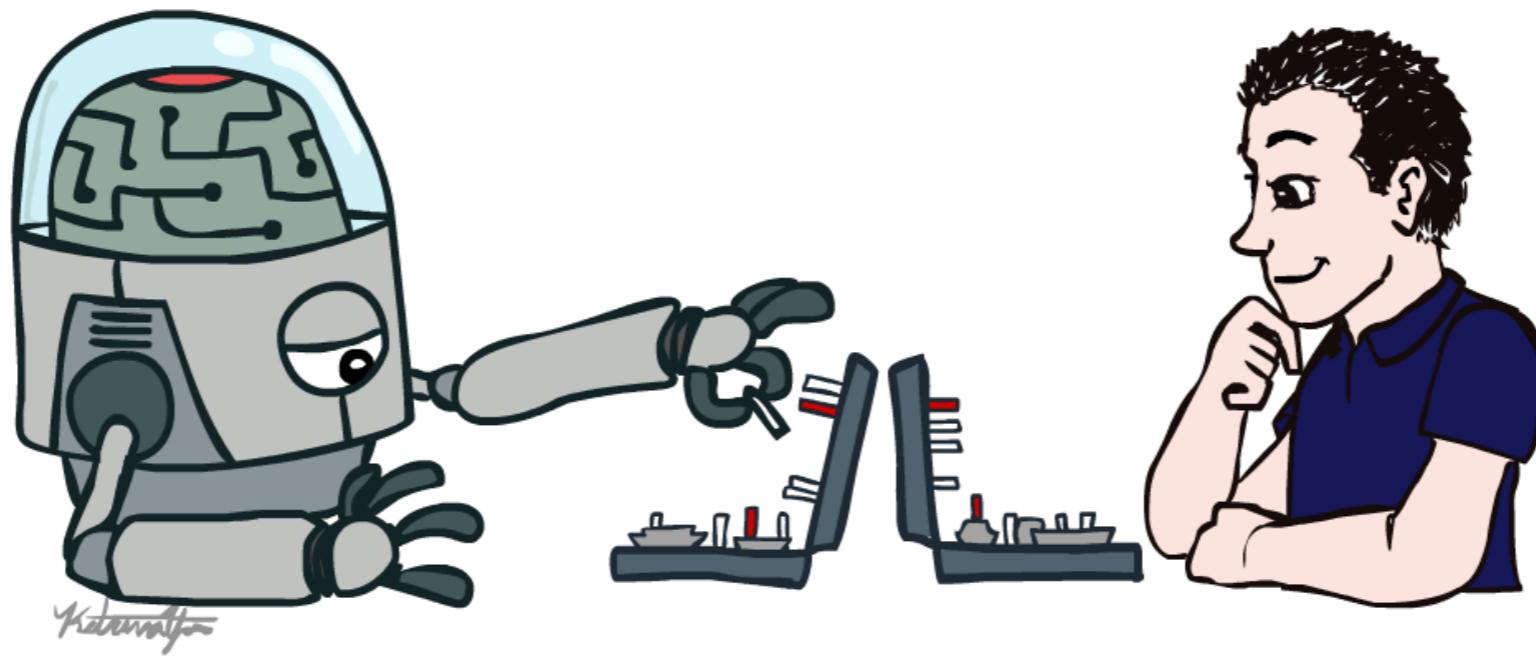


人工智能

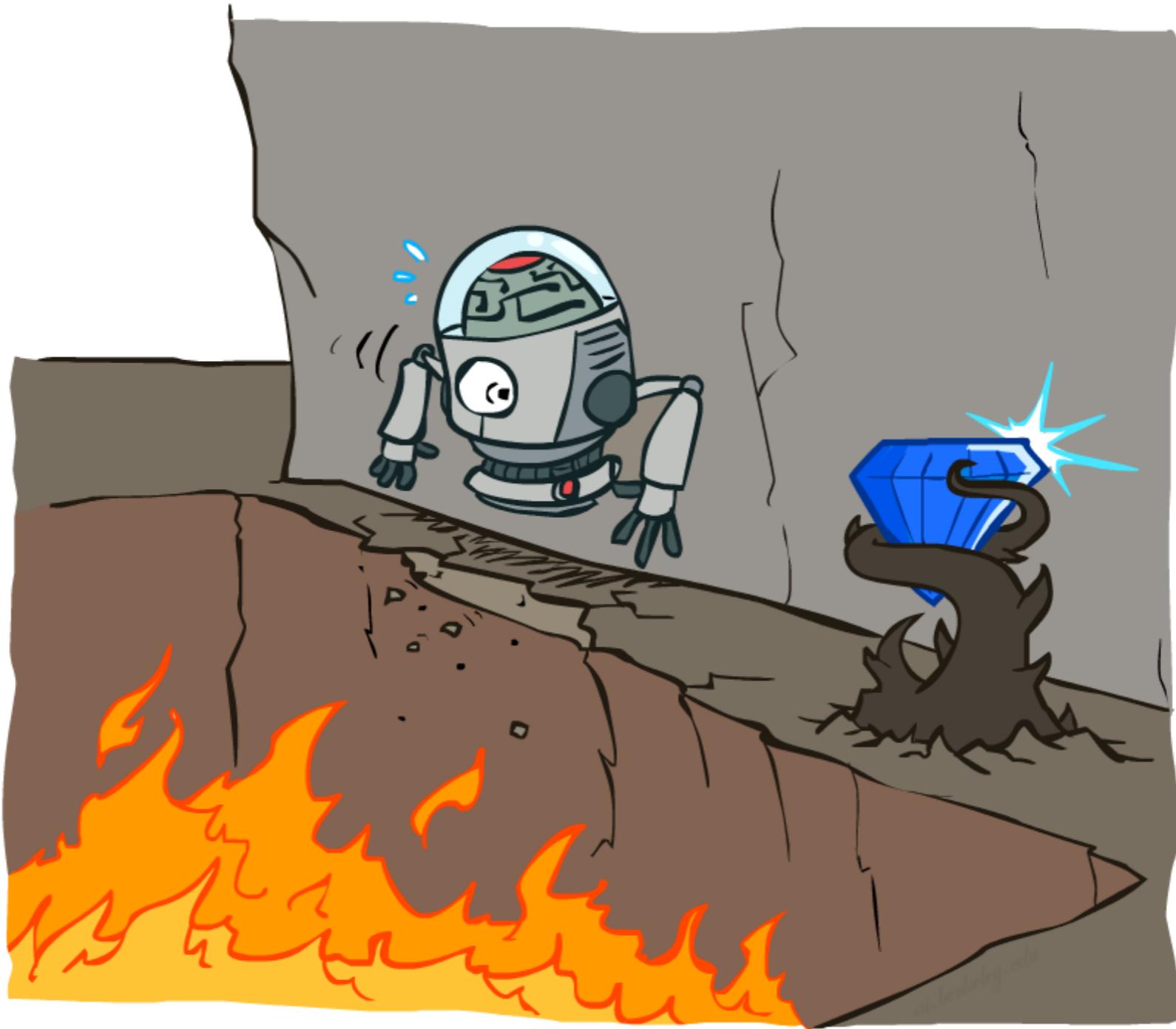


第六章·马尔科夫决策过程

- 非确定性搜索
- 求解马尔科夫决策过程
- 状态赋值迭代
- 策略迭代

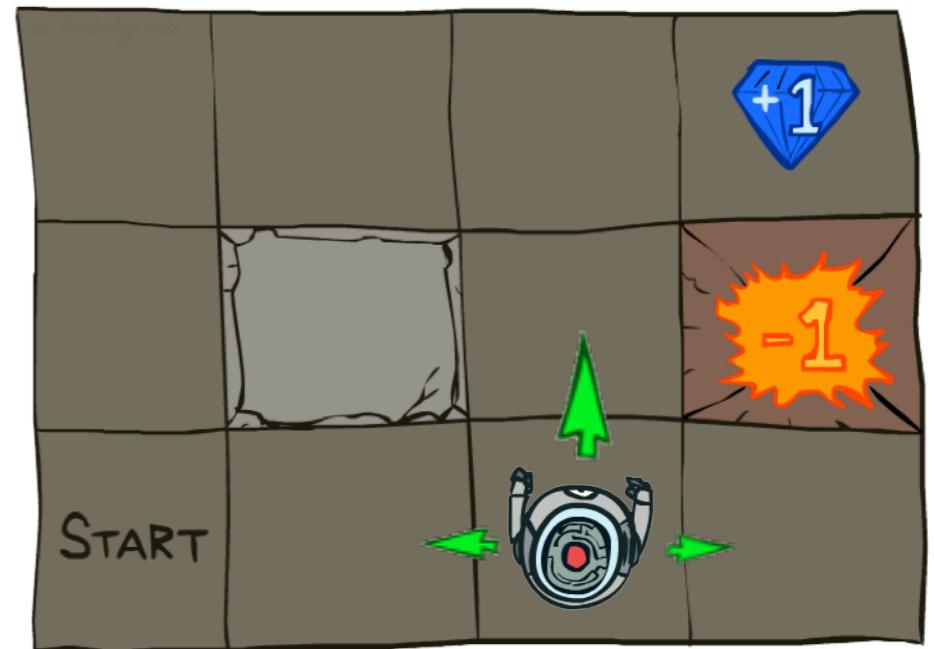


非确定性搜索 Non-Deterministic Search



举例: Grid World

- 一个像迷宫的问题
- 粗糙的行动: 行动不总是获得预先计划的结果
 - Agent采取North的行动, 只会有80%的几率移动到North, 分别10%的几率移动到East和West
- 智能体会在每个时刻获得奖赏 (得分)
 - 每时刻步长, 少量的“生存 living reward” 奖赏 (可能是负值)
 - 最大的奖赏分值在最后时刻 (执行退出行动时) (可能是奖赏也可能是惩罚)
- 目标: 最大化总的奖赏得分

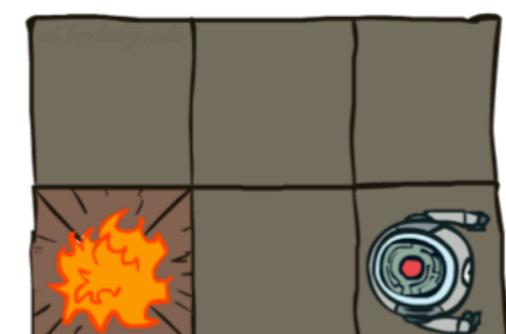
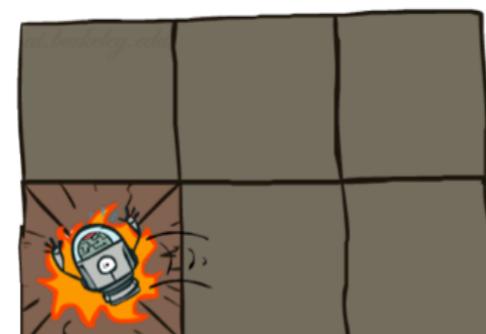
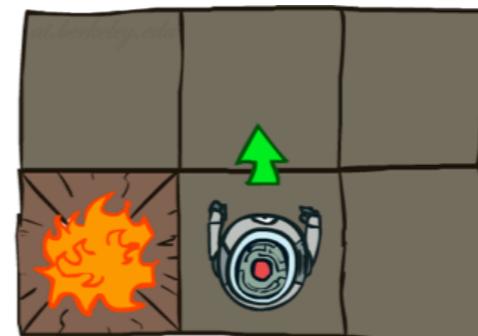


Grid World 行动

确定的 Grid World

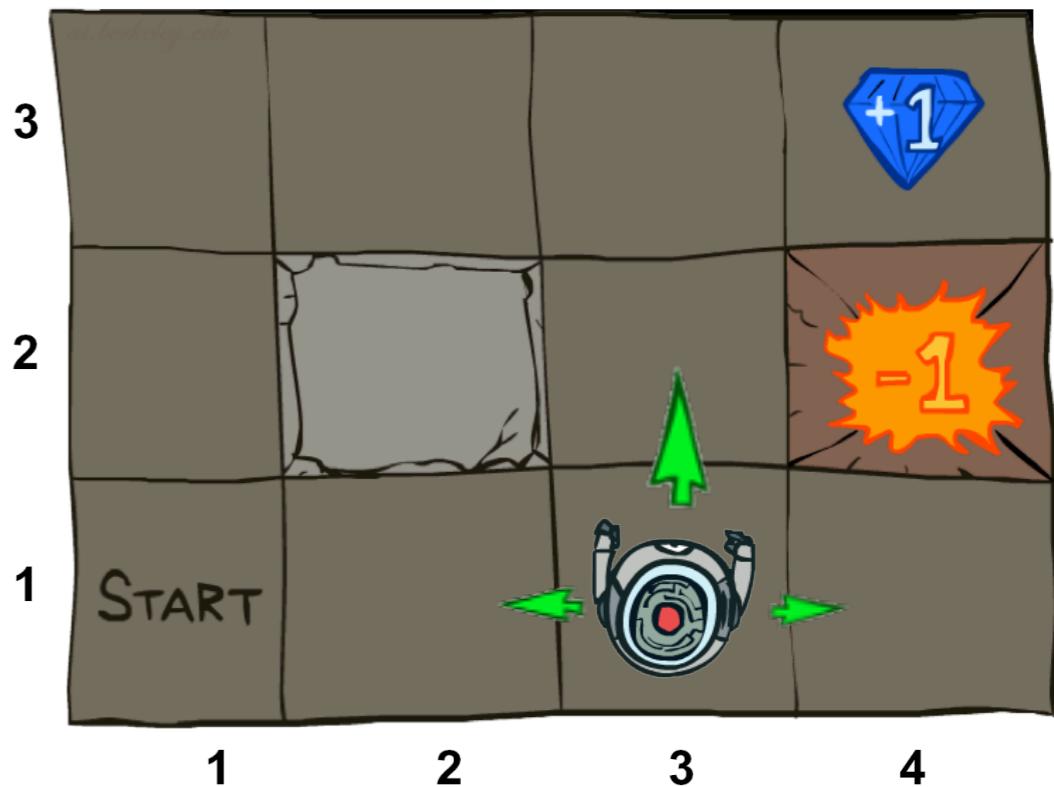


随机的(不确定的) Grid World



马可夫决策过程(MDP)

- 一个MDP由以下定义：
 - 状态集合 $s \in S$
 - 行动集合 $a \in A$
 - 转移函数 $T(s, a, s')$
 - 从状态 s 采取行动 a 到达 s' 的概率, 即 $P(s'|s, a)$, 也叫做动态模型
 - 奖赏函数 $R(s, a, s')$, 有时只是 $R(s)$ or $R(s')$
 - 一个开始状态, 一个终点状态 (有可能)
- MDPs 是非确定性搜索问题
 - 一种求解方法是通过期望最大值搜索



MDPs中的马可夫性质

- “Markov” 通常意味着给定当前状态, 未来状态独立于过去状态
- 对于马可夫决策过程, “Markov” 的含义是行动的输出结果只取决于当前状态

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

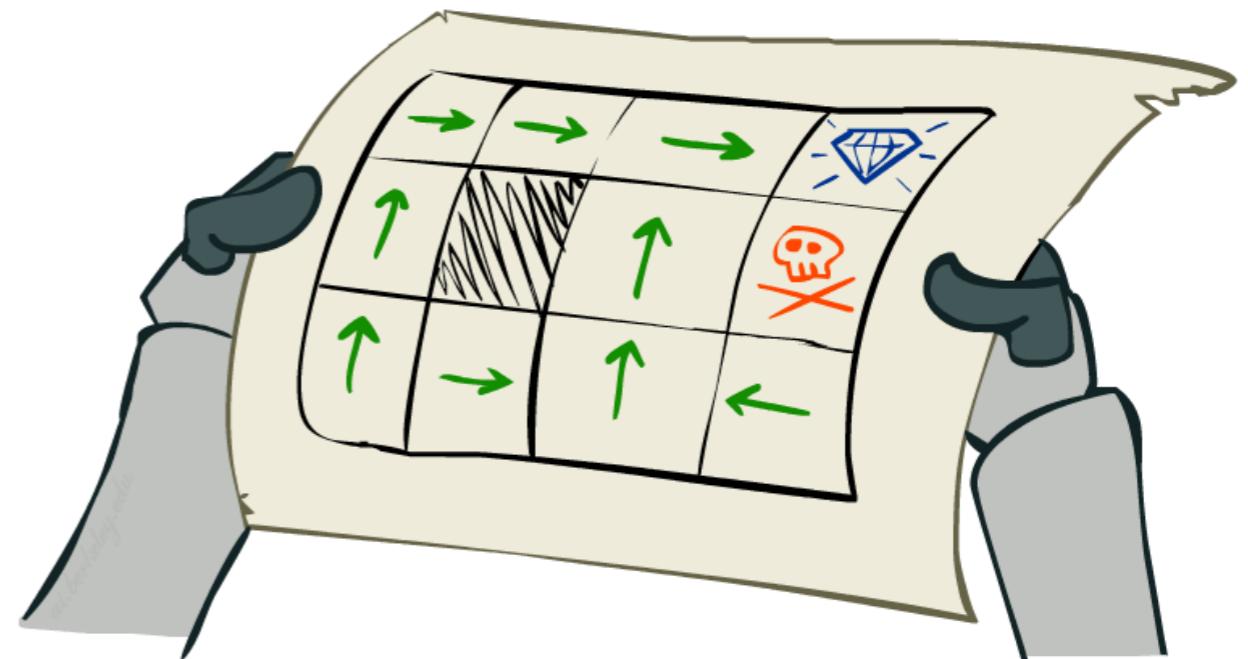


Andrey Markov
(1856-1922)

- 就像在搜索问题里, 后继函数只依赖于当前的搜索状态 (而不是历史状态)

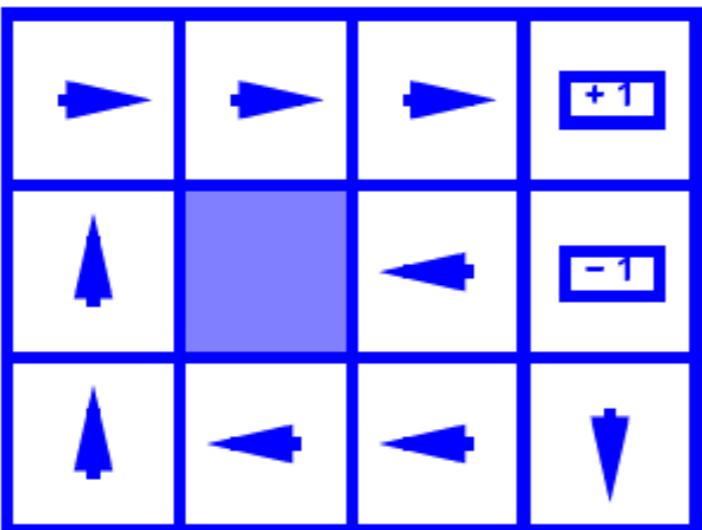
策略 Policies

- 在确定性的，单一智能体的，搜索问题里，我们想获得一个最优的规划 plan, 或是一个从开始到目标状态的行动上的序列
- 对于 MDPs, 我们想获得的是一个最优的策略 policy $\pi^*: S \rightarrow A$
 - 一个策略 π 对于每个状态给出了一个行动
 - 一个最优策略是这样一个策略，即遵循它可以获得更多期望功效值
 - 一个明确的策略定义了一个反射型的智能体

