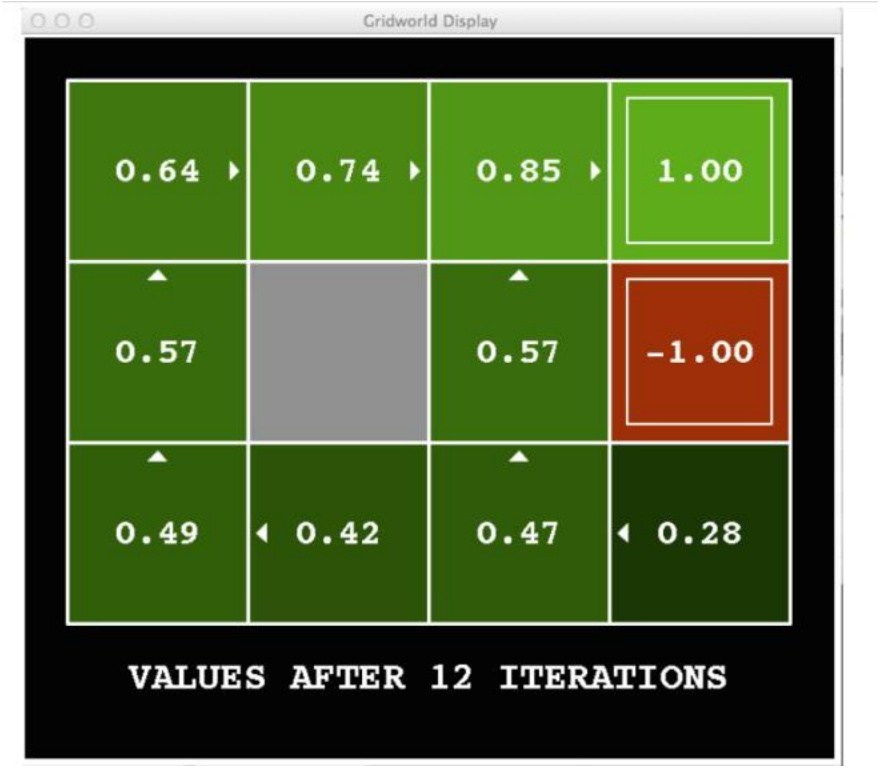

نویز = 0.2
ضریب کاهش = 0.9
پاداش زنده بودن = 0

# k = 11



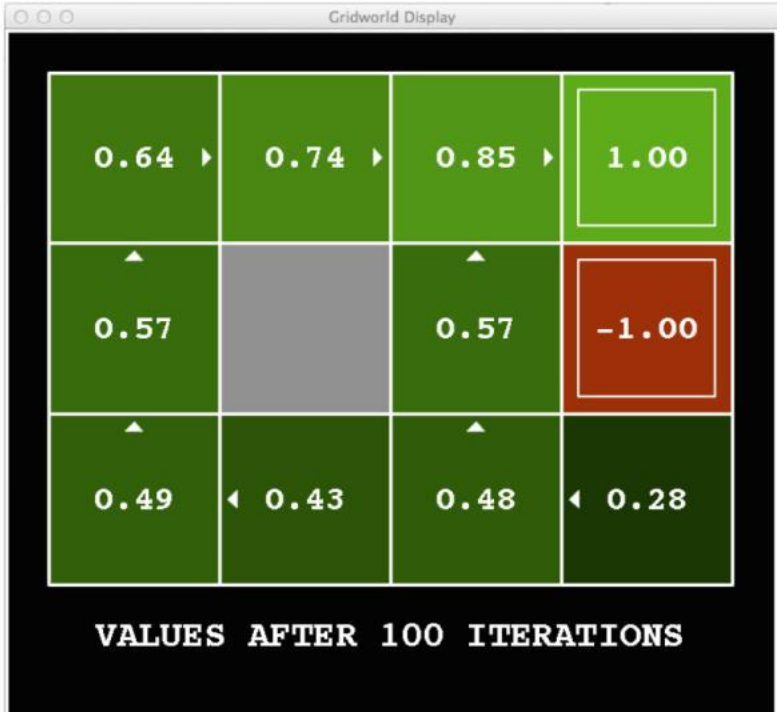

نویز = 0.2
ضریب کاهش = 0.9
پاداش زنده بودن = 0

# k = 12




نویز = 0.2
ضریب کاهش = 0.9
پاداش زنده بودن = 0

# k = 100




نویز = 0.2
ضریب کاهش = 0.9
پاداش زنده بودن = 0

# تکرار سیاست

- رویکردی دیگر برای مقادیر بهینه:
  - گام 1:ارزیابی سیاست: محاسبه مقادیر حالت‌ها برای یک سیاست ثابت (نه سیاست بهینه!) تا زمانی که مقادیر همگرا شوند.
  - گام 2: بهبود سیاست‌ها: به روز رسانی سیاست با استفاده نگاه به جلوی یک مرحله‌ای با مقادیر همگرا شده (ولی نه بهینه!) به عنوان مقادیر آینده
  - تکرار گام‌ها تا زمانی که سیاست همگرا شود.

