

# 提醒

- 作业1，9月29日课堂提交，进教室时把作业放到讲台前面

# 今天的内容

- A\* 搜索及其启发式函数
- 局部搜索
- 对抗性的（博弈）搜索

# A\* 搜索



# A\* 搜索



基于成本的统一搜索法 (UCS)



[www.shutterstock.com](http://www.shutterstock.com) · 172490930

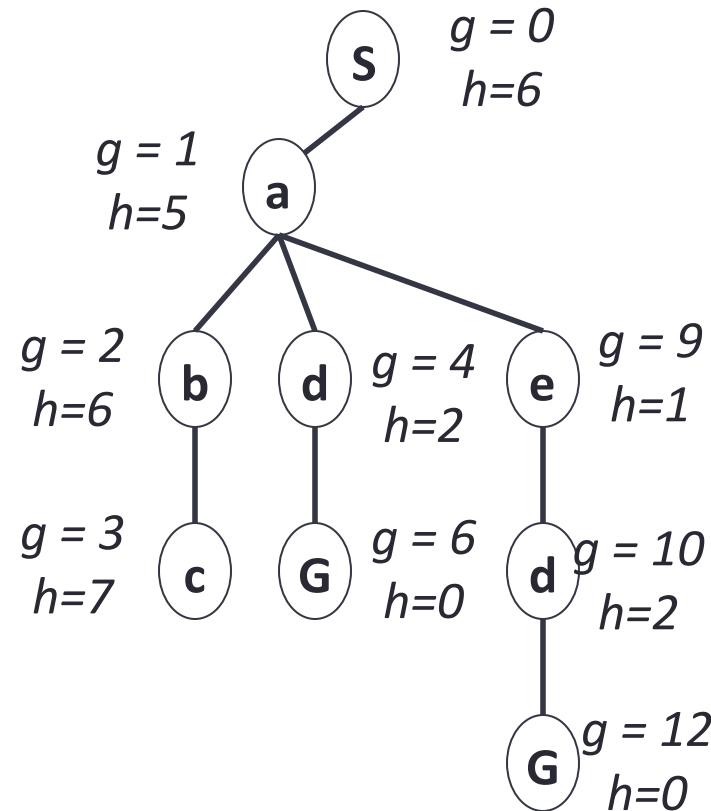
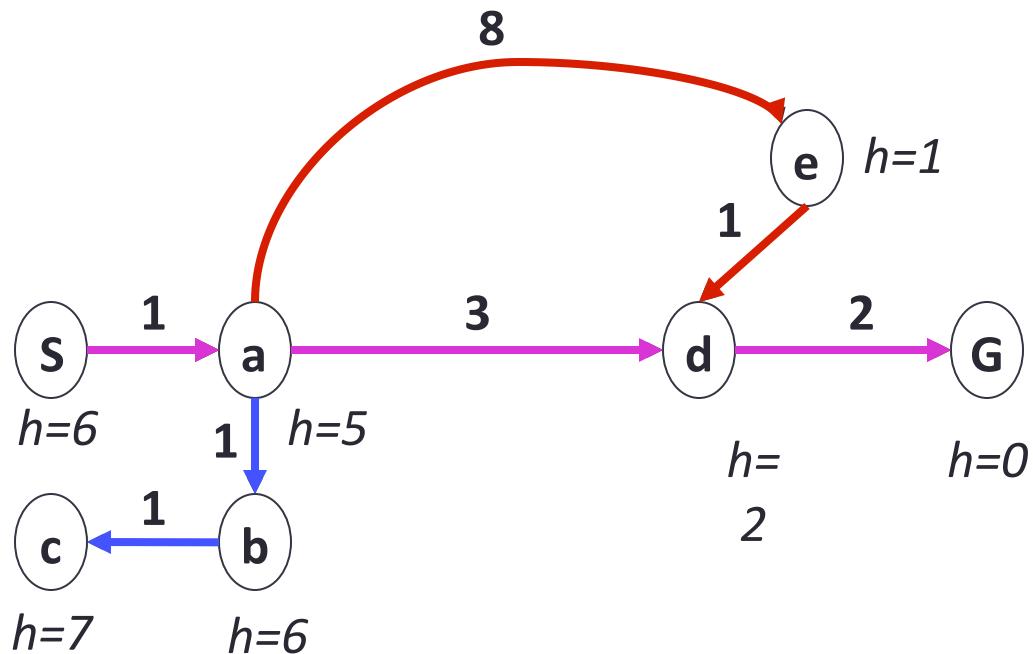


贪婪法 (Greedy)

A\*

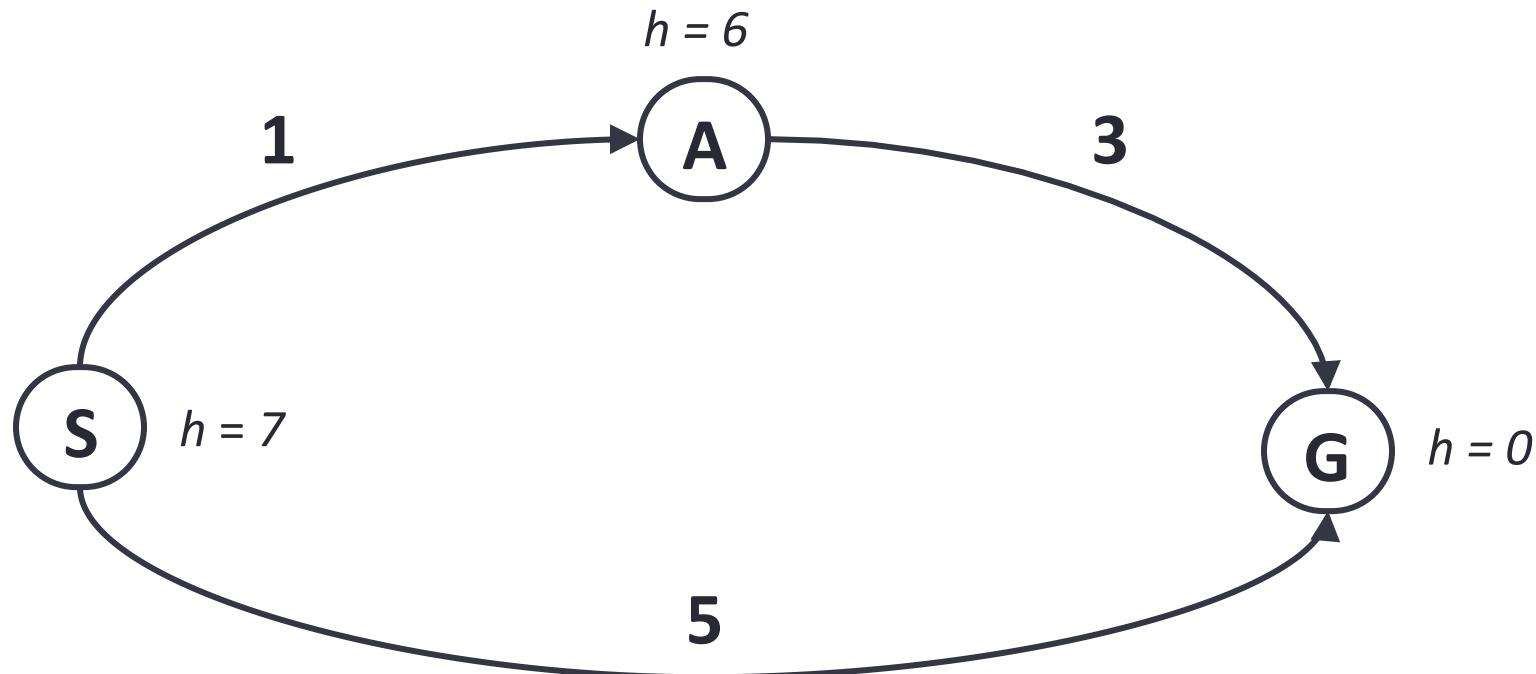
# 结合统一搜索和贪婪搜索

- 统一：把路径成本排序，即过来路程的成本  $g(n)$
- 贪婪：排序按照与目标的临近性，即前程成本  $h(n)$



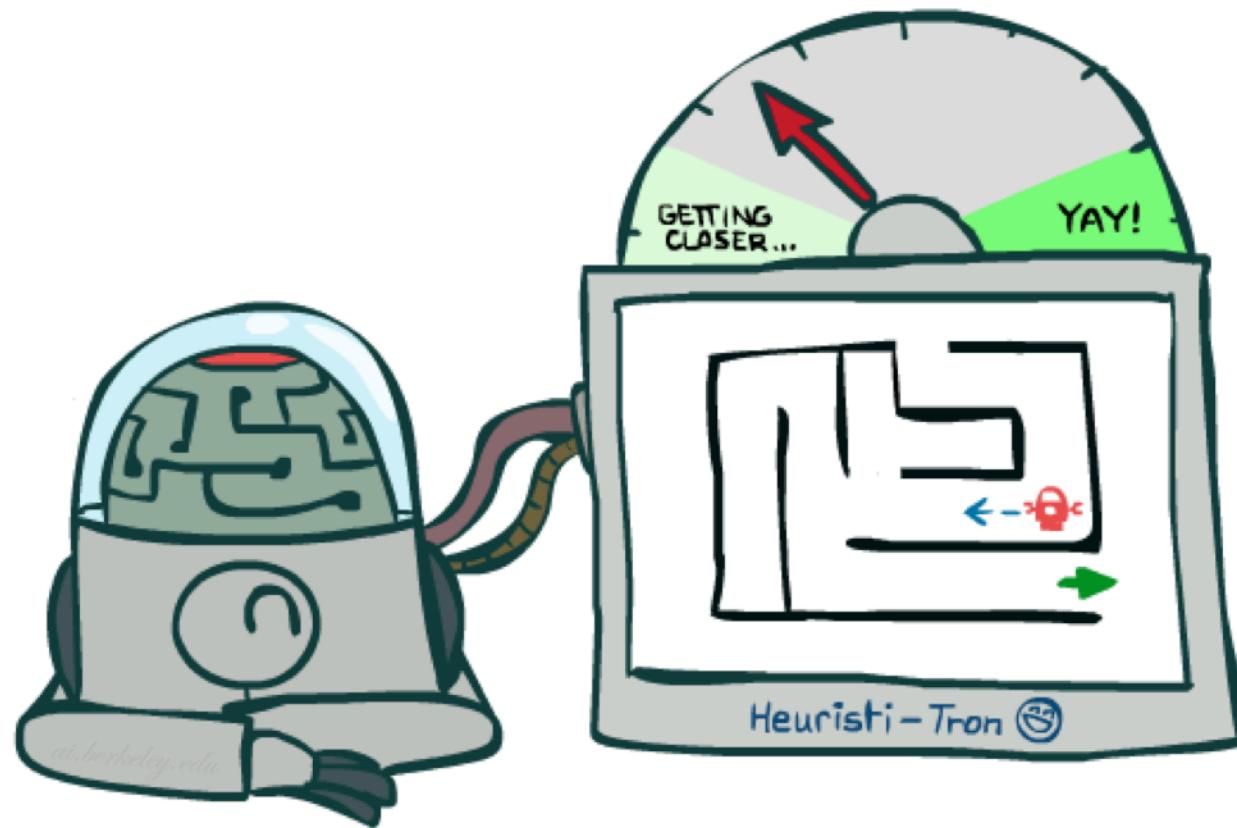
- A\* 搜索 :  $f(n) = g(n) + h(n)$

# A\* 是最优的吗?



- 哪个地方错了?
- 估计的到目标成本 > 实际到目标成本
- 我们需要估计值小于实际成本

# 可接纳的启发式函数



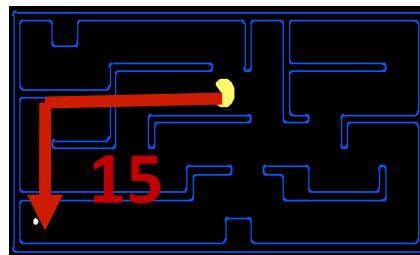
# 可接纳性的启发式函数 (Admissible Heuristics)

- $h$  是 可接纳的 (乐观的, 想的比实际好) :

$$0 \leq h(n) \leq h^*(n)$$

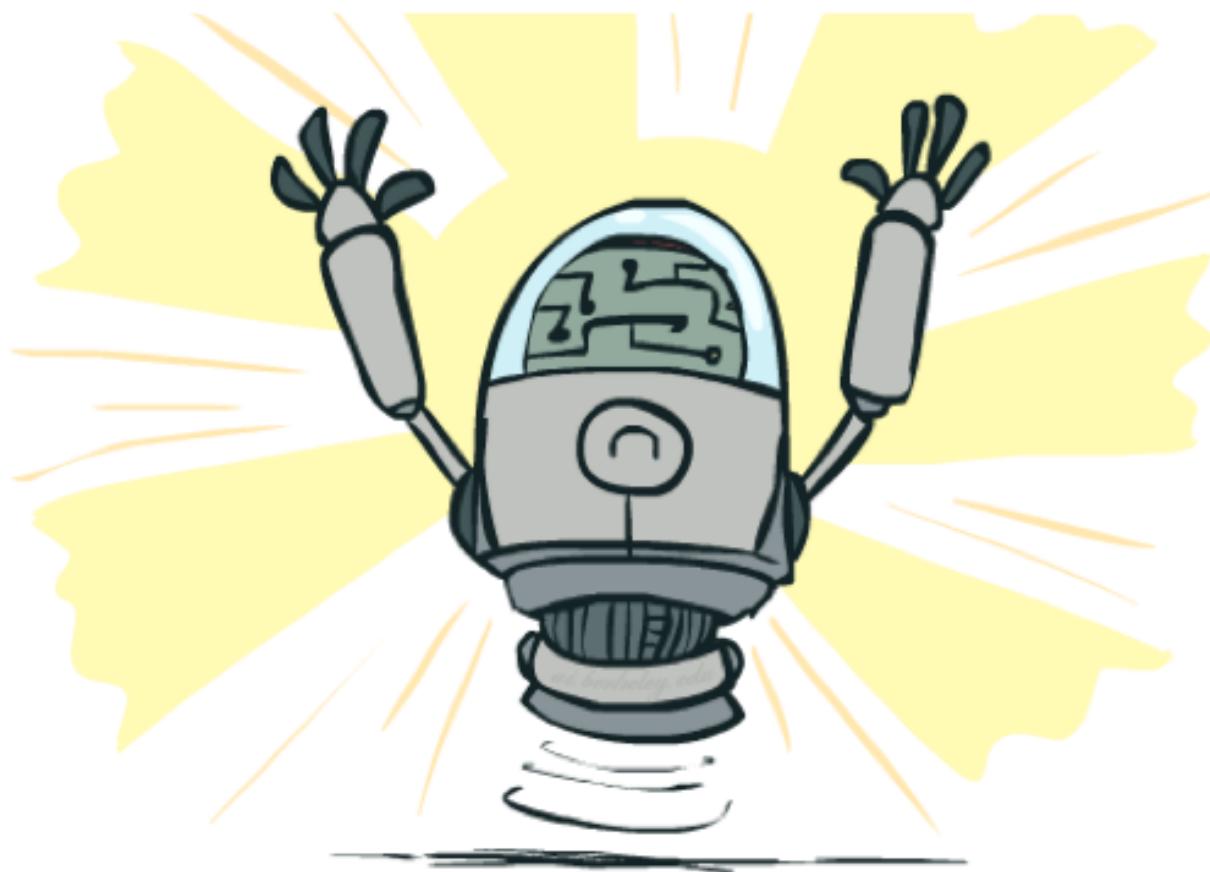
$h^*(n)$  是到一个最近的目标节点的真成本值

- 举例:



- 在实际应用A\*算法时，一个主要任务就是设计可接纳的启发式函数。

# A\* 树搜索的最优性



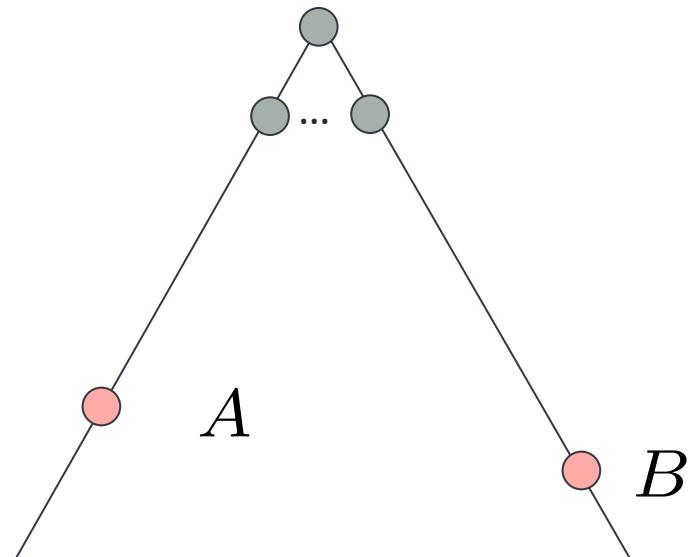
# A\* 树搜索的最优化

假定：

- A：一个最优目标节点
- B：一个次优目标节点
- $h$  是可接纳的

如果最优化成立，那么：

- A 将会在B之前被选择先扩展

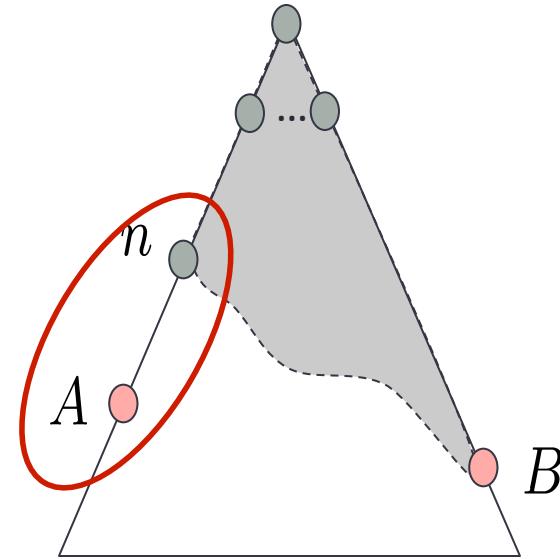


# A\* 树搜索的最优化

证明：

- 假设 B 在搜索前沿里
- A 的某个祖先节点  $n$  也在搜索前沿里
- 宣称:  $n$  会先于 B 被选择扩展

1.  $f(n)$  小于或等于  $f(A)$



$$f(n) = g(n) + h(n) \quad f \text{ 的定义}$$

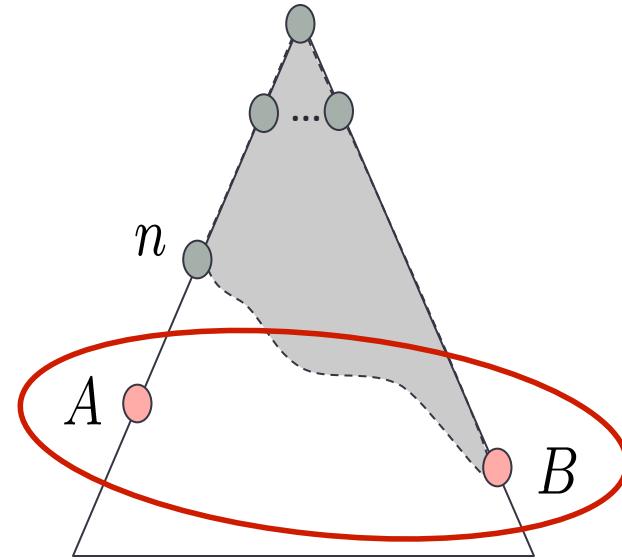
$$f(n) \leq g(A) \quad h \text{ 的可接纳性}$$

$$g(A) = f(A) \quad h = 0 \text{ 当在目标节点}$$

# A\* 树搜索的最优化

证明：

- 假设B在搜索前沿里
- A的某个祖先节点  $n$  也在搜索前沿里
- 宣称:  $n$  会先于 B 被选择扩展
  1.  $f(n)$  小于或等于  $f(A)$
  2.  $f(A)$  小于  $f(B)$



$$g(A) < g(B)$$

$$f(A) < f(B)$$

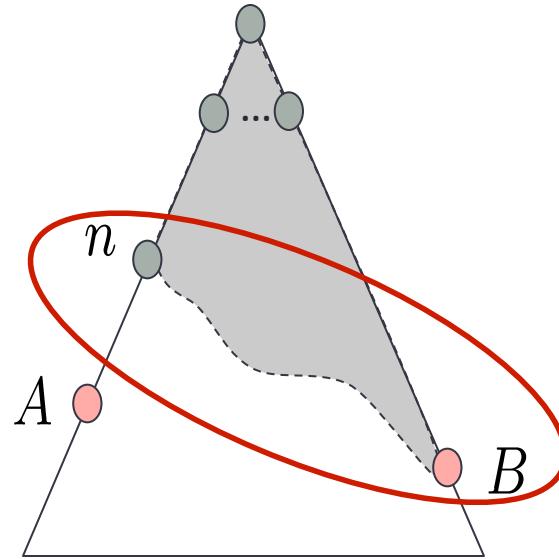
B 次优解

$h = 0$  在目标节点

# A\* 树搜索的最优性

证明：

- 假设 B 在搜索前沿里
- A 的某个祖先节点  $n$  也在搜索前沿里
- 宣称:  $n$  会先于 B 被选择扩展
  1.  $f(n)$  小于或等于  $f(A)$
  2.  $f(A) < f(B)$
  3.  $n$  先于 B 被扩展
- 所有 A 的祖先节点都会在 B 之前被扩展
- A 先于 B 被扩展
- A\* 搜索是最优的



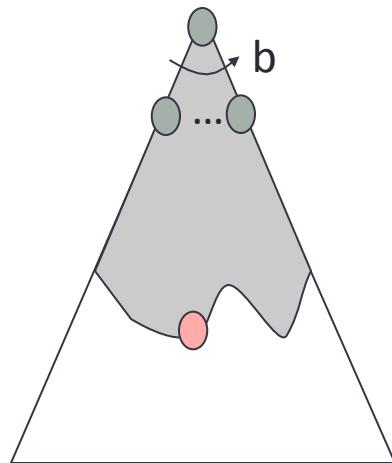
$$f(n) \leq f(A) < f(B)$$

---

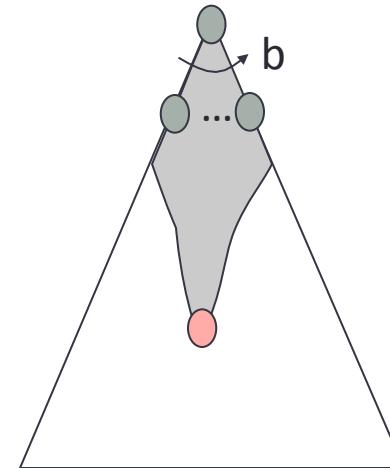
# A\* 搜索的属性

# A\*搜索的属性

成本统一搜索

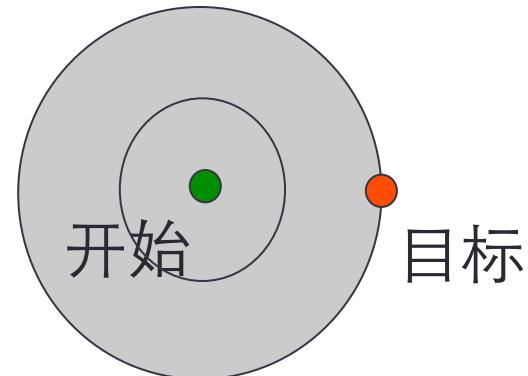


A\*搜索

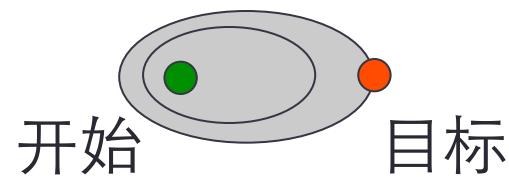


# 成本统一搜索 vs A\* 搜索轮廓

- 基于成本的统一搜索在各个方向上均匀探索



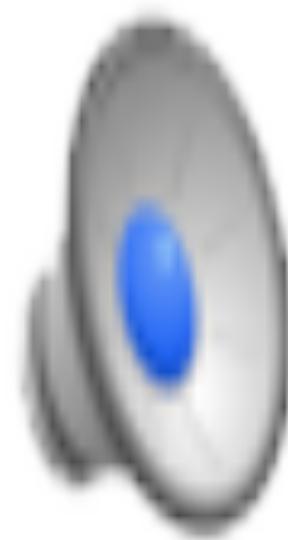
- A\* 在朝向目标的方向上进行探索，同时保证解的最优性



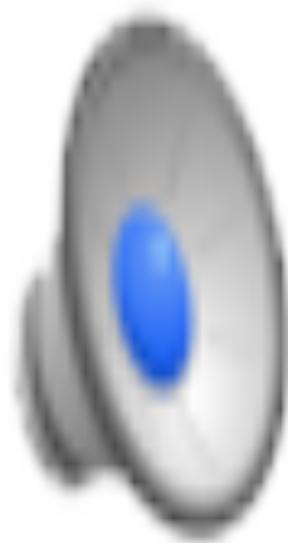
视频展示：搜索轮廓（空迷宫）– 基于成本的统一搜索



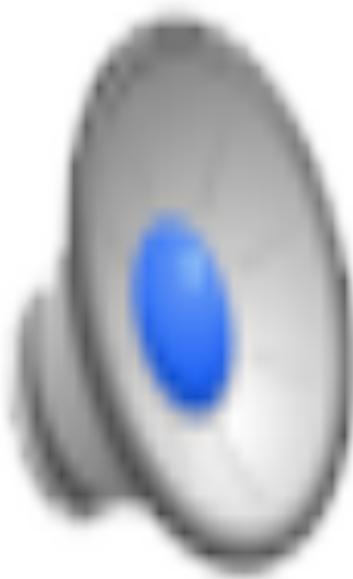
视频展示：搜索轮廓（空迷宫）- 贪婪搜索



视频展示：搜索轮廓（空迷宫）- A\*

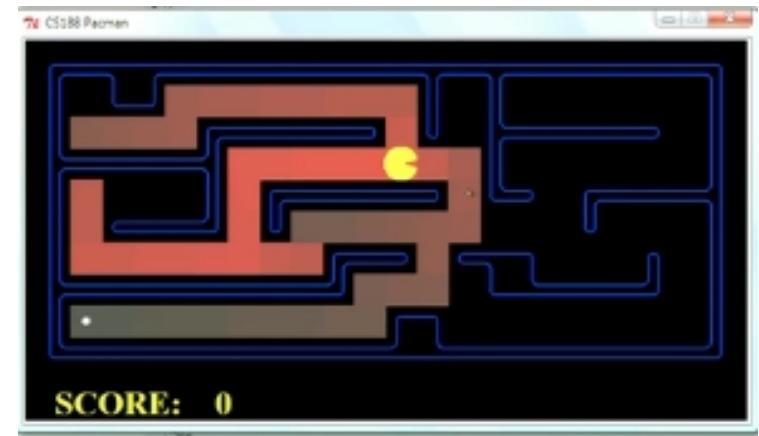


视频展示：搜索轮廓 (Pacman 迷宫) – A\*



# 比较

贪婪搜索

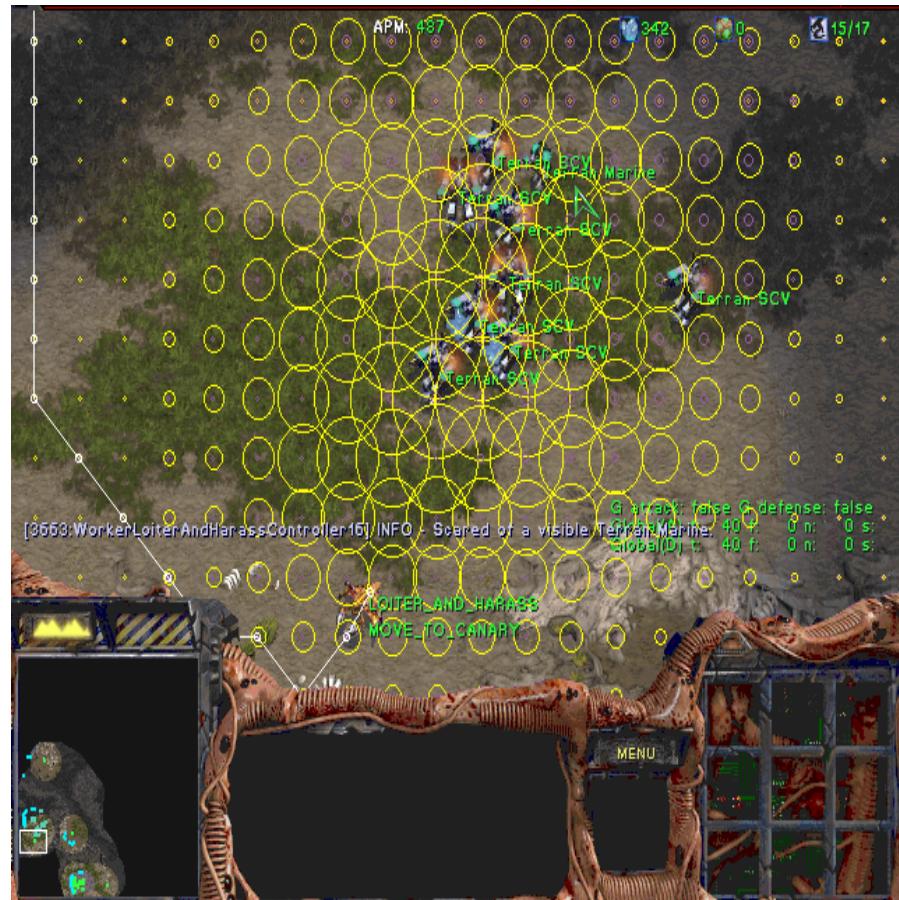


A\*

基于成本的统一搜索

# A\* 应用

- 视频游戏
- 路径搜索问题
- 资源规划问题
- 机器人移动规划
- 语言分析
- 机器翻译
- 语音识别
- ...



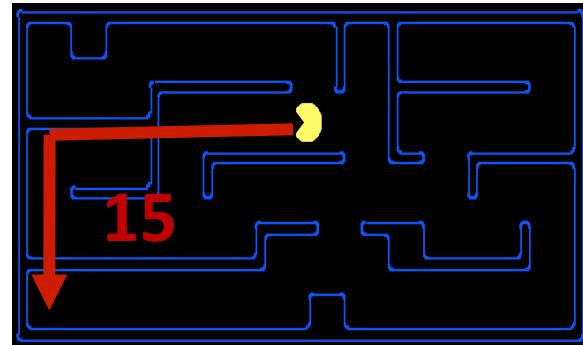
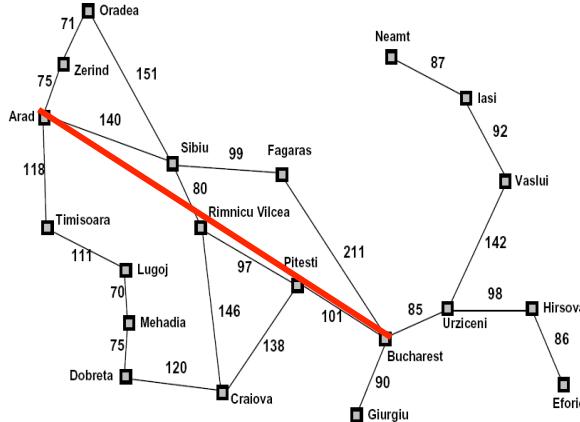
---

## 创建启发式函数

# 创建可接纳的启发式函数

- 在求解很难的搜索问题时，大部分的工作是找到可接纳的启发式函数。
- 可接纳的启发式函数信息，通常是对应的**松弛问题**的解，解除对行动的限制。

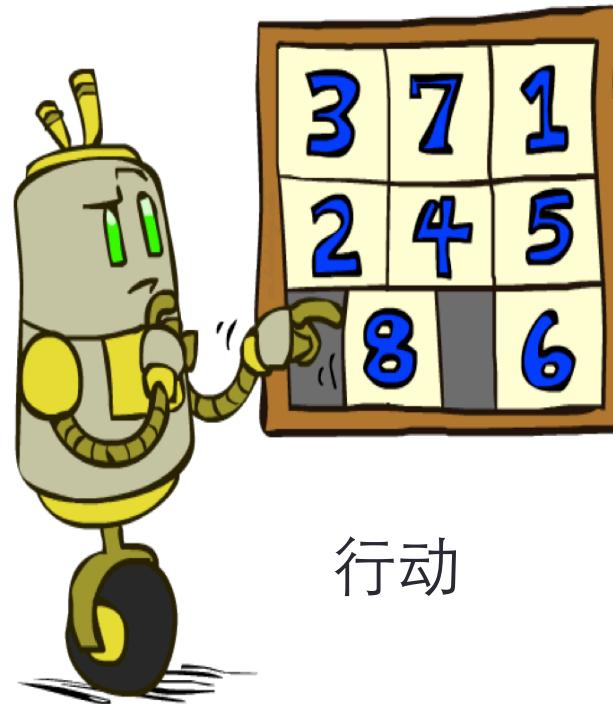
366



# 举例: 8 数字谜题

7	2	4
5		6
8	3	1

开始状态



行动

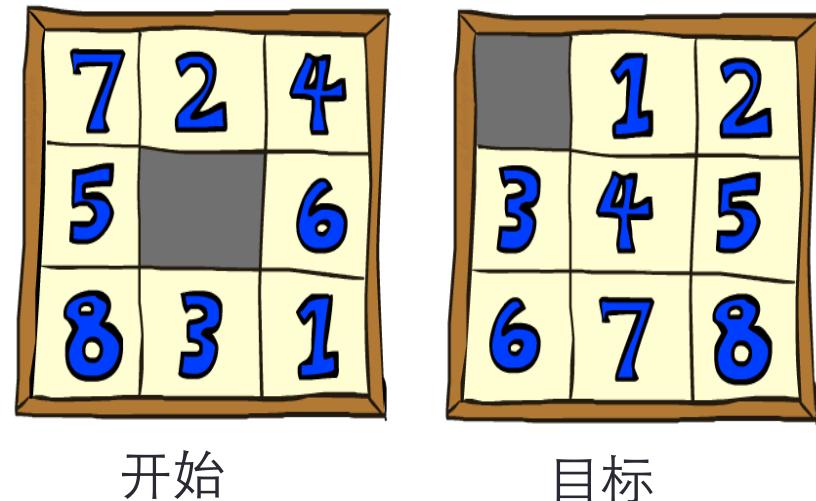
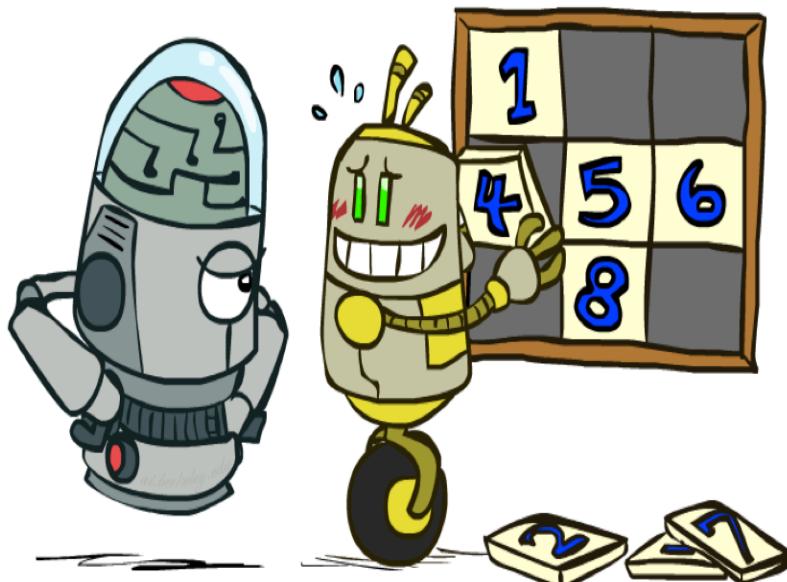
	1	2
3	4	5
6	7	8

目标状态

- 状态?
- 行动?
- 行动成本?