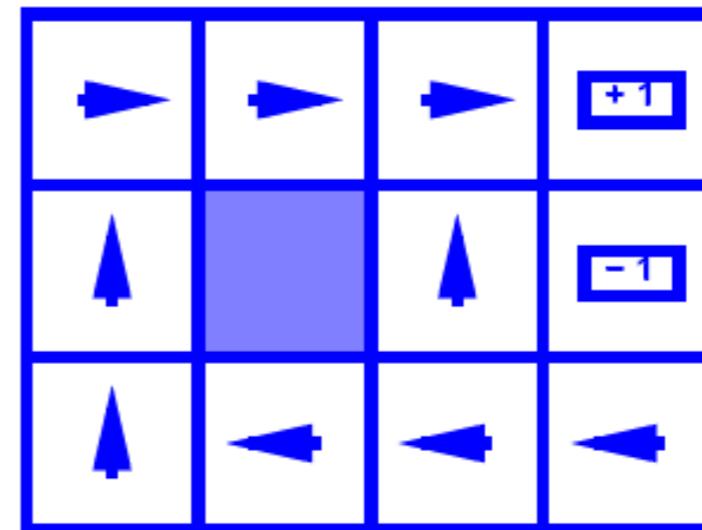


Optimal policy when $R(s, a, s') = -0.03$
for all non-terminals s

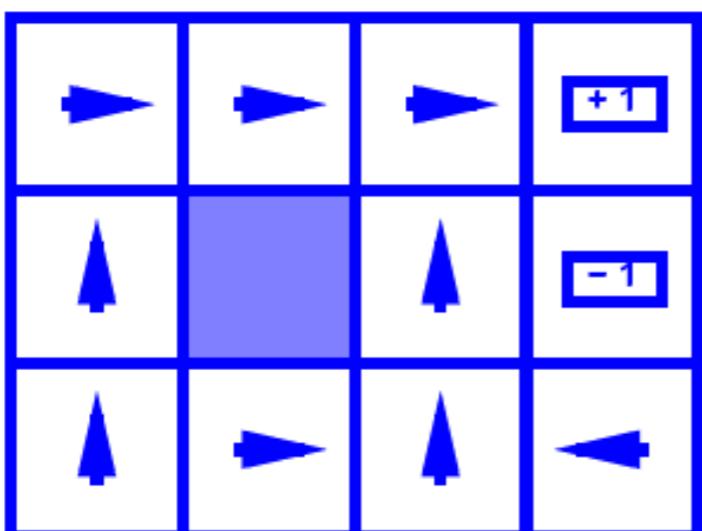
最优策略 Optimal Policies



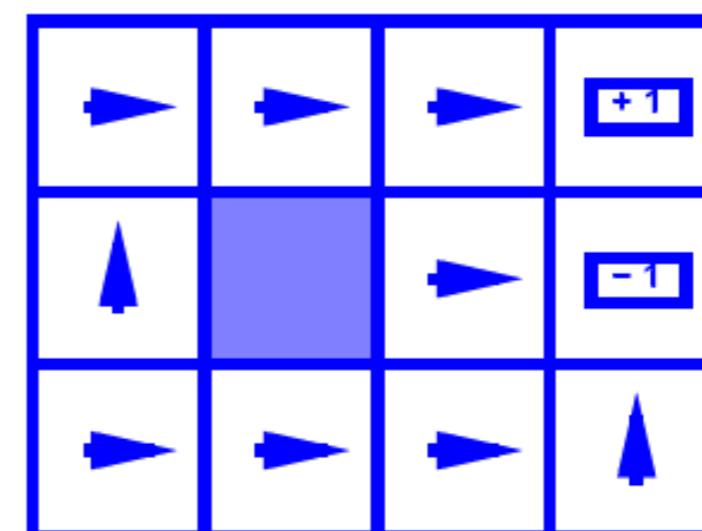
$$R(s) = -0.01$$



$$R(s) = -0.03$$



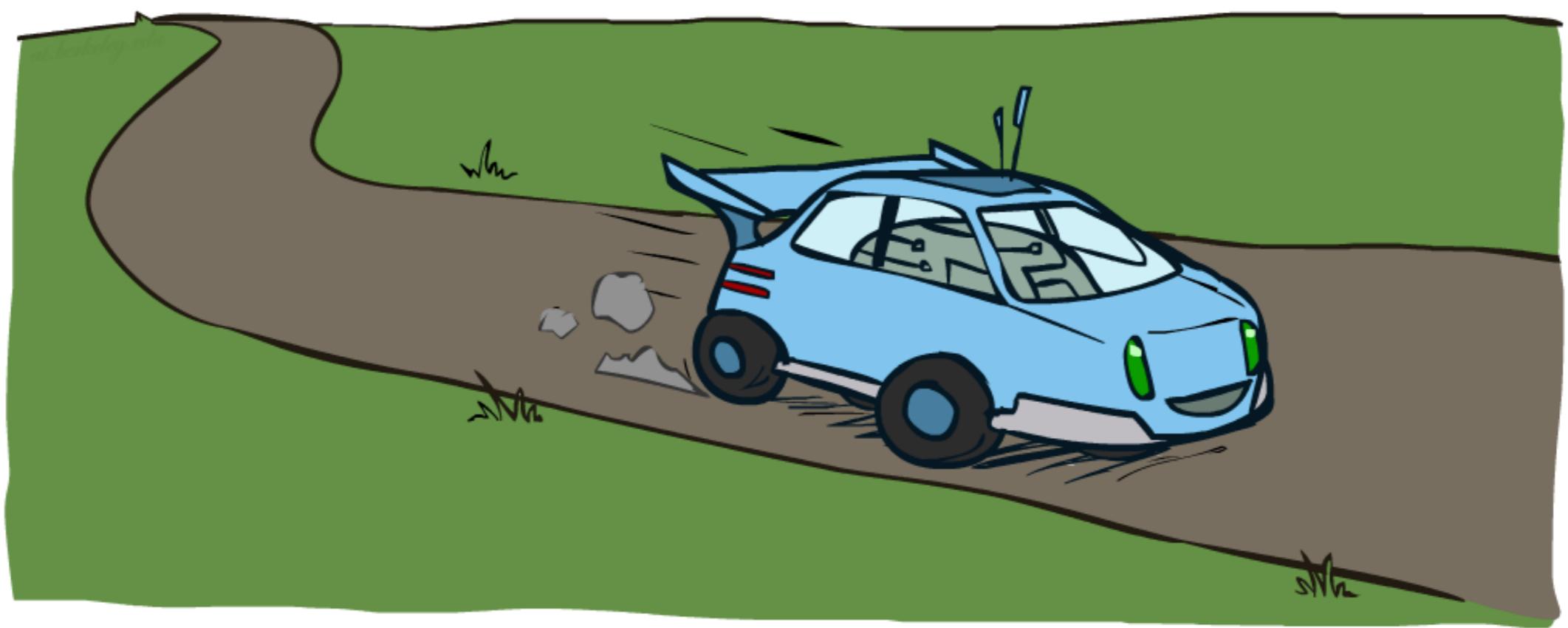
$$R(s) = -0.4$$



$$R(s) = -2.0$$

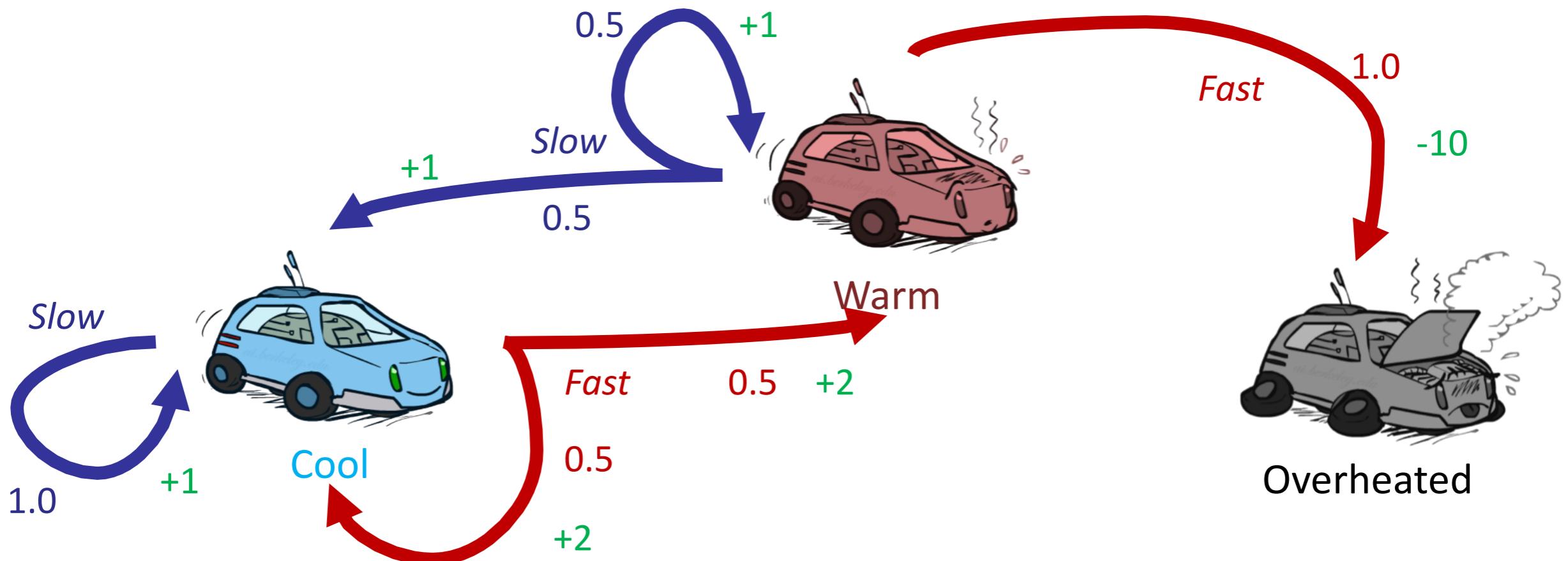
$R(s)$ = the “living reward”

举例：赛车

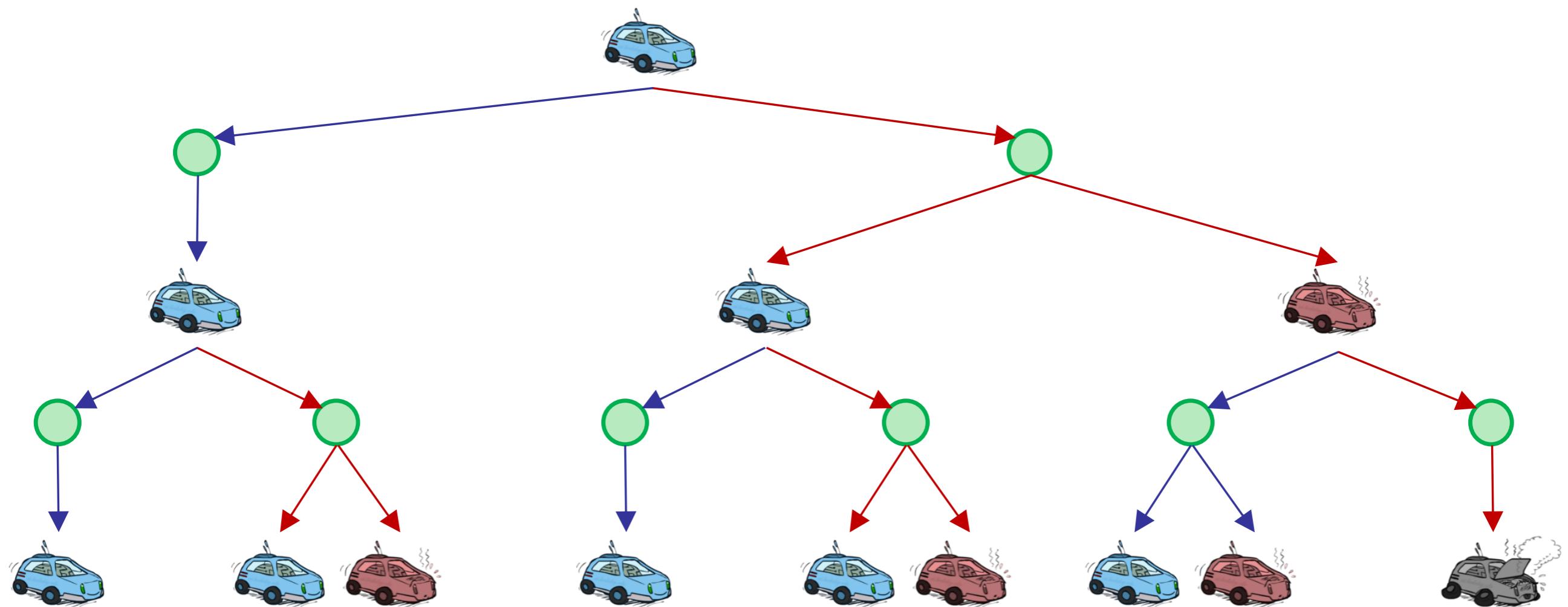


举例：赛车

- 一个机器人汽车想要开的既远又快
- 三个状态: Cool, Warm, Overheated
- 两个行动: Slow, Fast
- 速度快可以得到双倍的奖赏

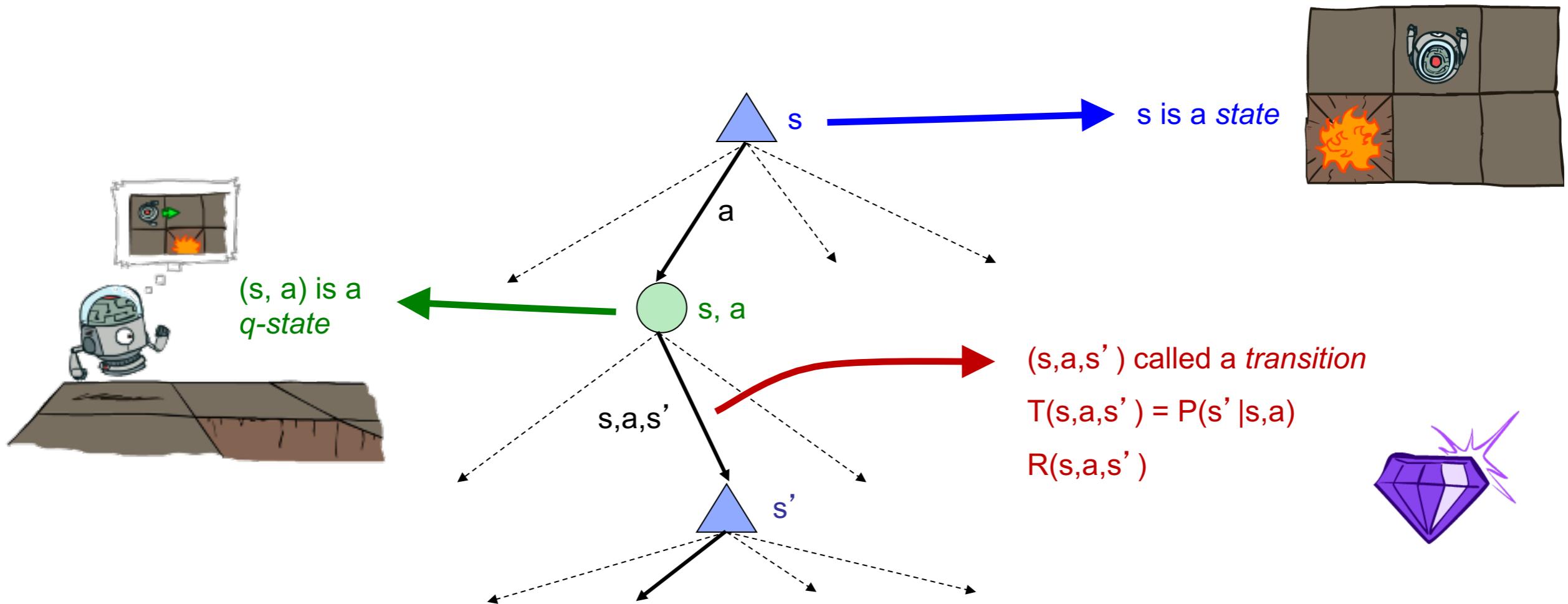


赛车例子里的搜索树

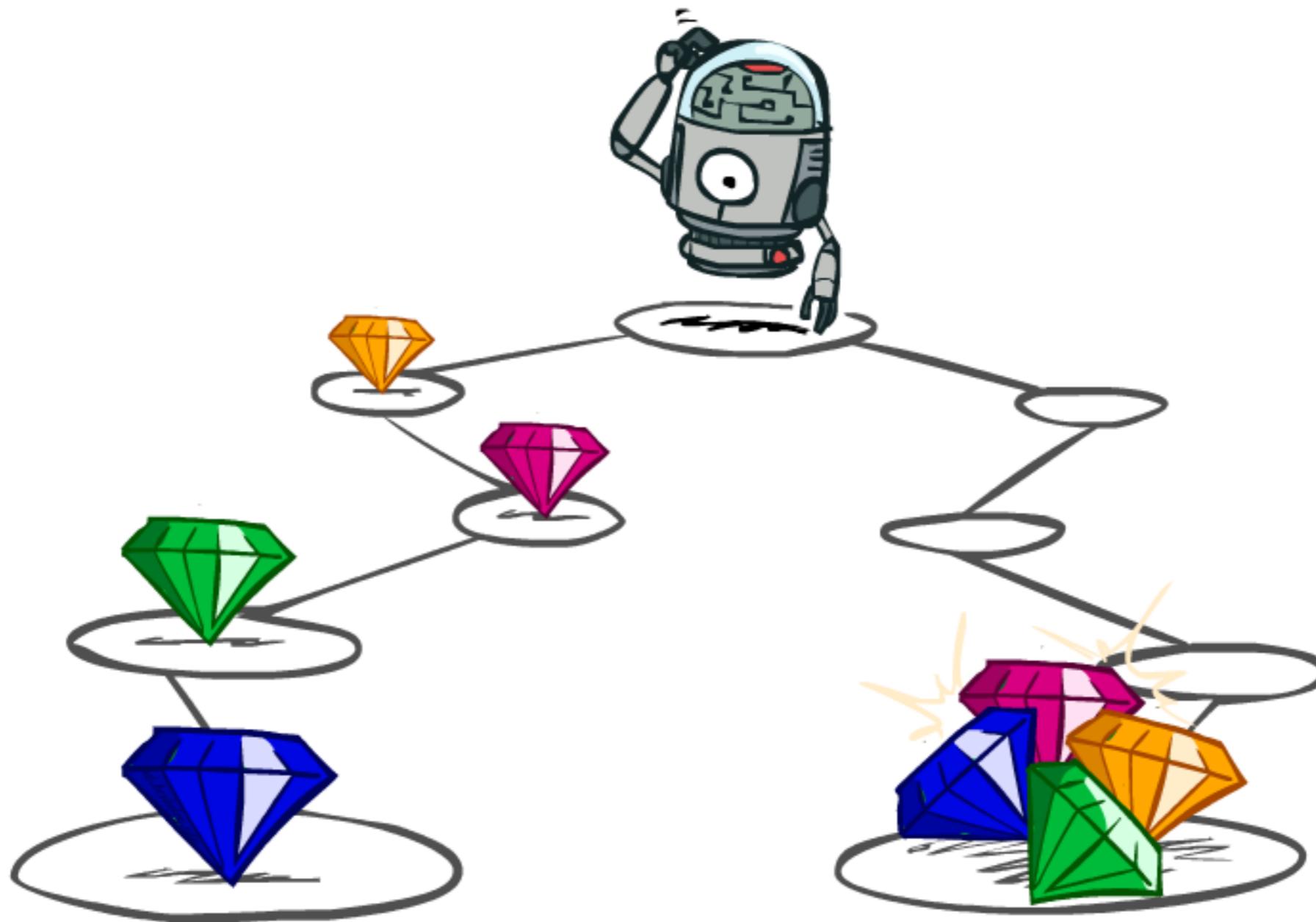


MDP 搜索树

- 一个类似-最大期望的 搜索树

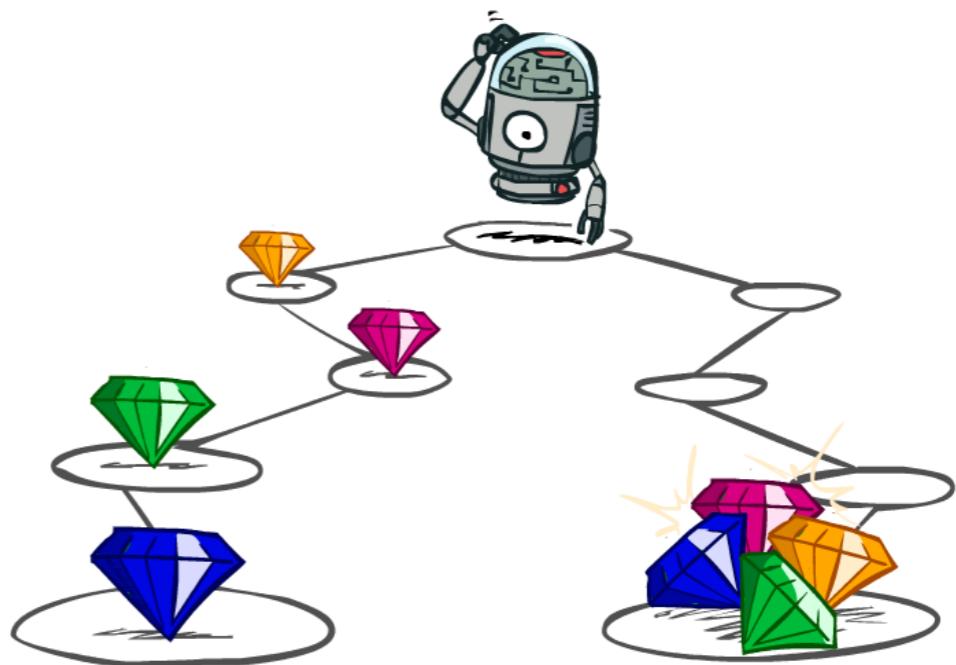


不同奖赏值序列的选择



如何选择：不同的奖赏值序列

- 不同选择的偏好，智能体应如何选择？
- 多或是少？ [1, 2, 2] or [2, 3, 4]
- 现在或是以后？ [0, 0, 1] or [1, 0, 0]



