机器学习

深 度 强 化 学 习

动态规划求解方法/GridWorld

徐哲 10月13日

课程目标

- 掌握MDP中常见符号的含义
  - V、Q、π
- 掌握通过动态规划求解MDP的两种方法
  - Value Iteration
  - Policy Iteration
- 理解MDP求解与强化学习的联系与区别

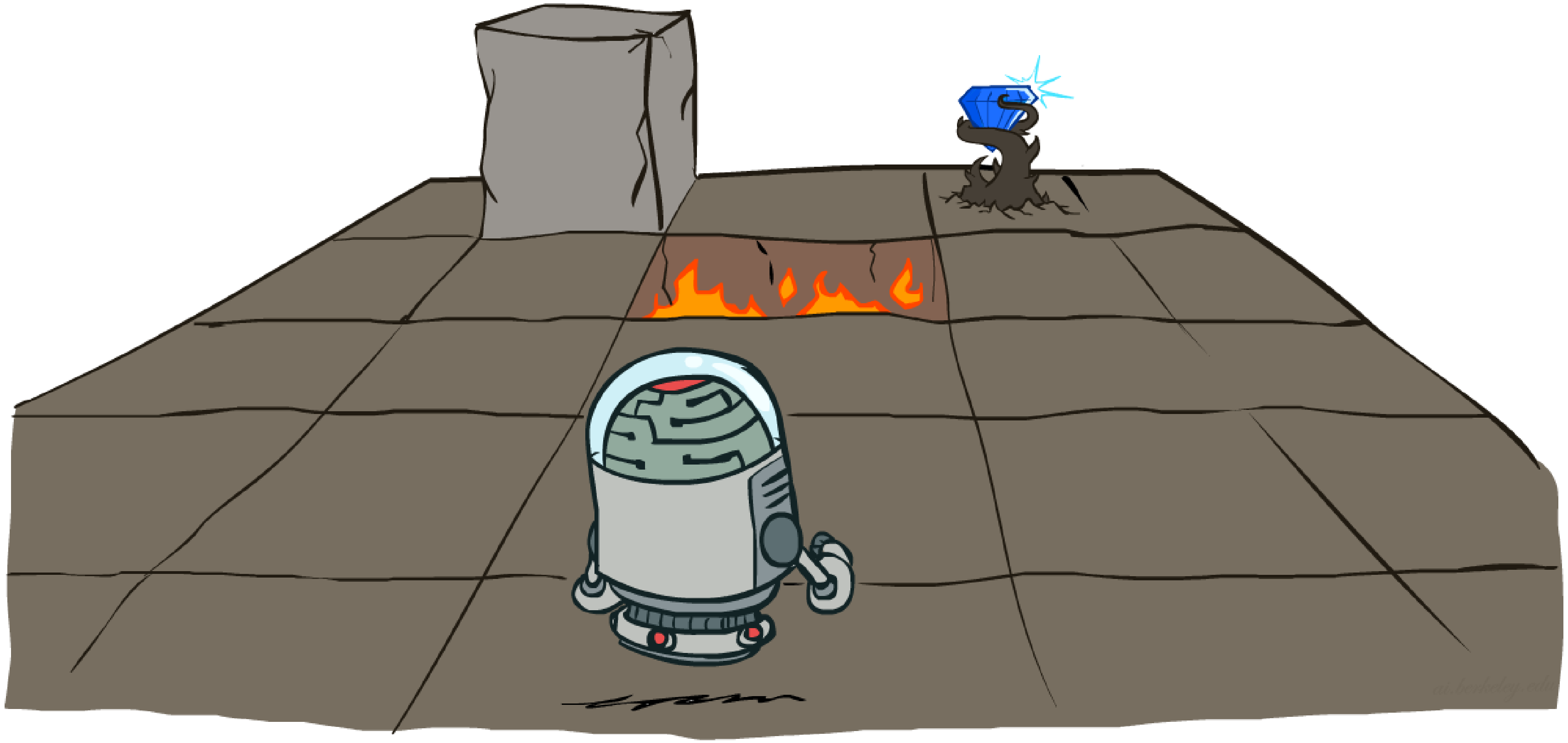
目录
Contents

01

第一章

主题：马尔可夫决策过程-价值函数

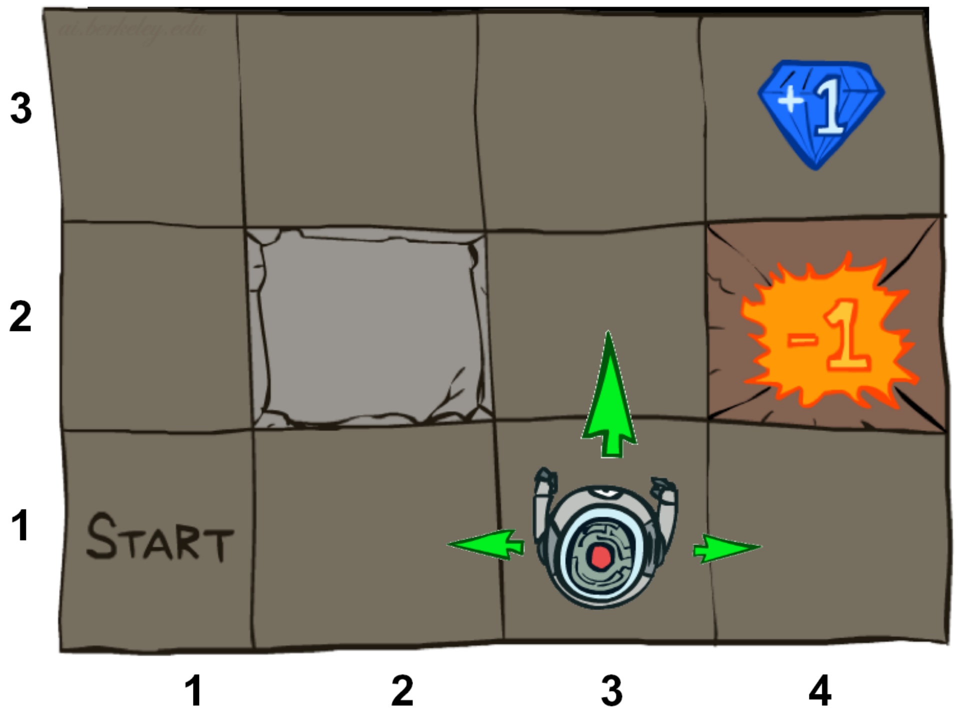
马尔可夫决策过程



PPT插图来源于Berkeley CS188课件 http://ai.berkeley.edu/home.html

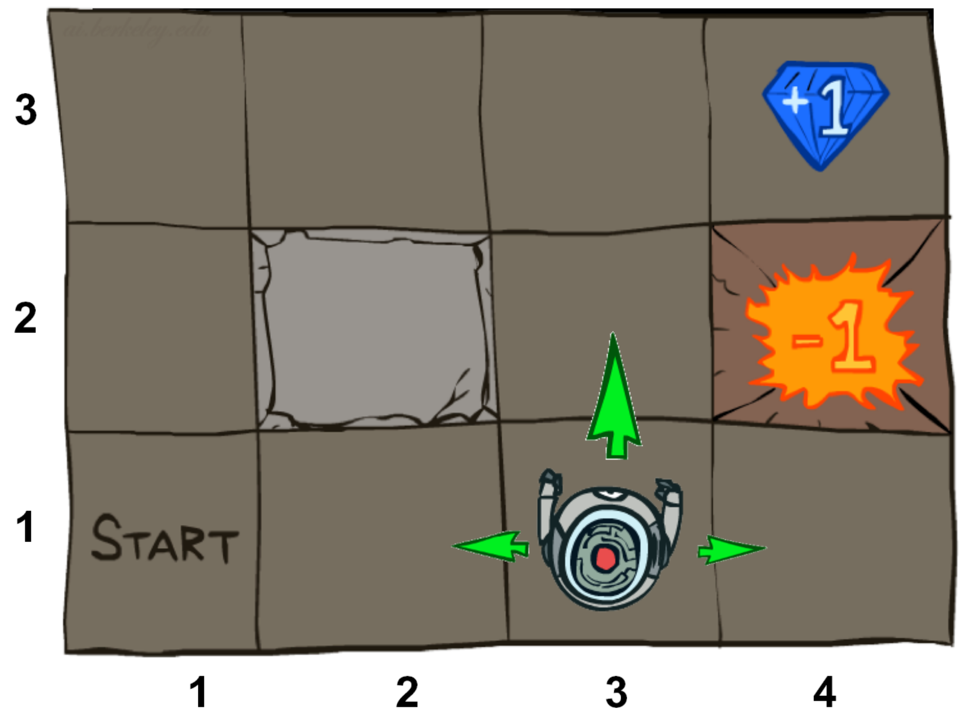
格子世界 (Grid World)

- 类迷宫问题
  - Agent在以格子为单位存在和移动
  - 不能移动到迷宫范围之外，墙会阻挡agent移动
- 奖励
  - 部分格子会有特殊的奖励，出现在这些格子会结束一轮游戏
  - 钻石 +1 表示正激励，火堆 -1 表示负激励
  - 每一步可能会有一个小小的生存激励（可能是负的）
- Agent的目标为最大化期望累积收益



（回顾）MDP

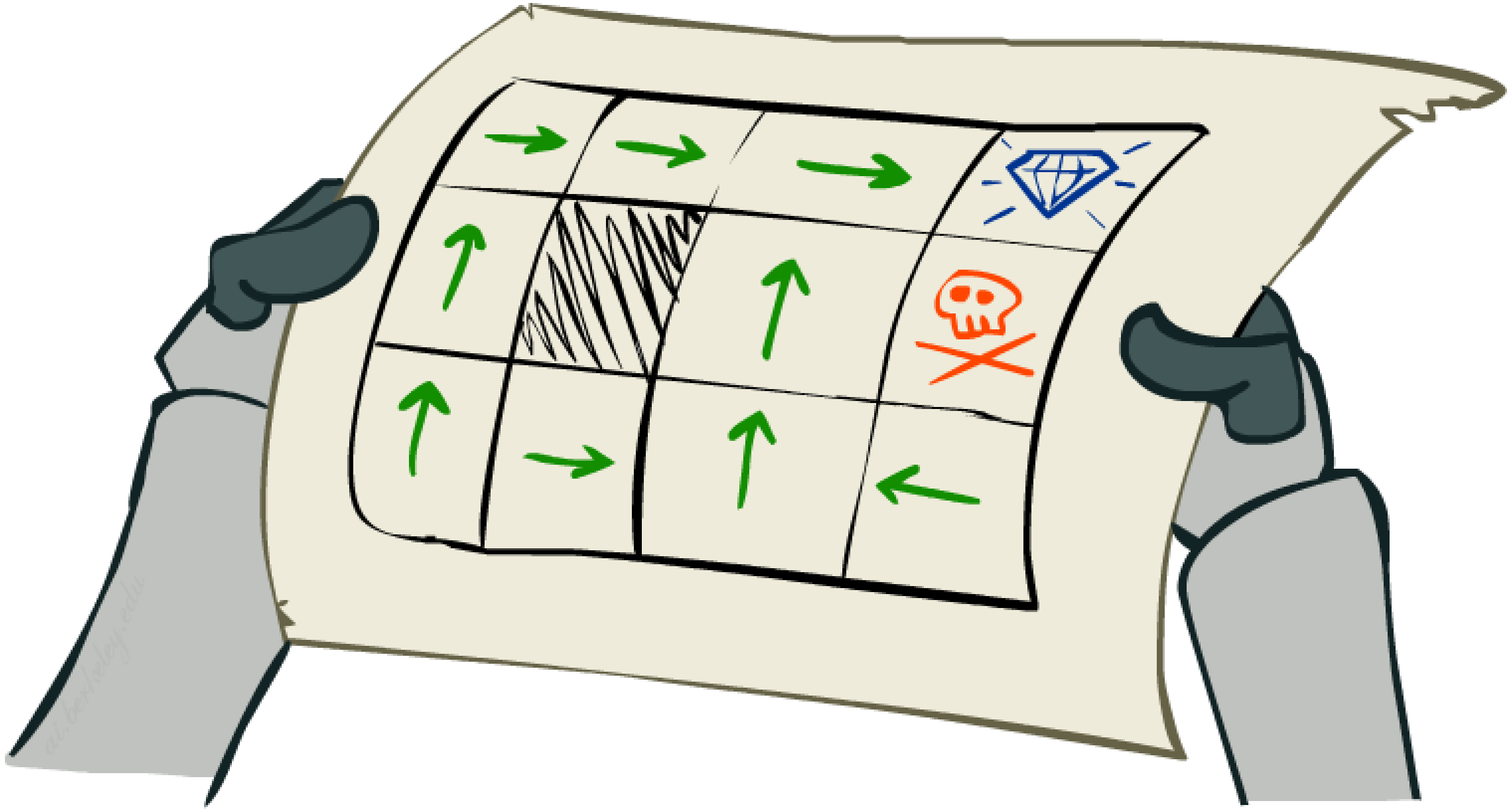
- **一个马尔可夫决策过程(MDP)由以下元素表示 $< \boldsymbol{S}, \boldsymbol{A}, \boldsymbol{P}, \boldsymbol{R}, \boldsymbol{\gamma} >$：**
  - 一系列状态 $s \in S$
  - 一组动作 $a \in A$
  - 状态转移函数 $\mathrm{P}_{ss'}^{a}$
    - 即$P(s'|s,a)$
  - 奖励函数 R_s^a
  - 衰减因子 γ
  - 一个完整的从起始状态到终止状态的过程，称为一个回合(episode)
    - 起始状态 s_0
    - （可能的）终止状态 s_{end}