# 8 数字谜题

- 启发式: 错位方块的数量
- 为什么是可接纳性的?
- $h(\text{开始}) = 8$
- 松弛问题的启发信息



平均节点扩展数，当最优解的深度是：			
	...4 步	...8	...12
UCS	112	6,300	$3.6 \times 10^6$
TILES	13	39	227

# 8 数字谜题

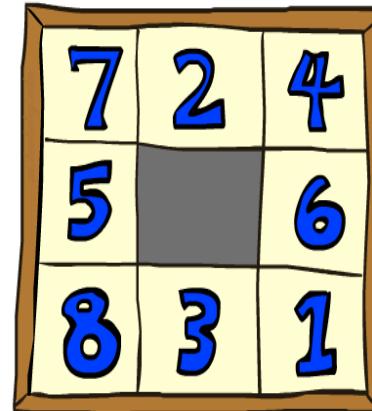
- 如果把条件再松弛一下，任何方块可以在任何时候朝任何方向滑动，无论那里有没有其他方块

- 曼哈顿距离

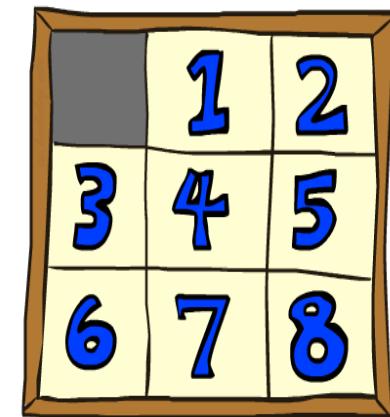
- 可接纳性的？

- $H(\text{开始}) =$

$$3 + 1 + 2 + \dots = 18$$



开始



目标

平均节点扩展数，当最优路径的长度是...

	...4 步	...8 步	...12 步
TILES	13	39	227
MANHATTAN	12	25	73

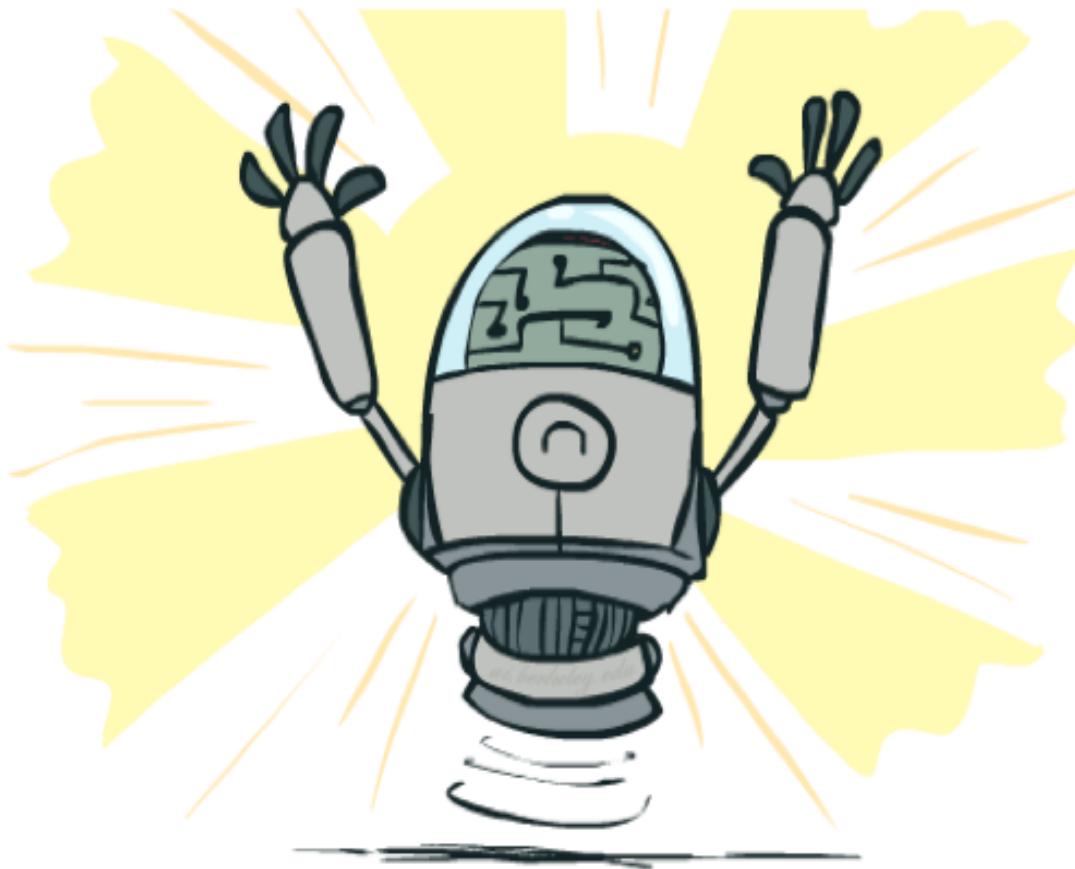
# 组合启发式信息

- 支配优势:  $h_a \geq h_c$  如果:

$$\forall n : h_a(n) \geq h_c(n)$$

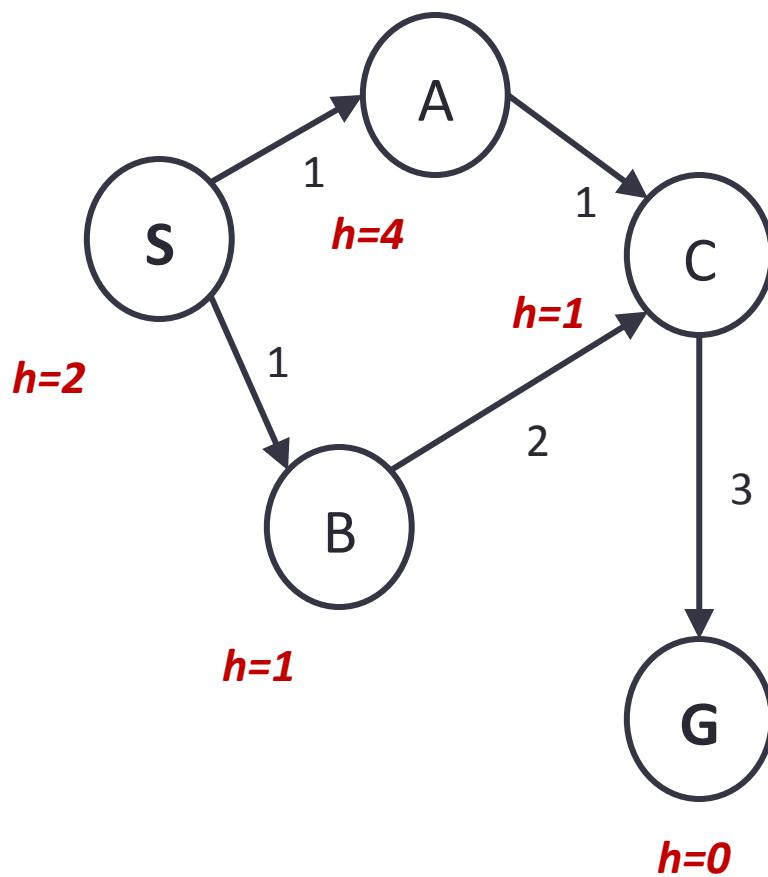
- 一般讲, 越大越好, 只要保持可接纳性。
- $H$  是 0 的话, 比较糟, ( $A^*$  变成什么, 如果  $h=0?$ )。
- 和真实目标成本相同最好, 但很难达到!
- 如果两个启发式函数, 都不支配对方?
  - 形成一个新的, 通过最大式组合:
$$h(n) = \max(h_a(n), h_b(n))$$
  - 这个既是可接纳的, 也是对之前任一个都有支配优势的, 启发式函数。

# A\* 图搜索的 优化性

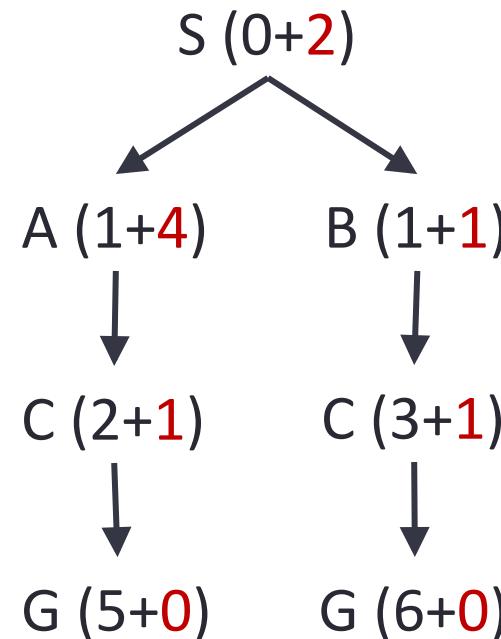


# A\*图搜索走错了?

状态空间图

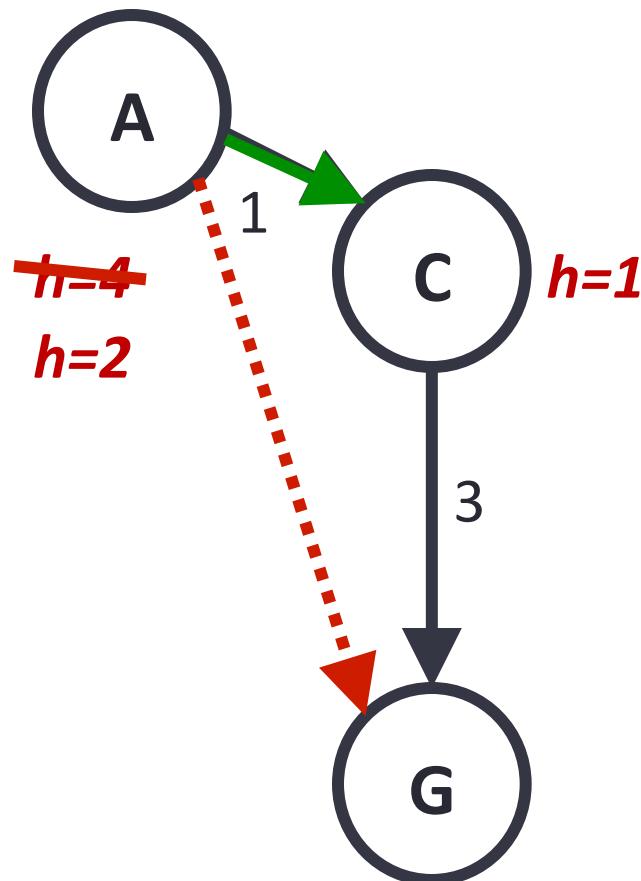


搜索树



A-C → G 不会被扩展，因为 C 已被扩展访问过  
(已存储在访问过的节点集合里)；  
所以错过了最 (优) 短路径。

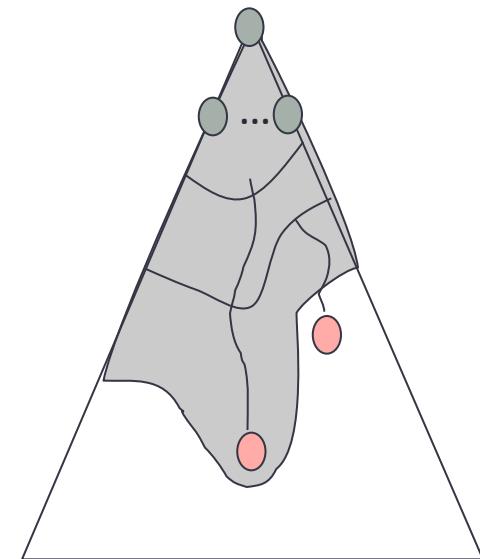
# 启发性函数的一致性



- 主体思想: 估计成本  $\leq$  实际成本
  - 可接纳性: 启发函数估计成本  $\leq$  实际路径成本
$$h(A) \leq \text{从 } A \text{ 到 } G \text{ 的实际路径成本}$$
  - 一致性: 启发函数估计的步骤成本  $\leq$  实际步骤成本
$$h(A) - h(C) \leq \text{cost}(A \text{ 到 } C)$$
- 一致性的结果:
  - f 值 在同一路径搜索中不会减少
$$h(A) \leq \text{cost}(A \text{ 到 } C) + h(C)$$
  - A\* 图搜索是最优的 (找到的解是最优的)

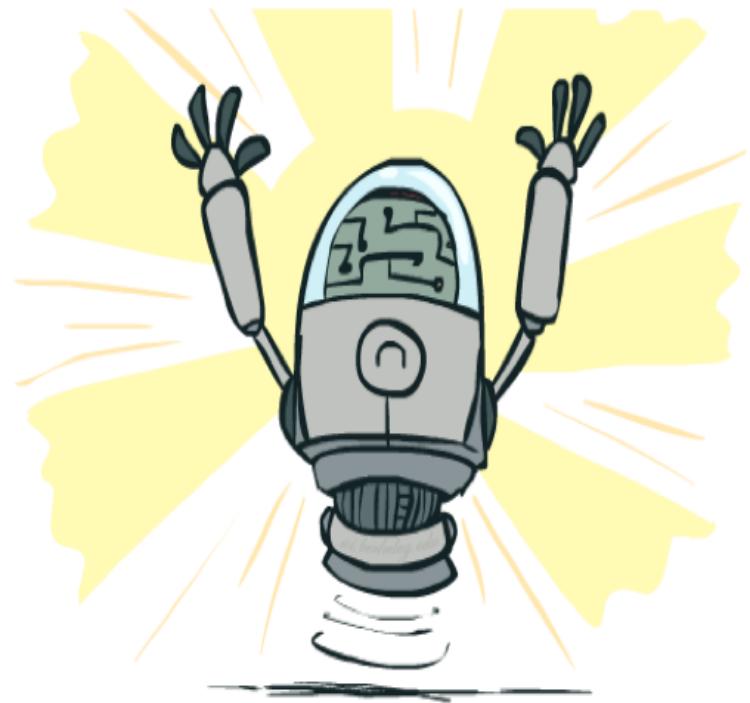
# A\* 图搜索的最优性 (Optimality)

- 如果 A\* 和一个一致性的启发式函数，那么：
  - 事实 1: A\* 扩展节点过程中， $f$  值递增 ( $f$ -轮廓)
  - 事实 2: 对于每个状态  $s$ , 到达  $s$  的最优路径上的节点先于次优路径节点被扩展
  - 结果: A\* 图搜索是最优的



# 最优化

- 树搜索:
  - A\* 最优, 如果启发式函数是可接纳性的
  - 基于成本的统一搜索(UCS) 是一个特例 ( $h = 0$ )
- 图搜索:
  - A\* 最优, 如果启发式函数是一致性的
  - UCS 最优 ( $h = 0$  也是一致性的)
- 一致性 => 可接纳性
- 通常, 从条件松弛问题中得到的启发式信息既具有可接纳性, 又趋向于具有一致性

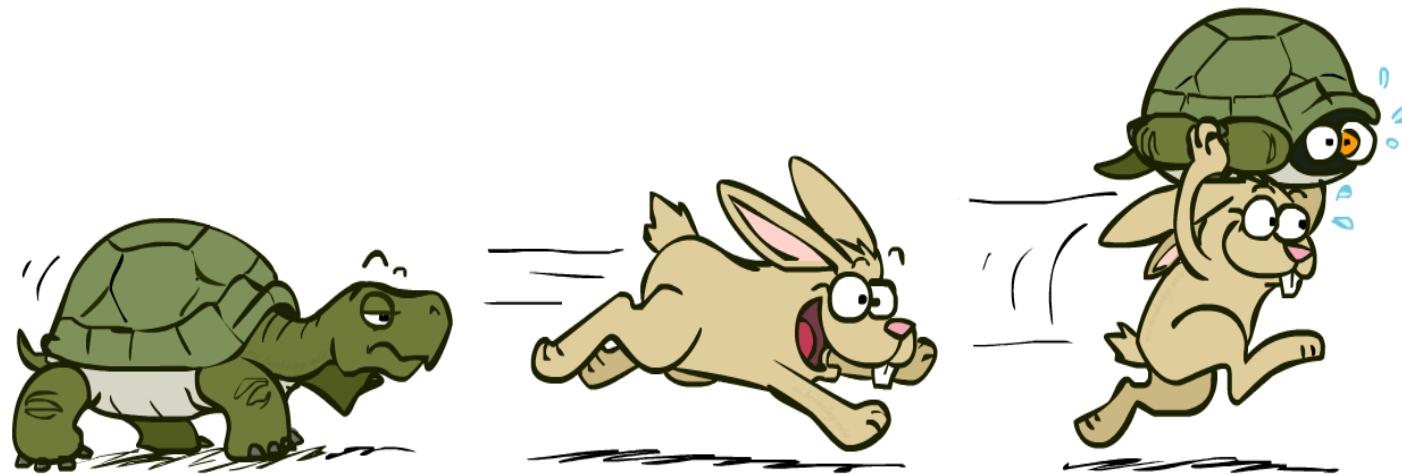


# A\*: 总结



# A\*: 总结

- A\* 既使用了来程路径成本，又使用了前程路径成本的估计
- A\* 是最优的，如果伴随使用可接纳性的和一致性的启发式函数
- 启发式函数的设计是关键：通常运用条件松弛问题的解



