

# 08 - Markov Decision Processes

## Bài toán ví dụ: Grid World

- Một mê cung với agent bên trong grid và có những bức tường ngăn chặn đường đi agent
- Có yếu tố ngẫu nhiên: mỗi hành động có phân bố xác suất
  - ví dụ: 80% hành động North thực sự khiến agent đi về North  
10% North thì đi West, 10% North đi East
- Agent nhận được phần thưởng ở mỗi bước đi
- **Goal:** tối đa hóa phần thưởng

## Policies

- Một chiến lược  $\pi : S \rightarrow A$  (truyền vô trạng thái trả 1 hàng động)
- Để giải bài toán MDP, ta cần 1 chiến lược tối ưu  $\pi^* : S \rightarrow A$ 
  - Chiến lược tối ưu không có nghĩa phải thắng
  - Khi mang chiến lược tối ưu chơi nhiều lần, tính kì vọng  $\rightarrow$  ra tỉ lệ thắng cao nhất  $\Rightarrow$  cực đại hóa độ hữu dụng kì vọng
- Cấu trúc dữ liệu: array/dict
- Cấu trúc dữ liệu cho số lượng trạng thái: hash table  $\rightarrow$  colision  $\Rightarrow$  dùng mạng neuron để xử lý số lượng trạng thái lớn
  - Input: bức hình
  - Output: phân lớp lên/xuống/trái/phải

## MDP

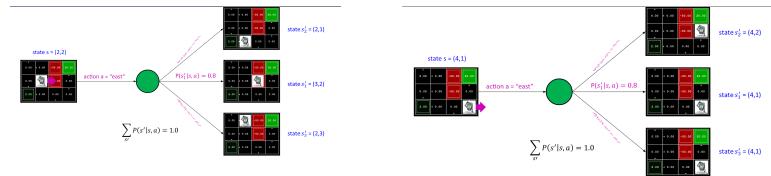
- Được định nghĩa:
  - 1 tập hợp các trạng thái  $S$
  - 1 tập các hành động  $A$
  - 1 hàm triển  $T$ 
    - xác suất với hành động  $a$  từ trạng thái  $s$  dẫn tới  $s'$
  - 1 hàm điểm thưởng  $R$
  - Trạng thái bắt đầu
  - Có thể có trạng thái kết thúc
- Có thể giải bằng expectimax nhưng chạy nhiều lần
  - mỗi lần đặt agent ở 1 vị trí bắt đầu khác nhau (bởi vì đây là policy chứ không phải plan)
- **Tính chất Markov:** khi có thông tin về hiện tại, quá khứ và tương lai độc lập với nhau

- ví dụ không có tính chất markov: việc phát sinh câu văn, sự phát sinh của chữ tiếp theo phụ thuộc vào chữ phía trước

## MDP Search Tree

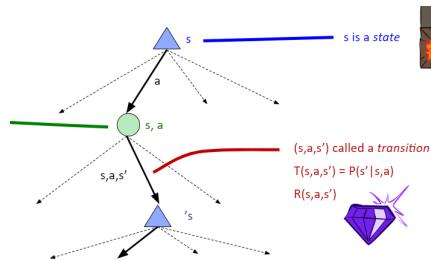


Sau khi thực hiện hành động  $\rightarrow$  tới node cơ hội/kì vọng  $\rightarrow$  đợi phản ứng từ môi trường trả về phân phối xs

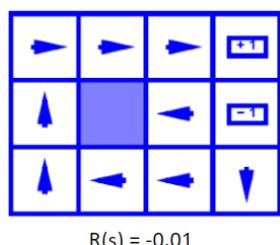


90% sẽ đứng yên, 10% đi lên

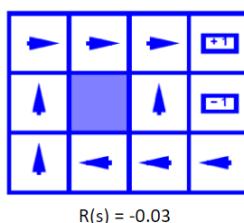
- Mỗi trạng thái MDP tương tự như cây expectimax



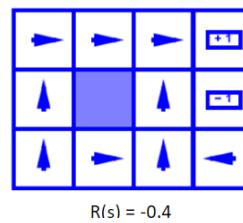
- Chiến lược tối ưu:



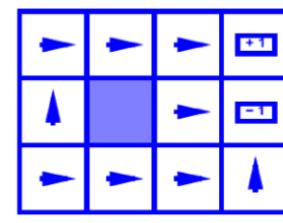
(2,3): 80% sẽ đứng yên,  
10% sẽ lên trên



(2,3): không đâm vào  
tường nữa vì mỗi lần  
đâm vào tường mất 0.03  
thay vì 0.02 nên lên trên  
luôn



bắt buộc phải mạo hiểm

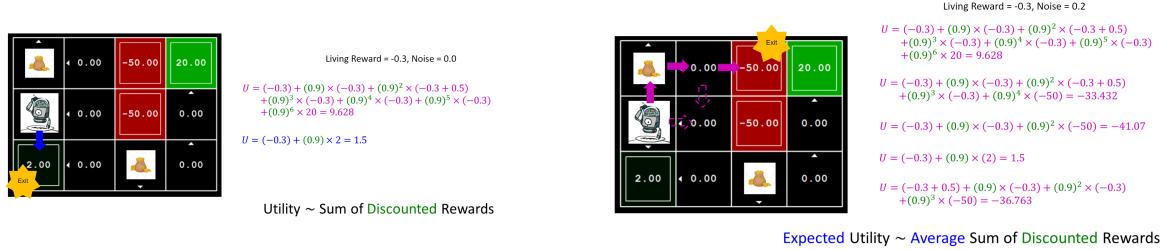


... dù chiến thắng cũng mất  
nhiều điểm nên quyết định  
kết thúc sớm

## Discounting - chiết khấu

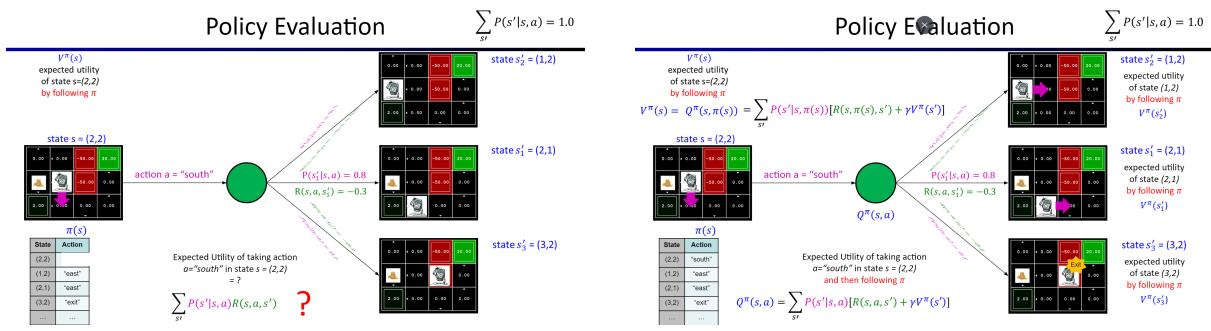
- Phần thưởng nhận sau n bước thì giá trị sẽ thay đổi
  - VD:  $U([1,2,3]) = 1 \times 1 + 0.5 \times 2 + 0.25 \times 3$
  - $U[(1,2,3)] < U[(3,2,1)]$
- Có 2 cách:
  - C1: Lấy tổng
  - C2: Cứ mỗi lần di 1 bước sẽ bình phương giá trị bước trước đó
  - Cách 2 xử lý được các dãy hành động vô tận

- chuỗi hình học:  $1 + \gamma^1 + \gamma^2 + \dots = \frac{1}{1-\gamma}$
- $U([r_0, \dots r_\infty]) = \sum_{t=0}^{\infty} \gamma^t r_t \leq \frac{R_{max}}{1-\gamma}$