目标为最大化累积收益(Return)

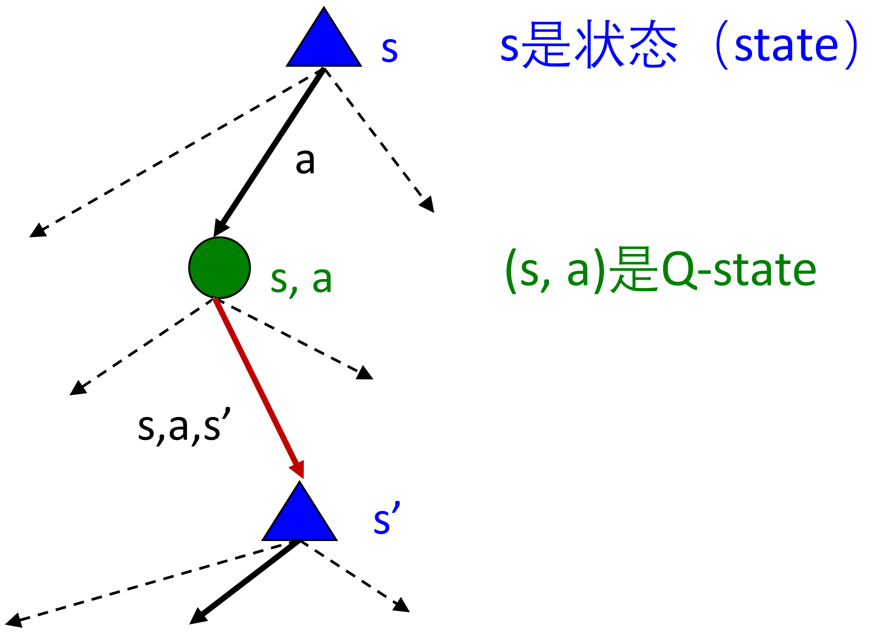
$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots$$

MDP求解



几个常用记号

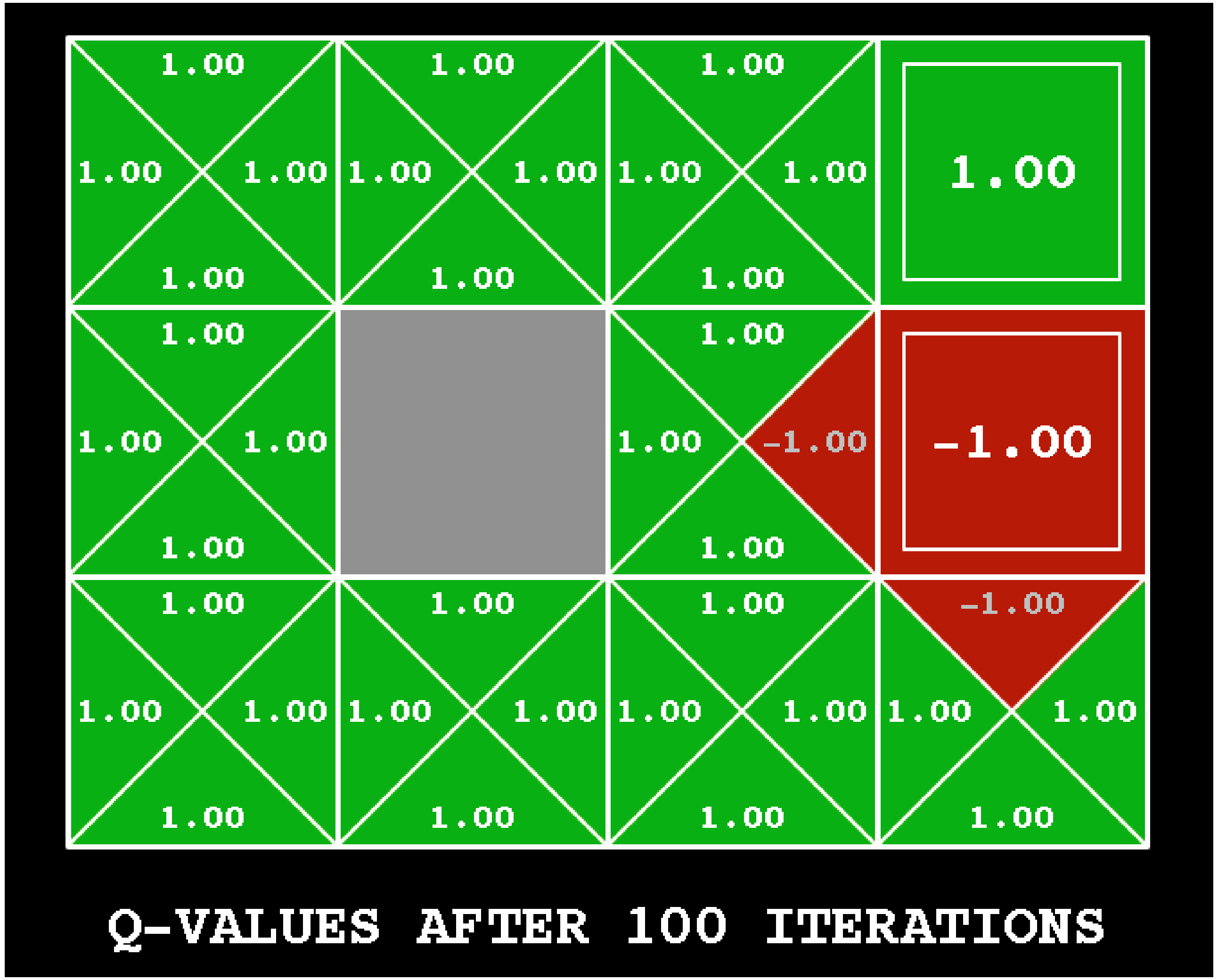
- 状态s的价值
  - V*(s)=从s开始，按照最优决策行动的期望总回报
- 状态-动作（Q-state）的价值：
  - Q*(s,a)=在s采取动作a，之后按照最优决策行动的期望总回报
- 最优策略：
  - $\pi^*(s)$ =在s应该采取的最优动作



示例：Grid World的V值

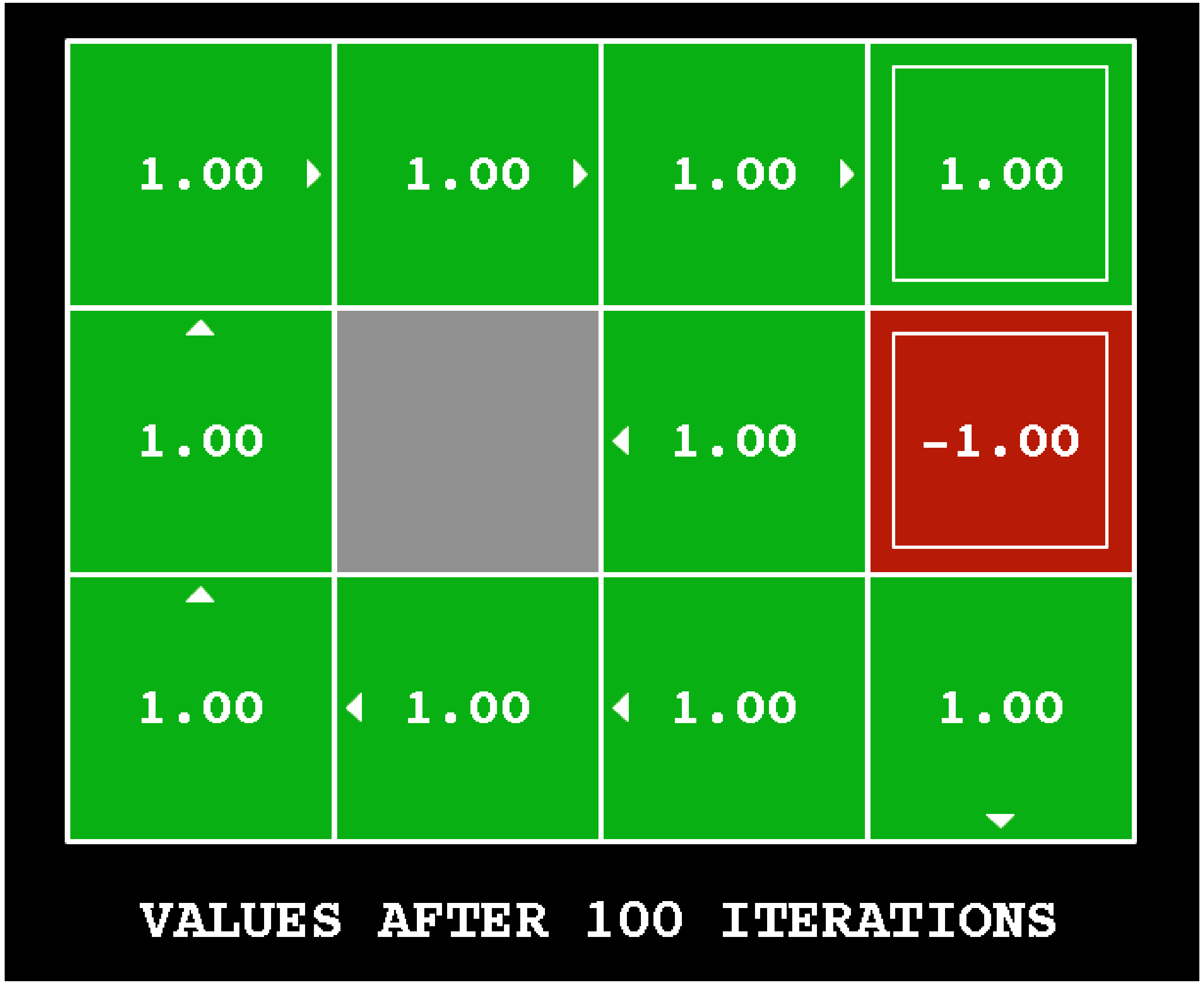
Noise = 0
Discount = 1
Living reward = 0

示例：Grid World的Q值



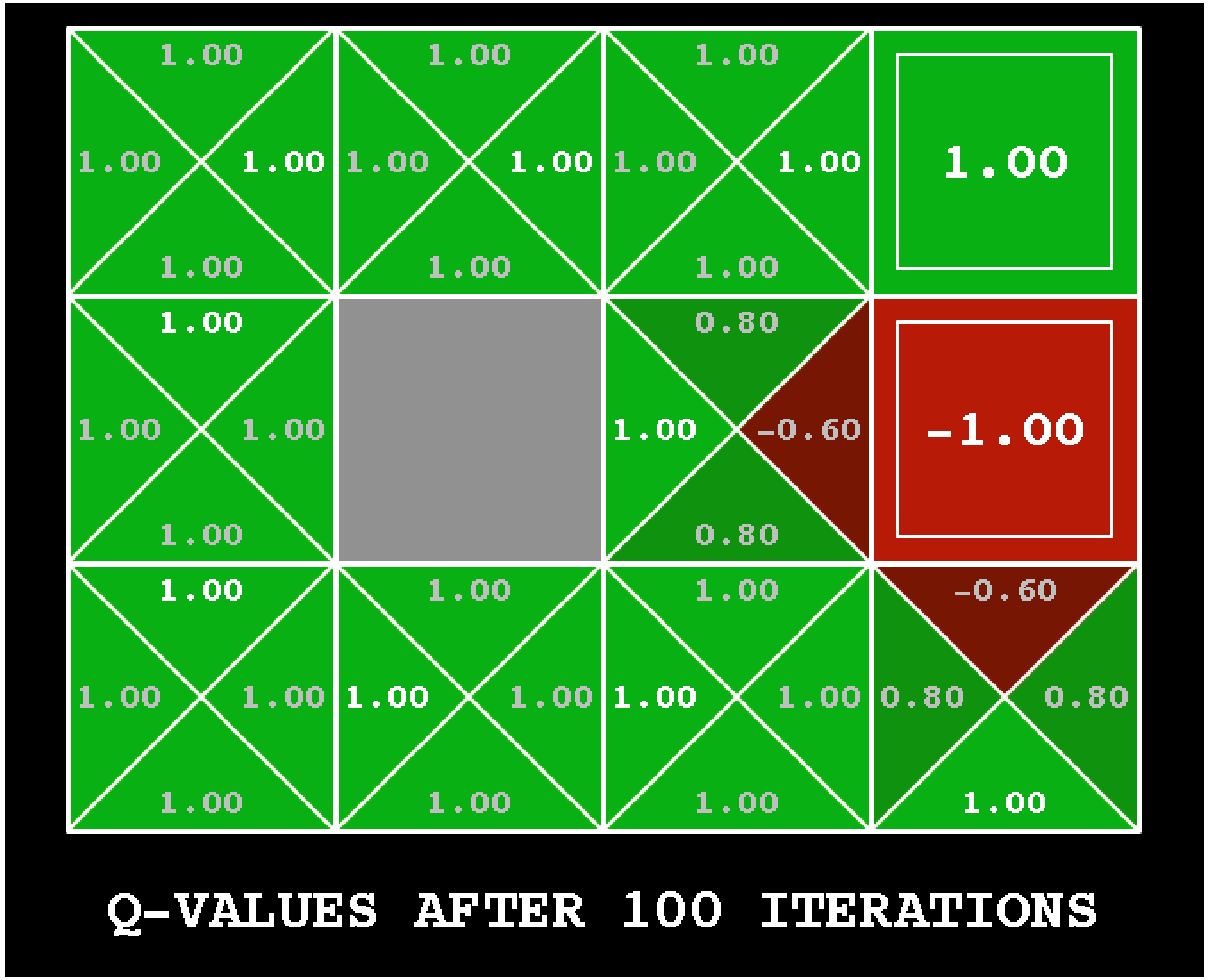
Noise = 0
Discount = 1
Living reward = 0