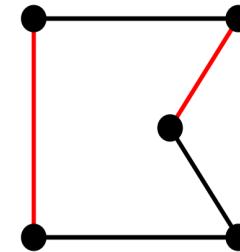
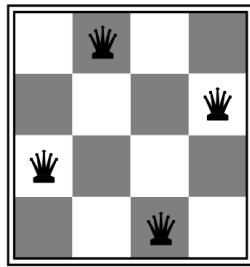
# 人工智能导论：局部搜索 和智能行为体

---

齐琦  
海南大学

# 局部搜索

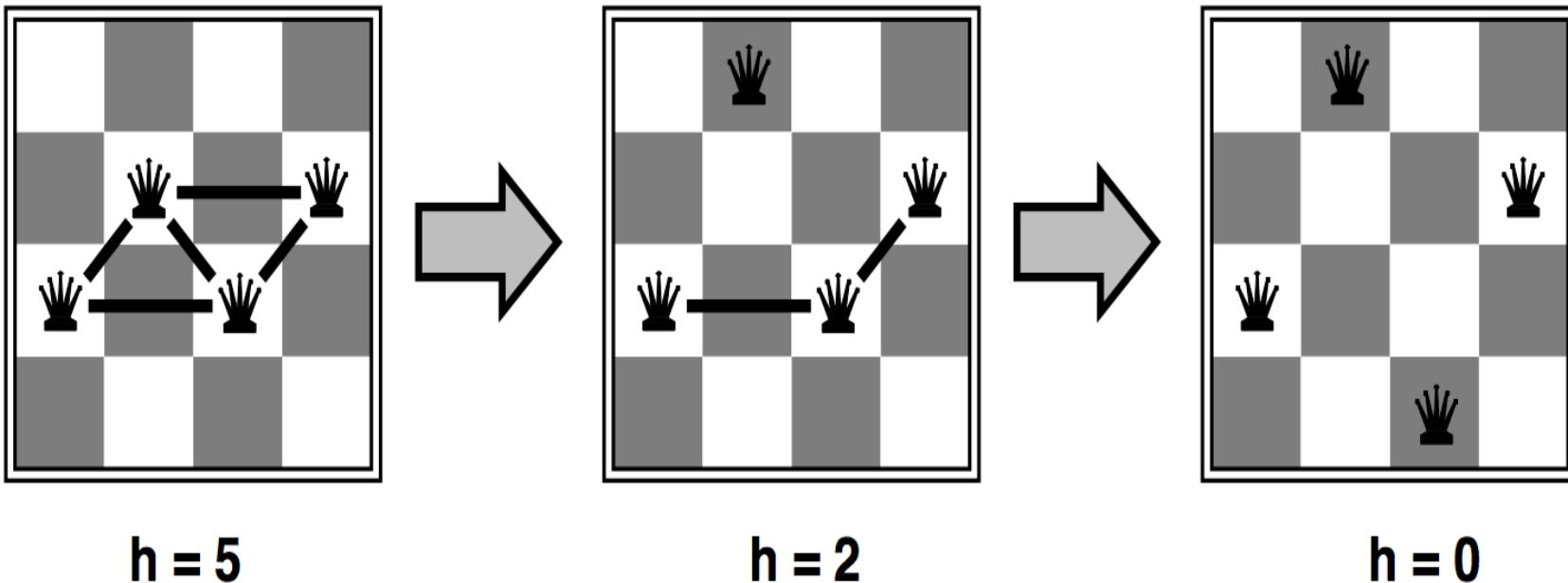
- 许多优化问题, **路径解** 不需要; 目标状态才是 问题的解
- 状态空间 = 完整结构布局的集合
- 找到 **满足约束的结构布局解**, 例如n个皇后问题; 或是找到**最优布局解**, 例如旅行推销商问题



- 在这些情况, 可以使用迭代改进算法; 保持一个单一当前状态, 并尝试改进它。

# $N$ 皇后问题的启发式信息

- 目标布局:  $n$  个皇后在棋盘上不互相冲突(攻击)
- 状态:  $n$  个皇后在棋盘上布局, 一列放一个
- 启发式函数: 相互冲突的皇后对数



# 爬山算法(Hill-climbing)

**function** HILL-CLIMBING(问题) **returns** 一个状态

当前节点  $\leftarrow$  make-node(问题.初始状态)

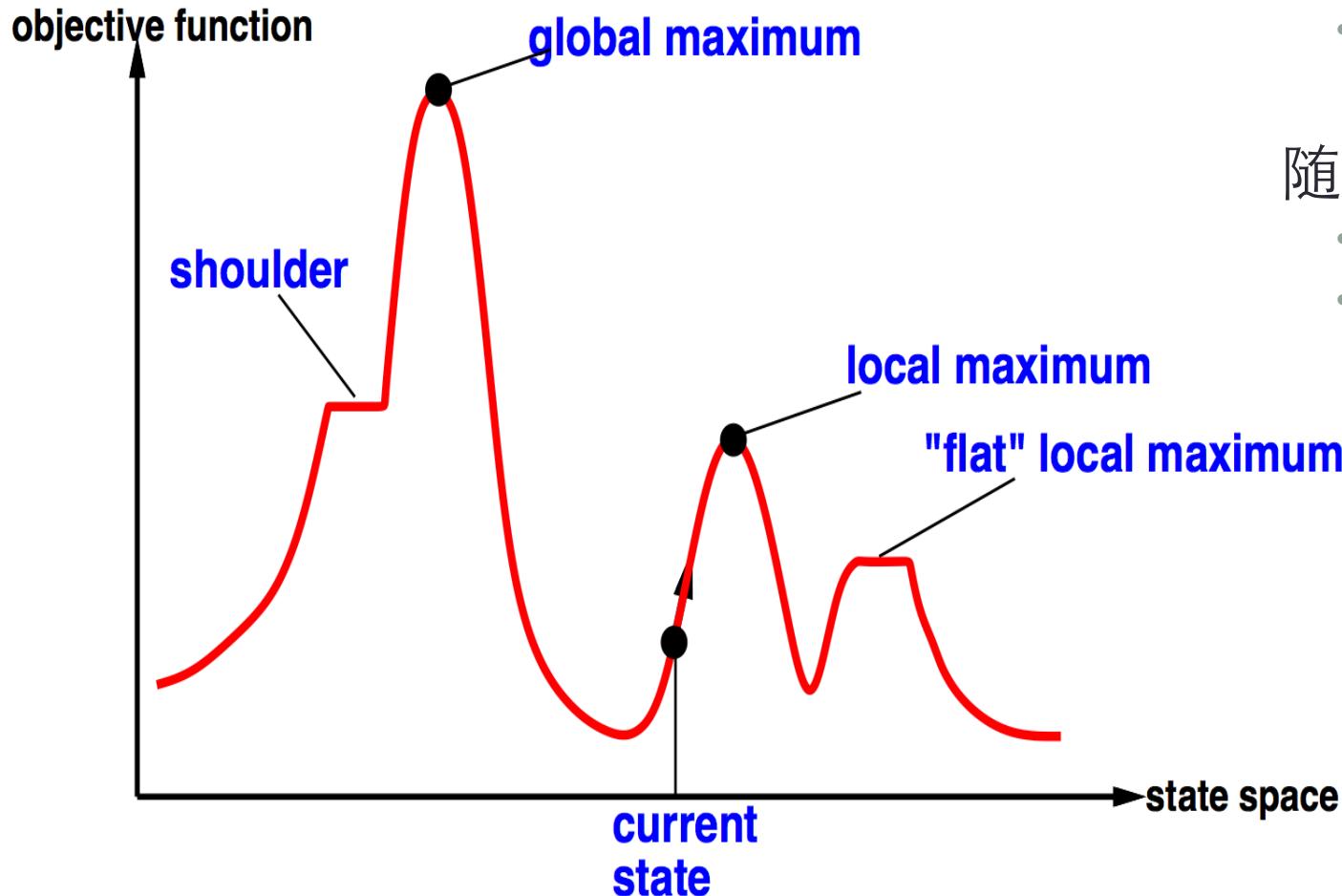
**loop do**

    邻居节点  $\leftarrow$  选一个 当前节点 的后继节点中评估值最大的节点

**if** 邻居节点.value  $\leq$  当前节点.value **then return** 当前节点.state

    当前节点  $\leftarrow$  邻居节点

# 全局和局部最优解



随机开始点

- 找到全局最优

随机水平移动

- 跳出“shoulder”
- 无限循环在“flat local maxima”

# 模拟退火(Simulated annealing)

- 退火过程用来缓慢冷却金属使之达到一个稳定状态
- 基本想法：
  - 允许偶尔的随机移动，依赖于“温度”
  - 高温 => 更多的随机移动，系统可能走出局部最优格局
  - 逐渐降低温度，根据一个冷却的时间调度
- 理论上：存在一个冷却时间调度，使得找到全局最优的可能概率为1。

# 模拟退火(Simulated annealing)算法

**function** SIMULATED-ANNEALING(**问题**,**冷却调度**) **returns** 一个状态

当前节点  $\leftarrow$  make-node(**问题**.initial-state)

**for**  $t = 1$  **to**  $\infty$  **do**

$T \leftarrow$  调度( $t$ )

**if**  $T = 0$  **then return** 当前节点

下一节点  $\leftarrow$  一个随机选择的 当前节点 的后继节点

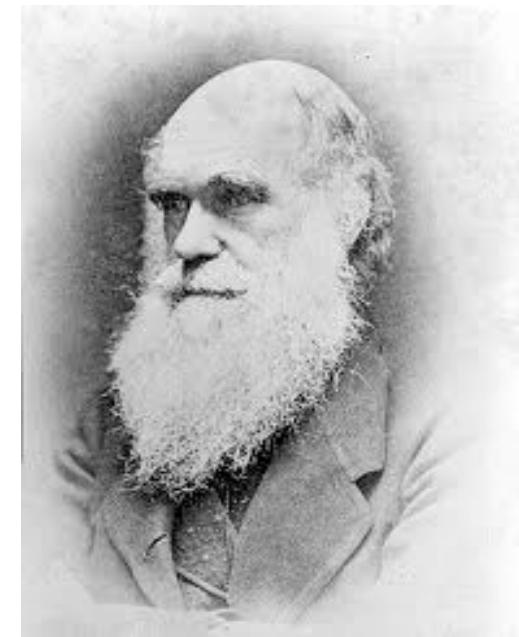
$\Delta E \leftarrow$  下一节点.value - 当前节点.value

**if**  $\Delta E > 0$  **then** 当前节点  $\leftarrow$  下一节点

**else** 当前节点  $\leftarrow$  下一节点, 如果随机概率大于等于  $e^{\Delta E/T}$

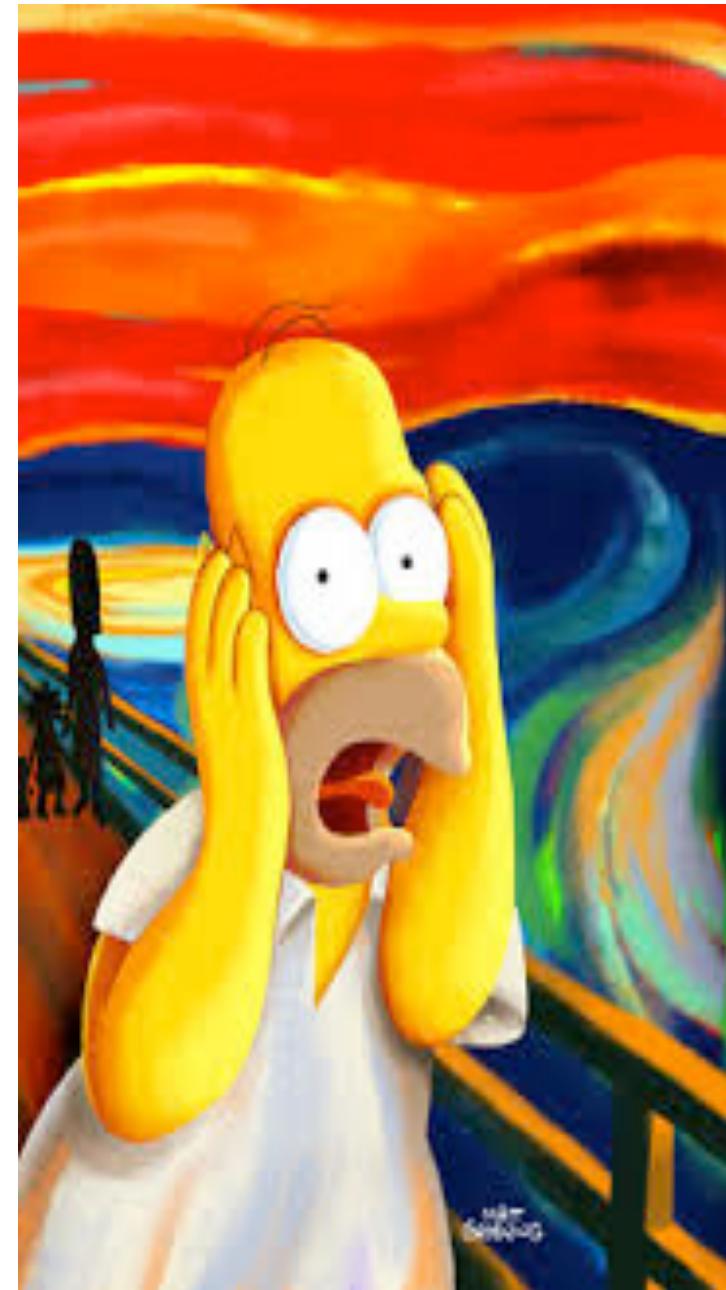
# 局部光束搜索(Local beam search)

- 基本思想:
  - $K$  拷贝某种局部搜索算法, 随机初始化
  - 在每一轮
    - 从  $k$  个当前状态产生所有的后继状态
    - 选出  $k$  个最好的, 赋给当前搜索状态
- 为什么和开始  $K$  个并行局部搜索不一样?
  - 搜索间相互 **交流!**
- 还有什么其他的著名算法采用相同的思想?
  - 进化算法!



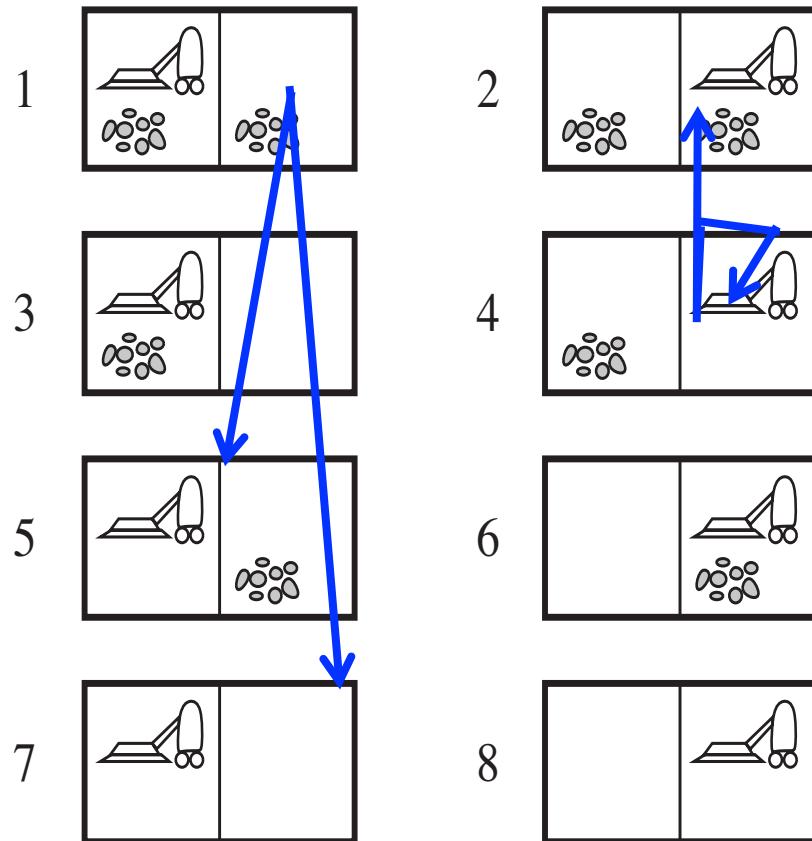
# 在真实世界里搜索

- 非确定性和部分可观察
- 非确定性: 行动有不可预测的影响
  - 问题构建需允许多个结果状态
  - 解则是 **条件化的（依情况而定的）规划**
  - 新算法: 与或搜索 (AND-OR )
  - 解规划中也可能有循环步骤!
- 部分可观察: 感知的不是整个状态