## Policy Evaluation

- Đánh giá chiến lược nào tốt hơn : tính giá trị trạng thái khi tuân theo chiến lược  $\pi \rightarrow$  so sánh các chiến lược
- $V^\pi(s)$ : khi truyền vào trạng thái  $s$  thì giá trị trả về là độ *hữu dụng kì vọng* nhận được khi xuất phát từ  $s$  tuân theo chiến lược  $\pi \rightarrow$  **important**
- $\pi_2 \geq \pi_1$  khi và chỉ khi  $V^{\pi_2}(s) \geq V^{\pi_1}(s)$  với mọi  $s \in S$
- Nếu có chi phí bước đi  $\rightarrow$  độ hữu dụng = tổng phần thưởng
- Nếu có nhiều và có chi phí bước đi  $\rightarrow$  độ hữu dụng kì vọng = trung bình tổng phần thưởng (phải thực hiện hành động nhiều lần do có nhiều)
- Dãy hành động có tổng điểm thưởng cao hơn sẽ tốt hơn
  - VD: [1,2,2] < [2,3,4]
- Dãy hành động có điểm thưởng càng gần càng tốt hơn  $\rightarrow$  chiết khấu
  - VD: [0,0,1] < [1,0,0]
- Cuối cùng, đánh giá chiến lược là đi tính  $V^\pi(s)$  cho mọi trạng thái  $s$ 
  - Ví dụ:



→ tính kì vọng cho hành động đi xuống = xác suất x điểm thưởng ngay lập tức

(điểm thưởng ngay lập tức nên không có discount) → thiếu vì độ hữu dụng của 1 hành động từ 1 trạng thái là xuất phát từ trạng thái đó đến hết trò chơi → thêm phần hệ quả tương lai

- Ta có:

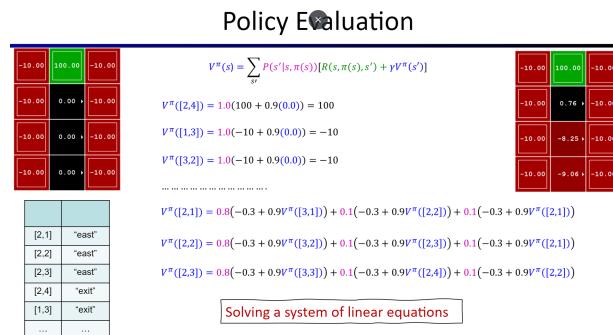
- $Q^\pi(s, a)$ : hàm giá trị hành động của chiến lược  $\pi$ , là độ hữu dụng kì vọng nhận được khi **làm hành động a** trong trạng thái  $s$  và **kể từ s'** trở đi tuân theo chiến lược  $\pi$  → cô lập hành động a ra khỏi dãy các hành động để thử hành động khác, nhưng sau đó tuân theo chiến lược  $\pi$  → xem xét giá trị hành động khác ⇒ khi thay đổi 1 thời điểm, tương lai sẽ thay đổi

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

- $V^\pi(s)$ : hàm giá trị trạng thái của chiến lược  $\pi$ , là độ hữu dụng kì vọng nhận được khi xuất phát từ trạng thái s tuân theo chiến lược  $\pi$  **từ trạng thái s**

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

- Ví dụ:



## Iterative Policy Evaluation

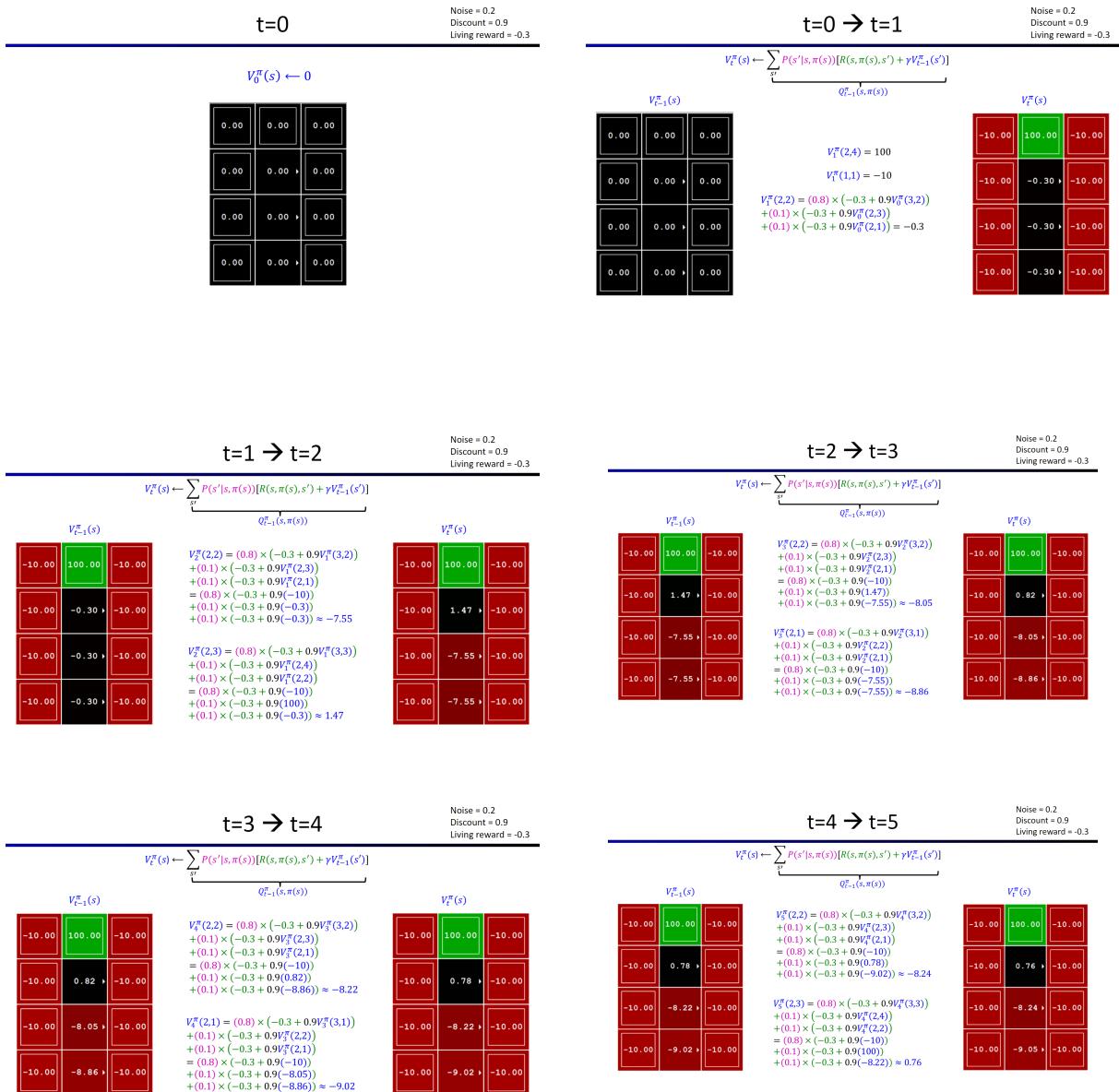
- Xấp xỉ các giá trị  $V^\pi(s)$

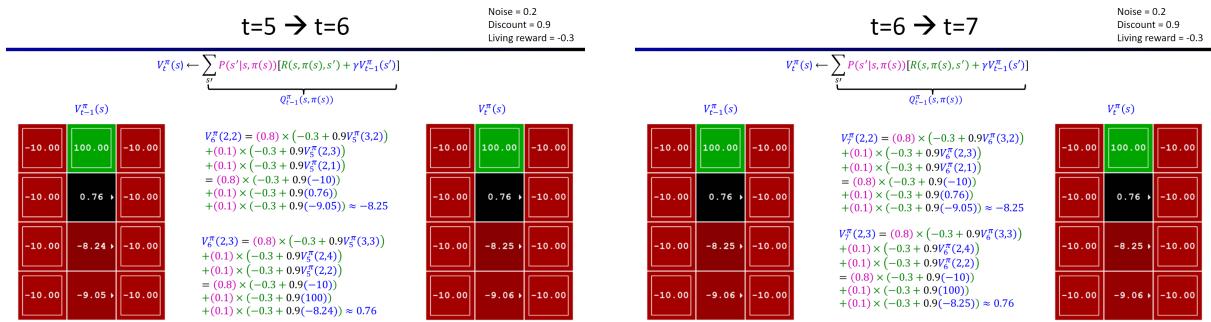
1. Initialize  $V_0^\pi(s) \leftarrow 0$  for all states  $s$ .
2. Repeat until convergence:  $t = 1, 2, 3, \dots$   
 $\text{// While } \max_s (|V_t^\pi(s) - V_{t-1}^\pi(s)|) > \varepsilon \approx 0.00001$