示例：Grid World的V值



Noise = 0.2
Discount = 1
Living reward = 0

示例：Grid World的Q值



Noise = 0.2
Discount = 1
Living reward = 0

示例：Grid World的V值

Noise = 0.2
Discount = 0.9
Living reward = 0

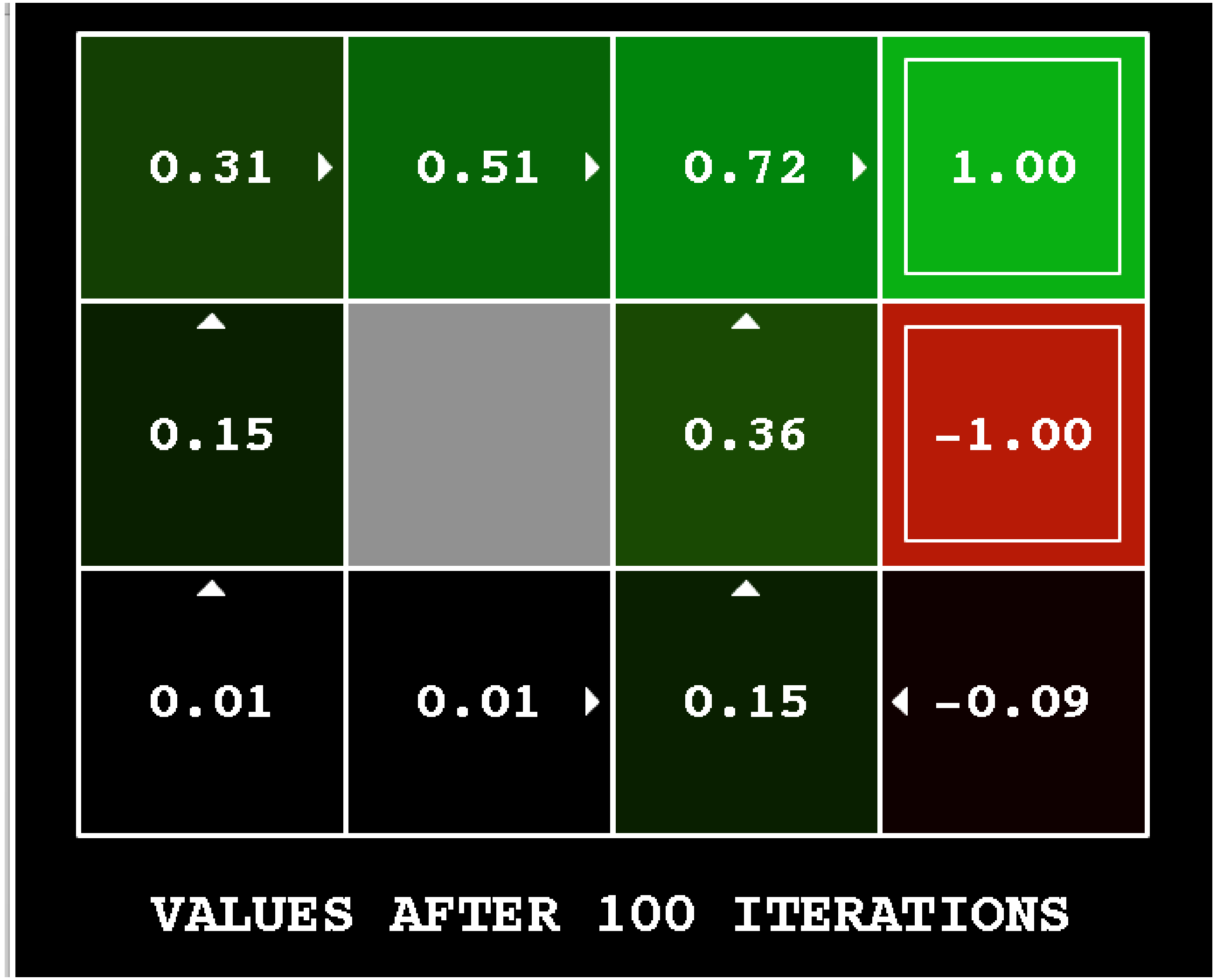
示例：Grid World的Q值

Noise = 0.2

Discount = 0.9

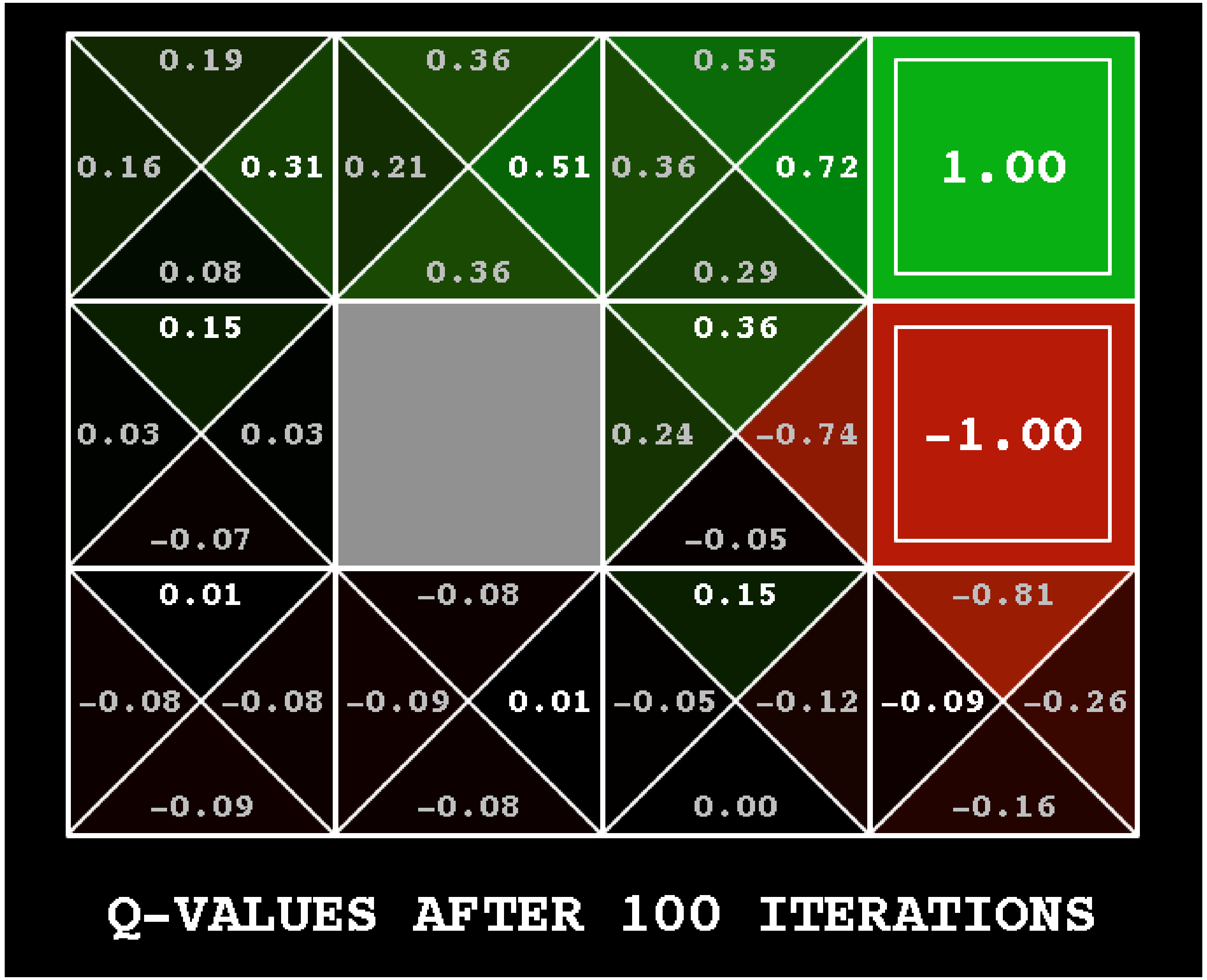
Living reward = 0

示例：Grid World的V值



Noise = 0.2
Discount = 0.9
Living reward = -0.1

示例：Grid World的Q值



Noise = 0.2
Discount = 0.9
Living reward = -0.1

状态的价值

- V*(s)：状态s在最佳策略下的期望收益
  - 实际上就是采取最优动作a之后的平均收益
- 可以如下计算：

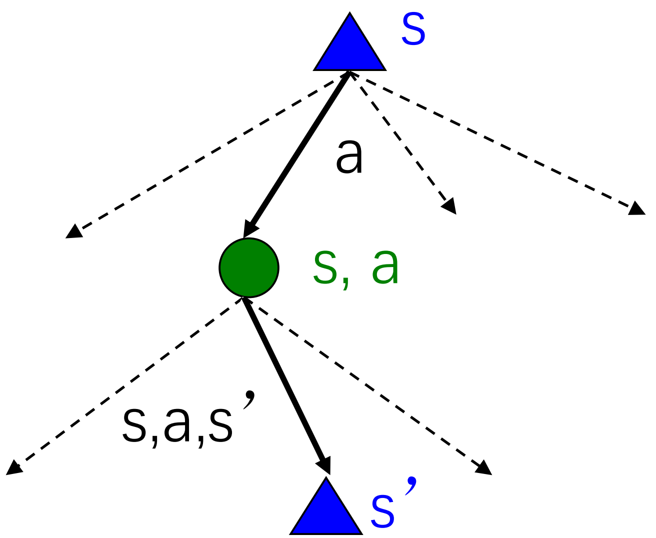
$$V^*(s) = \max_a Q^*(s,a)$$

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

在s做a，转移到s’ 的概率　　在s做a，转移到s’ 的直接奖赏

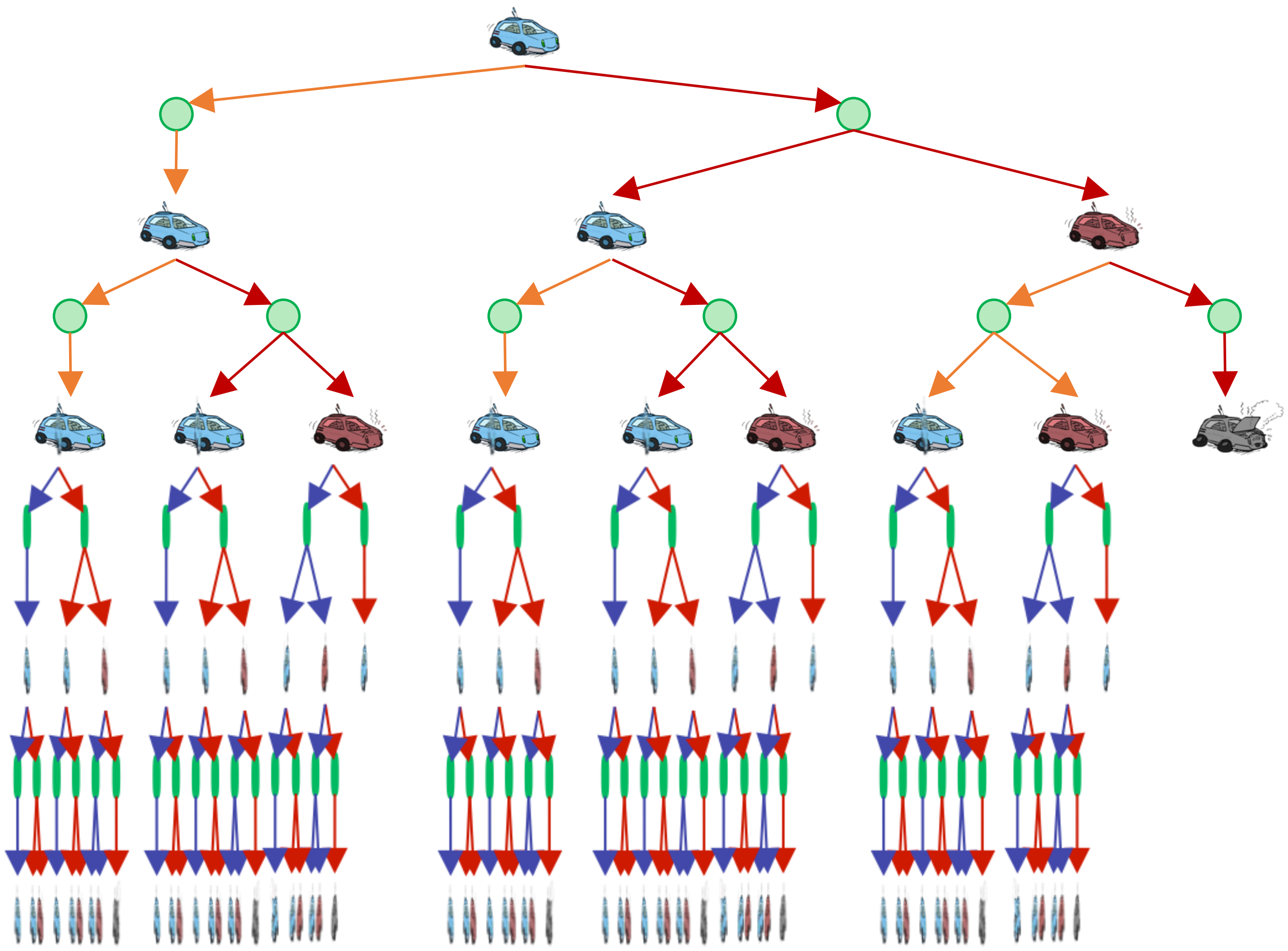
$$V^*(s) = \max_a \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma V^*(s')\right]$$