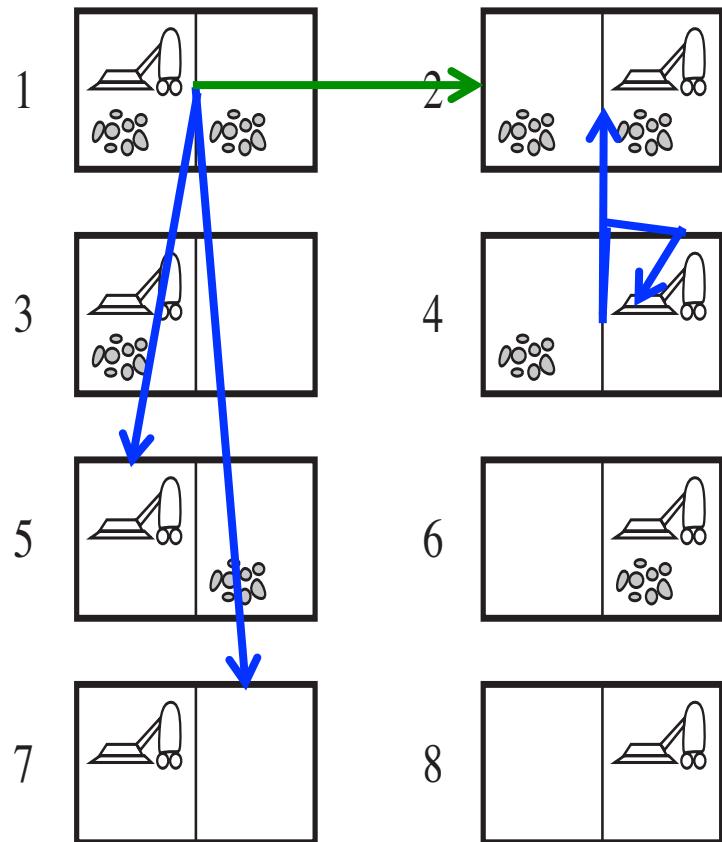
# 会出故障的吸尘器

- 如果格子有灰尘, 吸尘 可能也吸掉邻近格子里的灰尘
  - 例如, 状态 1 可能会到 5 或 7
- 如果格子是干净的, 吸尘 也可能发生故障, 放些灰尘在里面
  - 例如, 状态 4 可能会到 4 或 2



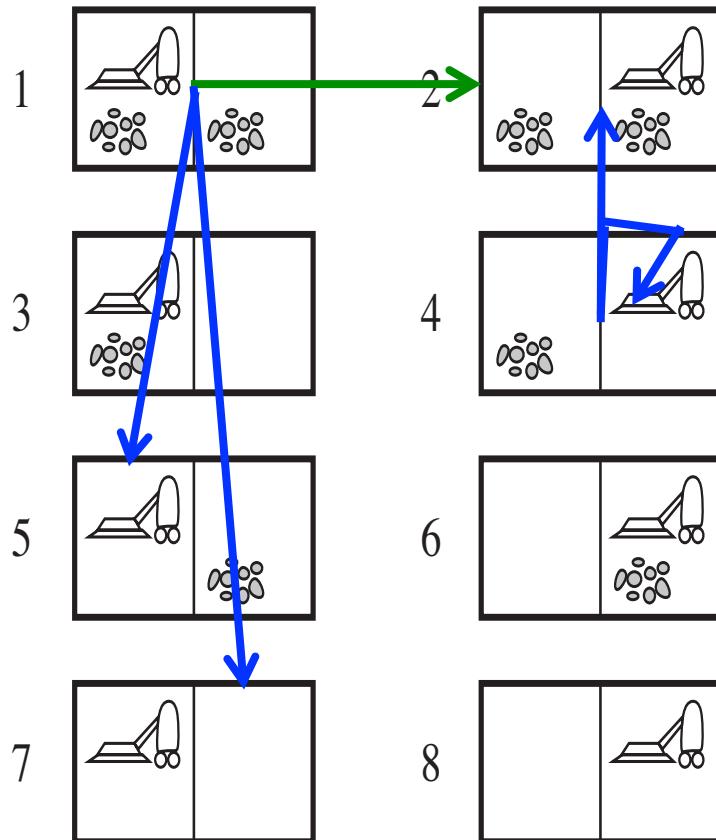
# 问题建立

- **Results(s,a)** 返回一个状态 集合
  - Results(1,吸尘) = {5,7}
  - Results(4,吸尘) = {2,4}
  - Results(1,向右) = {2}
- 其他的都和以前一样



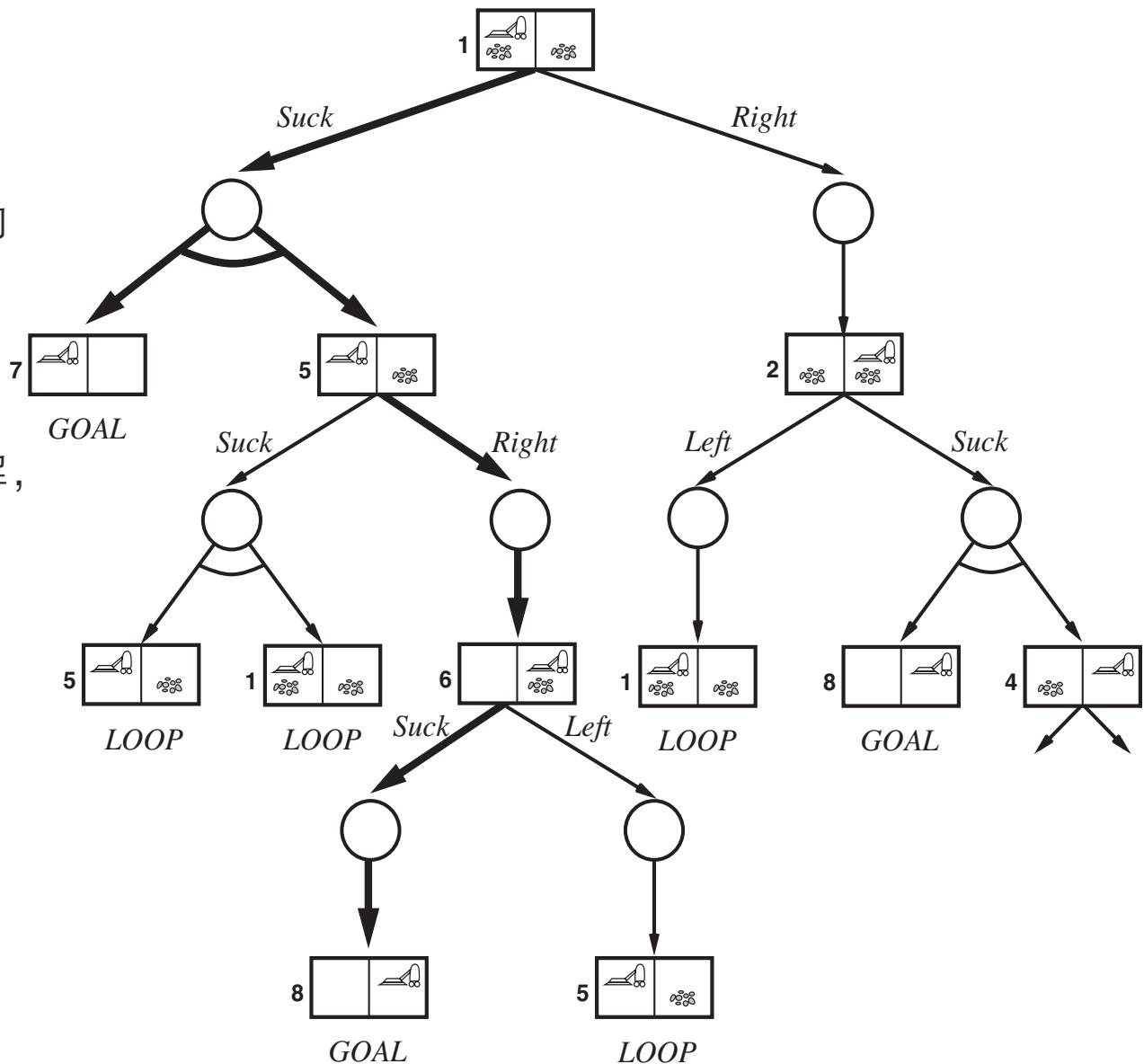
# 条件化的解

- 从状态 1, 行动 [吸尘] 能解决问题吗?
  - 不必然!
- 那么 [吸尘,右移,吸尘]?
  - 不必然!
- [吸尘; **if** 状态=5 **then** [右移,吸尘] **else** []]
- 这是一个 **依情况而定的解**  
**( 条件性规划 )**
- 如何找到这样的解?



# 与或搜索树

- 或-节点:
  - 智能体选择行动;
  - **至少一个分支** 能得到解即成功
- 与-节点:
  - 自然选择行动的影响;
  - **所有分支** 必须都能被求解, 否则失败
- 两种节点交替排列



# 与或搜索算法 (递归，深度优先)

**Function** 与或-图-搜索(问题) **returns** 一个条件性规划, 或失败  
或-搜索(问题.初始状态,问题,[])

**Function** 或-搜索(状态,问题,路径) **returns** 一个条件性规划, 或失败

**if** 问题.goal-test(状态) **then return** 空规划

**if** 状态 曾出现在 路径 **then return** 失败

**for each** 行动 **in** 问题.actions(状态) **do**

规划  $\leftarrow$  与-搜索(results(状态,行动),问题,[状态 | 路径])

**if** 规划  $\neq$  失败 **then return** [行动 | 规划]

**return** 失败

# 与或搜索，继续

**Function** 与-搜索(状态集,问题,路径) **returns** 一个条件规划, 或失败

**for each**  $s_i$  **in** 状态集 **do**

$plan_i \leftarrow$  或搜索( $s_i$ ,问题,路径)

**if**  $plan_i =$  失败 **then return** 失败

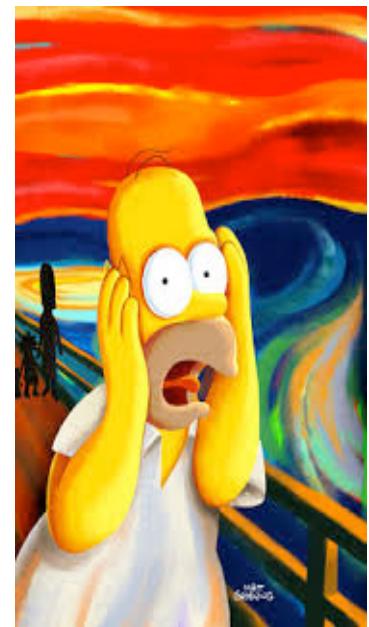
**return** [**if**  $s_1$  **then**  $plan_1$  **else if**  $s_2$  **then**  $plan_2$  **else** ... **if**  $s_{n-1}$  **then**  $plan_{n-1}$  **else**  $plan_n$ ]

# 部分可观察环境

- 信念状态
  - 智能体可能会处在的状态集合
- 搜索问题需在信念空间里搜索；并添加观察模型
- 遵循通常的智能行为体设计过程

# 总结

- 非确定性（行动导致的） 要求解是条件化的规划
  - 通过与或搜索寻找解
- 无感知问题 的解是通常的行动规划
  - 通过在信念状态空间中去寻找解
- 普通的部分可观察环境 导致 感知上的非确定性
  - 在信念状态空间里进行与或搜索



# 人工智能导论： 对抗性的搜索

---

# 计算机游戏/比赛的当前水平

- 国际跳棋
  - 1950年，第一个计算机跳棋程序
  - 1994年，计算机击败人类冠军
  - 2007年，游戏搜索问题被解决。总共有39万亿个终局状态
- 国际象棋
  - 1945-1960年，计算机国际象棋程序
  - 1997年，象棋机器深蓝击败人类冠军
- 围棋
  - 1968年，计算机围棋程序出现
  - 2005-2014年，蒙特卡罗树搜索提高了程序性能；当前水平能击败高水平的业余选手，和部分职业选手
- 吃豆子（Pacman）

# 游戏类型

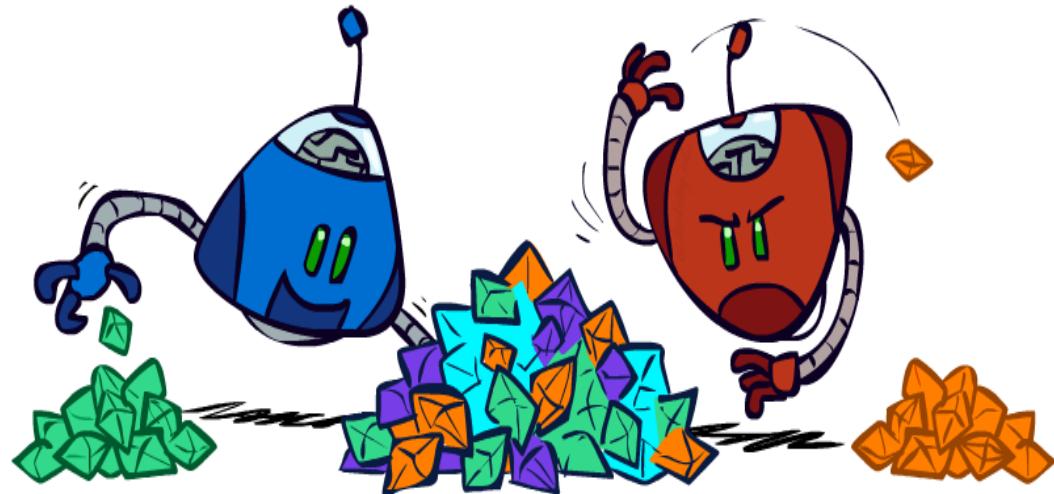
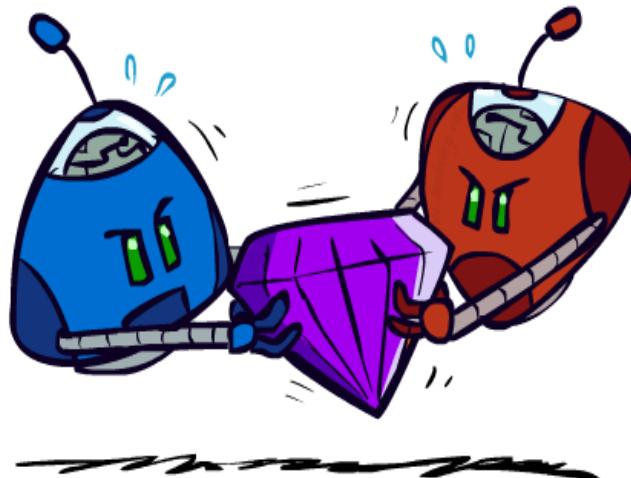
- 描述角度
  - 环境变换确定的，或不确定的？
  - 一个，两个，或多个玩家？
  - 轮流的，或是即时的？
  - 零和的？
  - 信息完全可观察的？
- 算法的目的是计算一个依情况而定的行动计划（策略），为每一个可能的事件推荐一个相应的行动。



# 人工智能所研究的游戏比赛 (games)

- 标准的游戏是，确定的，全局可观察的，轮流行动的，两人的，零和的。
- 游戏问题的数学模型建立：
  - 初始状态:  $s_0$
  - 玩家:  $\text{Player}(s)$  显示在当前状态轮到哪一个玩家行动
  - 行动:  $\text{Actions}(s)$  当前轮次的玩家可能的移动
  - 状态转换模型:  $\text{Result}(s,a)$
  - 终局状态检测:  $\text{Terminal-Test}(s)$  是否是终局
  - 终局得分:  $\text{Utility}(s,p)$  玩家  $p$  的得分
    - 或只用  $\text{Utility}(s)$  代表游戏一开始时最先移动的玩家的得分