- این کار تکرار سیاست است.
  - این کار همچنان بهینه است!
  - تحت شرایطی می تواند (خیلی) سریعتر همگرا شود.

# تکرار سیاست

- ارزیابی: برای سیاست فعلی ثابت π ، مقادیر را با ارزیابی سیاست پیدا کنید:
  - تکرار تا زمانی که مقادیر همگرا شوند:

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

- بهبود: برای مقادیر ثابت، با استخراج سیاست یک سیاست بهتر بدست بیاورید.
  - نگاه به جلوی یک مرحله ای:

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

# مقایسه

- هردوی تکرار مقدار و تکرار سیاست یک چیز را محاسبه می کنند(تمام مقادیر بهینه)
- در تکرار مقدار:
  - هر تکرار هم مقادیر و هم (به طور ضمنی) سیاست را به روزرسانی می کند
  - ما دنبال سیاست نیستیم، ولی بیشینه گرفتن روی اعمال به طور ضمنی آن را دوباره محاسبه می کند.

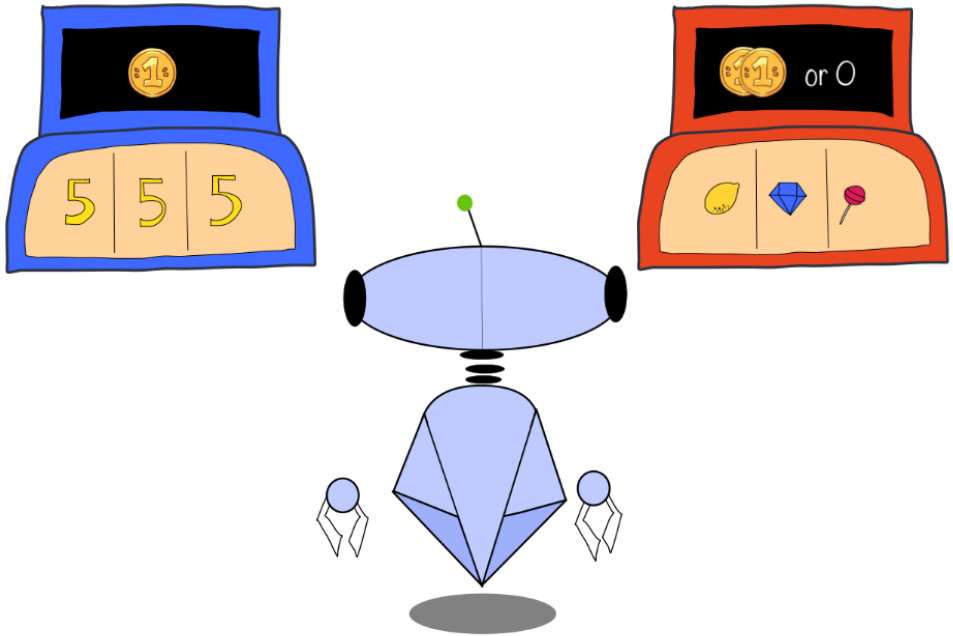
- در تکرار سیاست:
  - چند تکرار وجود دارد که در آنها مقادیر سودمندی را تنها برای یک سیاست ثابت به روز رسانی می کنیم (هر تکرار سریع است زیرا ما تنها یک عمل را در نظر می گیریم، نه تمام آن ها را)
  - بعد از اینکه سیاست ارزیابی شد، یک سیاست جدید انتخاب می شود (کند مانند گام از تکرار مقدار)
  - سیاست جدید بهتر خواهد بود (یا اینکه کار تمام است)
- هر دو برنامه های پویا برای حل MDP ها هستند.