- 这就是贝尔曼方程（Bellman equations）



求解V*(s)

- 可以靠搜索来求解
  - 但计算复杂度太大了
- 有很多重复状态
  - 可以记忆下来！
- 搜索要搜很多层才结束
  - 可以规定最多只搜多少层，如果γ<1，深层的其实就没什么用了
  - 更好的方法是自底向上地迭代更新



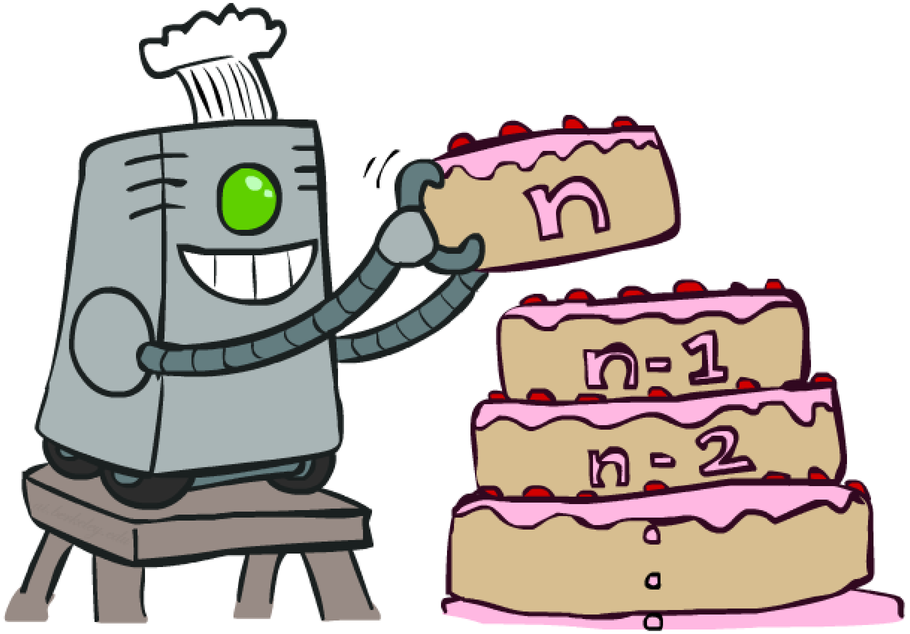
02 第二章

主题：价值迭代 Value Iteration

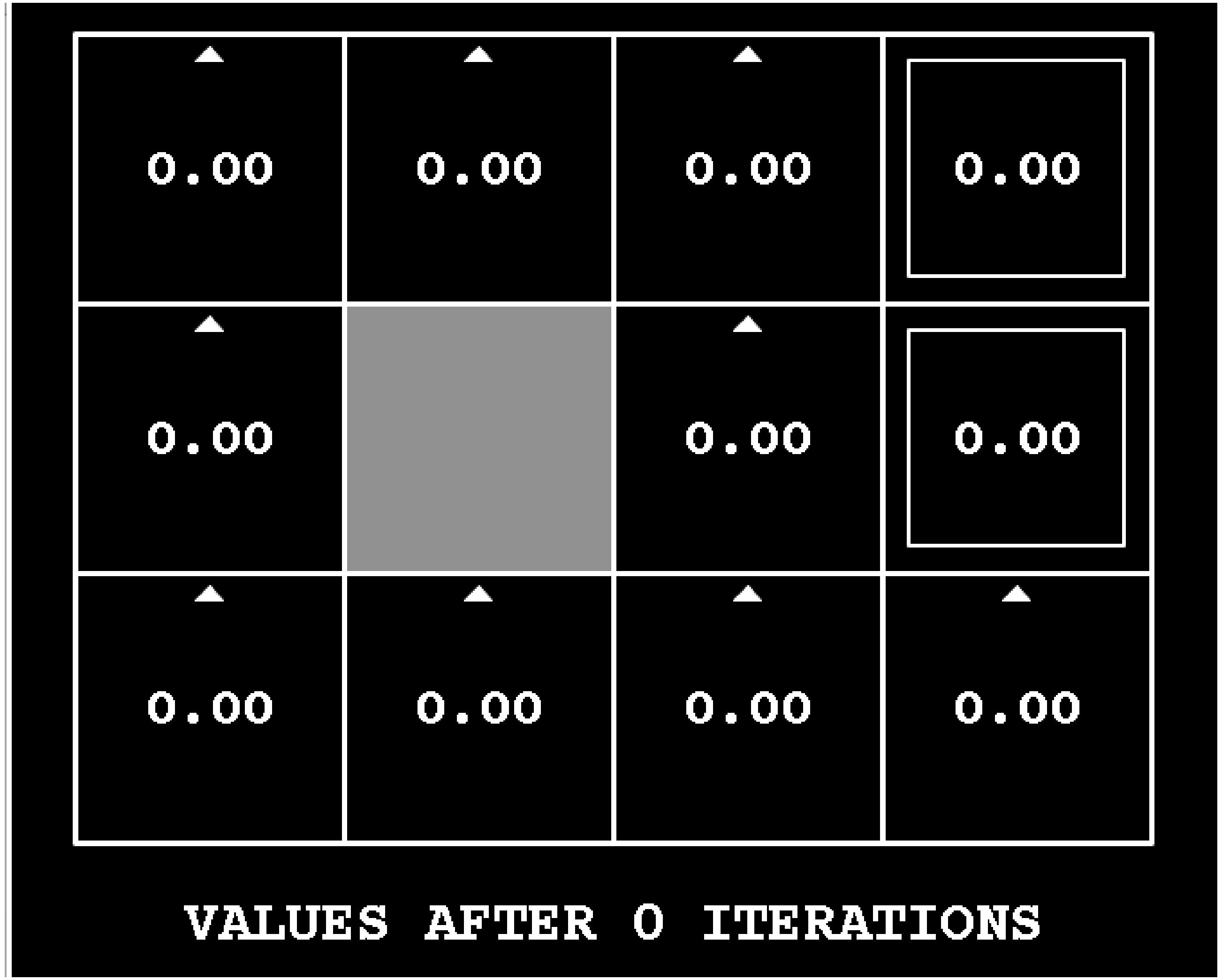
Value Iteration

- 从最后状态开始往前更新
- $V_k(s)$：k步以后游戏就结束，这种状态下的$V^*(s)$

- $V_0(s)=0$
  - 游戏结束时的期望回报=0
- $V_{k+1}(s)$可以从$V_k(s)$推出

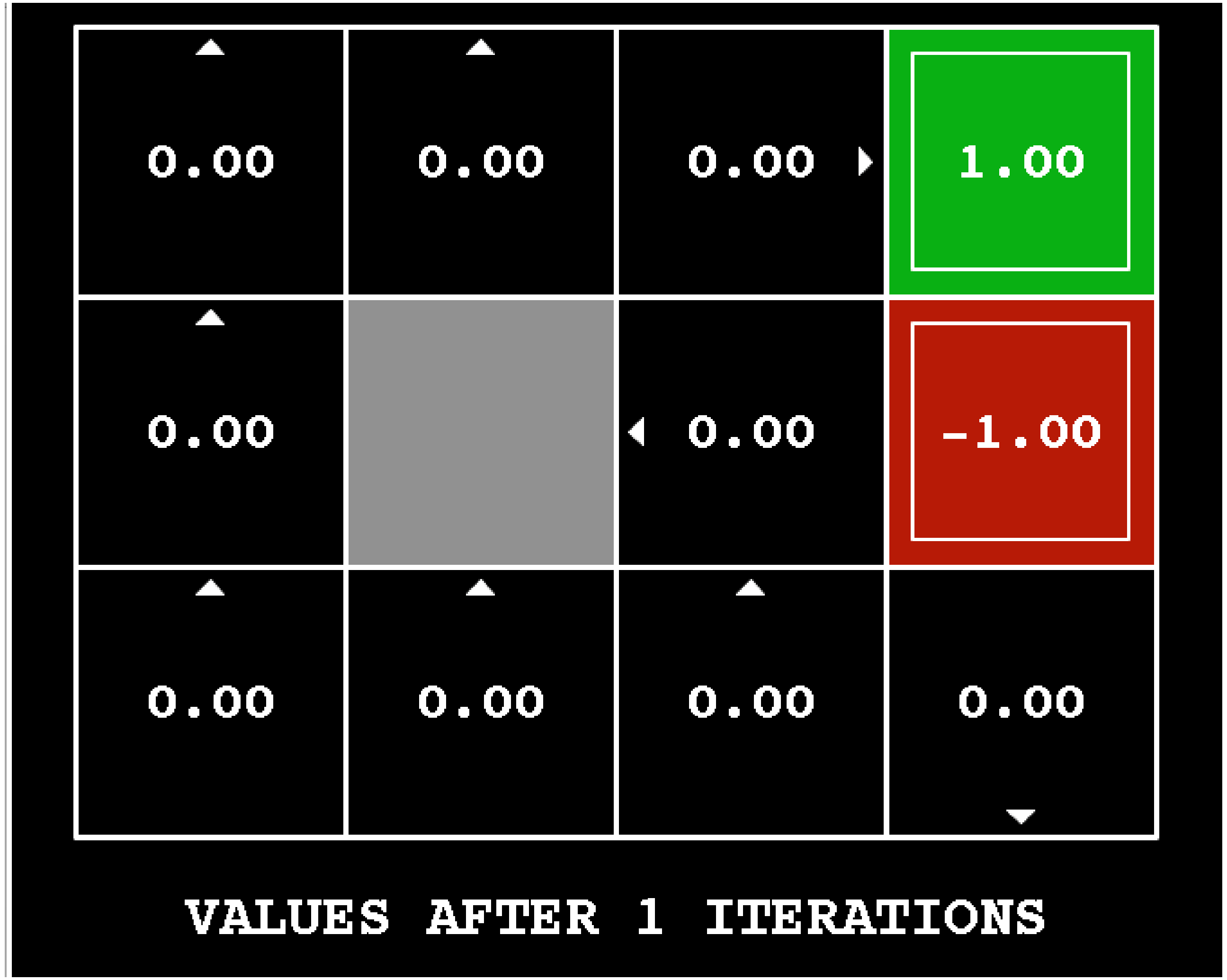


K=0



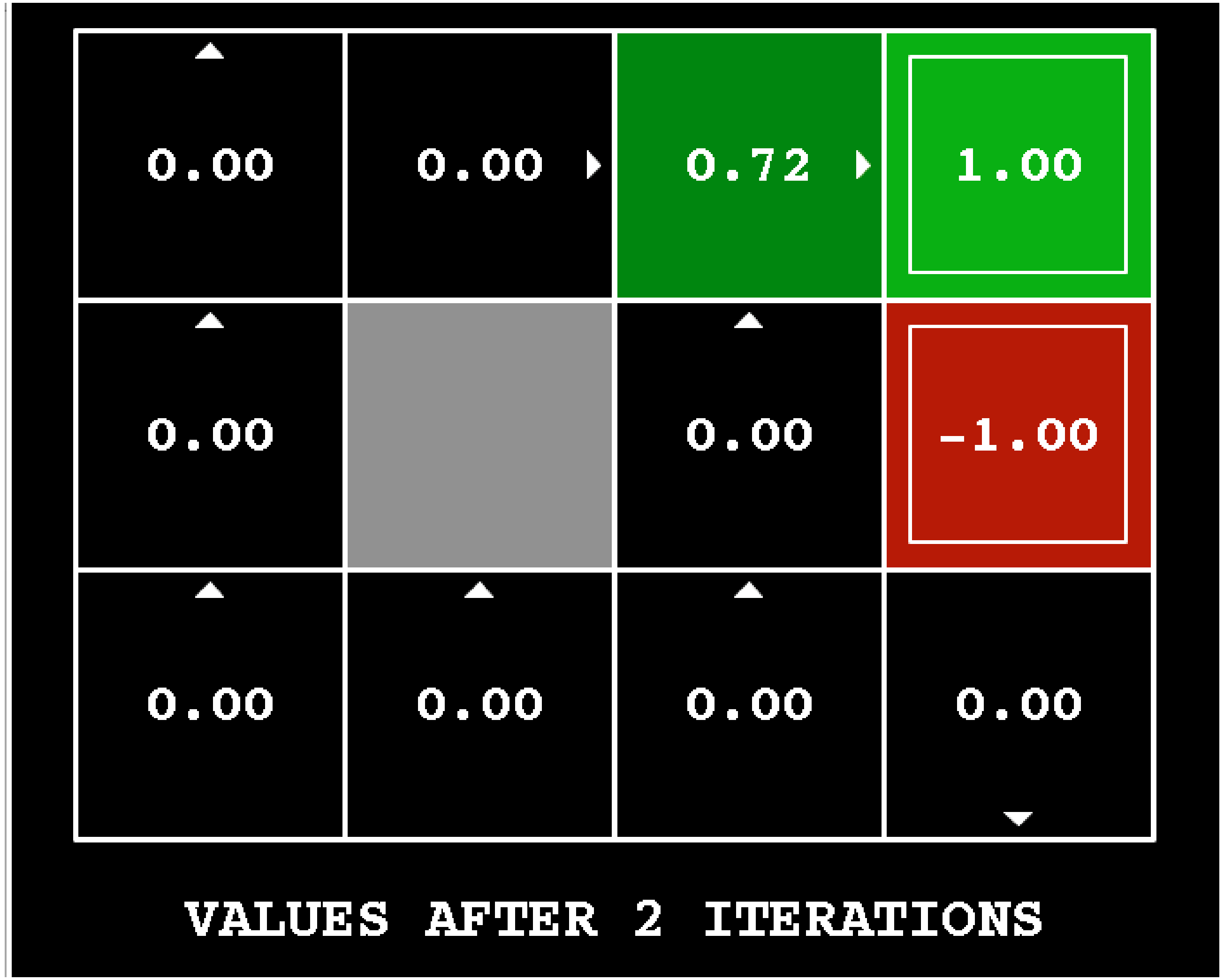
Noise = 0.2
Discount = 0.9
Living reward = 0

k=1



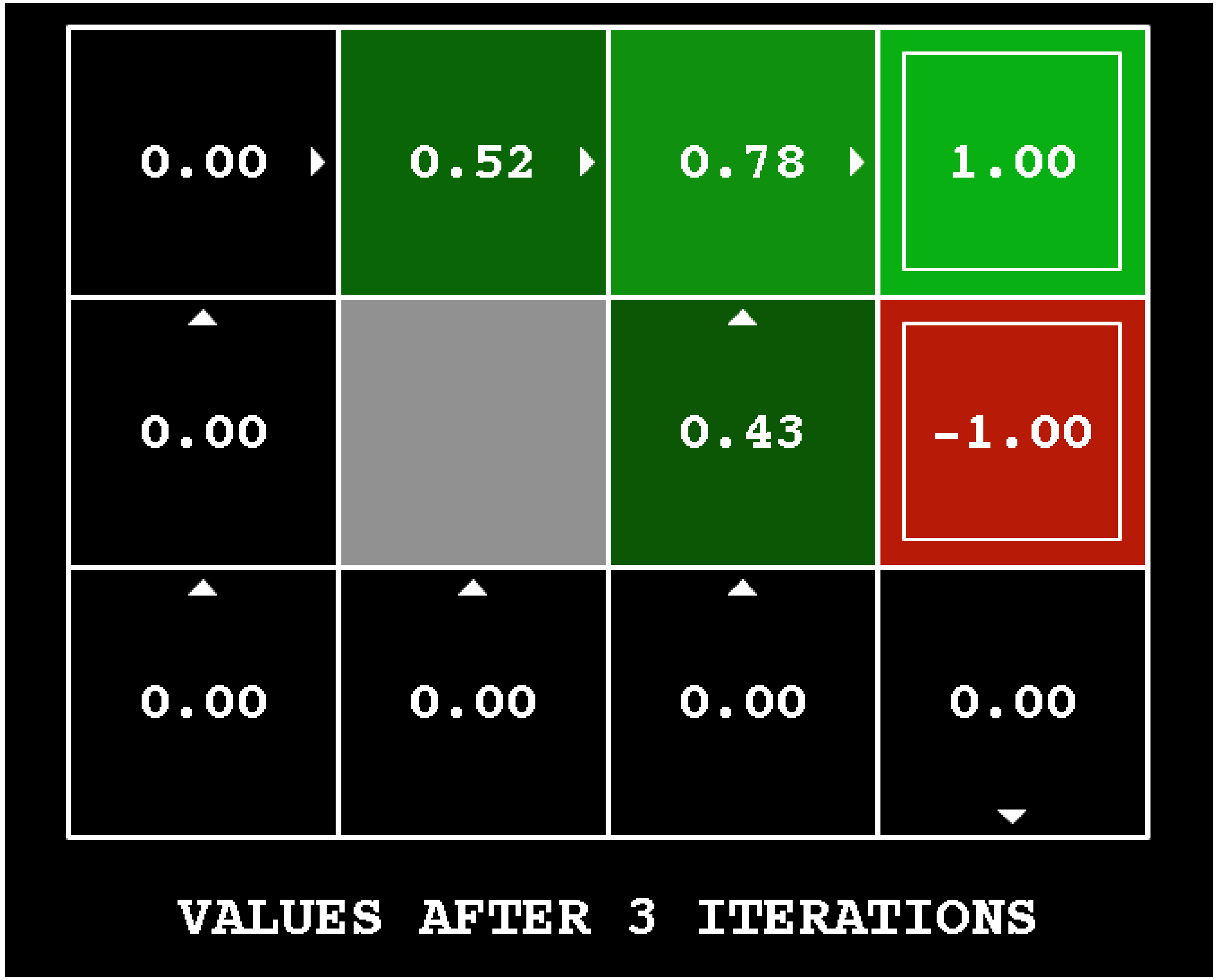