# 零和游戏



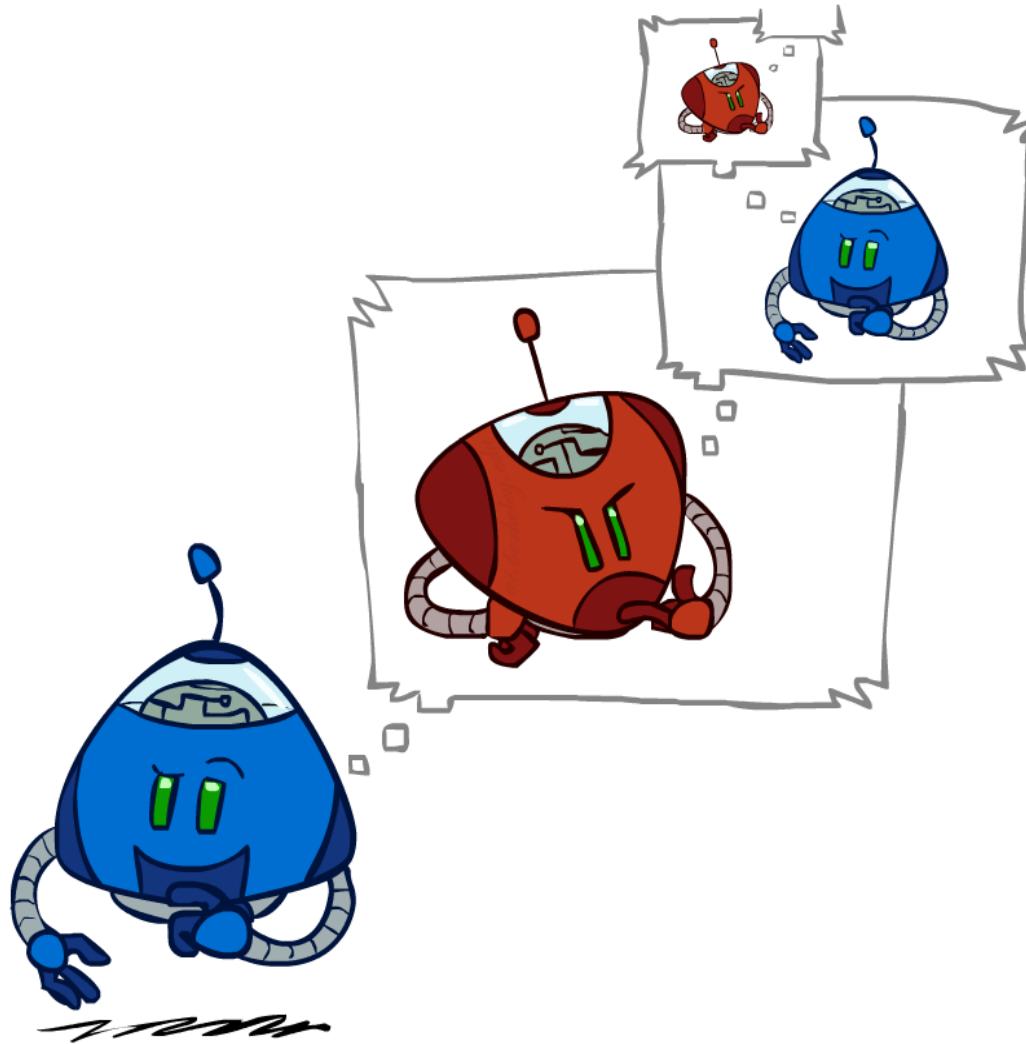
- 零和游戏

- 智能体竞争实现相反的利益
- 一方 **最大化**这个利益, 另一方 **最小化**它

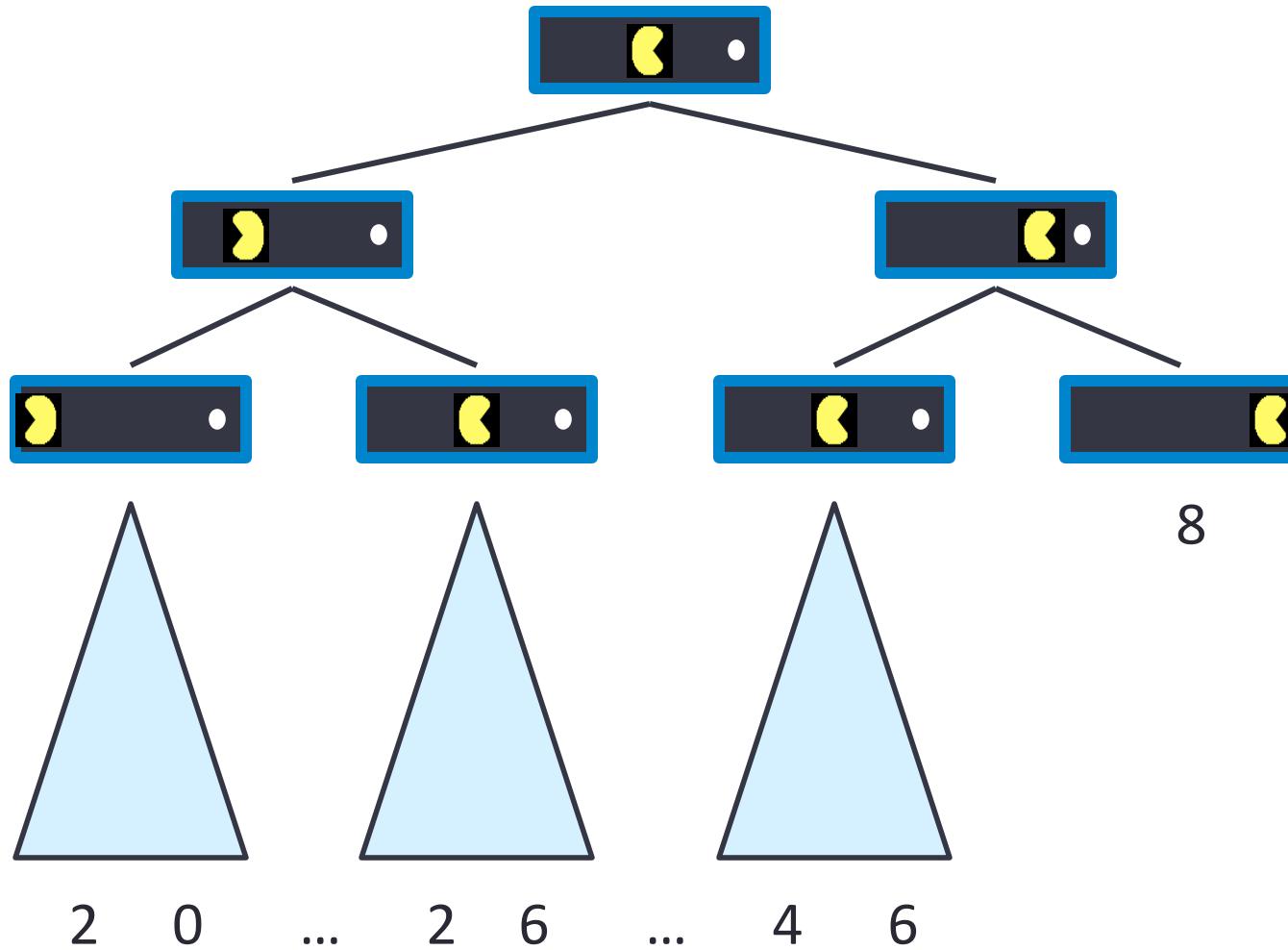
- 通常游戏

- 智能体有 **独立的** 利益
- 合作, 竞争, 联盟等相互关系, 都有可能

# 对抗性搜索

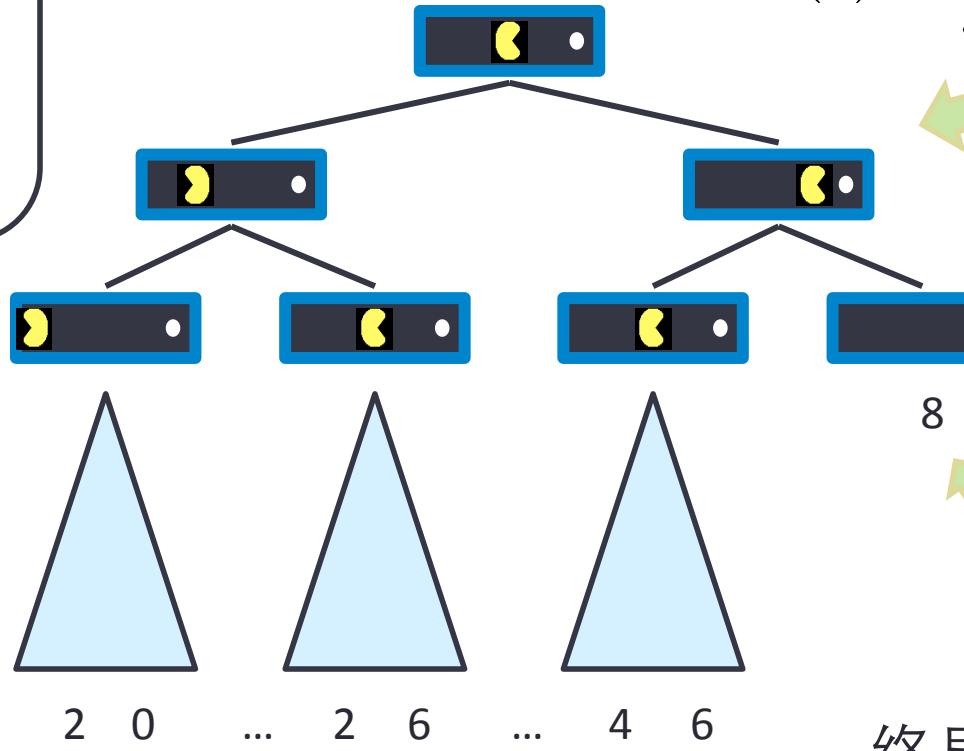


# 单一智能体搜索树



# 状态值

一个状态的  
值: 从这个状  
态往下可能  
达到的最大  
利益值

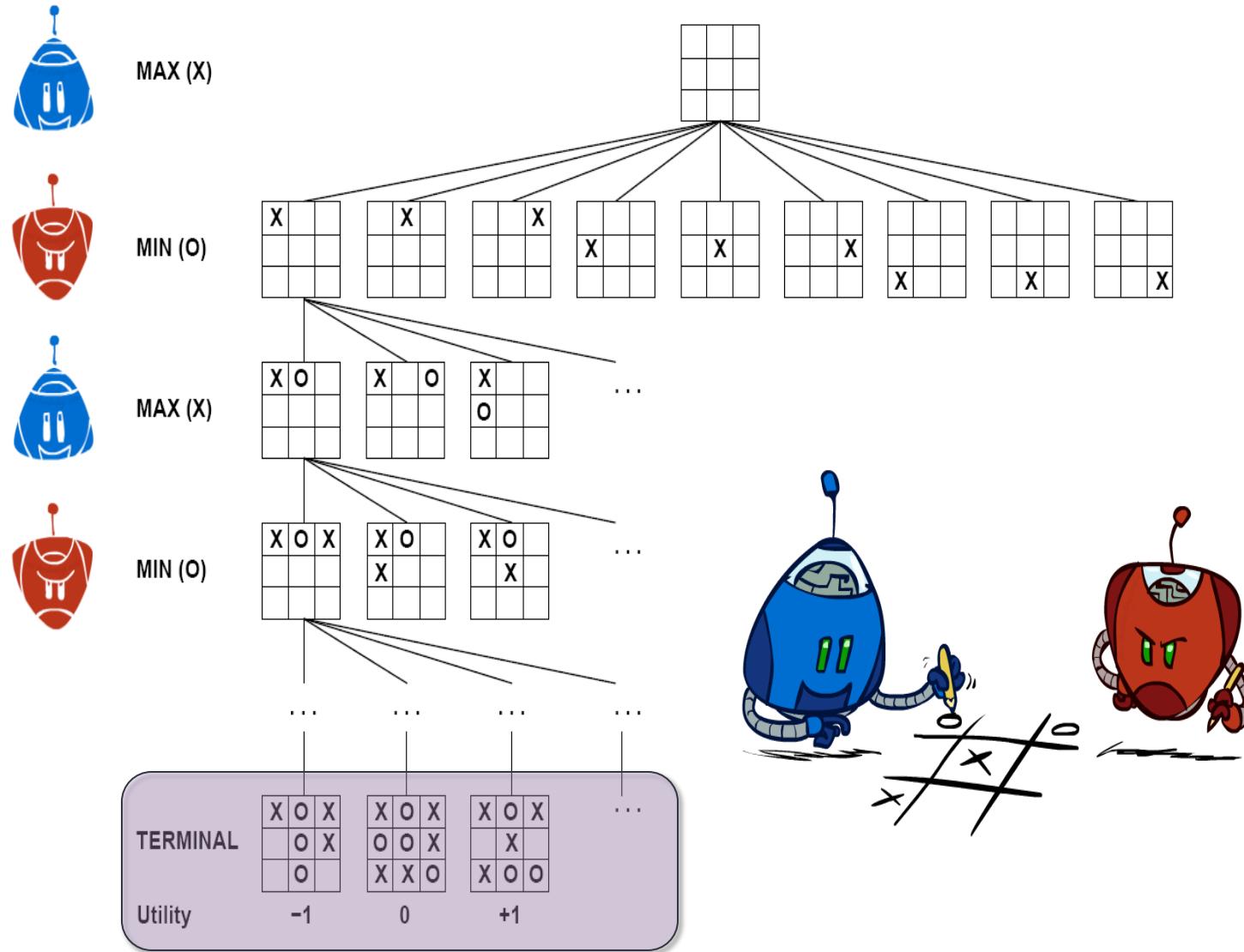


中间状态:

$$V(s) = \max_{s' \in \text{children}(s)} V(s')$$

终局状态的值已知

# 三连棋游戏搜索树



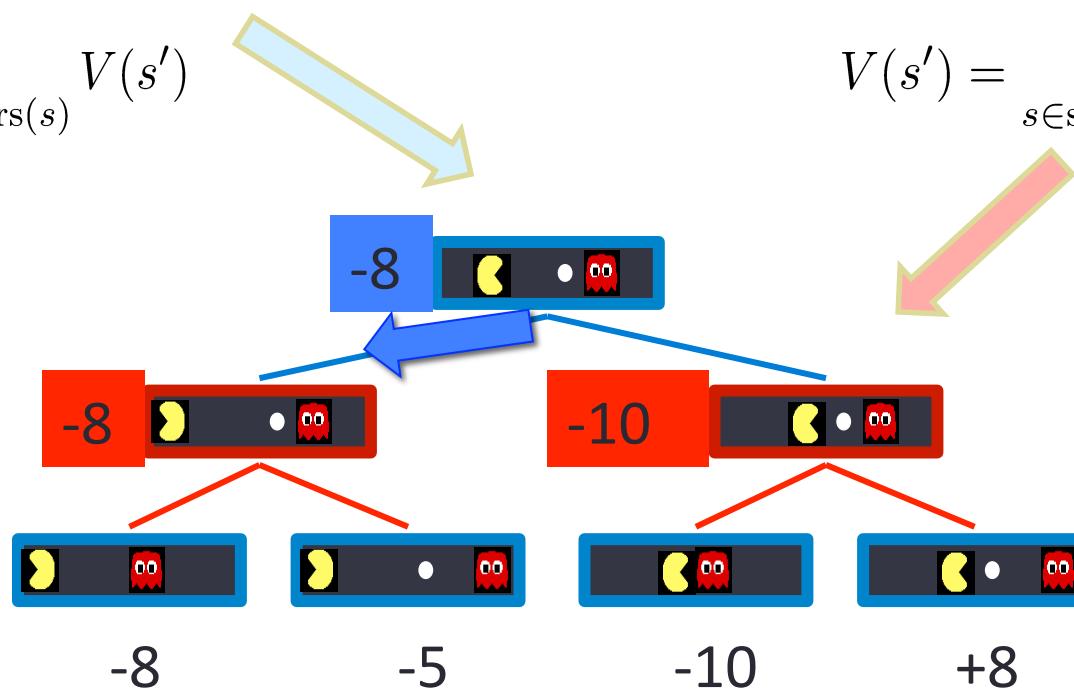
# 最小最大值 (Minimax values)

MAX 节点: 智能体控制下的状态

$$V(s) = \max_{s' \in \text{successors}(s)} V(s')$$

MIN 节点: 对手控制下的节点

$$V(s') = \min_{s \in \text{successors}(s')} V(s)$$



终局状态的值是已知的

# 最小最大算法(Minimax) 实现

深度优先搜索

Function 最小最大-决策( $s$ ) returns 一个行动

return 行动  $a$  in  $\text{Actions}(s)$  , 它能导致最大的  
最小-值(Result( $s,a$ ))的返回值

Function 最大-值( $s$ ) returns 一个值  
If 终局-检测( $s$ ) then return Utility( $s$ )  
初始化  $v = -\infty$   
for each  $a$  in  $\text{Actions}(s)$ :  
     $v = \max(v, \text{最小-值}(\text{Result}(s,a)))$   
return  $v$

Function 最小-值( $s$ ) returns 一个值  
If 终局-检测( $s$ ) then return Utility( $s$ )  
初始化  $v = +\infty$   
for each  $a$  in  $\text{Actions}(s)$ :  
     $v = \min(v, \text{最大-值}(\text{Result}(s,a)))$   
return  $v$

$$V(s) = \max_{s' \in \text{successors}(s)} V(s')$$

$$V(s') = \min_{s \in \text{successors}(s')} V(s)$$

# 另一种实现方法

Function 最小最大-决策( $s$ ) returns 一个行动

return 行动  $a$  in  $\text{Actions}(s)$  , 它能导致最大的  
 $\text{value}(\text{Result}(s,a))$  的返回值



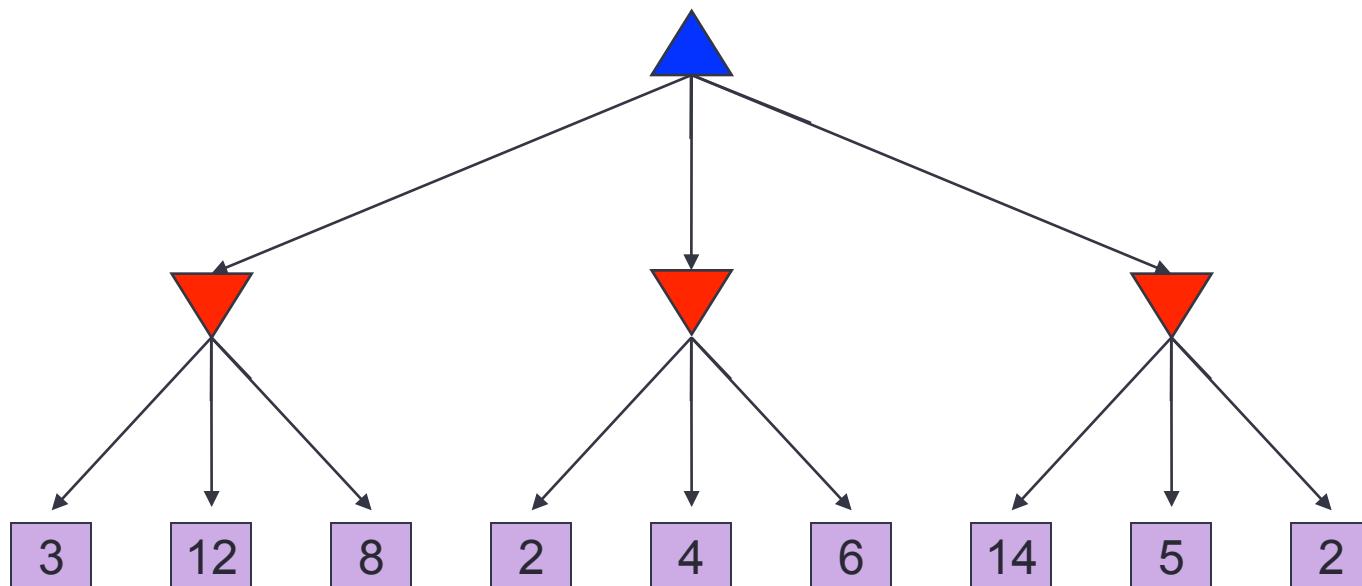
function  $\text{value}(s)$  returns 一个值