- For each state  $s$ :

$$V_t^\pi(s) \leftarrow \sum_{s'} P(s'|s, \pi(s)) [R(s, \pi(s), s') + \gamma V_{t-1}^\pi(s')] \\ Q_{t-1}^\pi(s, \pi(s))$$

- Ví dụ:





## Tóm tắt đánh giá chiến lược

- Để đánh giá chiến lược, sử dụng phương trình Bellman
  - Sử dụng iterative → Độ phức tạp:  $O(S^2)$  mỗi vòng lặp

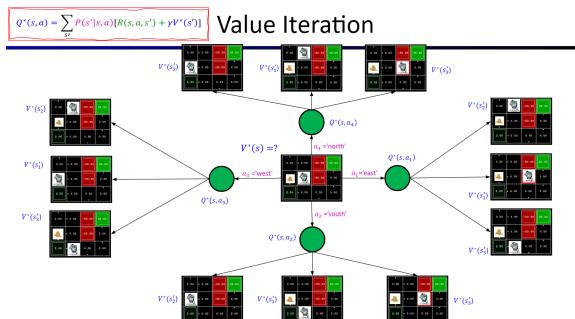
$$V_0^\pi(s) \leftarrow 0$$

$$V_t^\pi(s) \leftarrow \sum_{s'} P(s'|s, \pi(s)) [R(s, \pi(s), s') + \gamma V_{t-1}^\pi(s')]$$

- Giai hệ phương trình tuyến tính

## Value Iteration

- Xác định  $V^*(s)$ : giá trị trạng thái tối ưu - độ hữu dụng kì vọng nhận được khi bắt đầu trạng thái s, tuân theo chiến lược tối ưu  $\pi^*$  → ta không biết chiến lược tối ưu là gì nhưng có thể tính được giá trị tối đa kì vọng nhận được



sau khi tính được Q → chọn hành động có Q lớn nhất  
(giá trị hành động lớn nhất)

$$\pi^*(s) = \underset{a}{\operatorname{argmax}} Q^*(s, a)$$

trả về hành động làm Q\* lớn nhất

$$V^*(s) = \underset{a}{\operatorname{max}} Q^*(s, a)$$

trả về giá trị Q\* lớn nhất

- $Q^*(s, a)$ : giá trị hành động tối ưu - độ hữu dụng kì vọng khi bắt đầu ở trạng thái s làm hành động a và kề từ s' trở đi tuân theo chiến lược tối ưu
- $\pi^*(s)$  - chiến lược tối ưu, hành động cần làm trong trạng thái s này là gì