Noise = 0.2
Discount = 0.9
Living reward = 0

k=2



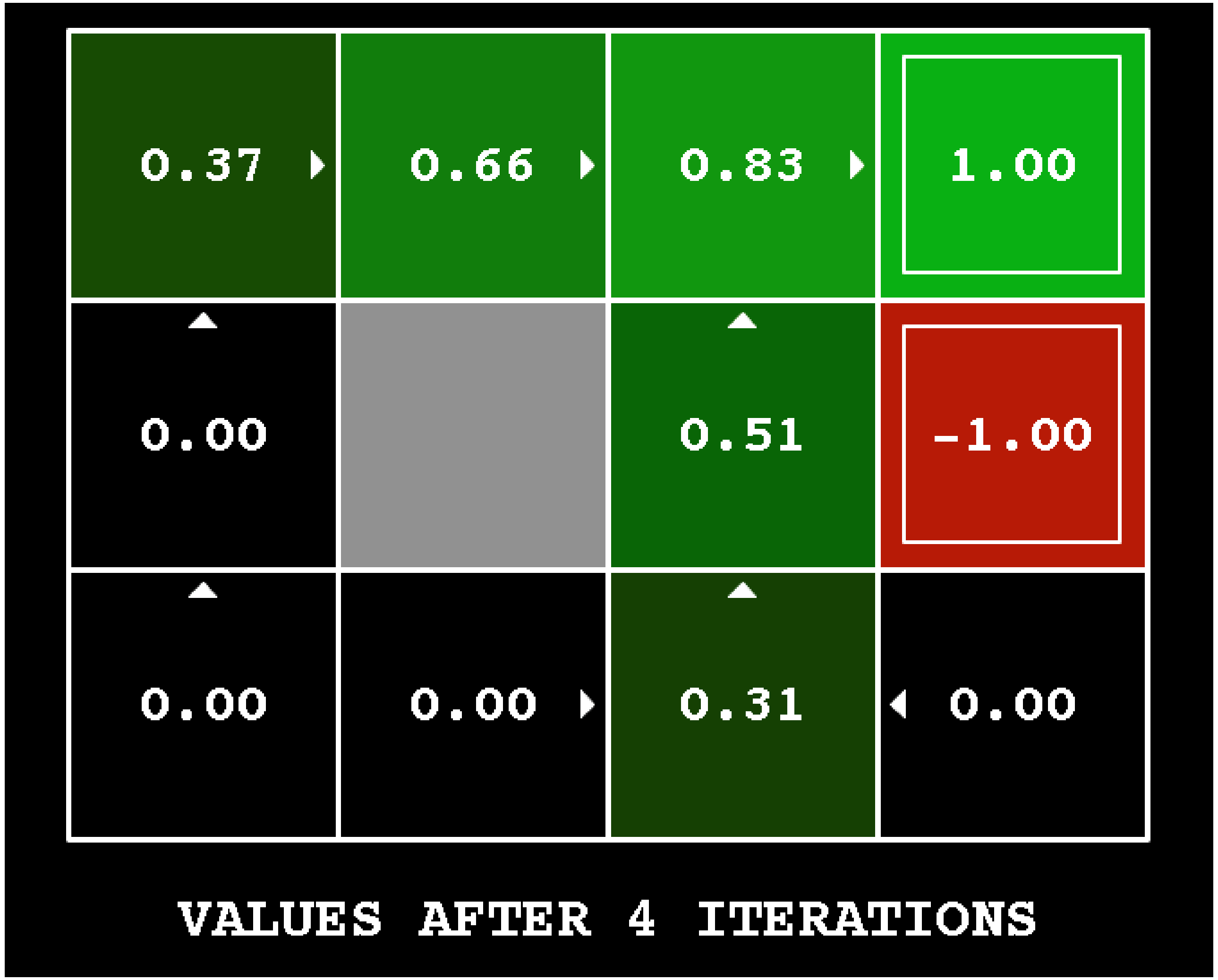
Noise = 0.2
Discount = 0.9
Living reward = 0

k=3



Noise = 0.2
Discount = 0.9
Living reward = 0

k=4



Noise = 0.2
Discount = 0.9
Living reward = 0

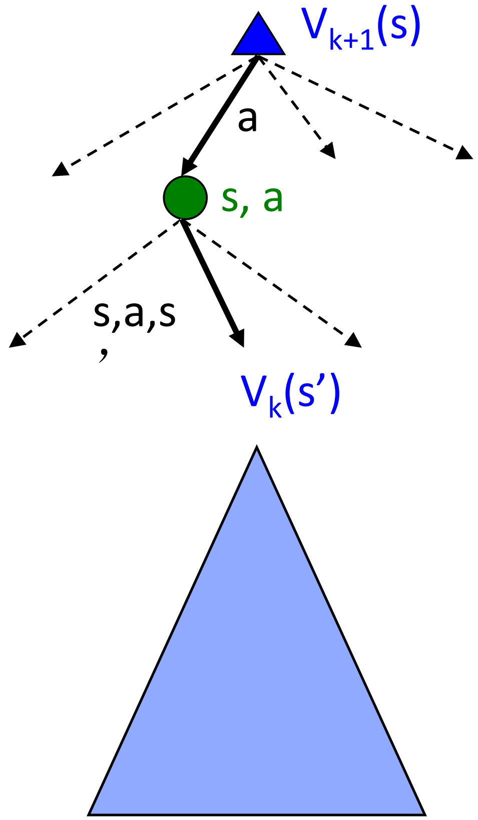
k=100

Noise = 0.2
Discount = 0.9
Living reward = 0

Value Iteration

- 从$V_0(s)$=0开始不断往前推算
- 已知$V_k(s)$，根据定义就可以很容易地计算$V_{k+1}(s)$：

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



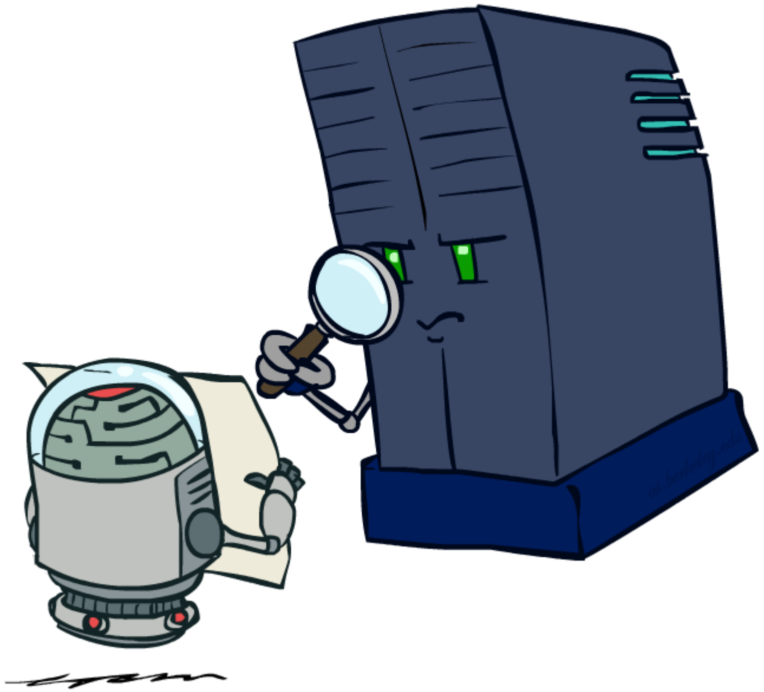
- 一直迭代，直到收敛
- 定理：一定会收敛到最优值
  - 基本思想：估计值会不断地向最优值靠近
  - 比起V，策略π要收敛得快得多