折扣 Discounting

- 效用可以等于奖赏值的总和
- 效用也可以趋向于越早获得奖赏越好
- 一种方法是：给奖赏打折，奖赏值随时间成指数递减



1

现在的价值



γ

下一步的

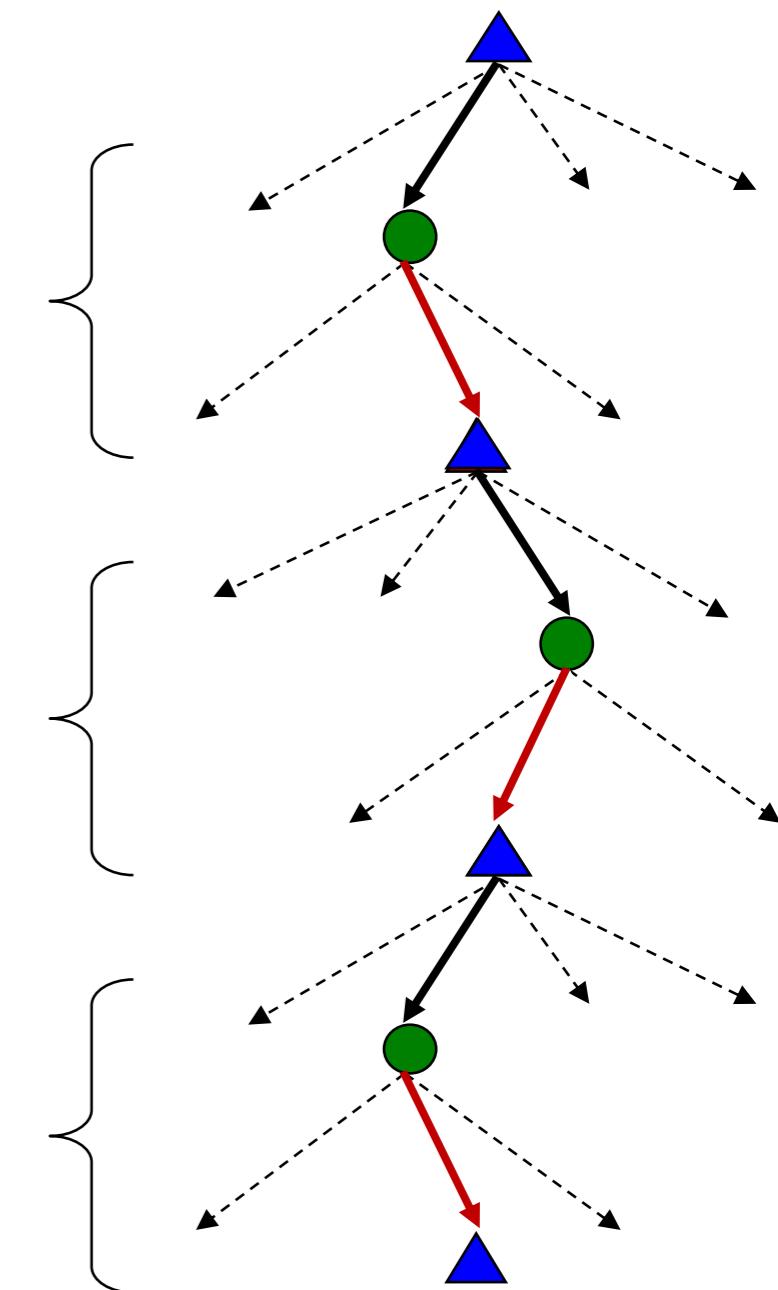


γ^2

两步以后的

折扣下的效用

- 如何打折?
 - 在搜索树上每下降一层，乘上一次折扣系数
- 为什么要进行折扣?
 - 为了实现倾向于更早地获得奖赏
 - 还能帮助算法收敛
- 例如: 折扣系数 0.5
 - $U([1,2,3]) = 1 * 1 + 0.5 * 2 + 0.25 * 3$
 - $U([1,2,3]) < U([3,2,1])$



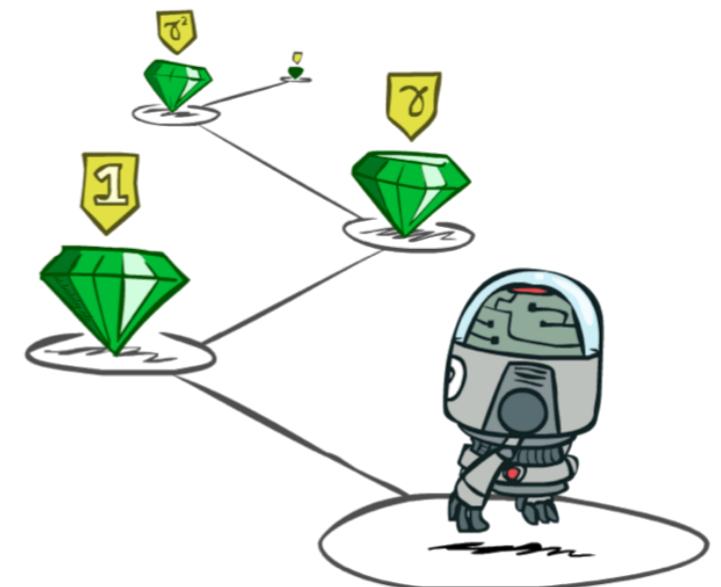
理性的偏好 Stationary Preferences

- 定理: 如果我们想要偏好保持理性:

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

↔

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



- 那么: 只有两种方式定义其上的功效值 (utilities)

- 相加功效值: $U([r_0, r_1, r_2, \dots]) = r_0 + r_1 + r_2 + \dots$

- 折扣的功效值: $U([r_0, r_1, r_2, \dots]) = r_0 + \gamma r_1 + \gamma^2 r_2 \dots$

示例: 折扣

- 给定:



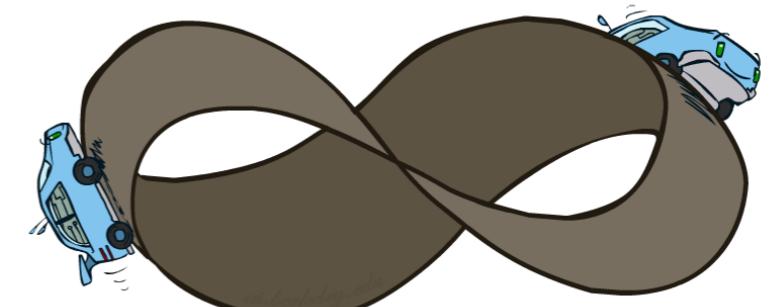
- 行动: 向东走, 向西走, 和 退出 (退出行动只在退出状态 a, e上有效)
- 状态转移: 确定性的
- Question 1: For $\gamma = 1$, 最优策略是什么?
- Question 2: For $\gamma = 0.1$, 最优策略?
- Question 3: γ 为何值时, 在状态d上, 向西和向东的行动是一样的?

10				1
----	--	--	--	---

10				1
----	--	--	--	---

无穷大功效值?!

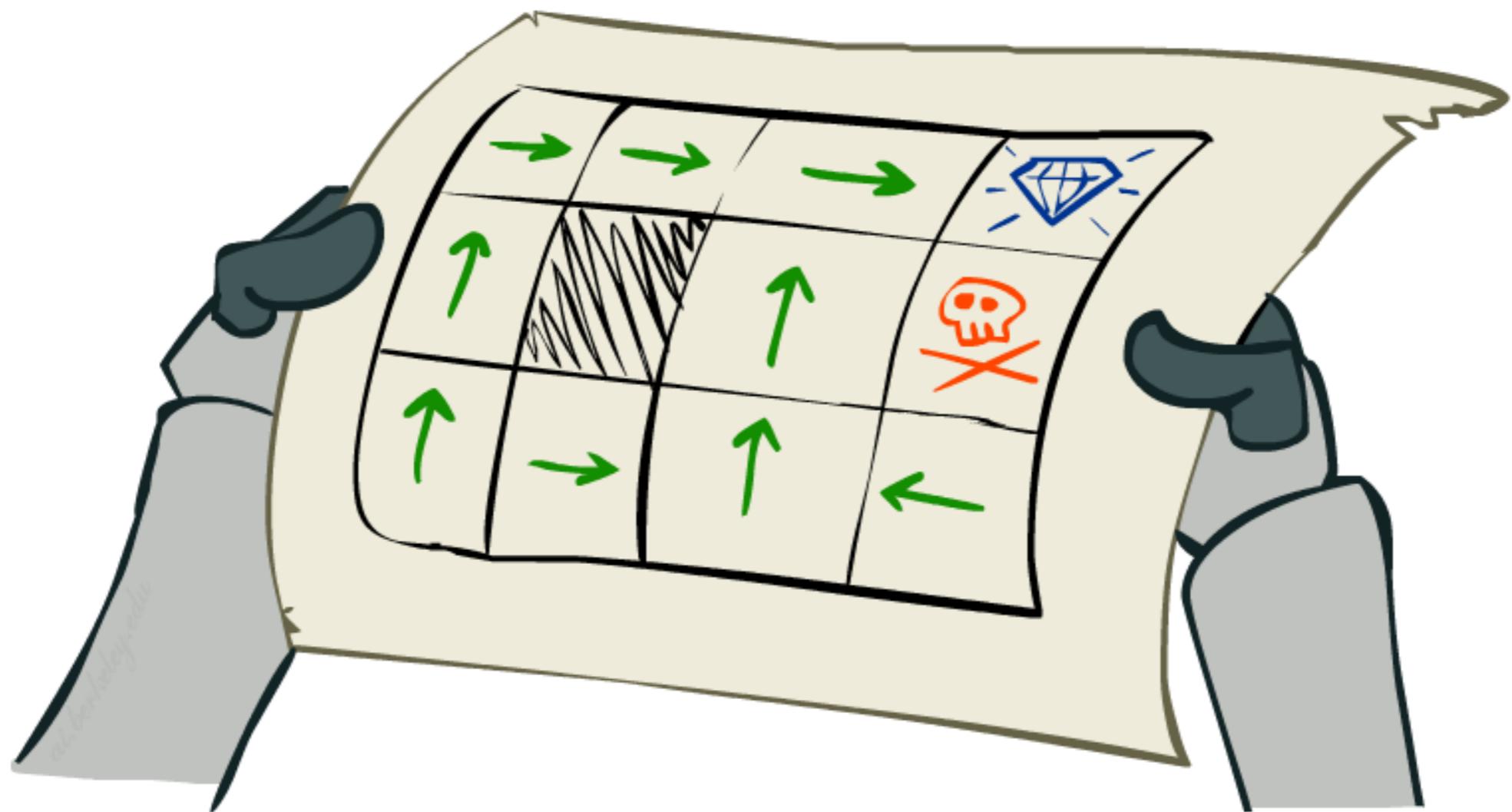
- 问题: 如果游戏一直进行下去会怎么样? 是否我们会得到无穷大的奖赏值?
- 解决方法:
 - 有限的游戏时间: (类似于有限深度的搜索)
 - 游戏终止在一个固定的 T 步长时间后
 - 使用非稳定的策略 (π 依赖于所剩游戏时间)
 - 折扣的奖赏: 使用 $0 < \gamma < 1$



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

- 吸收状态: 对于每一个策略都保证, 一个终点状态将会被最终达到 (像在之前赛车例子里的“overheated”状态)

求解马可夫决策过程



定义最优量值

- 一个状态 s 的(效用)值:

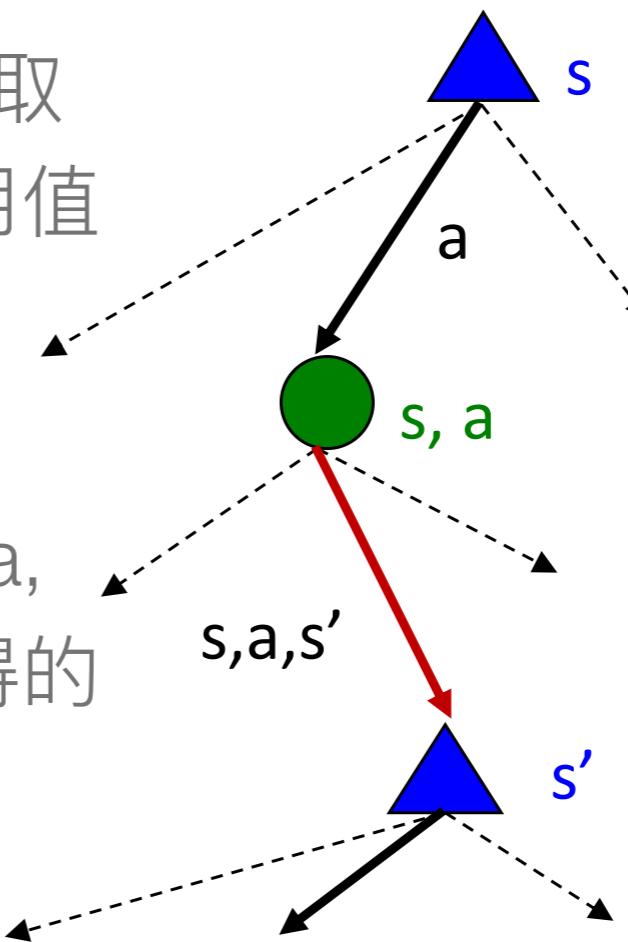
- $V^*(s) =$ 开始于状态 s 然后随后都采取最优化行动, 最后获得的期望效用值

- 一个 q-状态 (s,a) 的(效用)值:

- $Q^*(s,a) =$ 开始于状态 s 并采取行动 a , 随后都采取最优化行动, 最后获得的期望效用值

- 最优策略:

- $\pi^*(s) =$ 给定状态 s , 返回最优的行动

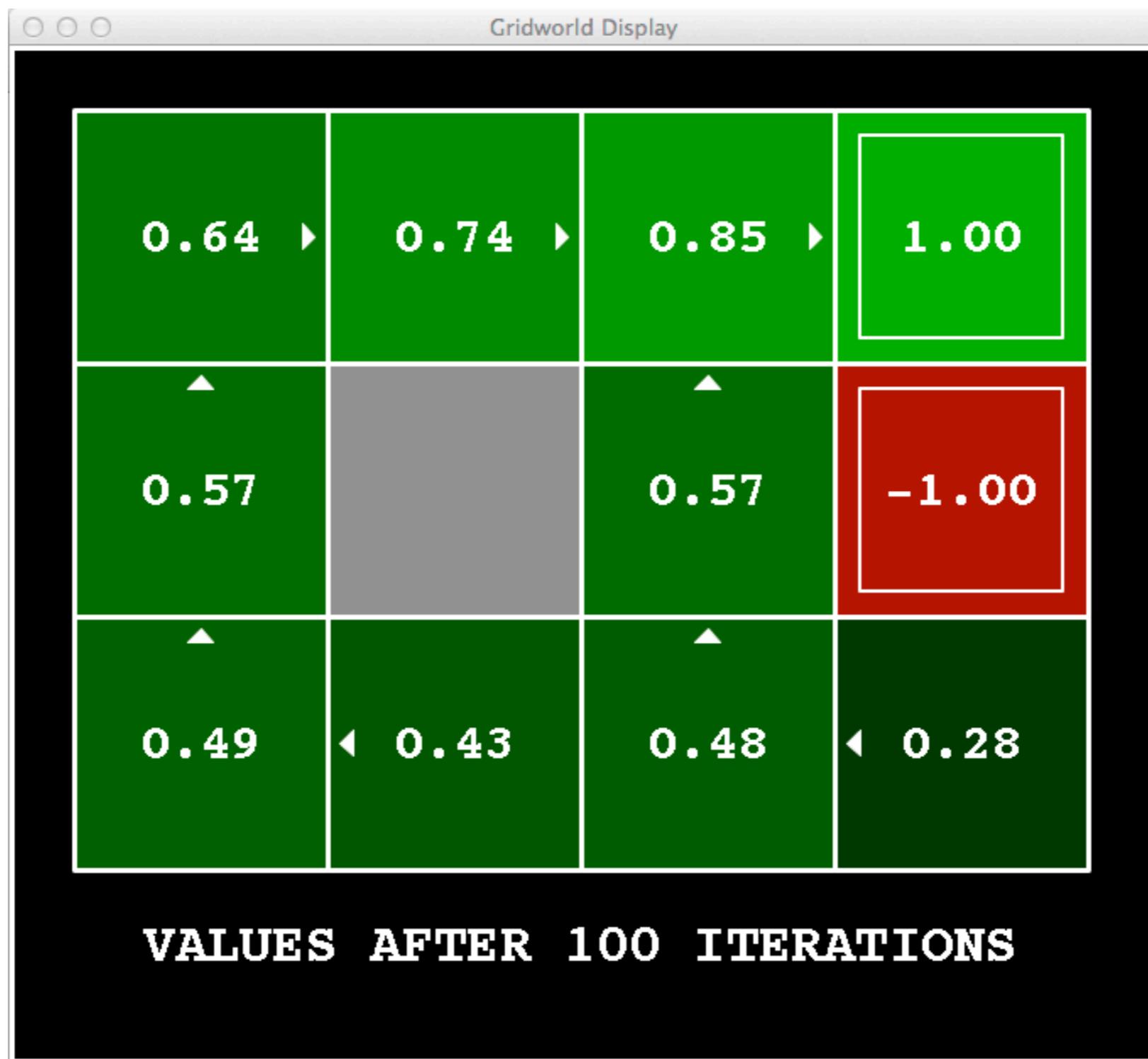


s is a state

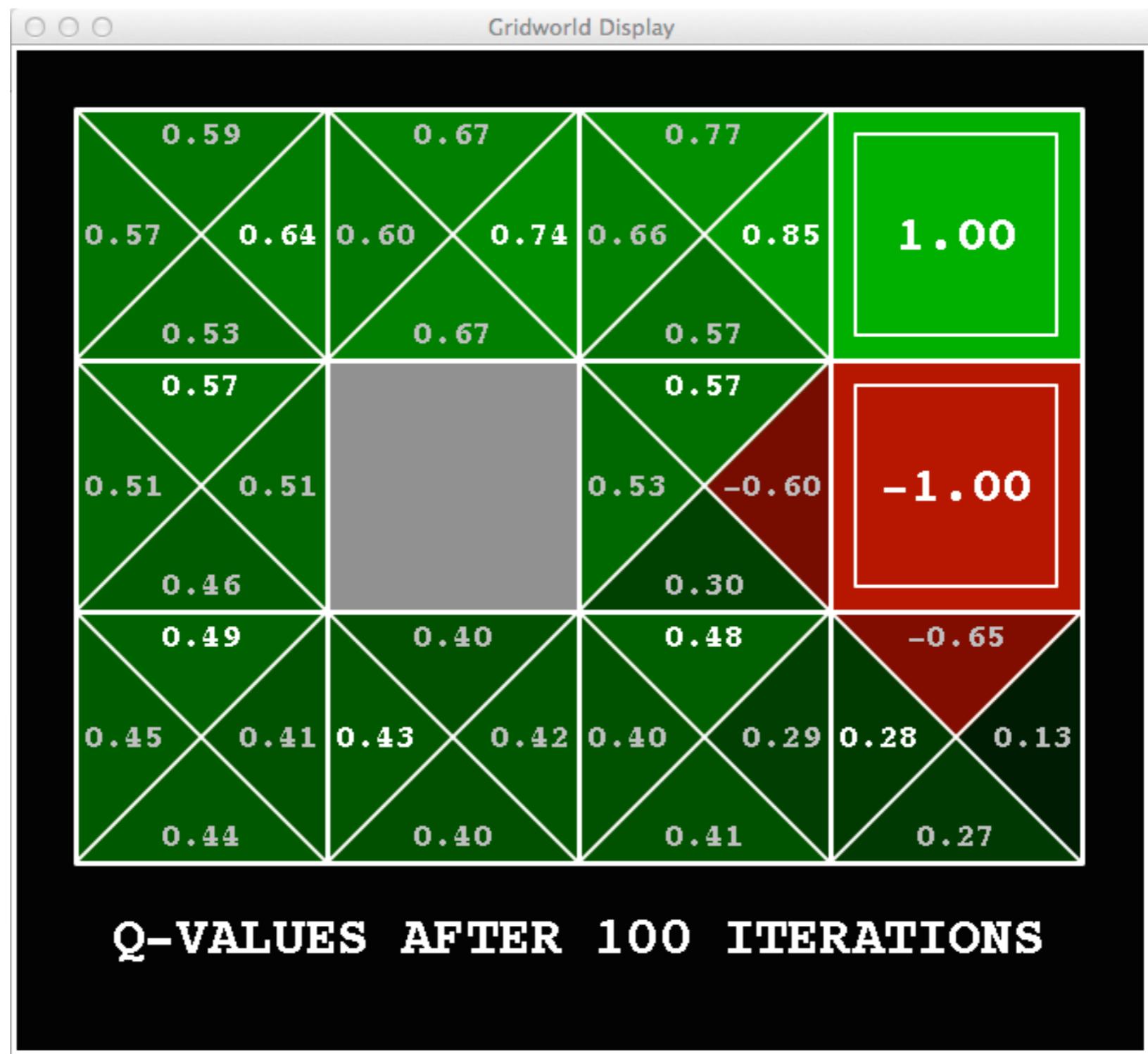
(s, a) is a q-state

(s, a, s') is a transition

Snapshot of Demo – Gridworld V Values



Snapshot of Demo – Gridworld Q Values



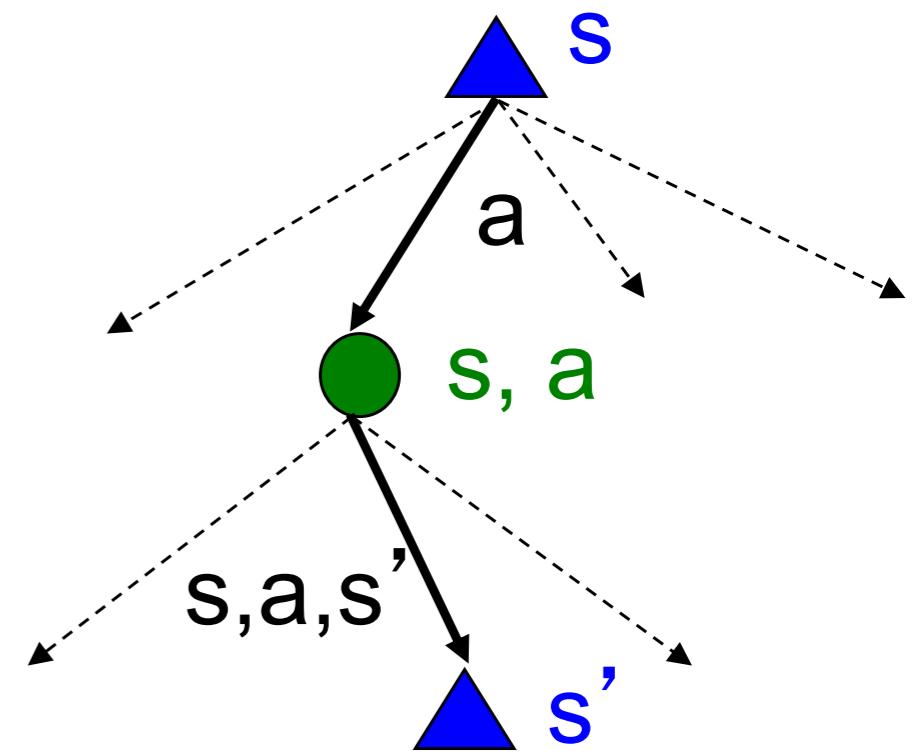
计算两种状态的值

- 基本操作: 计算一个状态的(期望最大)值
- 递归定义:

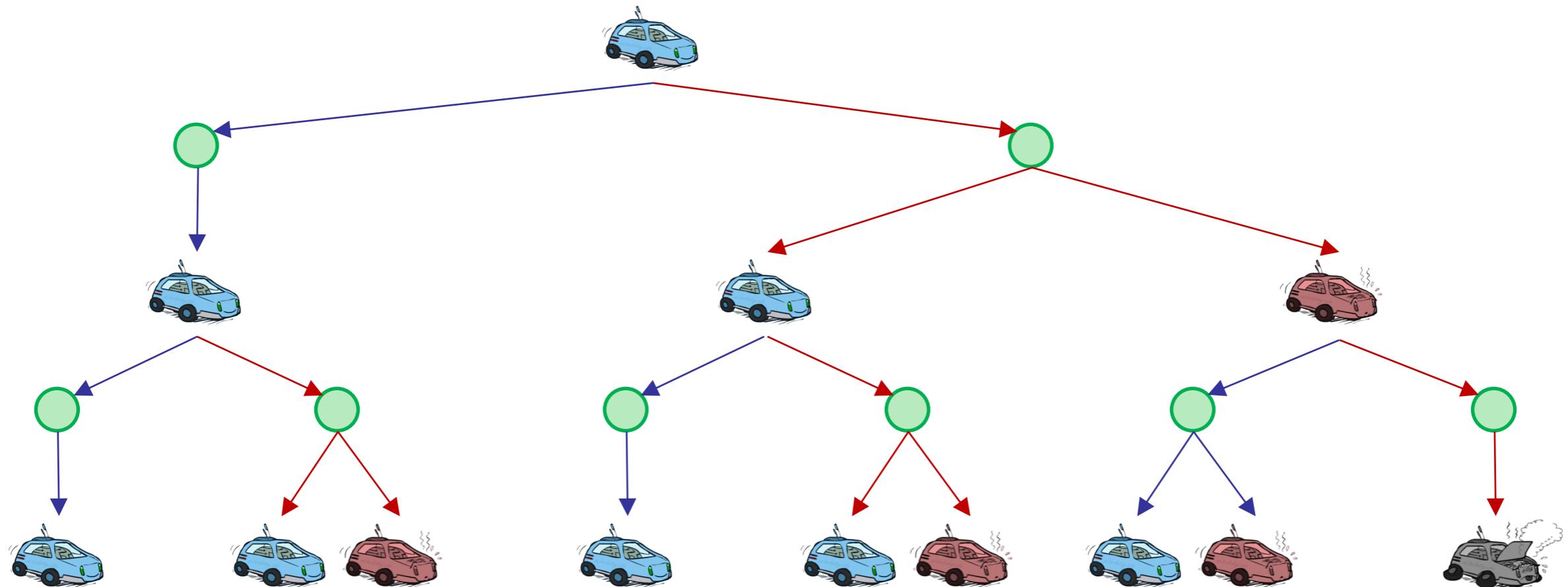
$$V^*(s) = \max_a Q^*(s, a)$$

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

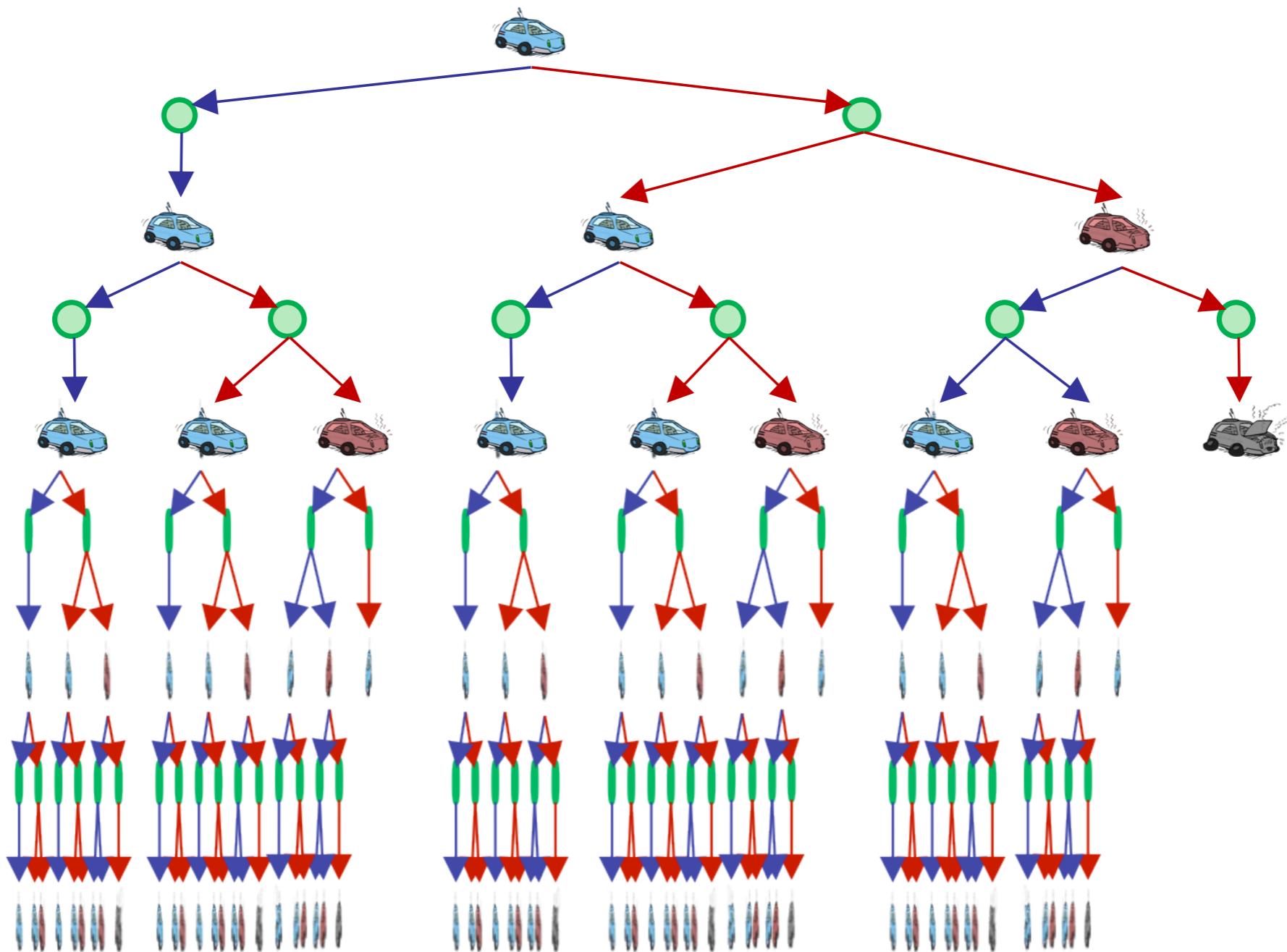
$$V^*(s) = \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^*(s')]$$



赛车搜索树

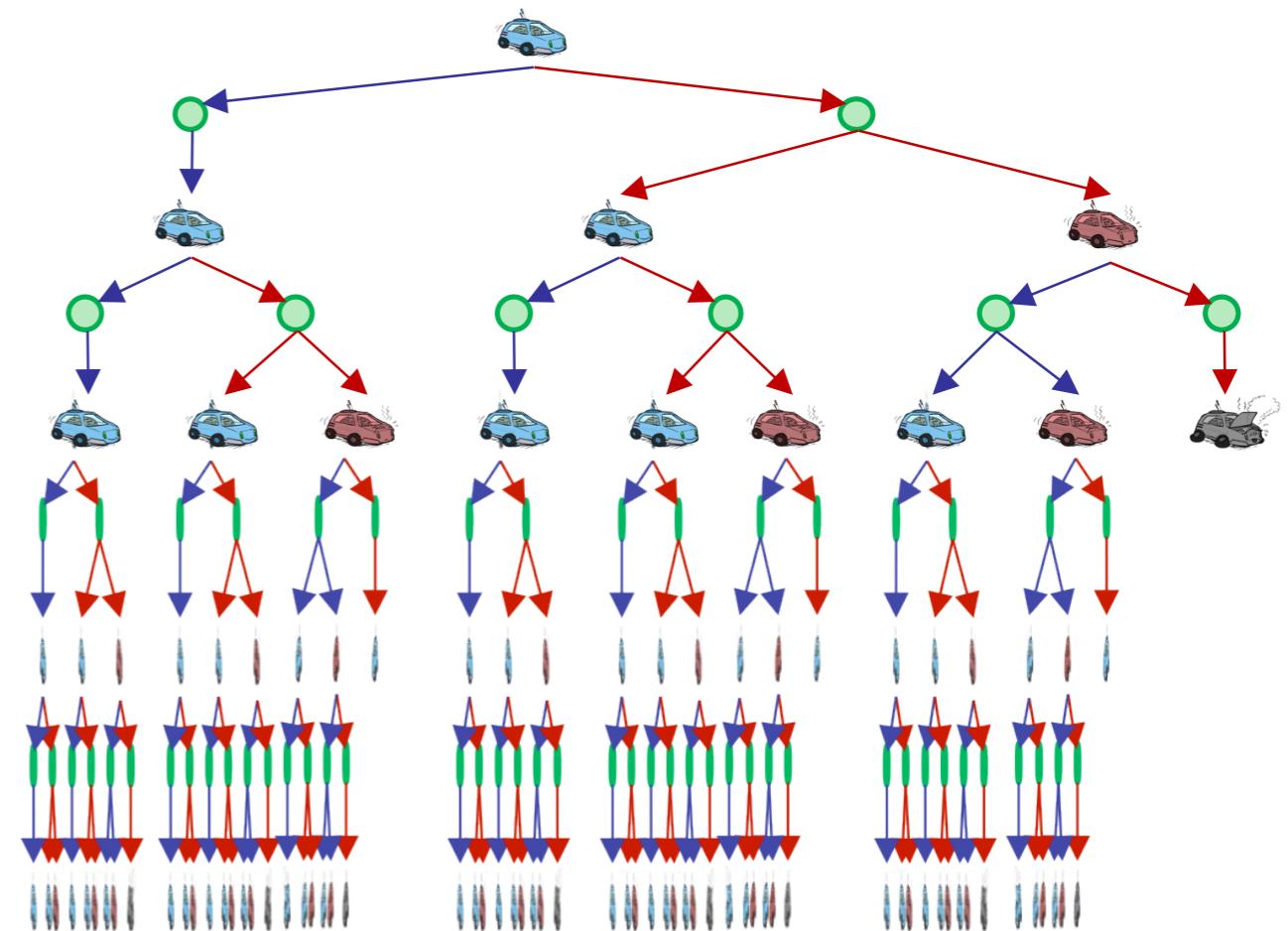


赛车搜索树



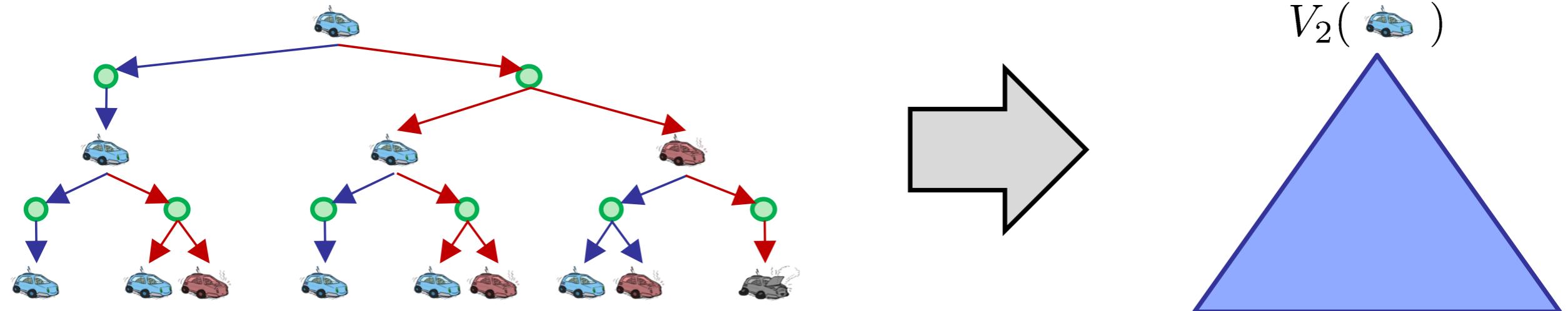
赛车搜索树

- 用expectimax会做太多的计算!
- 问题：状态是重复的
 - 想法：只计算一次
- 问题：树永远在继续
 - 想法：做一个深度有限的计算，但随着深度的增加，直到变化很小
 - 注意：如果 $\gamma < 1$ ，树的深部最终不重要