# خلاصه : الگوریتم های MDP

- پس می خواهید...
- مقادیر بهینه را محاسبه کنید : از تکرار مقدار یا تکرار سیاست استفاده کنید.
- مقادیر را برای یک سیاست به خصوص محسابه کنید : از ارزیابی سیاست استفاده کنید.
- مقادیر خود را به یک سیاست تبدیل کنید: از استخراج سیاست استفاده کنید(نگاه به جلوی یک مرحله ای)

- این ها همه یکسان به نظر می رسند!
  - آن ها در واقع یکسان هستند-آن ها گونه مختلفی از به روز رسانی معادلات بلمن هستند.
  - آن‌ها همه از قطعه‌های نگاه به جلوی یک مرحله ای expectimax استفاده می کنند.
  - آن‌ها فقط در این متفاوتند که آیا از یک سیاست ثابت استفاده می کنیم یا در هر حالت بر روی عملیات ممکن بیشینه می گیریم.

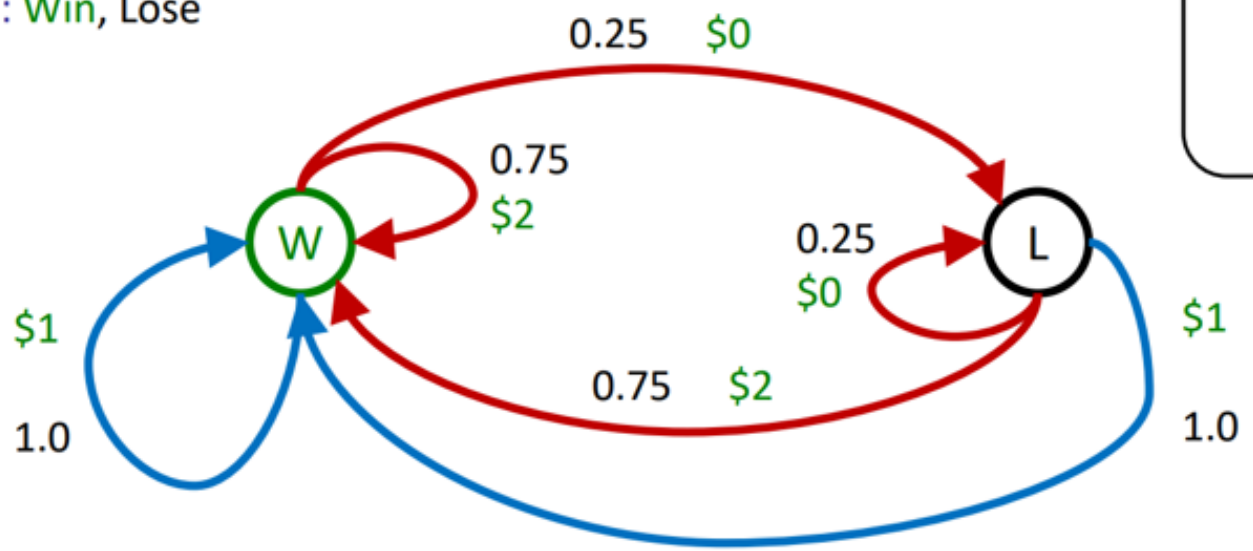
# مسئله دابل بندیت



# Double Bandit MDP

- اعمال : *Blue, Red*
- حالات : Win, Lose

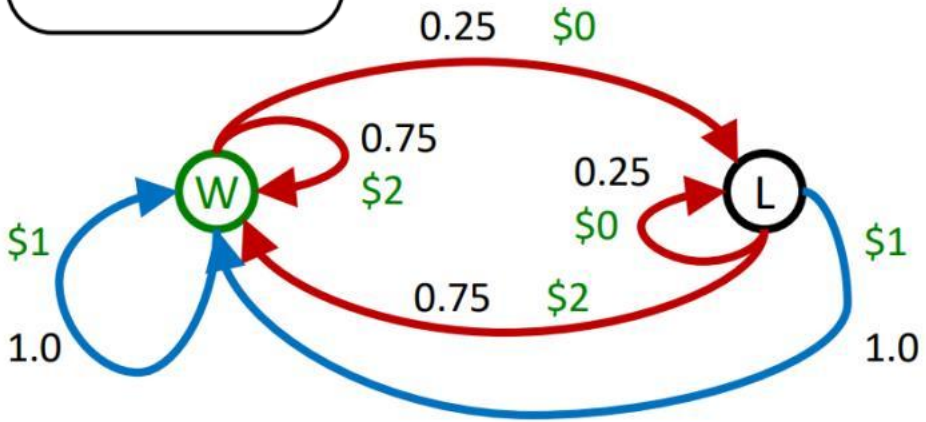
بدون کاهش
۱۰۰ گام زمانی
هر دو حالت
مقدار