03 第三章

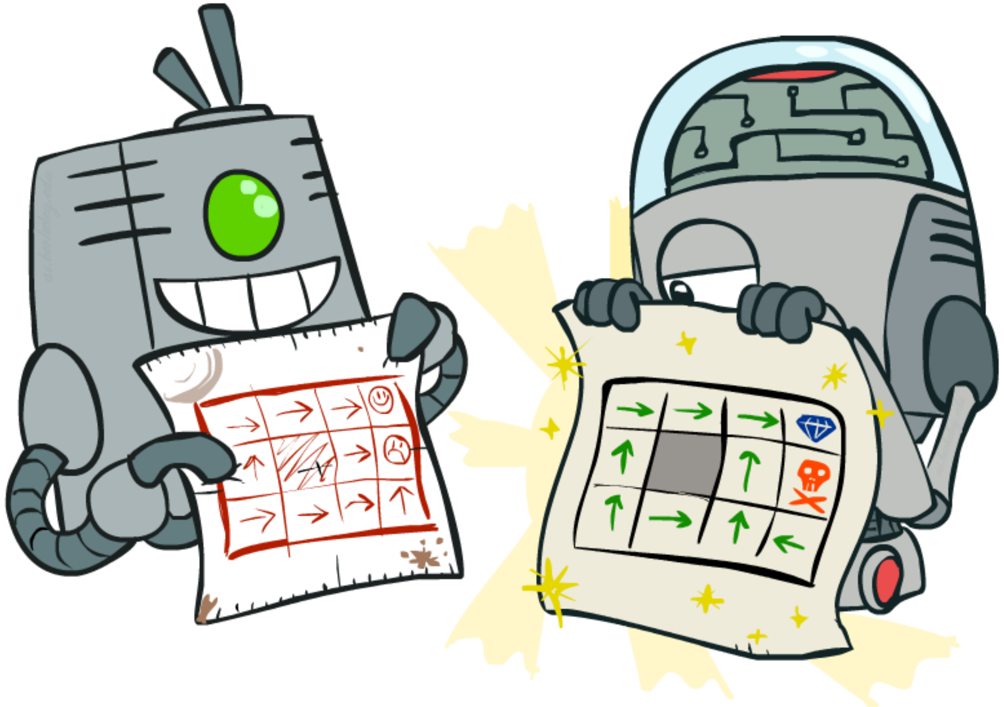
主题：策略迭代 Policy Iteration

基于Policy的方法

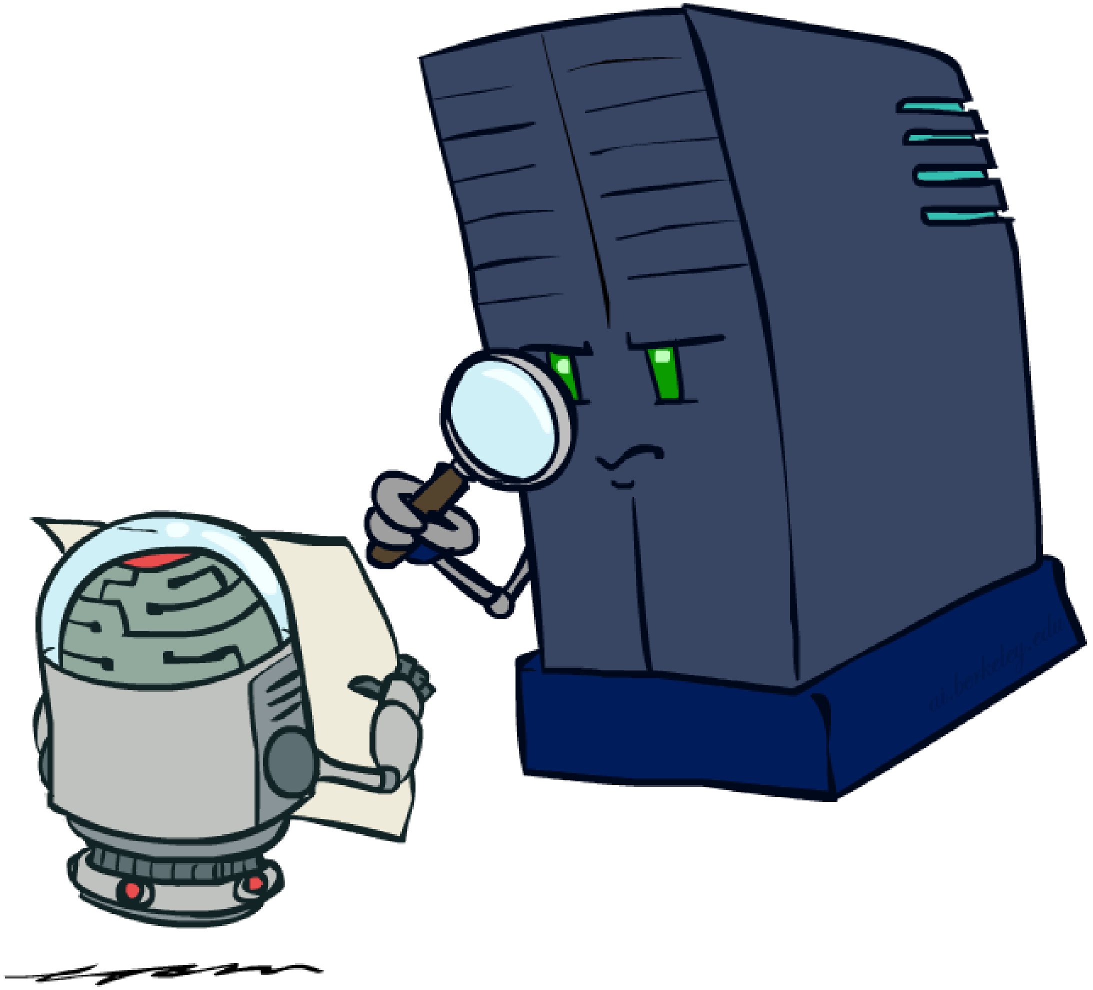
评估 (Evaluation)



优化 (control)



Policy Evaluation

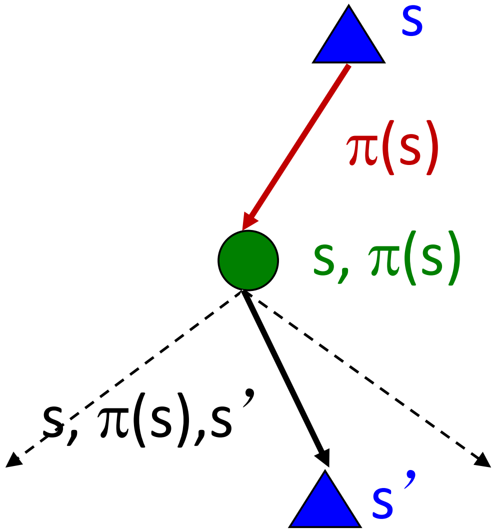


给定策略下的V值

- 当给定一个策略π的时候（不一定是最优策略）也可以计算V值
- $V^\pi(s)$=从s开始，完全按照π的指示行动，这样的期望收益

- 同样也可以用Bellman Equation更新：

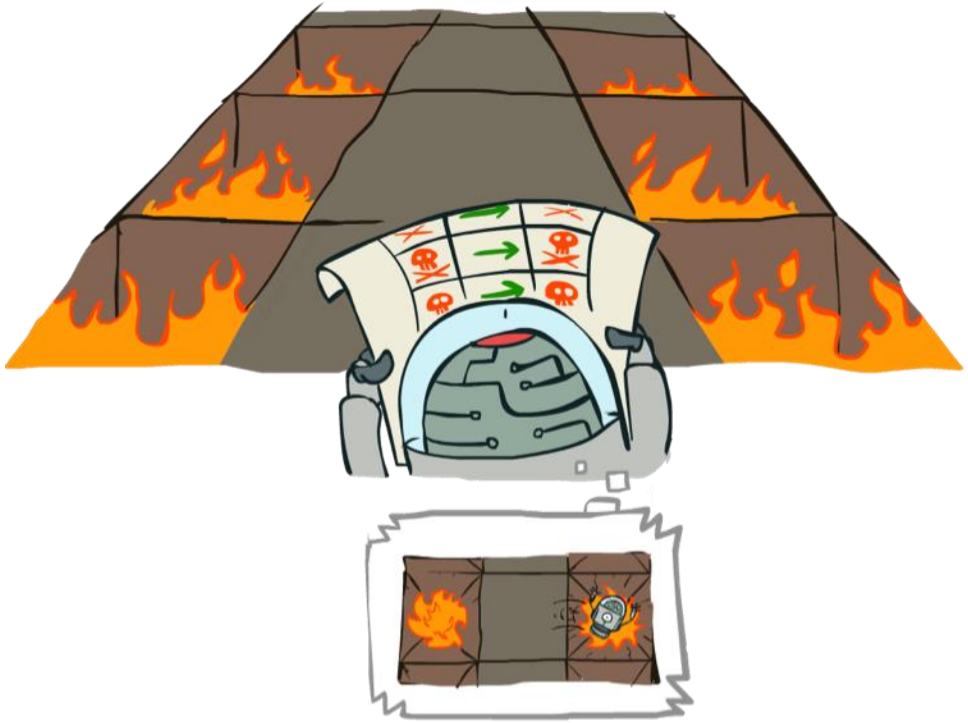
$$V^\pi(s) = \sum_{s'} T(s,\pi(s),s')[R(s,\pi(s),s') + \gamma V^\pi(s')]$$



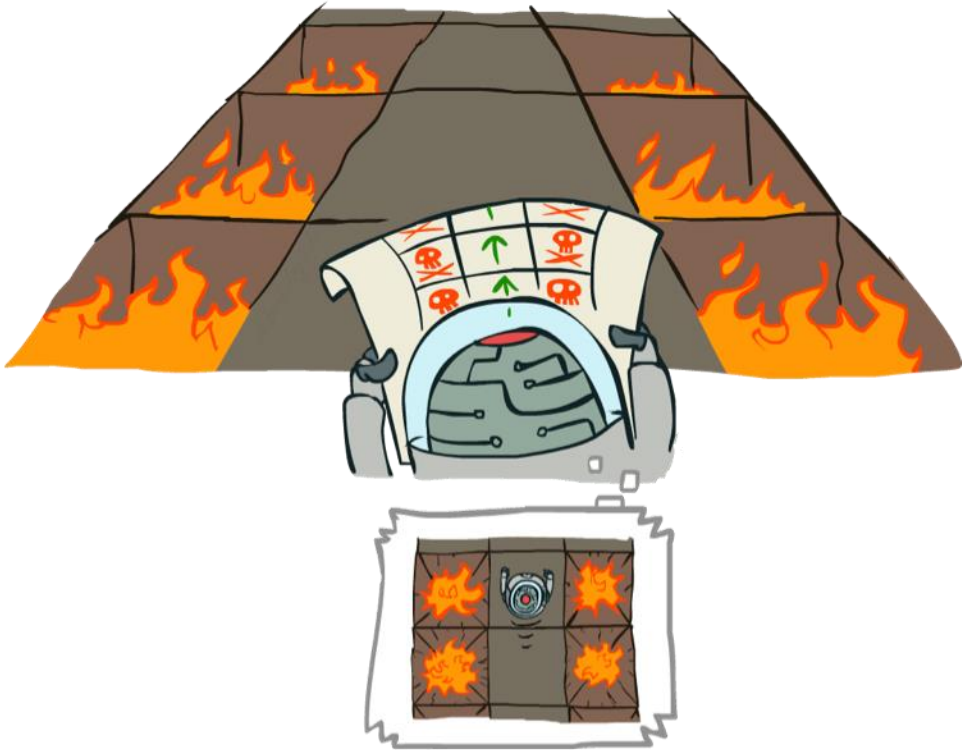
- 这实际上是个线性方程组
  - 可以不迭代，直接给出解析解
  - Value Iteration不行，因为里面有max

示例：Policy Evaluation

"永远往右走"