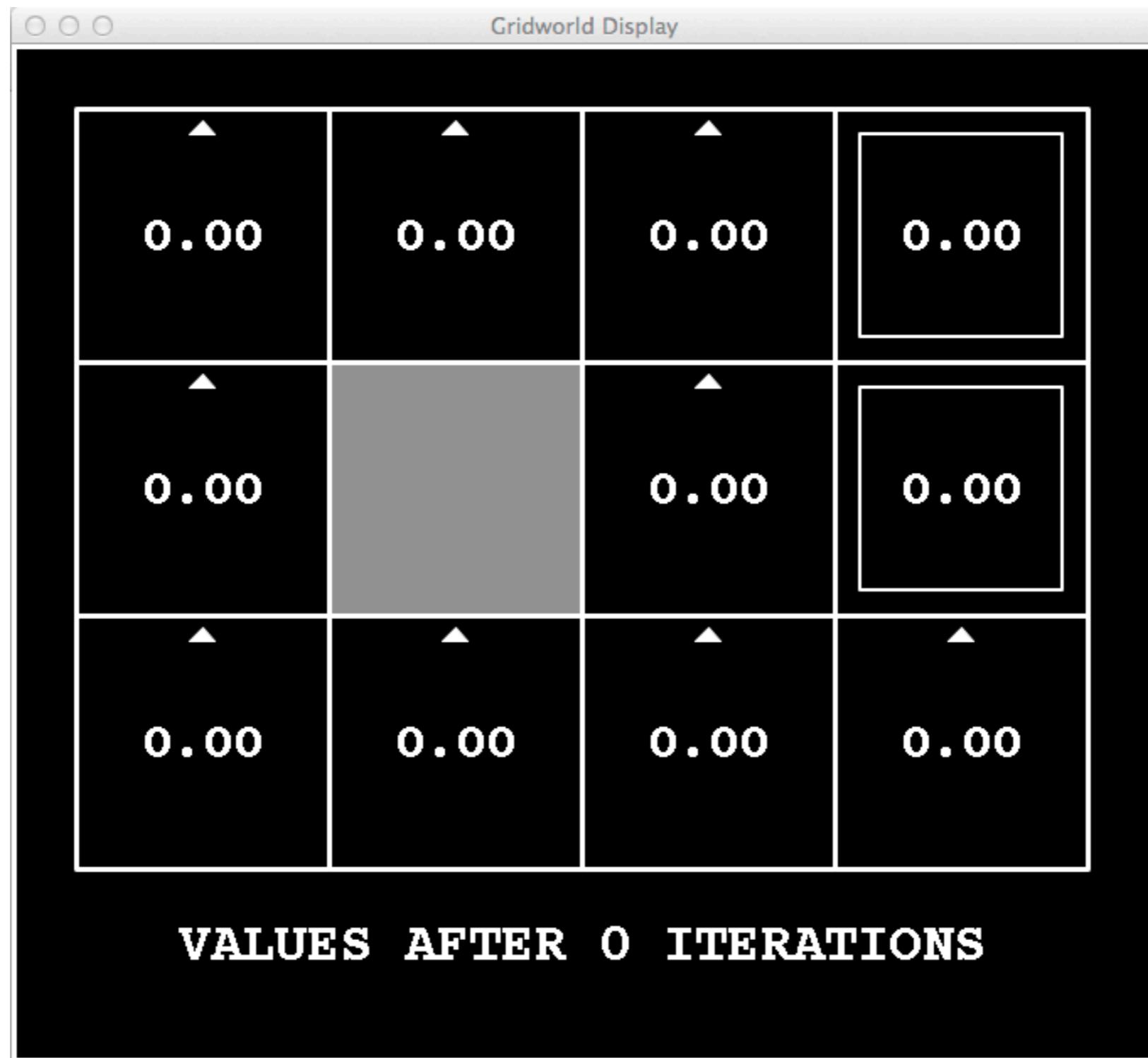


有限时间步长下的V值

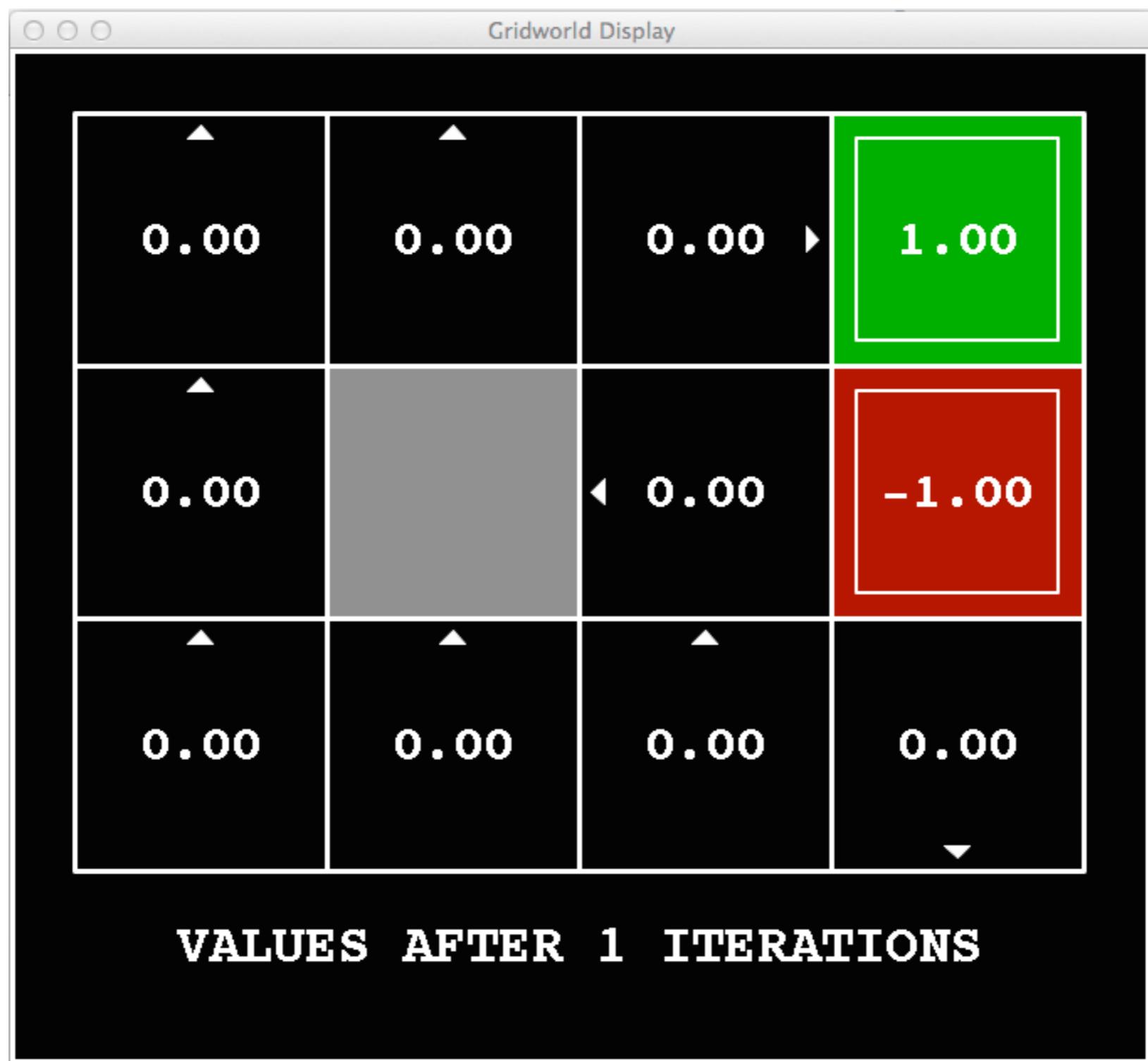
- Key idea: 限制时间步长
- Define $V_k(s)$ 状态s的最优值, 如果游戏在K步长后结束
 - 相当于 it's what a depth-k expectimax would give from s