If 终局-检测( $s$ ) then return  $\text{Utility}(s)$

if  $\text{Player}(s) = \text{MAX}$  then return  $\max_{a \text{ in } \text{Actions}(s)} \text{value}(\text{Result}(s,a))$

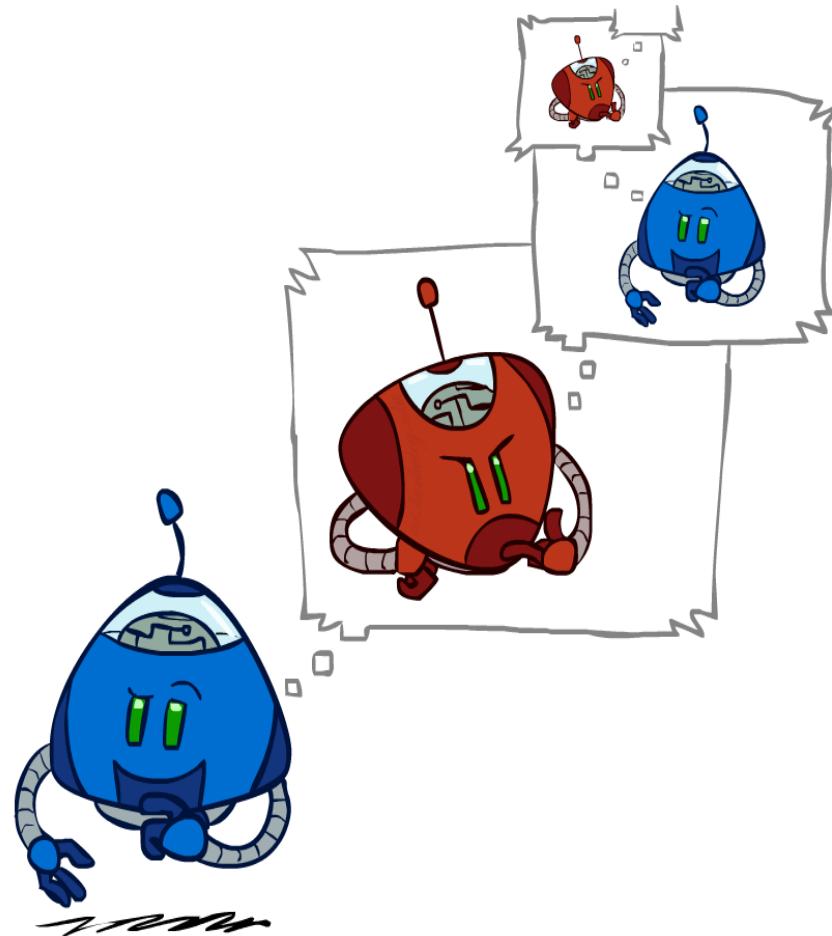
if  $\text{Player}(s) = \text{MIN}$  then return  $\min_{a \text{ in } \text{Actions}(s)} \text{value}(\text{Result}(s,a))$

# 最小最大值 (Minimax) 举例



# Minimax 的效率

- Minimax 的效率?
  - 深度优先穷尽搜索
  - 时间复杂度:  $O(b^m)$
  - 空间复杂度:  $O(bm)$
- 举例: 国际象棋,  $b = 35, m = 100$ 
  - 找到准确解, 不可行
  - 有必要探索整棵树吗?

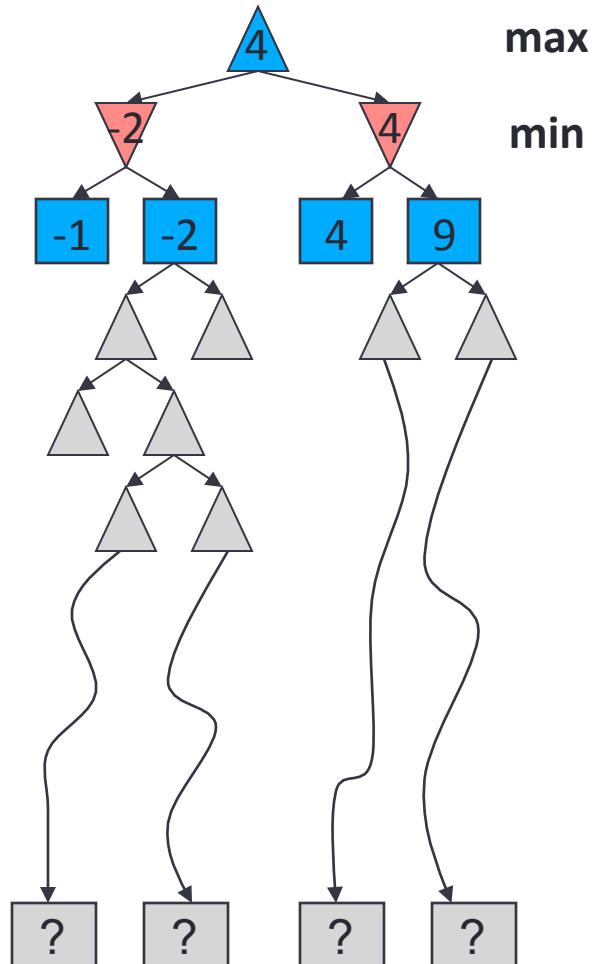


# 资源有限



# 资源局限

- 问题: 现实中, 几乎不能搜索到叶节点!
- 办法之一: 有界搜索加预测
  - 搜索只到预定深度层次
  - 使用 **评估函数** 预测搜索边界节点的值
- 失去了最优解的保证
- 搜索的层次越多结果就越不相同
- 例如:
  - 假设计算时间为100 秒, 每秒能探索1万个节点
  - 所以每步能检查1百万个节点
  - 在国际象棋中,  $b \sim 35$ , 搜索大致能达到搜索树的第4层 – 还是不够好

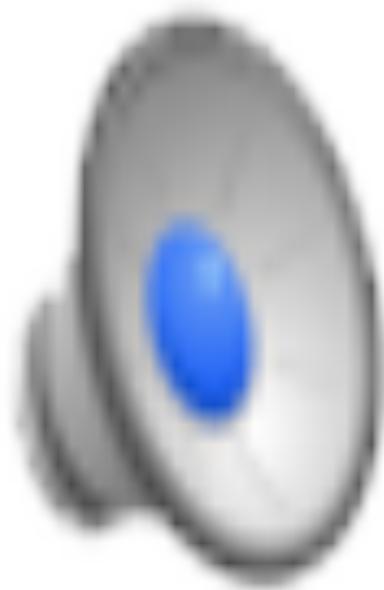


# 探索深度的重要性

- 评估函数总是不完美的  
(不准确的)
- 通常，越深的搜索 => 越好的表现
- 或者说，更深的搜索能够弥补相对不准确的评估函数
- 评估函数设计复杂度和计算复杂度之间的权衡



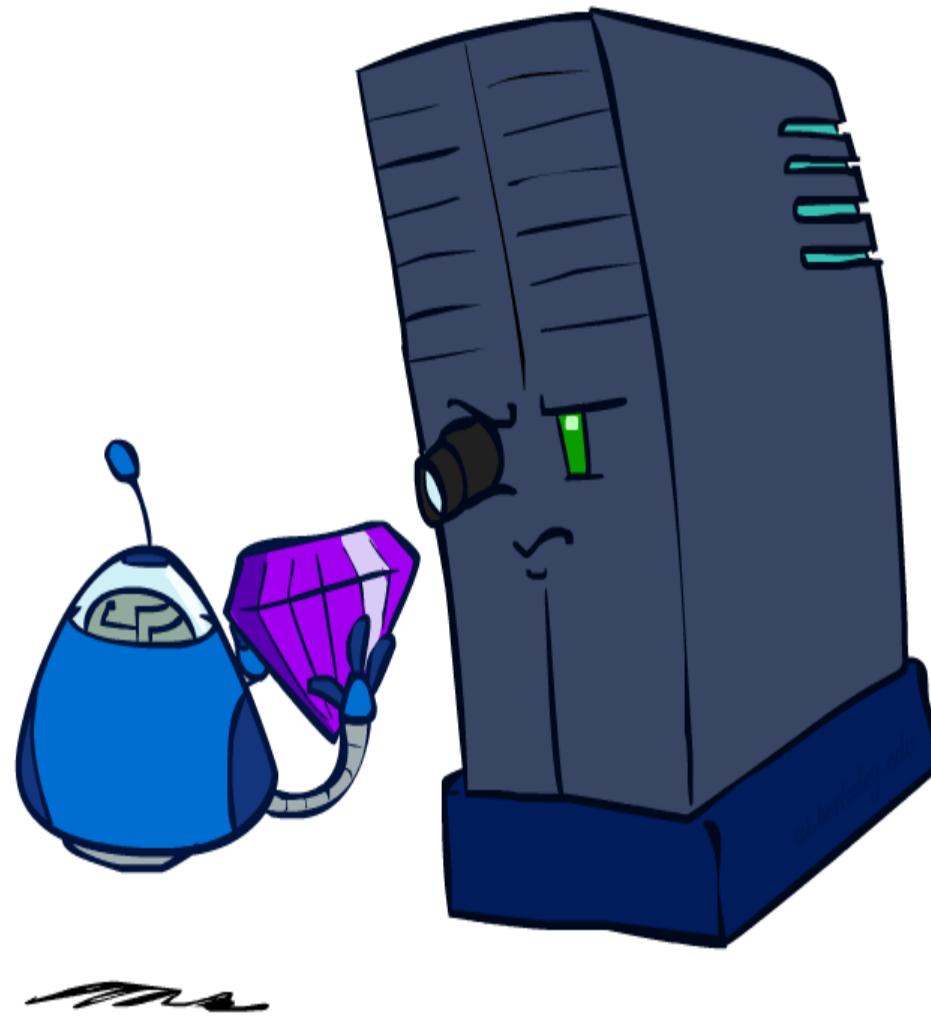
# 视屏演示：有限搜索深度（深度=2）



视屏演示：有限搜索深度（深度=10）

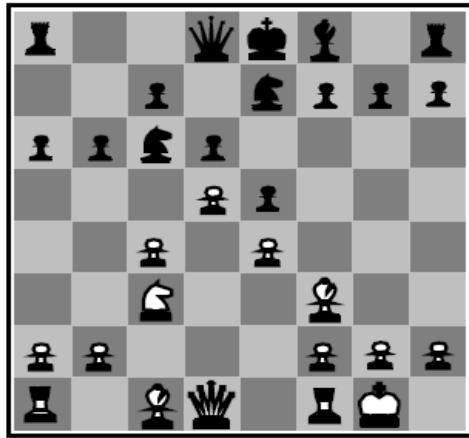


# 评估函数

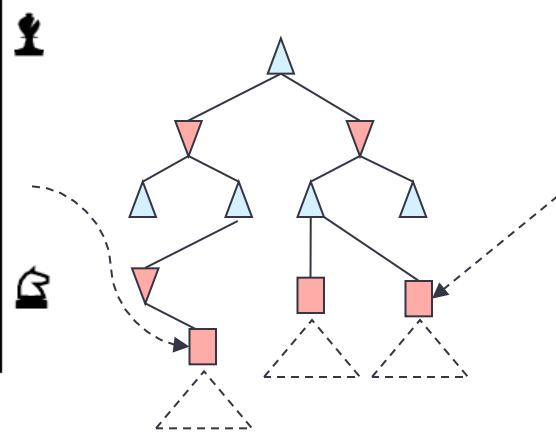


# 评估函数

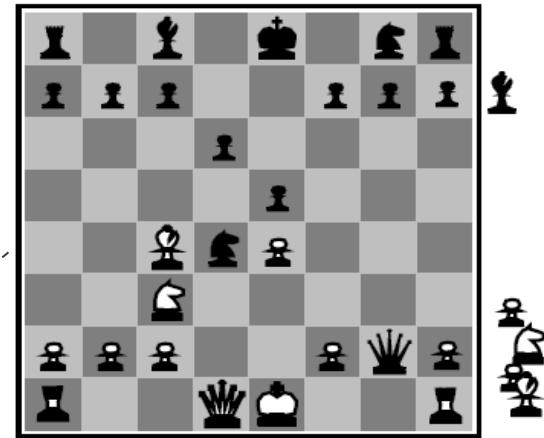
- 用来给非终局状态打分，在一个深度有限搜索中。



Black to move



White slightly better

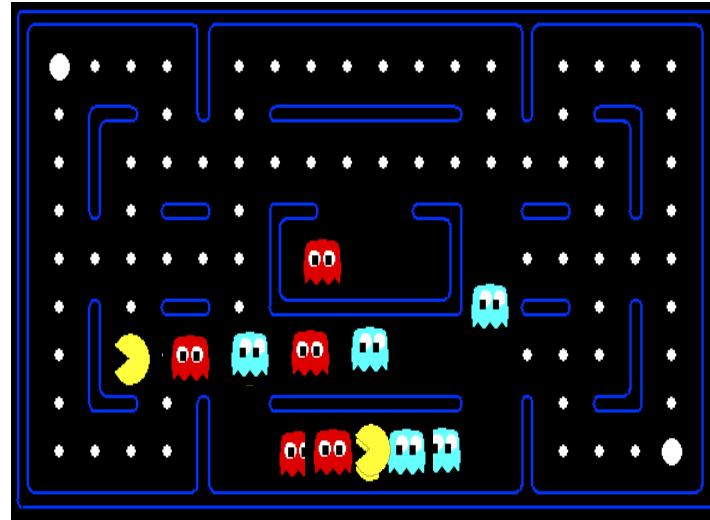


White to move

Black winning

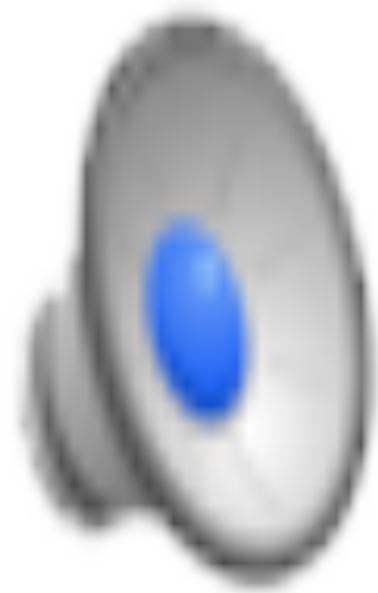
- 理想函数: 返回这个状态的实际最小最大值
- 实践中: 特征函数值的加权线性和:
  - $\text{EVAL}(s) = w_1 f_1(s) + w_2 f_2(s) + \dots + w_n f_n(s)$
  - 例如  $w_1 = 9$ ,  $f_1(s) = (\text{白皇后数量} - \text{黑皇后数量})$ , 等。
- 评估函数应作用在 **沉寂** 状态, 即评估值在其后继状态相对变化不大。

# Pacman游戏状态评估

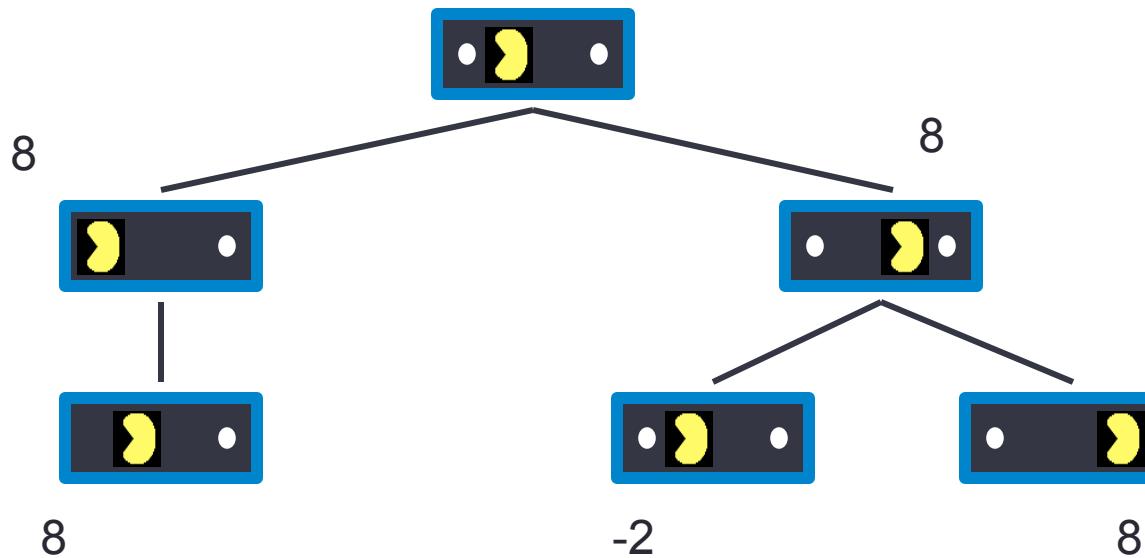


评估函数值应反映所处的局面。

# 评估函数实例演示：犹豫的情况 (深度=2)



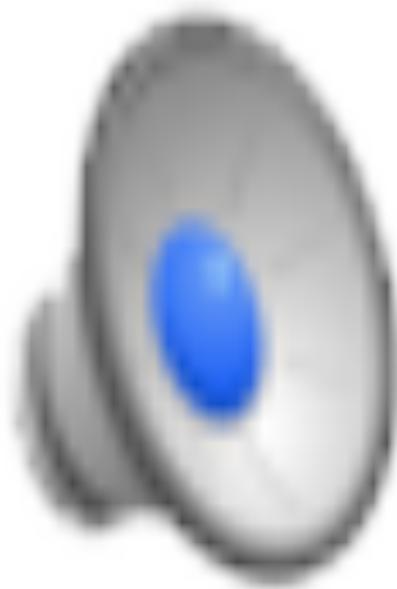
# Pacman 为什么会犹豫？



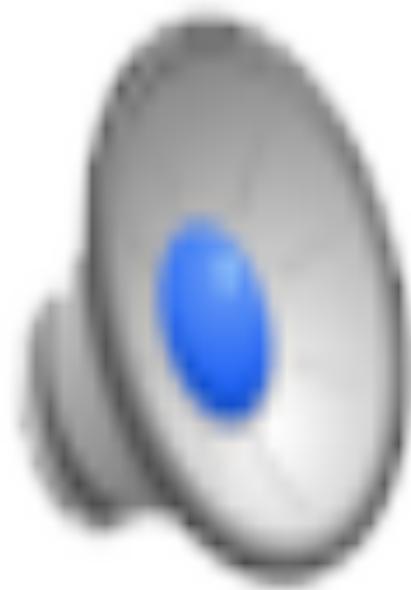
- 智能体重新计划可能面临的处境

- 在当前两步搜索的情况下，向左吃掉豆子的得分和向右移动的是一样的（后面可能会吃掉豆子）
- 得分是用最小最大值方法得到的
- 所以这个时候等待和吃掉豆子的选择结果似乎一样； 它有可能向右移动，然后下一轮再规划时，可能又走回来了！