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

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

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

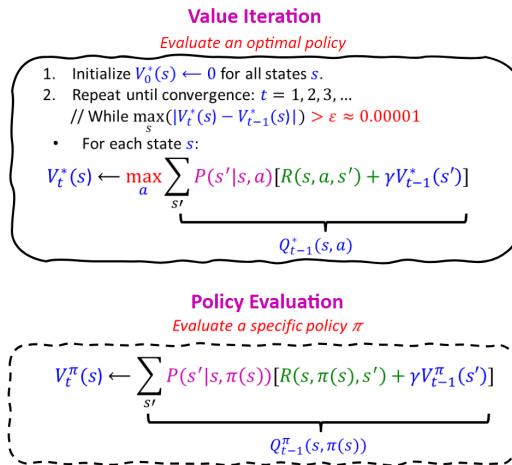
## Phương trình Bellman tối ưu

- Làm hành động đầu tiên đúng → Tiếp tục tối ưu

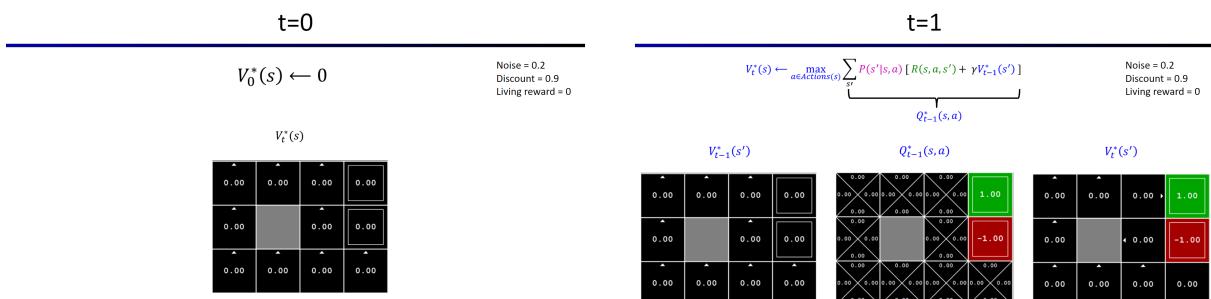
$$V^*(s) = \max_a \sum_{s'} P(s, a, s')[R(s, a, s') + \gamma V^*(s')]$$

⇒ Phương trình phi tuyến tính ⇒ Dùng qui hoạch động và xấp xỉ

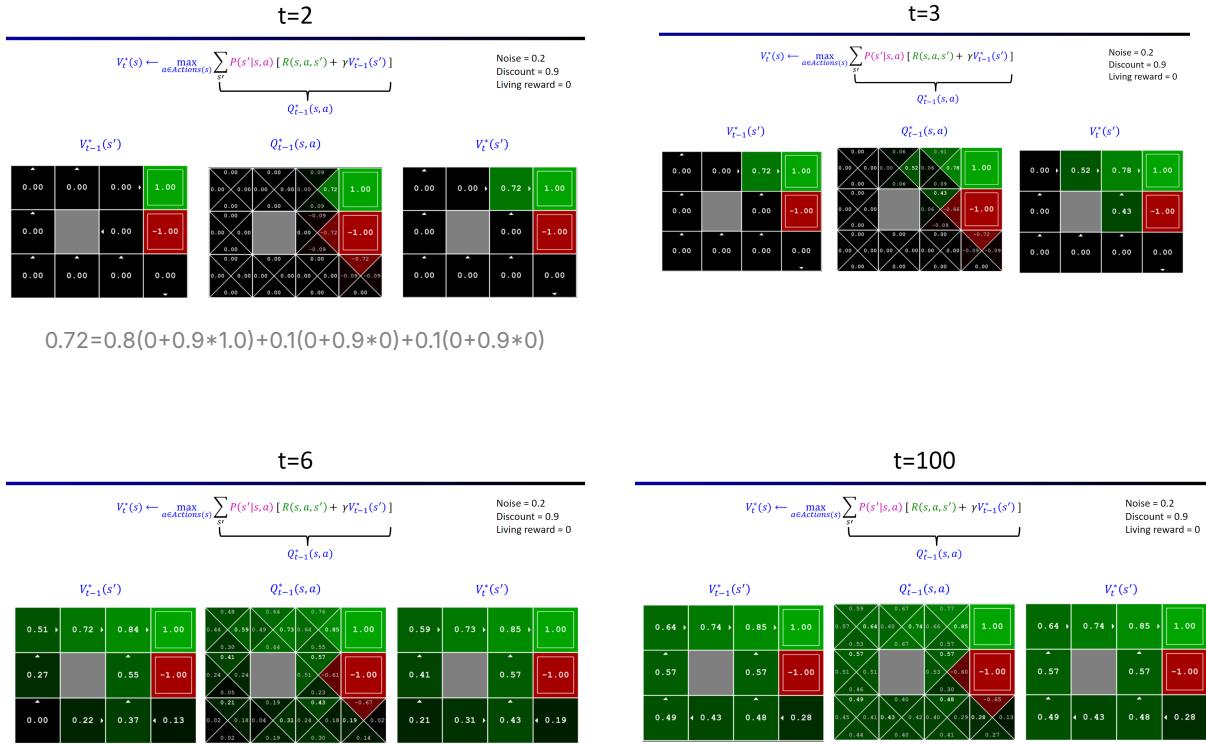
- Thuật toán:



- Ví dụ:



giả sử chi phí mỗi bước di (living reward)=0 và  $V^*(t-1) = 0$  hết nên  $Q^*(t-1)=0$  hết

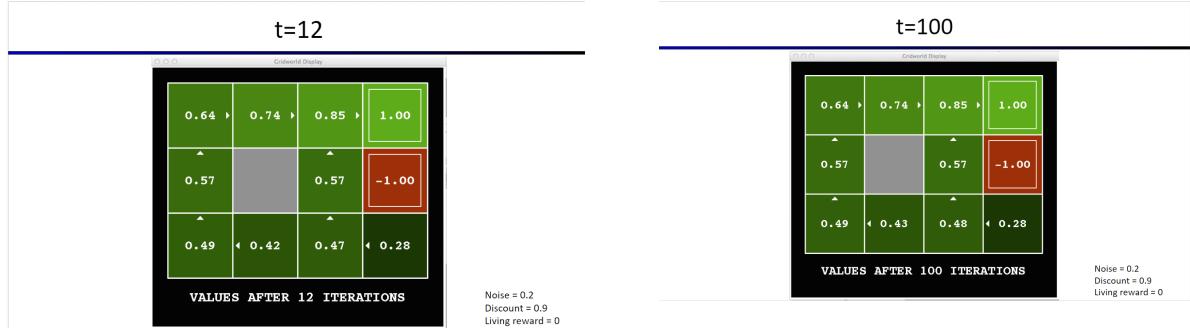


## Policy Extraction

- Rút chiến lược tối ưu  $\pi^*$  từ  $V^*$  bằng cách chạy thêm 1 bước rút trích chiến lược tối ưu
- $V^*(s, a) \rightarrow \pi^*(s) = \arg \max_a Q^*(s, a)$

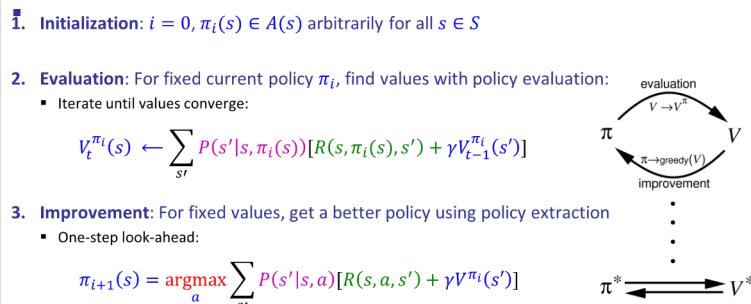
## Vấn đề của Value Iteration

- Rất chậm:  $O(S^2 A)$  mỗi vòng lặp
- Gia trị tối đa ở mỗi trạng thái ít khi thay đổi
- Chiến lược thường hội tụ chậm hơn giá trị



# Policy Iteration

- Có 2 bước chính:
  1. Policy Evaluation: tính toán giá trị hữu dụng tới khi hội tụ
  2. Policy Improvement: cập nhập chiến lược sử dụng giá trị hữu dụng nhìn trước 1 bước đã hội tụ như là giá trị tương lai → chiến lược mới sẽ tốt hơn chiến lược cũ
  3. Lặp lại đến khi chiến lược hội tụ
- Thuật toán:



## Tóm tắt: các thuật toán MDP

- Tính toán giá trị tối ưu: dùng value iteration/policy iteration
- Tính toán giá trị cho 1 chiến lược cụ thể: policy evaluation
- Chuyển giá trị thành chiến lược: policy extraction

⇒ Tất cả thuật toán đều là các biến thể của phương trình Bellman tối ưu, đều sử dụng nhìn trước 1 bước của expectimax