k=0



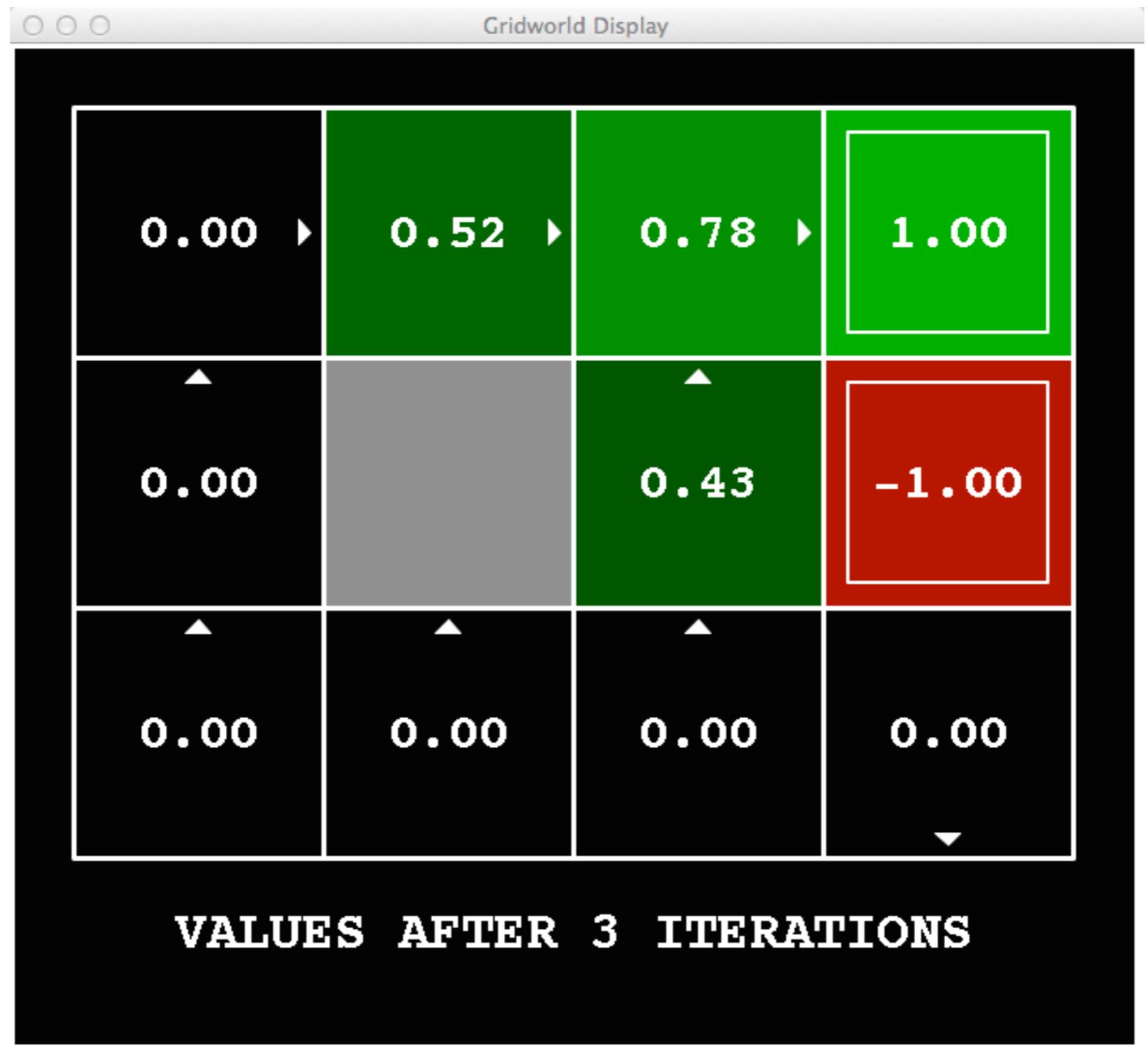
$k=1$



k=2



k=3



k=4



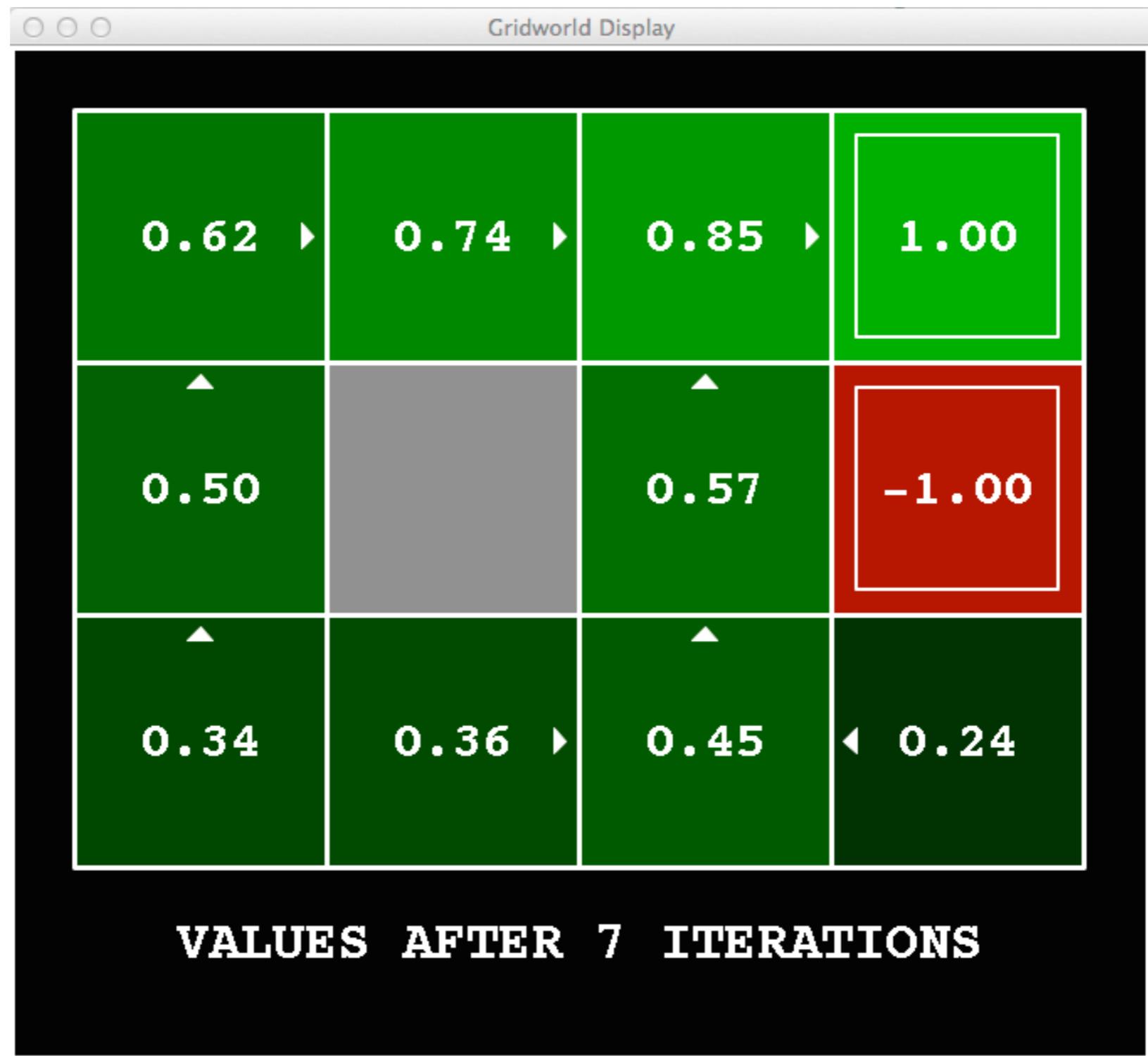
k=5



k=6



k=7



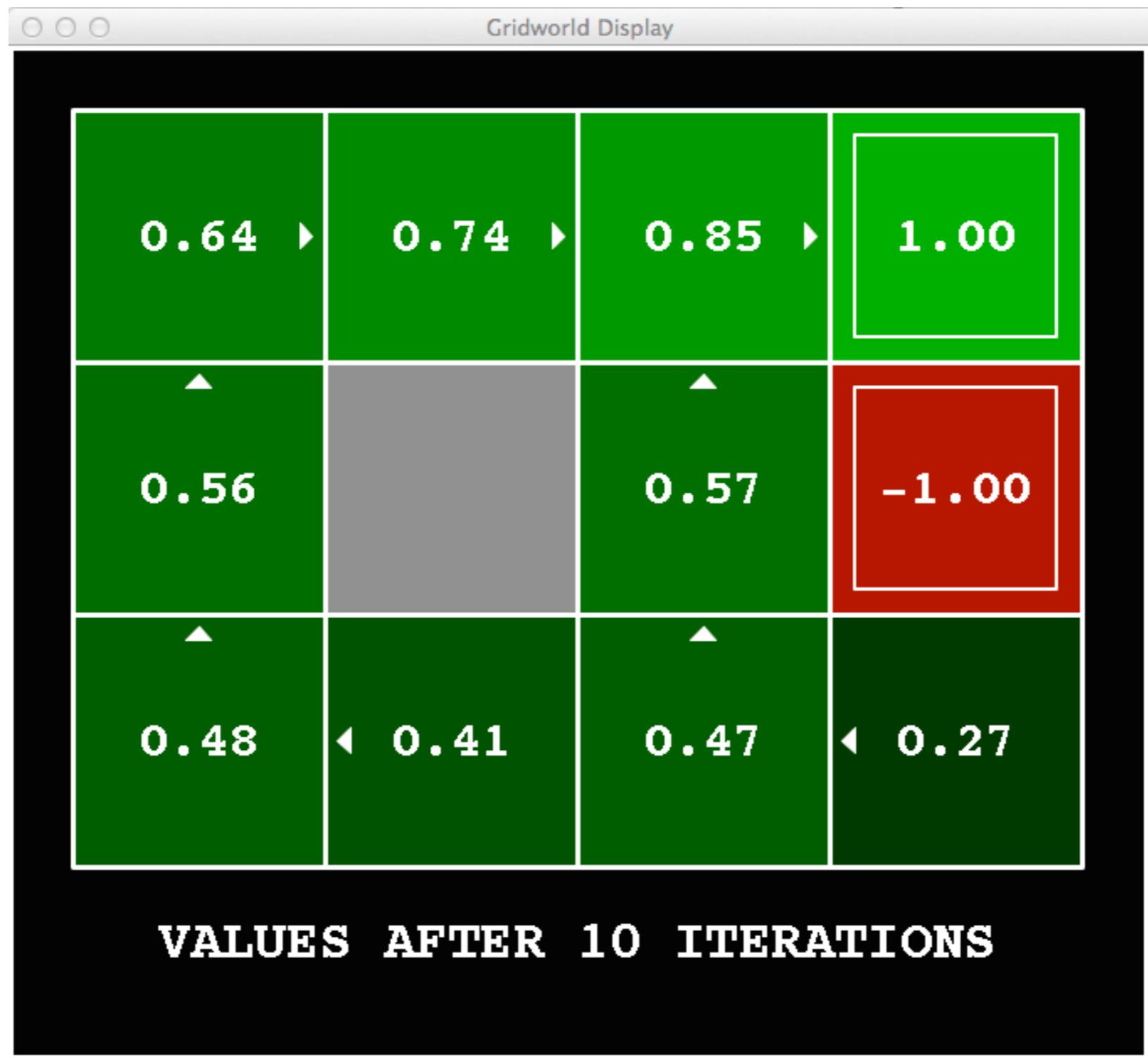
k=8



k=9



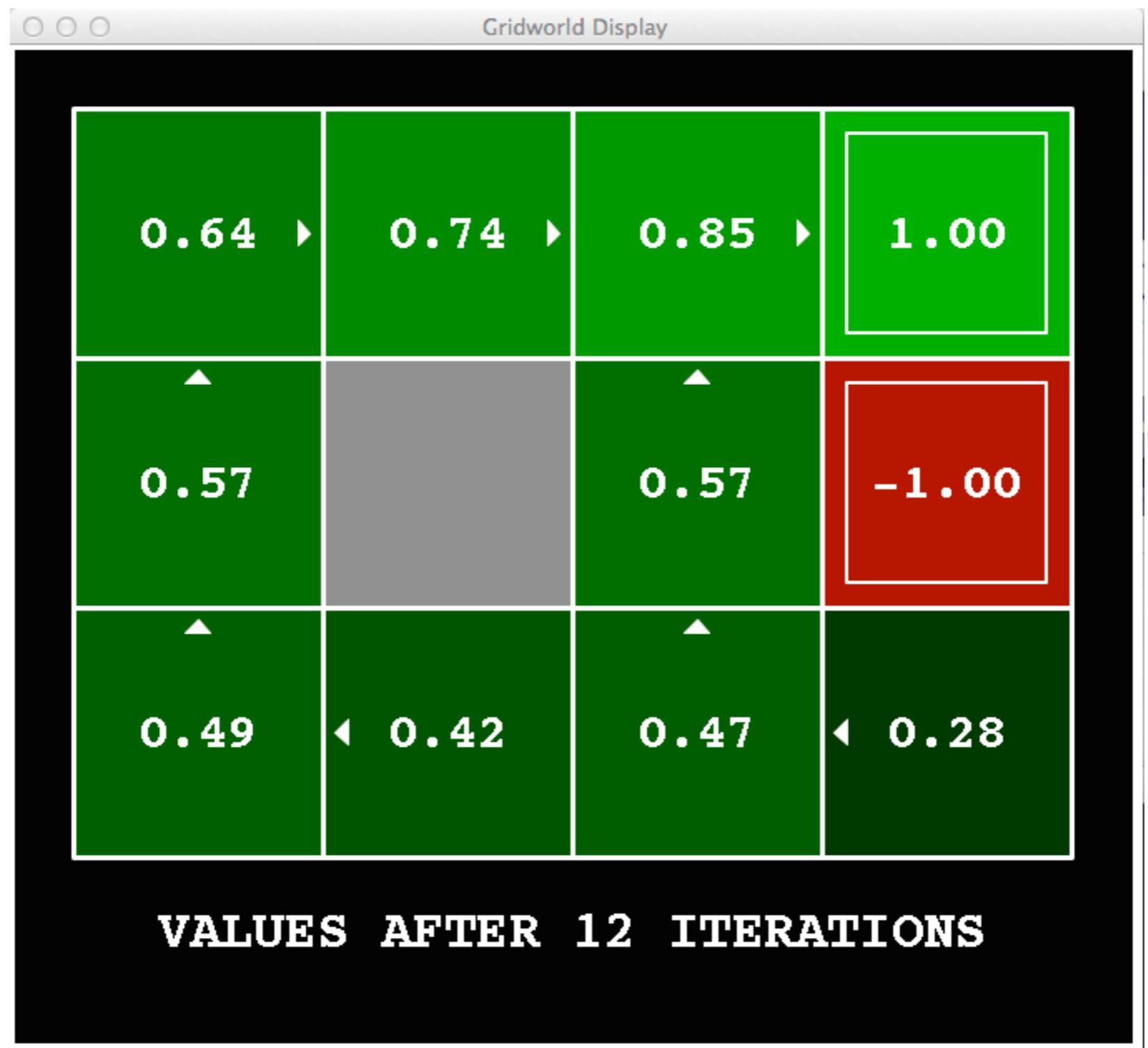
k=10



k=11



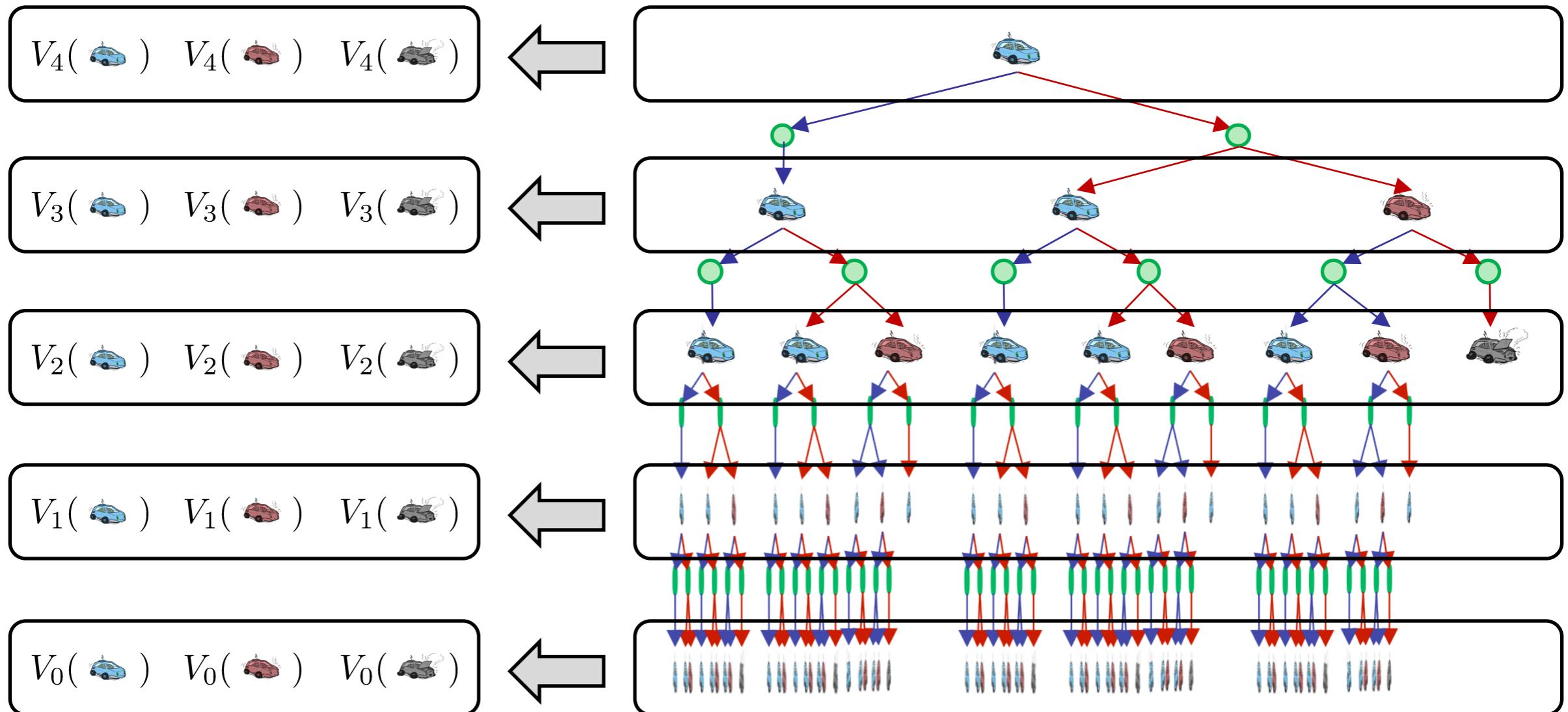
k=12



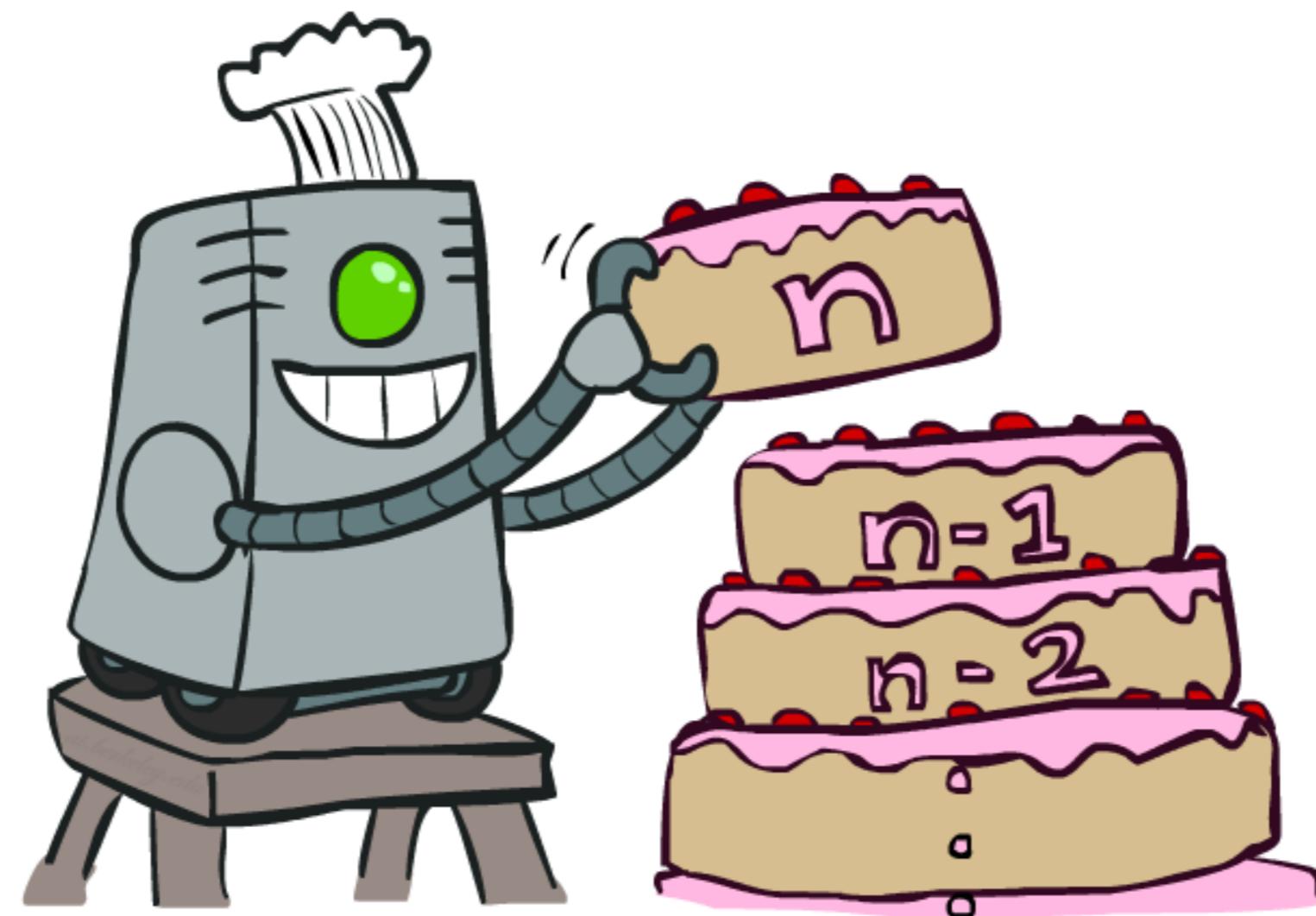
k=100



计算时间限制值



状态赋值迭代 Value Iteration

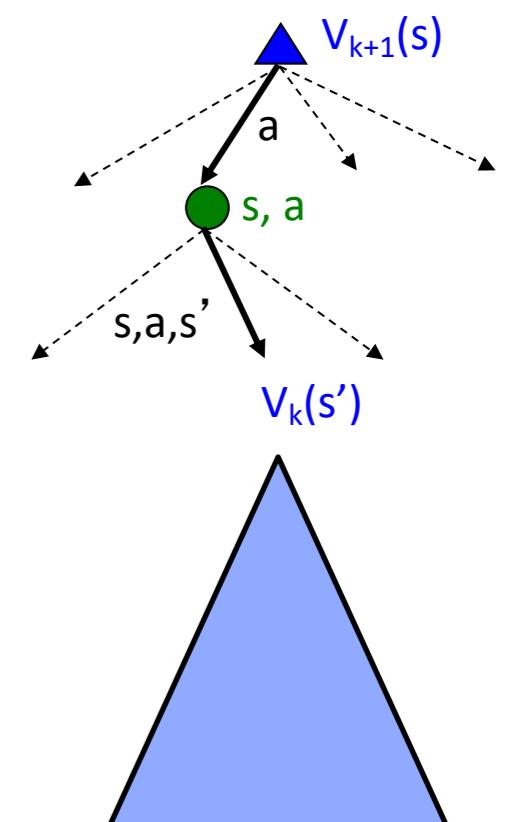


状态赋值迭代

- 开始于 $V_0(s) = 0$
- 给定向量 $V_k(s)$ 值, 计算一步期望最大值,
根据以下公式:

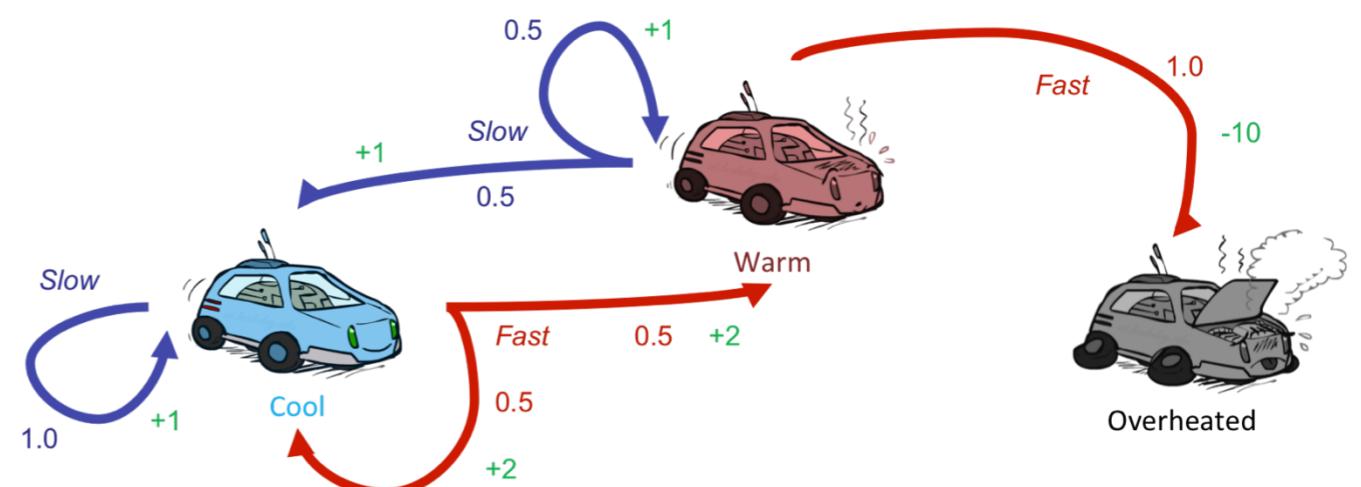
$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V_k(s')]$$

- 重复这个过程, 直到收敛
- 每步迭代的计算时间复杂度: $O(S^2A)$
- 定理: 这个过程将会收敛于唯一的最优值



举例: 赋值迭代

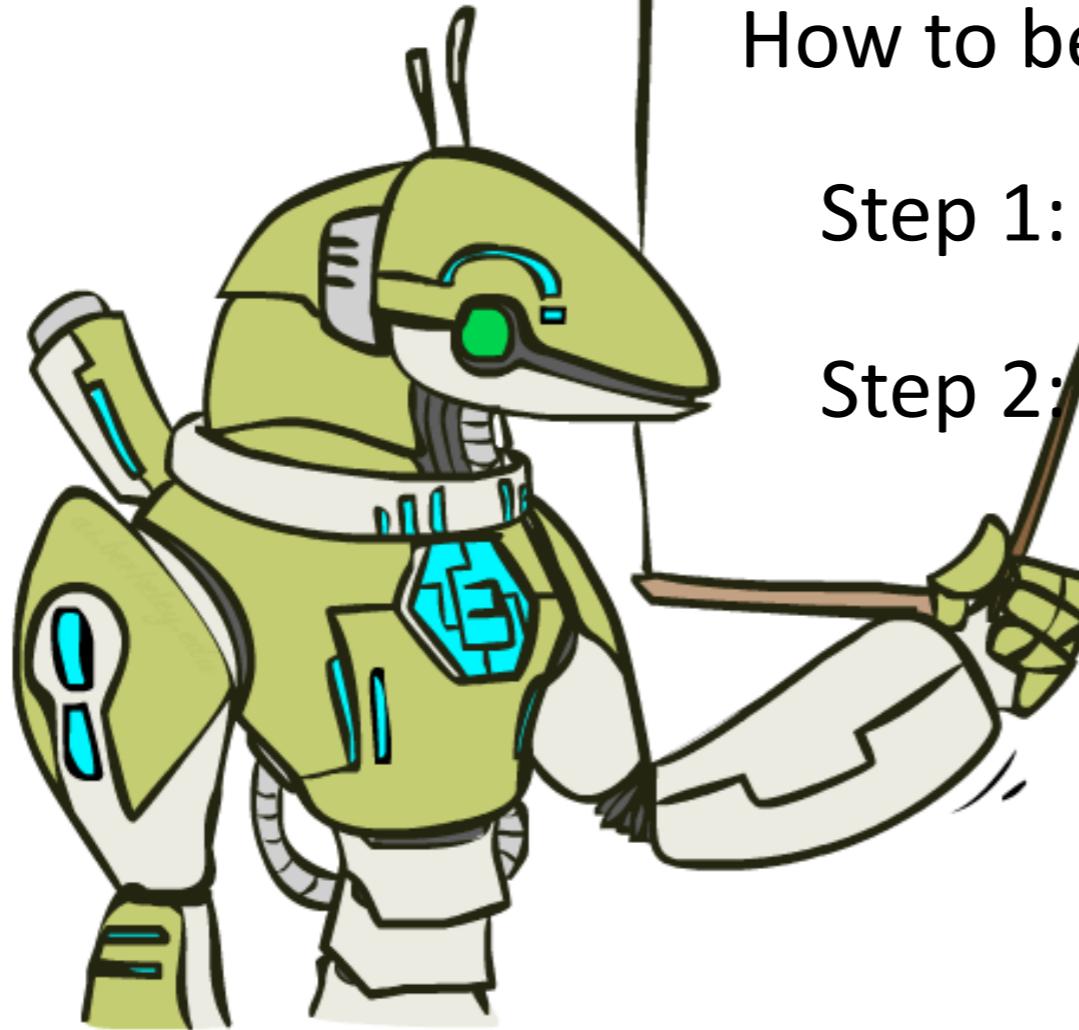
V_2	3.5	2.5	0
V_1	2	1	0
V_0	0	0	0



Assume no discount!