# برنامه ریزی برخط

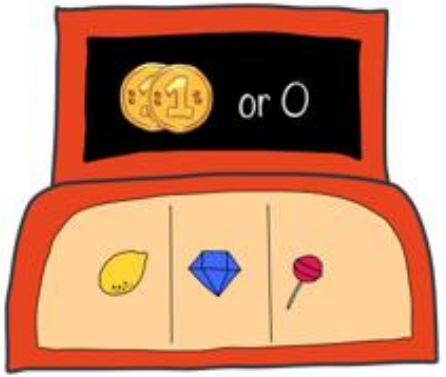
- حل MDP ها برنامه ریزی آفلاین است
  - شما همه‌ی کمیت‌ها را با محاسبات تعیین می‌کنید
  - شما باید جزئیات MDP را بدانید
  - شما واقعا بازی را بازی نمی‌کنید!

بدون کاهش
100 گام زمانی
هر دو حالت مقدار
یکسان دارند




| | مقدار |
|---|---|
| انجام قرمز | 150 |
| انجام آبی | 100 |

# بیاید بازی کنیم!

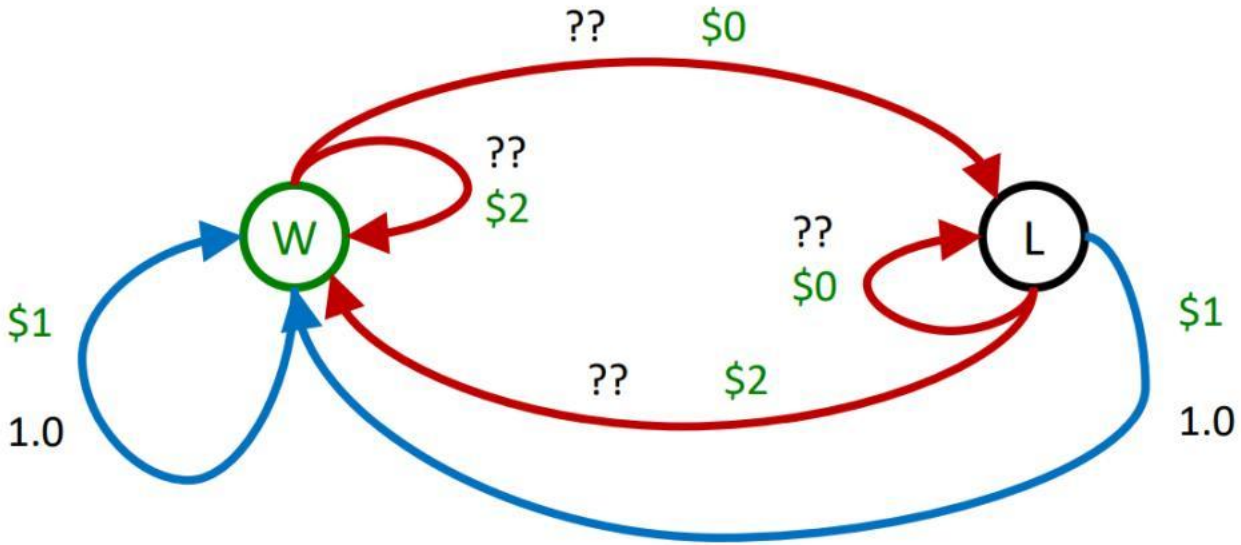




$2 $2 $0 $2 $2

$2 $2 $0 $0 $0

# برنامه ریزی آنلاین

- قوانین عوض شد! شانس برد قرمز عوض شد.



# بیاید بازی کنیم!



$0 $0 $0 $2 $0

$2 $0 $0 $0 $0

# چه اتفاقی افتاد؟

- اتفاقی که افتاد برنامه ریزی نبود، یادگیری بود!
  - به بیان دقیق‌تر، یادگیری تقویتی
  - یک مسئله‌ی MDP است، ولی شما نمی‌توانستید تنها با محاسبات آن را حل کنید.
  - برای حل آن مجبور بودید واقعاً عمل کنید.

- ایده‌های مهم در یادگیری تقویتی
  - کاوش (exploration): باید اعمال ناشناخته را امتحان کنید تا اطلاعات به دست آورید.
  - بهره‌برداری (exploitation): سرانجام، باید از چیزی که می دانید استفاده کنید.
  - پشیمانی (regret): حتی اگر هوشمندانه یاد بگیرید، باز هم ممکن است اشتباه کنید.
  - نمونه برداری (sampling): به دلیل عدم قطعیت، باید یک عمل را بارها آزمایش کنید.
  - دشواری: یادگیری می‌تواند بسیار سخت تر از حل یک MDP شناخته شده باشد.

# دفعه ی بعد: یادگیری تقویتی!