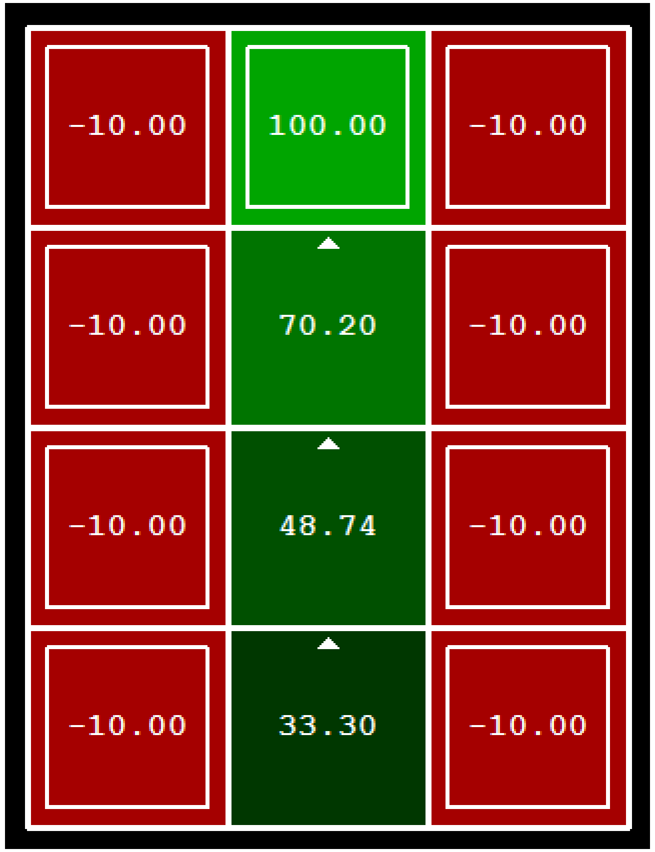
"永远往前走"



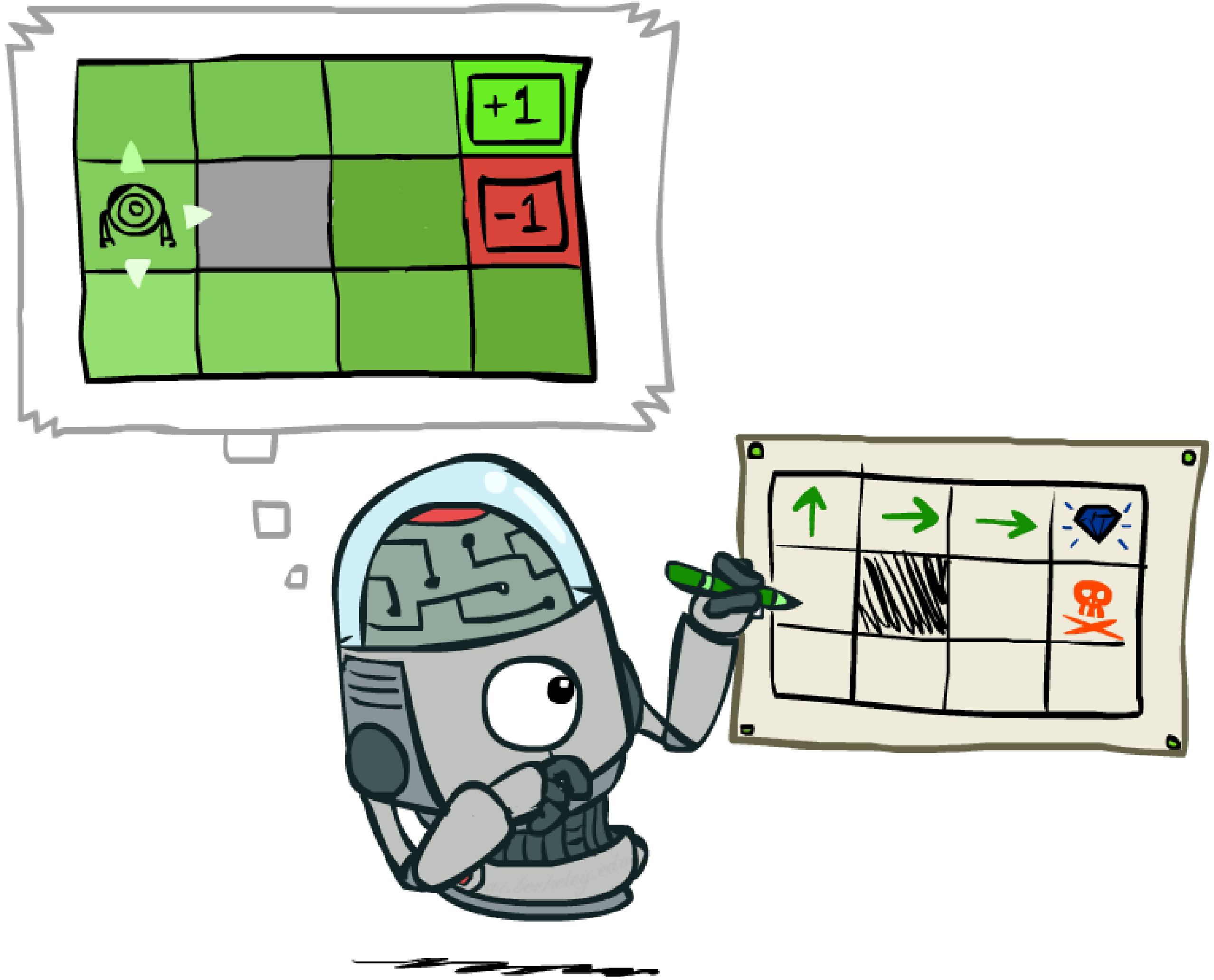
示例：Policy Evaluation

“永远往右走”

“永远往前走”



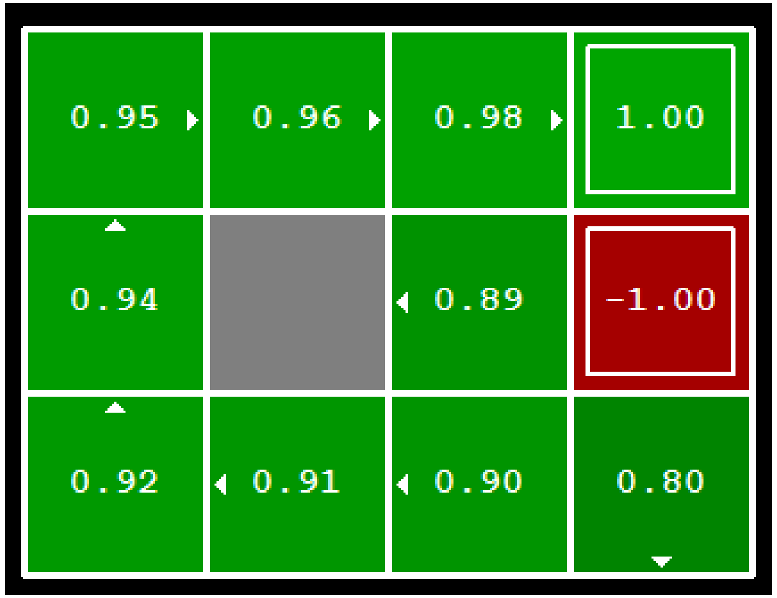
从值函数导出策略



从V值导出策略

- 如果已经有最优策略下的状态值V*(s)了，怎么走呢？
- 这事没有那么容易……
- 我们需要对每个action求个期望，取最大的走：

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

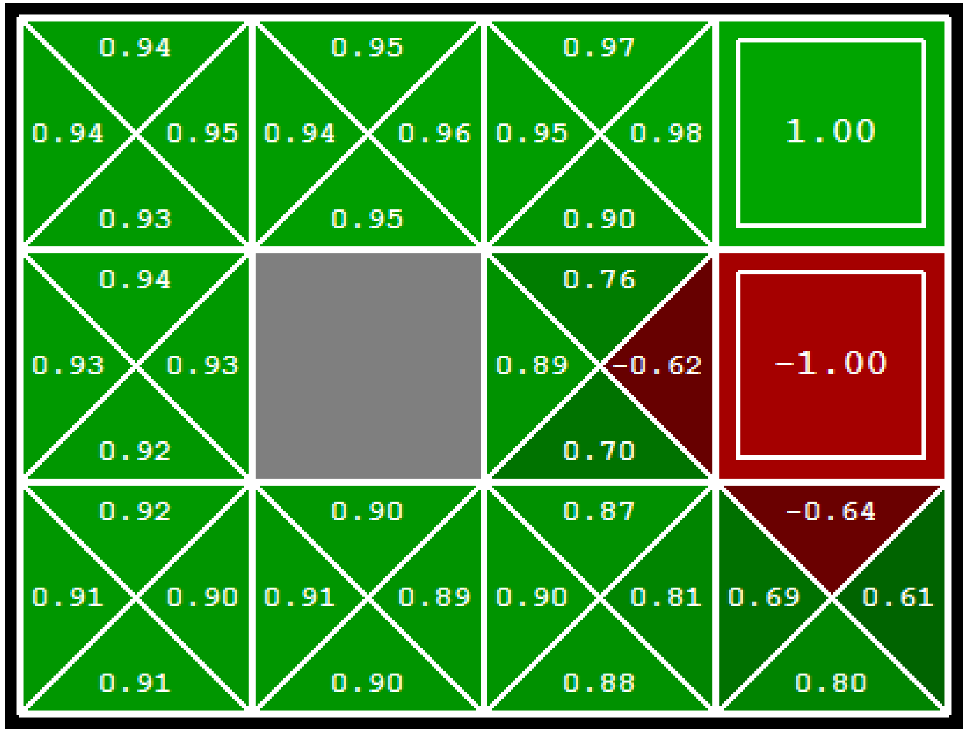


从Q值导出策略

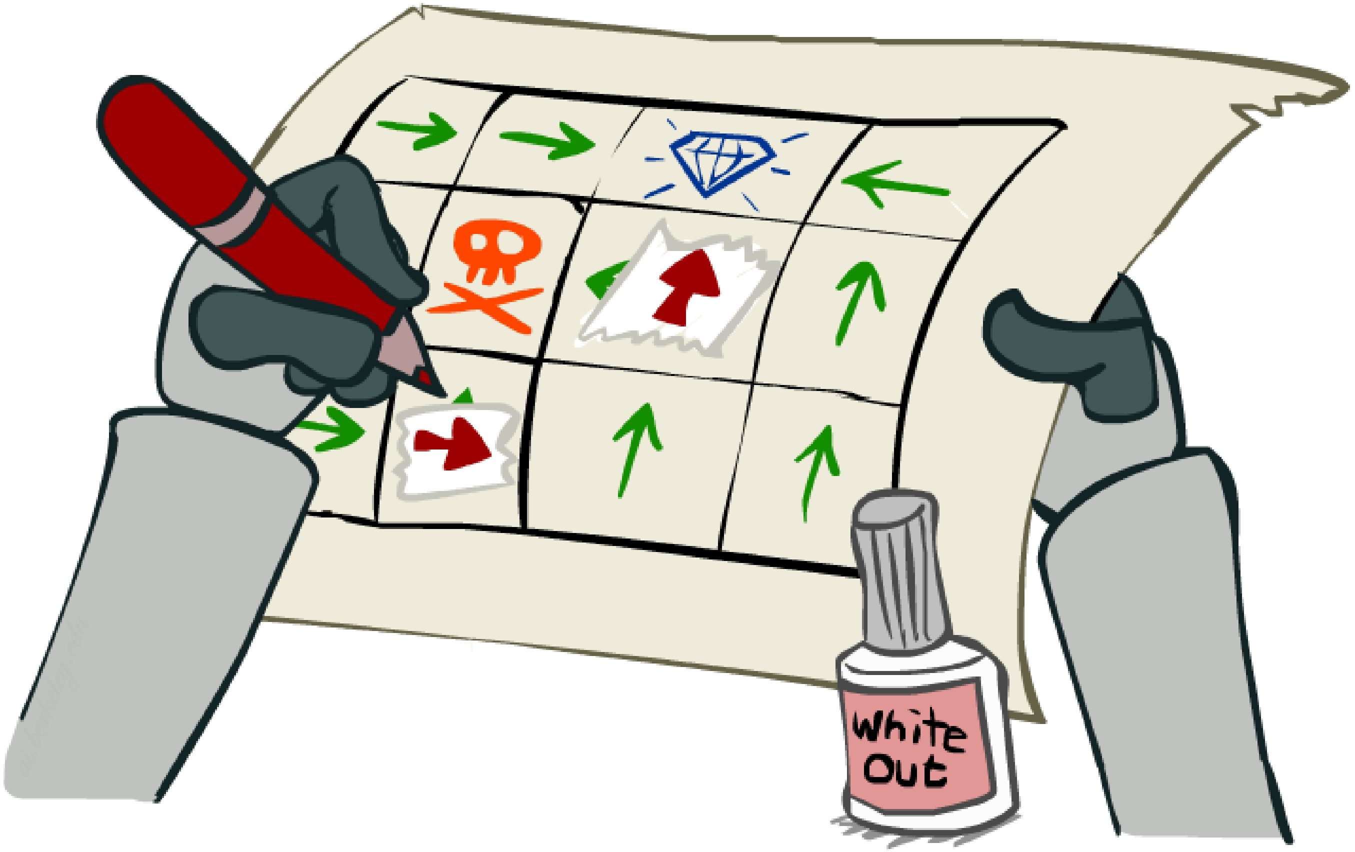
- 如果我们已经有最优策略下的Q函数了，怎么行动？
- 太简单了！

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

- 这说明什么？
  - **用Q值选动作要简单得多！**



Policy Iteration



Value Iteration的缺点

- Value Iteration不断地使用Bellman Equation更新：

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

- 问题1：太慢，一步的复杂度是$O(S^2A)$
  - 每一步s个状态，a种动作，下一个状态还有可能是s种

- 问题2：每步取max，其实取的动作到后期很少改变
  - 策略经常比值函数收敛得快得多

K=5


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


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


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=10


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=100

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

Policy Iteration

- 第一步：Policy Evaluation
  - 给定当前的策略π，计算V值/Q值直到收敛
- 第二步：Policy Improvement
  - 使用这些V/Q值，来决定新的策略