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

贝尔曼方程 The Bellman Equations



How to be optimal:

Step 1: Take correct first action

Step 2: Keep being optimal

贝尔曼方程

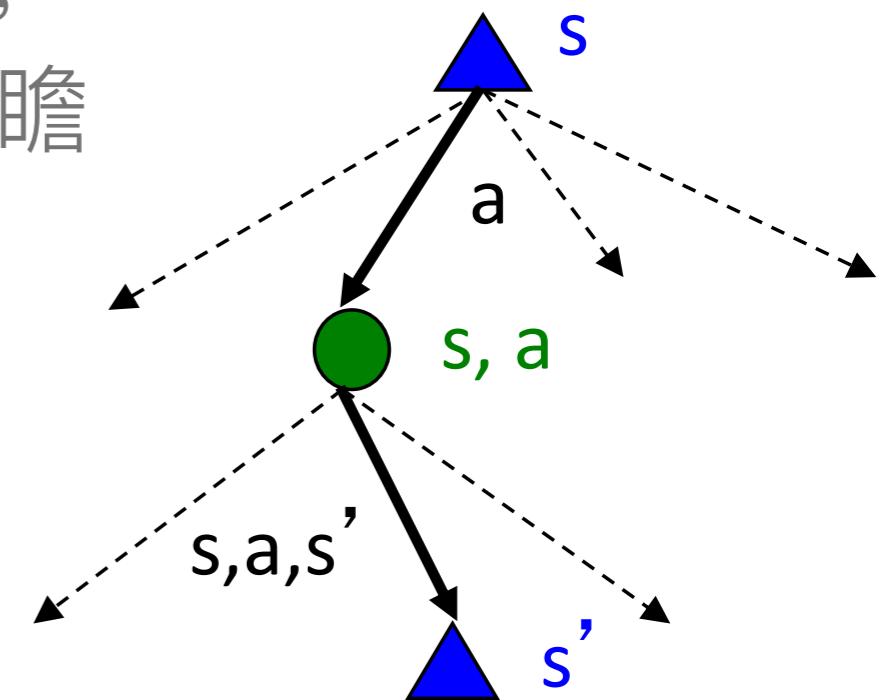
- 通过expectimax递归定义“最效用”，给出了最效用值之间简单的一步前瞻关系

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

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

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

- 这就是贝尔曼方程，它以迭代的方式描述了最优值



值迭代 Value Iteration

- Bellman方程描述了最佳值：

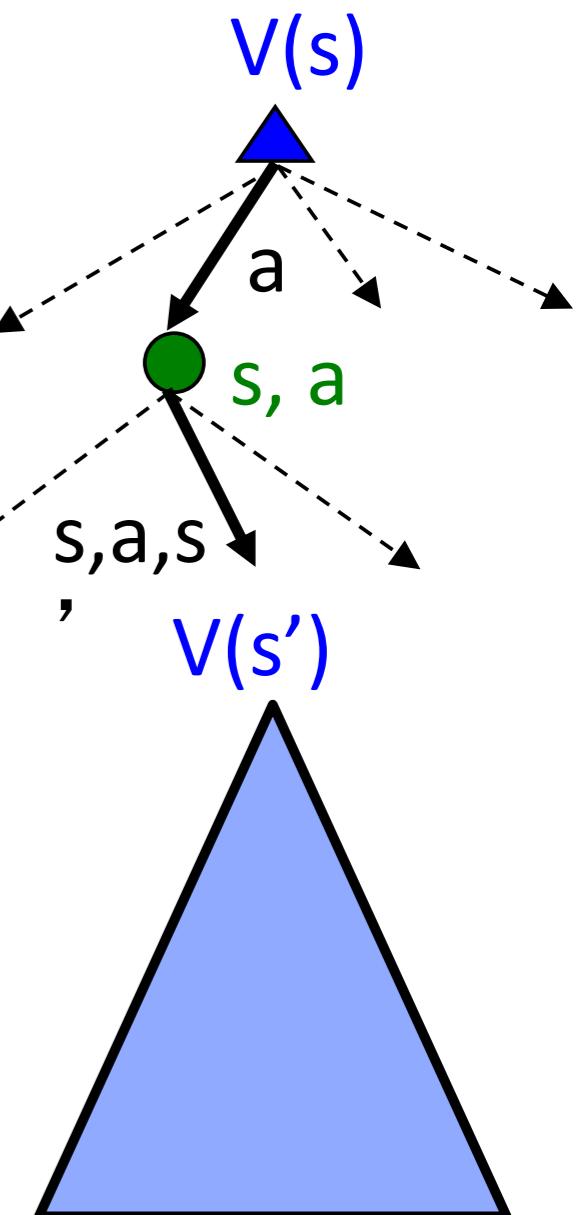
$$V^*(s) = \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^*(s')]$$

- 值迭代计算它们：

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

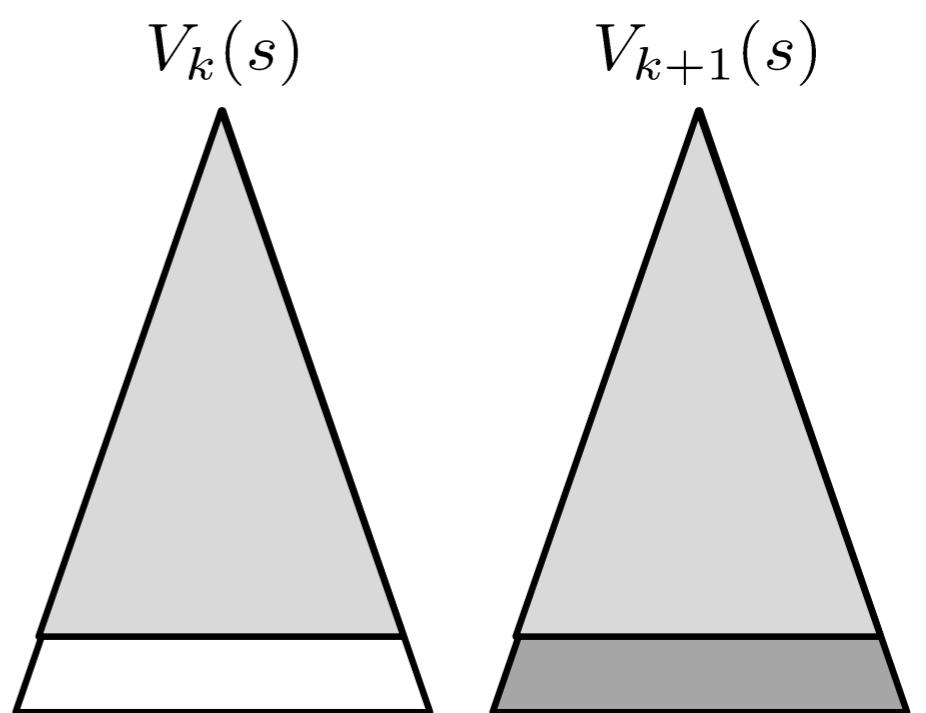
- 值迭代只是一种不动点求解方法

- ...虽然 V_k 向量也可以解释为时间限制值

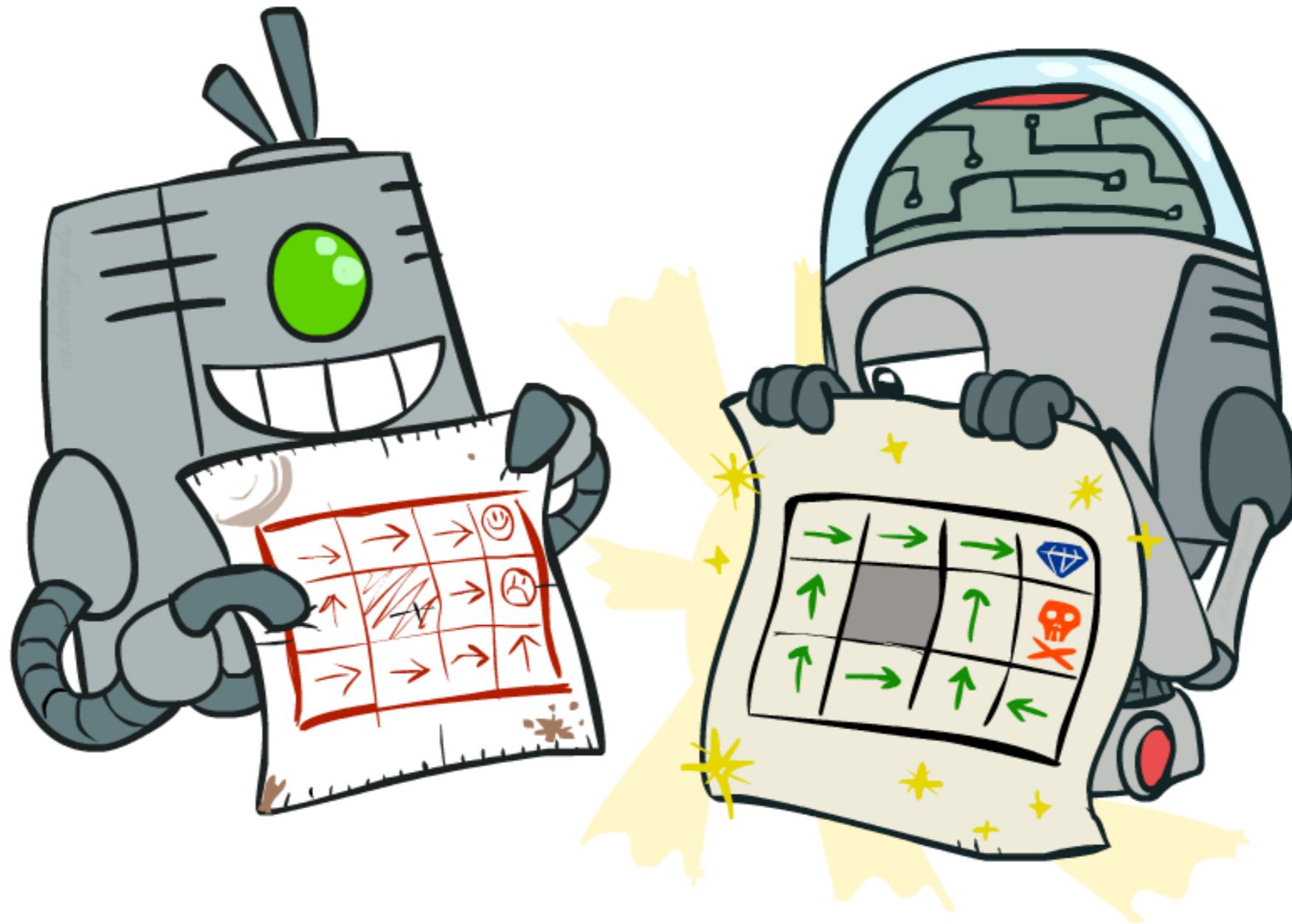


收敛 Convergence

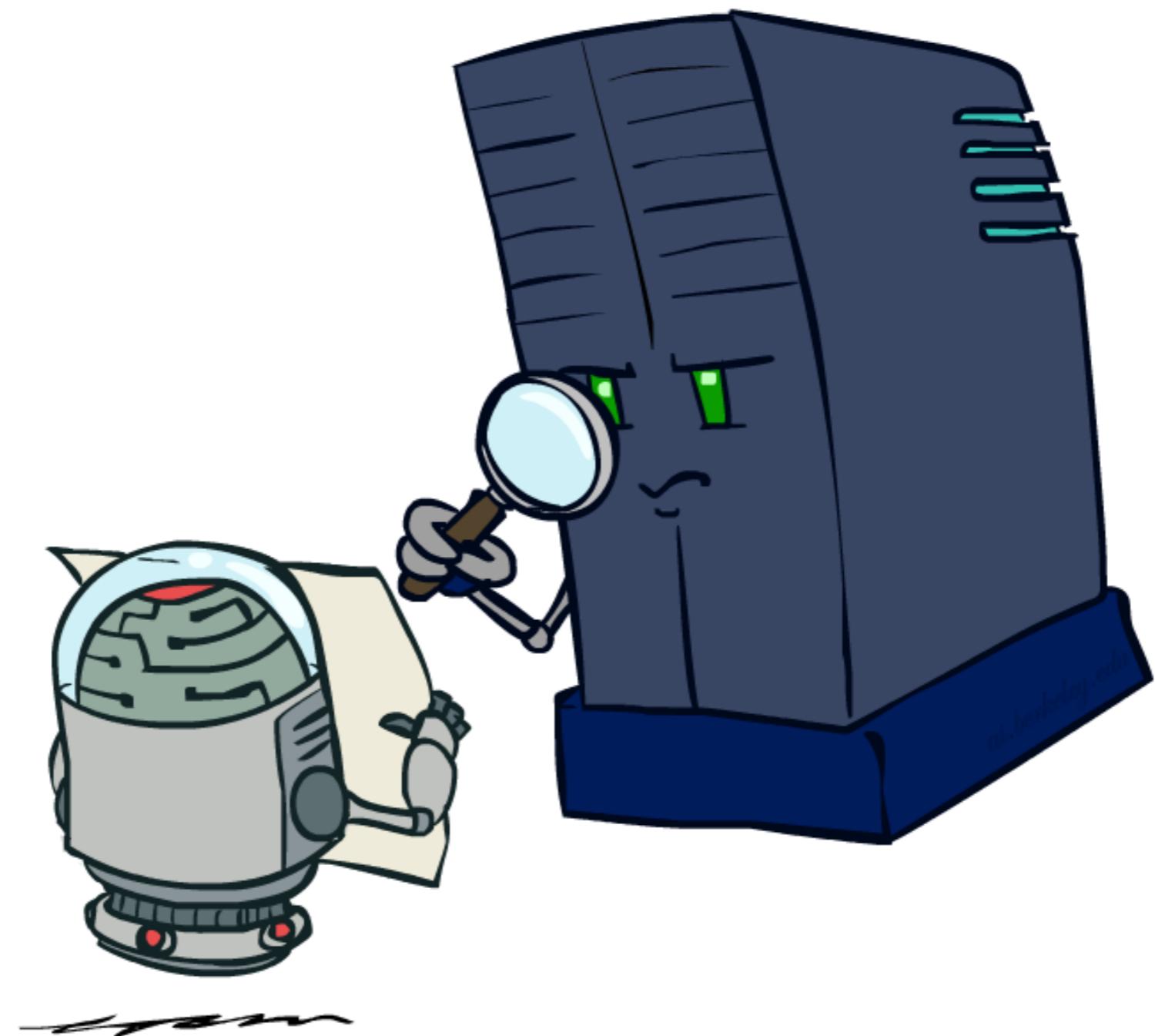
- 我们怎么知道 V_k 向量会收敛?
- 情况1：如果树有最大深度M，那么 V_M 计算得到实际的值
- 案例2：如果折扣小于1
 - 对于任意状态 V_k 和 V_{k+1} 可以看做一棵深度为 $k+1$ 的搜索树
 - 区别在于 V_{k+1} 在最后一层有奖励，而 V_k 没有
 - 最后一层奖励最大 R_{MAX}
 - 但是要乘以一个折扣 γ^k
 - 随着k的增加， $\gamma^k R_{MAX} \rightarrow 0$



基于策略的求解MDP方法

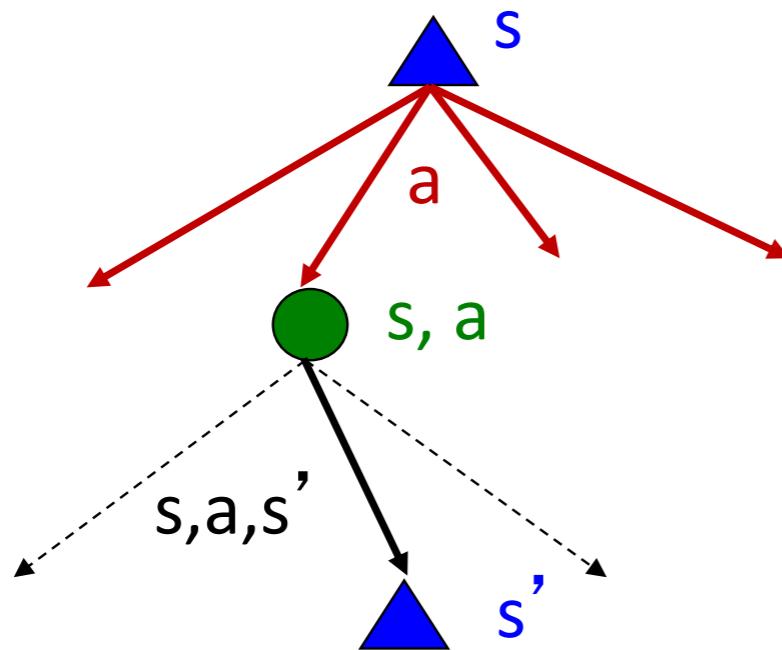


策略评价

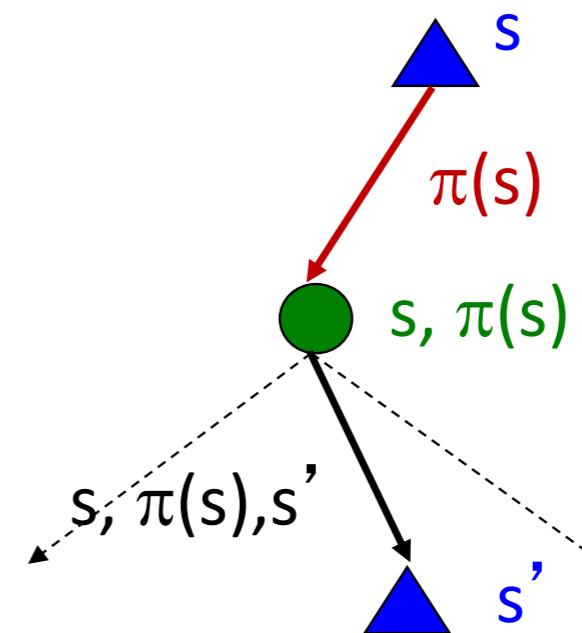


固定的策略

采取最优行动



按照策略 π 选择行动

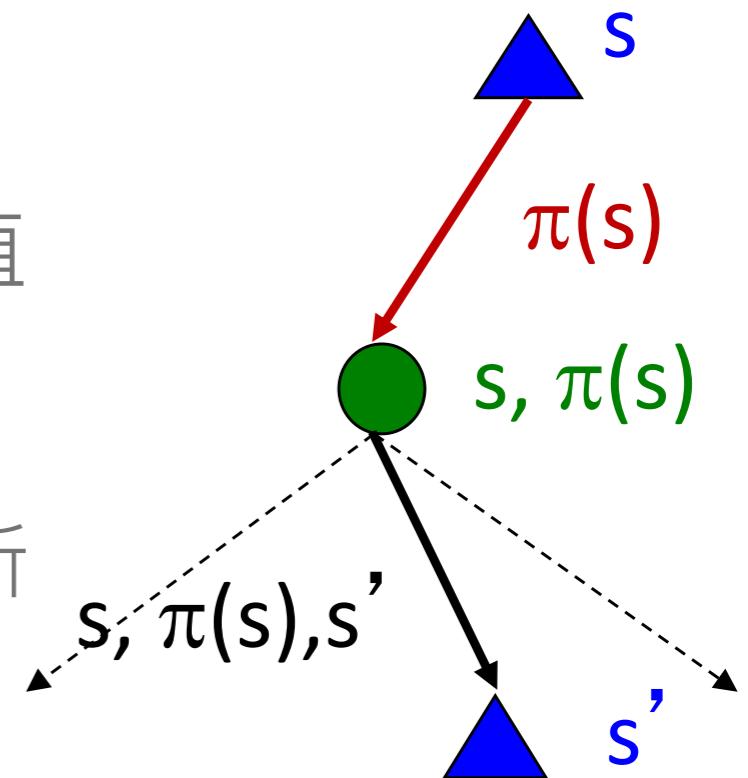


- 期望最大值搜索树在所有行动分支里选择最大功效值
- 如果使用一个给定的策略 $p(s)$, 那么这个搜索树变得比以前要简单 – 每个状态节点只有一个分支
 - ... 所以搜索树的状态节点值的计算将取决于我们所固定（使用）的那个策略

用来评价一个固定的策略的功效值

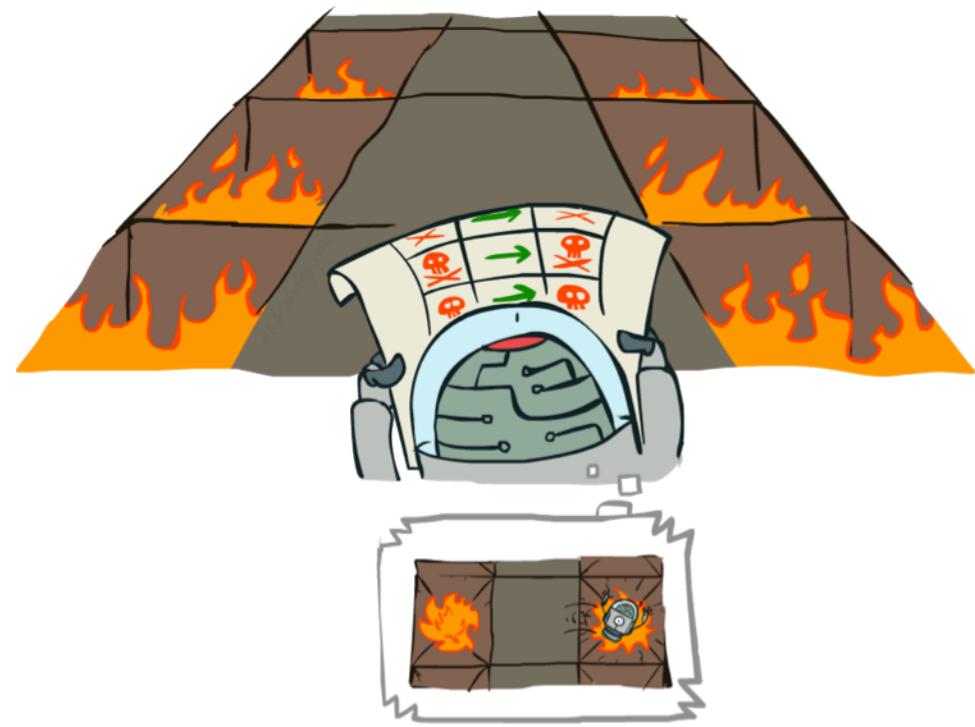
- 当给定一个策略（通常不是最优的）时，如何计算一个状态 s 的功效值
- 给定策略 π 下，定义一个状态 s 的功效值为：
 - $V^\pi(s) =$ 从状态 s 开始，按照策略 π 执行，所获得的期望折扣奖赏值的总计
- 迭代关系 (基于 Bellman 公式的一步计算):

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

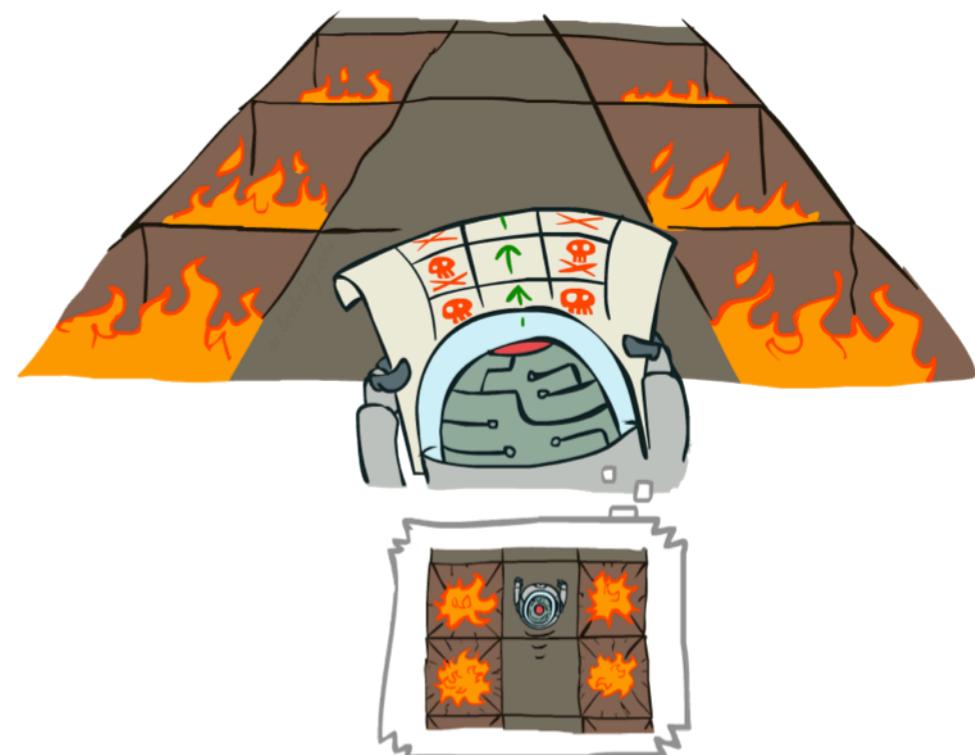


举例：策略评价

Always Go Right



Always Go Forward

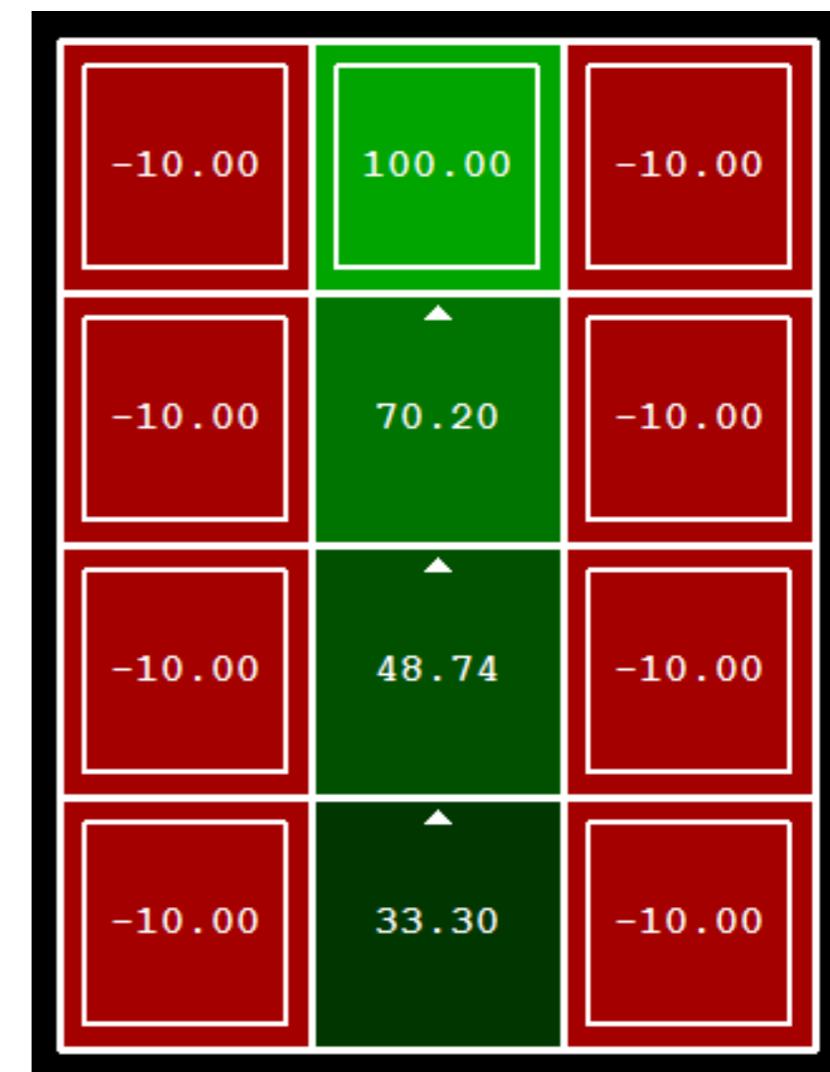


举例：策略评价

Always Go Right



Always Go Forward



策略评价

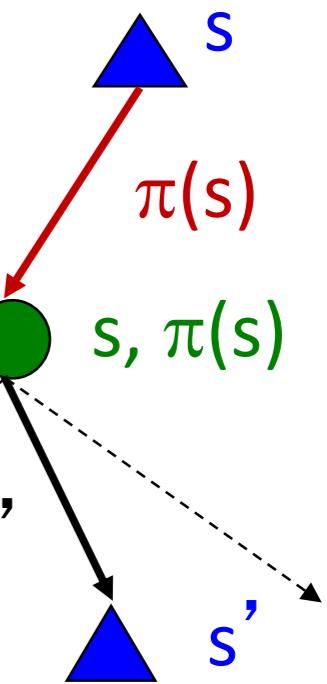
- 给定一个策略 π , 如何计算 V^π 值?

- 基于 Bellman 方程的赋值迭代更新

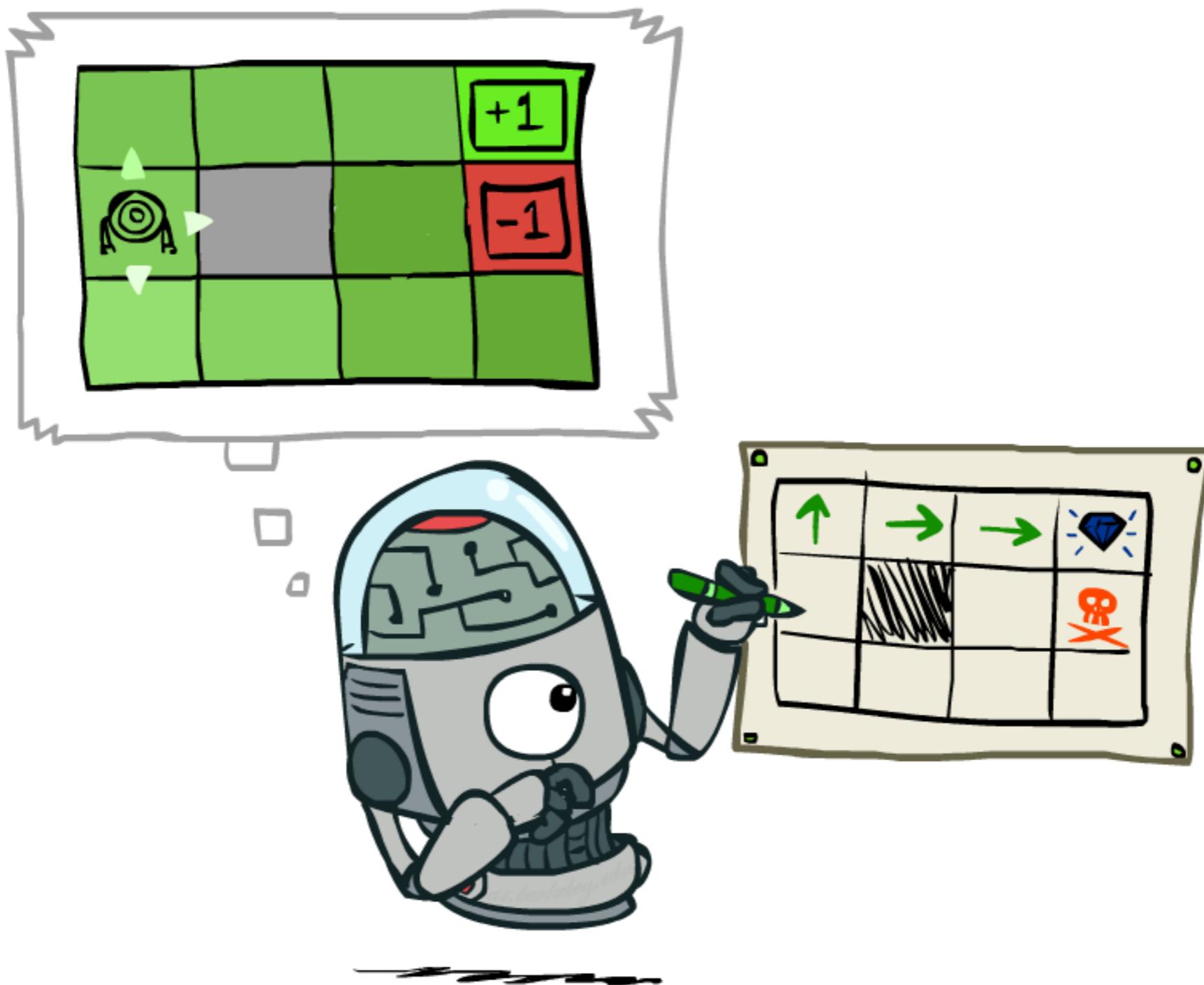
$$V_0^\pi(s) = 0$$

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

- 效率: $O(S^2)$ 每步迭代
 - 由于没有了“最大符号”, Bellman 方程变成了一个线性系统



策略提取 Policy Extraction



从状态值计算行动

- 假设我们已有了每个状态的最优值 $V^*(s)$
- 那么我们如何选择行动?
 - 不是很明显!
- 需要计算一步长的 mini-expectimax:
$$\pi^*(s) = \arg \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^*(s')]$$
- 这叫做 **策略提取**, 通过计算期望最大值, 间接地获得最优行动选择



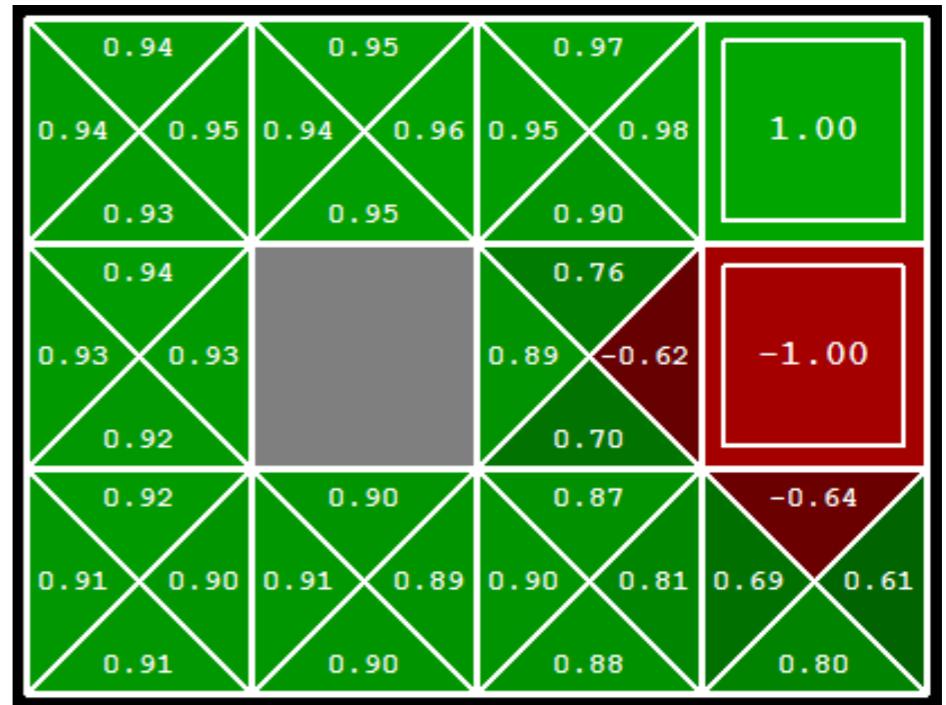
从Q-值计算行动

- 假设我们已有了最优的 q-values:

- 现在我们如何挑选行动?

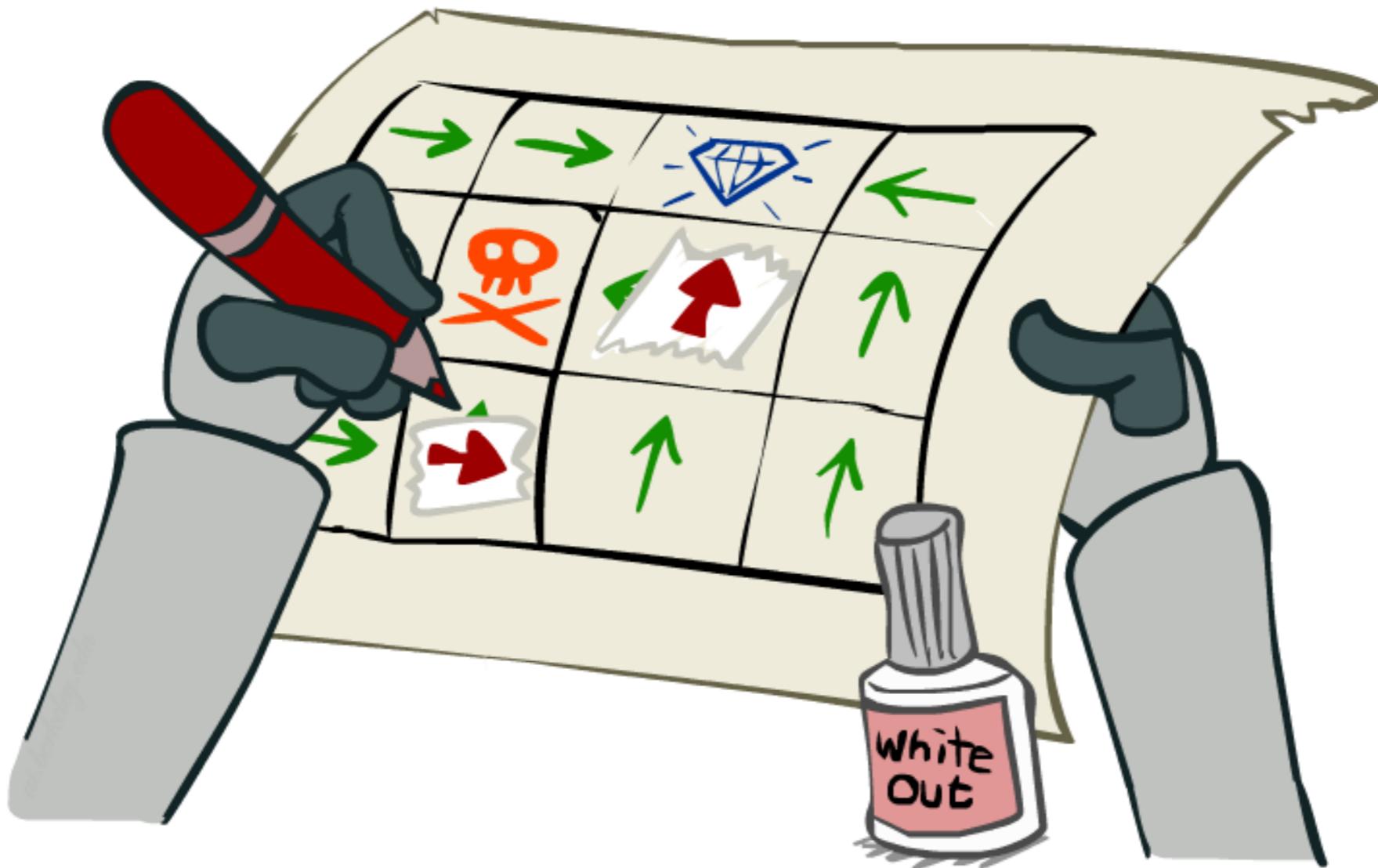
- 此时变得很简单!

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



- 因此: 为每个状态位置选择最优行动从Q-值中比在状态值中更容易!

策略迭代法 Policy Iteration



状态赋值迭代方法的缺点

- 赋值迭代实现的是基于 Bellman 方程的更新公式：

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

- 问题 1：很慢 – 每次迭代的复杂度是 $O(S^2A)$
- 问题 2：在每个位置（状态）的“最大”选项很少改变
- 问题 3：策略 policy 通常比状态值收敛的更快

策略迭代法 Policy Iteration

- 另一种方法求解最优值:
 - 步骤 1: 策略评价: 给定某个固定的行动策略, 计算各个状态值 (尽管他们不是代表最优的状态值!) , 迭代计算, 直到这个状态值收敛
 - 步骤 2: 策略改进: 更新行动策略, 使用向前一步的计算, 使用之前迭代计算收敛的 (but not optimal!) 状态值 (作为未来的状态值)
 - 重复这两步直到行动策略收敛

策略迭代法 (Policy iteration)

- 策略评价: 对于当前的策略 π , 使用策略评价过程计算状态位置的值:
 - 迭代直至V-值收敛

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

- 策略改进: 给定计算出来的V-值, 通过策略提取过程, 计算获得更好一步的策略
 - 向前一步的计算:

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

比较

- 状态值迭代和策略迭代方法计算的是同一件事情 (所有状态节点的最优功效值)
- 在状态赋值迭代里:
 - 每次迭代, 更新状态V-值, 和策略(隐式地)
 - 没有直接追踪策略, 但是从不同行动分支中获取一个最大值时, 实际上隐式地计算了策略
- 在策略迭代里:
 - 我们在当前固定的策略下, 通过几次迭代, 更新计算状态的功效值(V-值) (每次迭代很快, 因为我们此时只考虑一个行动, 而不是所有的行动分支)
 - 在当前策略评价过程完成以后, 一个新的策略被挑选出来 (这一步较慢, 就像状态赋值迭代方法里的一次迭代)
 - 新的策略将会更优化 (否则的话, 迭代过程结束)
- Both are dynamic programs for solving MDPs

总结: MDP 算法

- 我们用到的算法:
 - 计算状态节点的最优功效值: 使用状态赋值迭代, 或策略迭代方法
 - 给定一个策略, 计算状态值: 使用策略评价方法
 - 通过状态值获取一个策略: 使用策略提取方法 (向下一步优化)
- 这些方法看上去都很像!
 - 它们本质上都是基于 Bellman 赋值更新表达式
 - 全都使用了基于期望最大值的向下一步优化计算模块
 - 它们区别只是在于是否固定一个策略, 或是在所有行动分支中进行最优挑选 (最大化期望功效值的行动)

作业

- 在一个 8×8 的迷宫世界中：
 - 随机设定2个出口，并且给出出口随机设定一个+10或者-10的奖励；
 - Agent在迷宫中的行动存在不确定性，例如向action=east时，会有 $(1-p)$ 的概率滑到north或south；
 - 随机指定 p 值、Living reward和折扣系数 γ ；
 - 求解该MDP问题，给出Agent的最佳策略