- 这就是Policy Iteration
  - 仍然能收敛到最优值！
  - 在很多情况下收敛要比单纯的Value Iteration快得多

比较

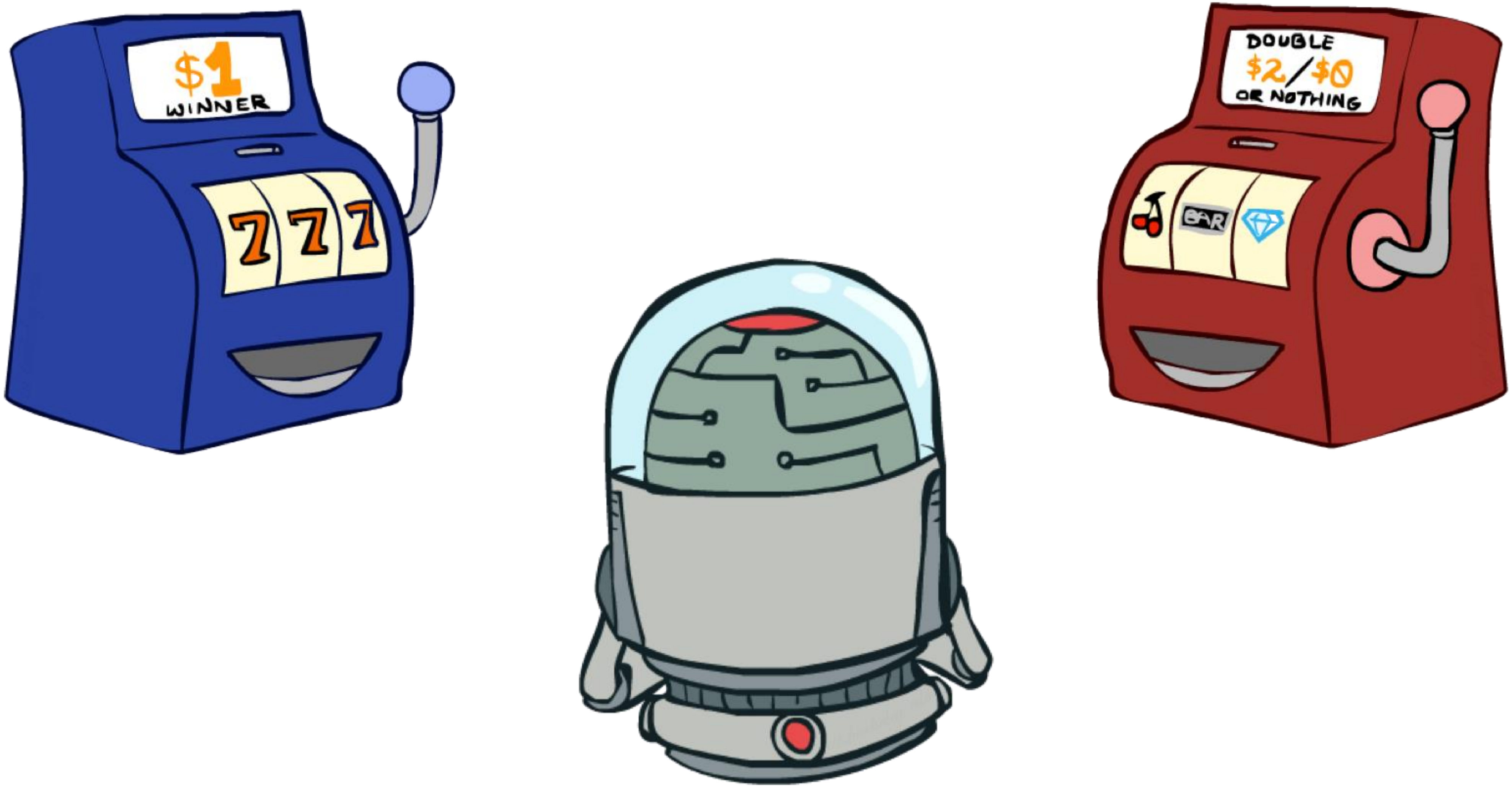
- Value Iteration和Policy Iteration想干的是同一件事
  - 计算所有的最优函数值（V/Q/π）
- Value Iteration：
  - 不记录π，根据值函数取max来确定应该怎么走
- Policy Iteration：
  - 先迭代几遍，确定当前策略下的值函数（快，因为动作已经定死了）
  - 值函数确定以后，导出一个新策略（慢，跟Value Iteration一样）
  - 新的策略应该比旧的策略好，否则就结束

- 这些都是解决MDP的动态规划（DP）算法

04 第四章

主题：马尔可夫决策过程与强化学习

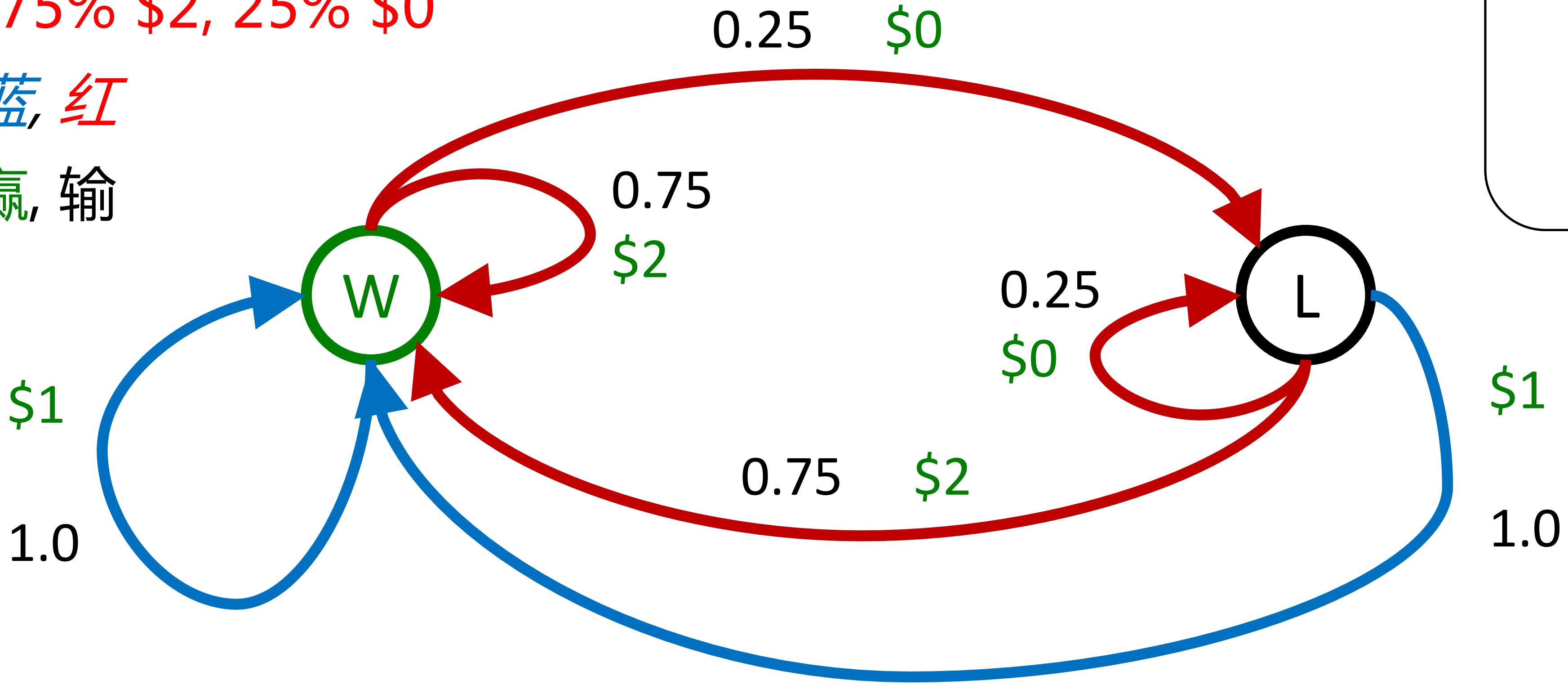
Double Bandits



Double Bandits MDP

- 蓝色：100% $1
- 红色：75% $2, 25% $0
- 动作: *蓝*, *红*
- 状态: 赢, 输

无discount
玩100次



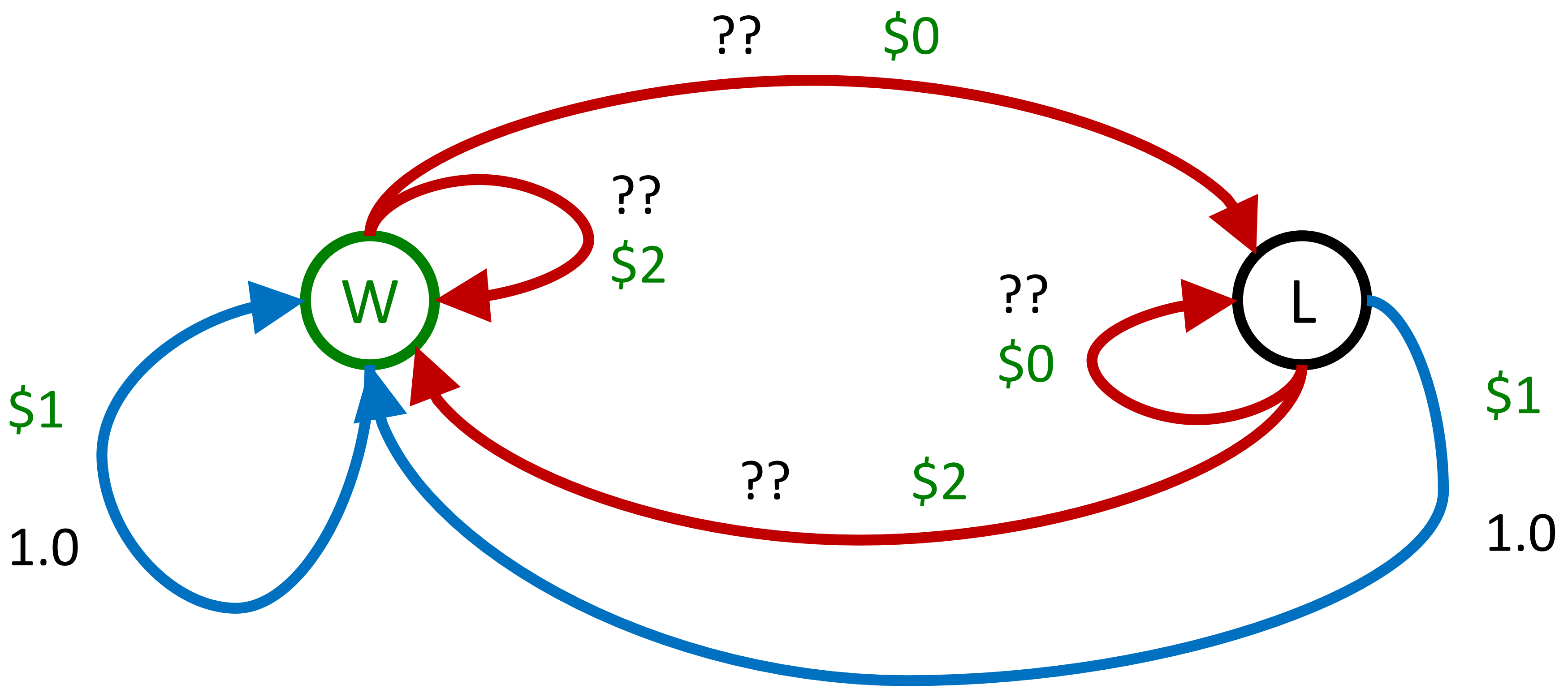
Offline Planning

- 玩红色：期望收益150
- 玩蓝色：期望收益100
- 最优策略：一直玩红色

- 解决MDP，实际上只是在离线求解
  - 你完全知道MDP的所有参数
  - 你没有玩过这个游戏就知道应该怎么玩了
  - 根本没有“学习”

Online Planning

- 现在不告诉你红色机器的出钱概率了！



那怎么办？

- 我们需要实际试一试才能知道应该怎么玩
  - 这其实就是强化学习（Reinforcement Learning）
  - 有一个MDP，但是转移概率未知，所以不能直接求解
  - 需要实际试一试才能求解

- 已经触及到了一些强化学习的基本思想
  - 探索(Exploration)：你需要尝试一些新的动作来收集信息
  - 利用(Exploitation)：你需要利用已有的知识
  - 采样(Sampling)：由于随机性，你需要多试几次才能确定一个动作好不好
  - 比直接求解一个已知的MDP要难得多

小结

1、价值函数的定义

2、动态规划方法——策略迭代和价值迭代

3、已知MDP参数的求解方法与强化学习的关系和区别

Q&A

下次课：Q Learning

THANK YOU

扫钉钉群二维码，加入我们