调整评估函数，解决这个问题（深度=2）



演示：共同的评估函数可自动形成合作

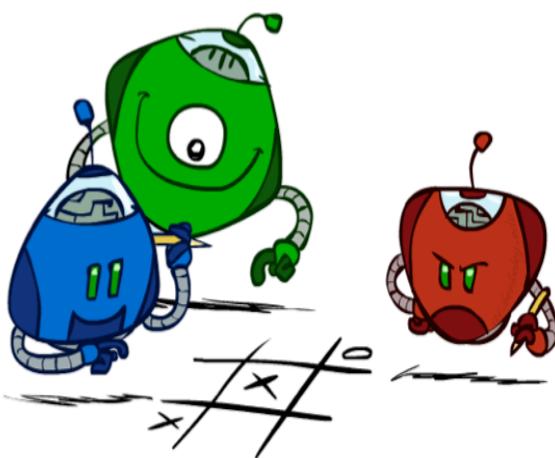
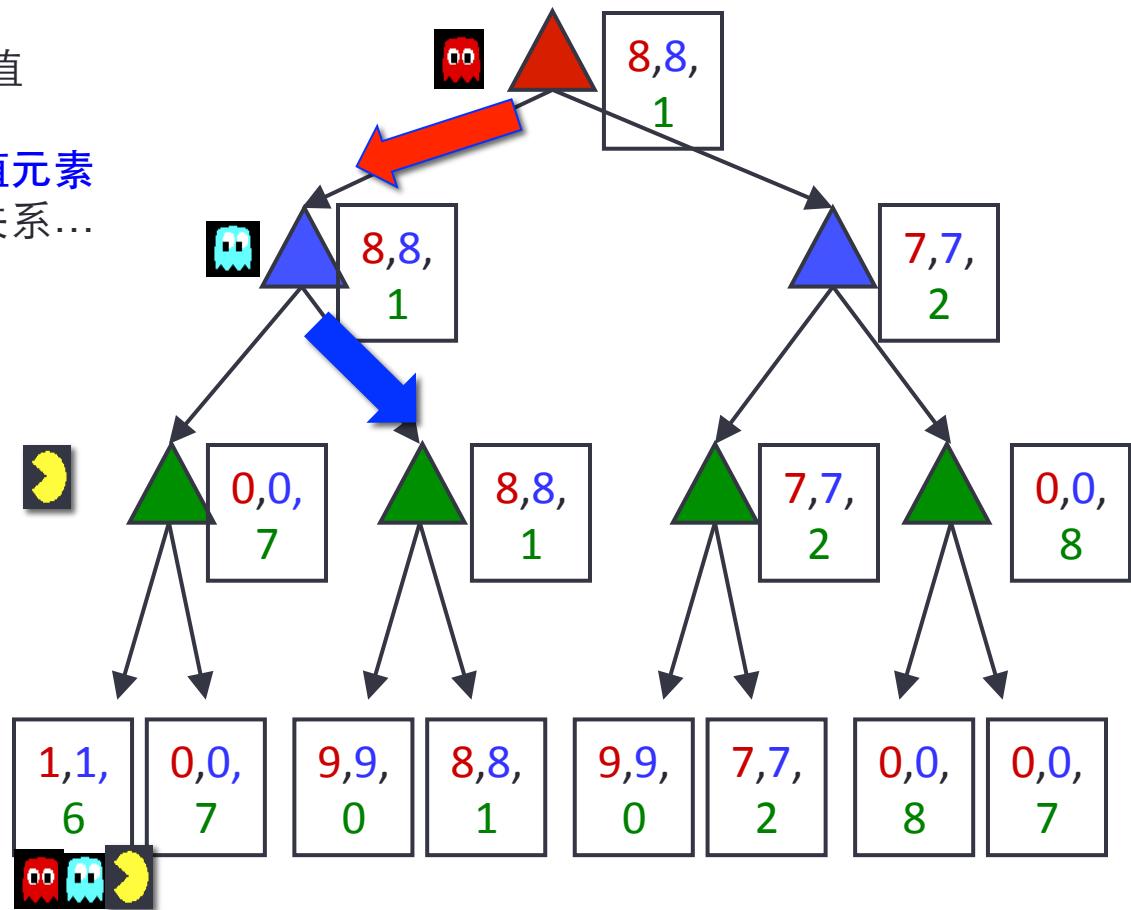


# 普遍化的最小最大值法 (minimax)

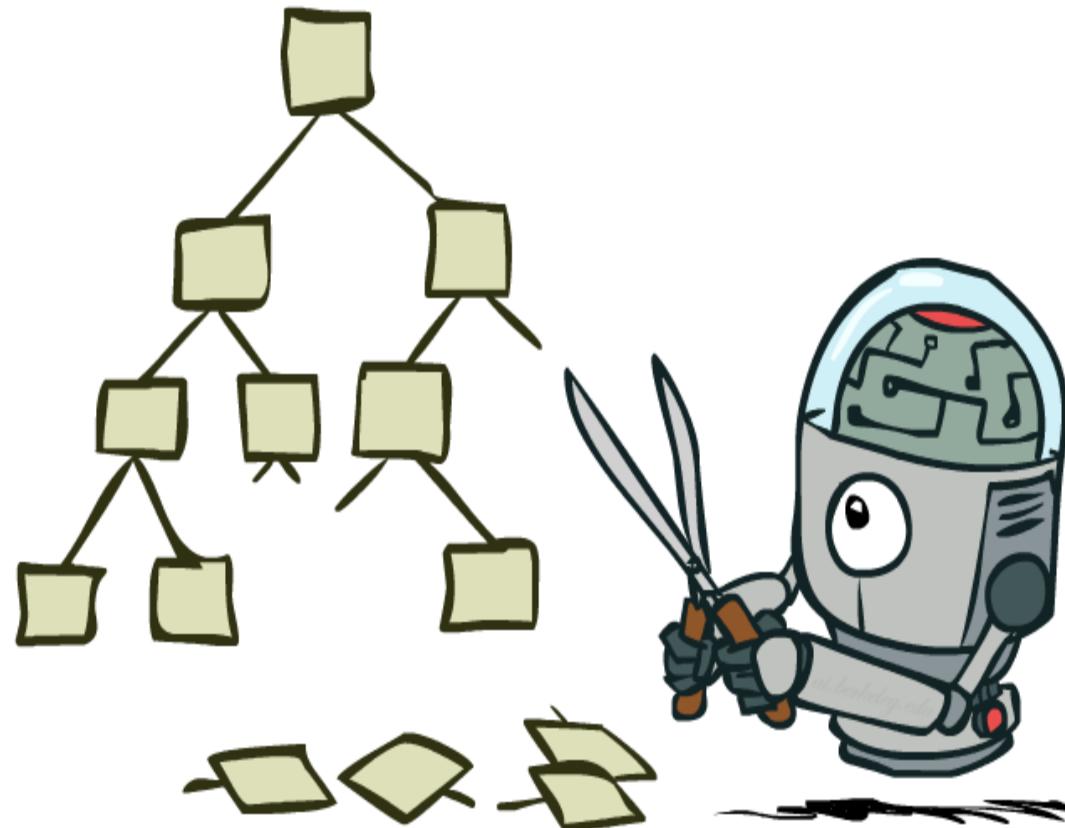
- 如果不是零和游戏，或有超过两个玩家？

- 普遍情况：

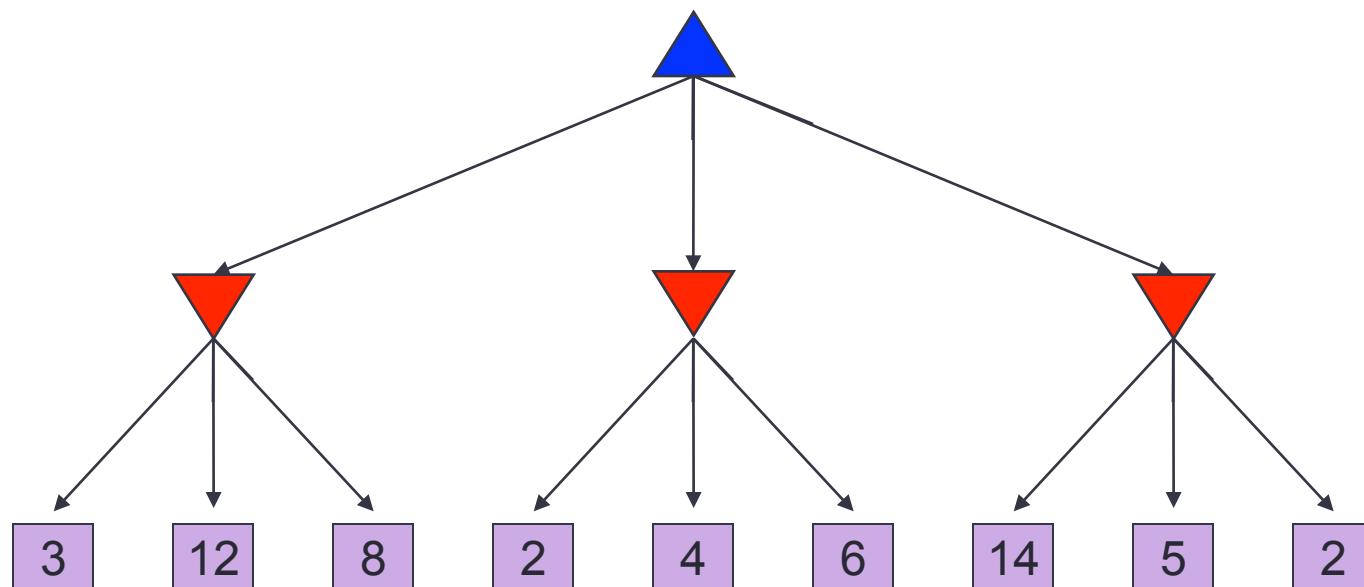
- 终局（叶结点）的值是一组值
- 中间节点的值也是一组值
- **每个玩家最大化自己的利益值元素**
- 能够动态产生协作和竞争等关系...



# 博弈（游戏）树的剪枝

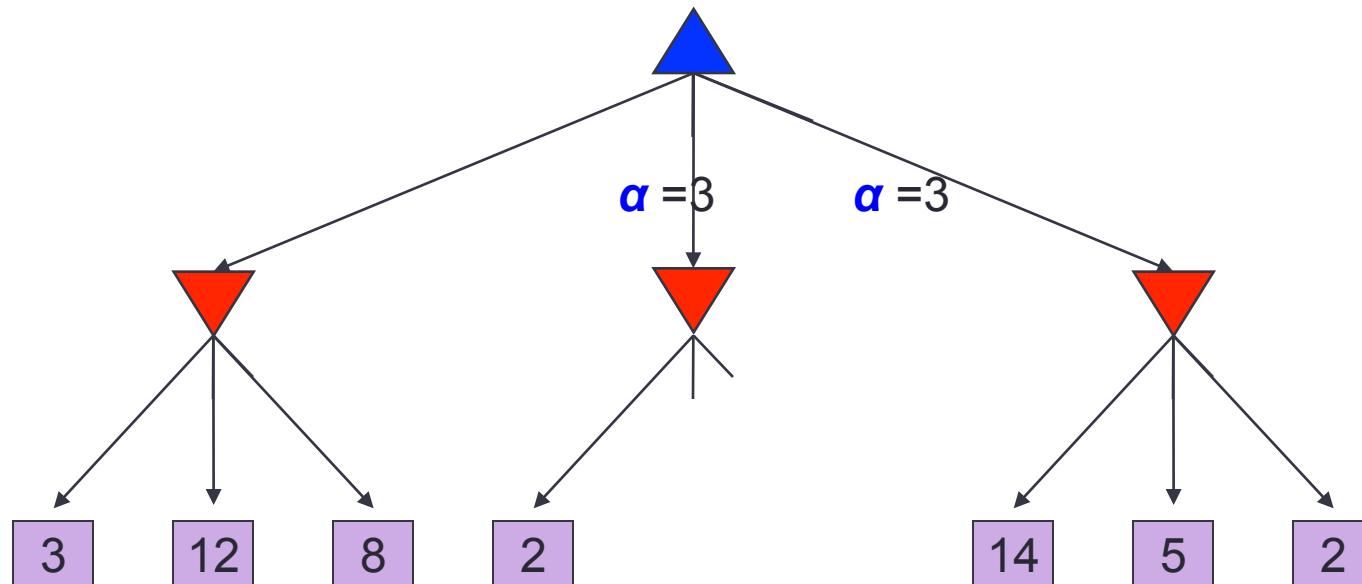


# 最小最大法举例



# Alpha-Beta 剪枝例子

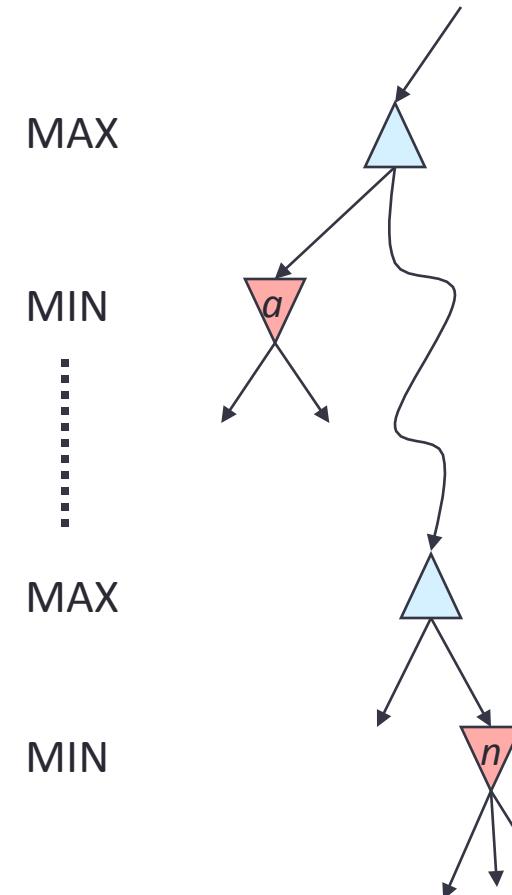
$\alpha$  = 在当前路径上所有MAX节点中最大的值



节点产生的顺序与结果有关: 可能导致不同的可被剪掉的节点数量

# Alpha-Beta 剪枝

- 假定修剪 MIN 节点的子节点
  - 假设正在计算节点  $n$  的 最小-值
  - $n$  节点的值在检查其子节点的过程中逐渐减小
  - 令  $\alpha$  是从根到当前节点路径上的 MAX 分支节点所能达到的最大值
  - 如果  $n$  的当前值比  $\alpha$  的小，那么路径上的 MAX 分支节点将会避开这条路径，所以我们可以剪掉(不去检查)  $n$  的其他子节点
- 对 MAX 节点剪枝是对称的
  - 令  $\beta$  是从根到当前节点路径中的 MIN 分支节点所能达到的最小值



# Alpha-Beta 剪枝算法实现

```
def alpha-beta-search(state) return 一个行动  
v <- max-value(state, -∞, +∞)  
return 导致值为 v 的行动
```

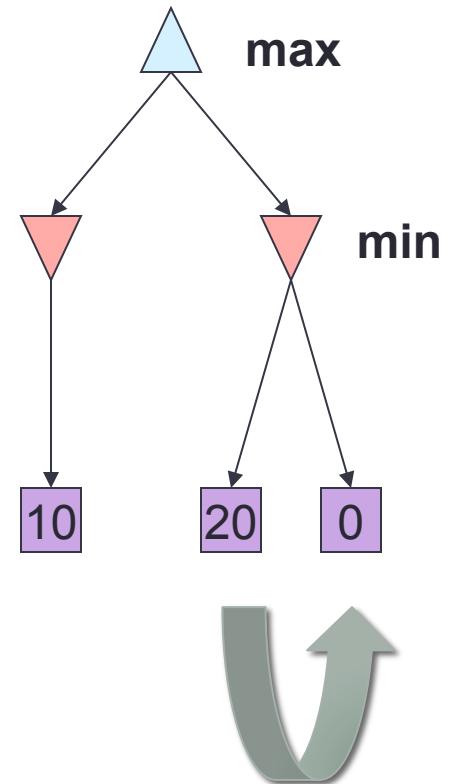
α: MAX节点的最大值, 当前路径上  
β: MIN节点的最小值, 当前路径上

```
def max-value(state, α, β):  
if 终局-检测(state) return Utility(state)  
初始化 v = -∞  
for each a in ACTIONS(state) do  
    v = max(v, min-value(Result(s,a),  
                           α, β))  
    if v ≥ β return v  
    α = max(α, v)  
return v
```

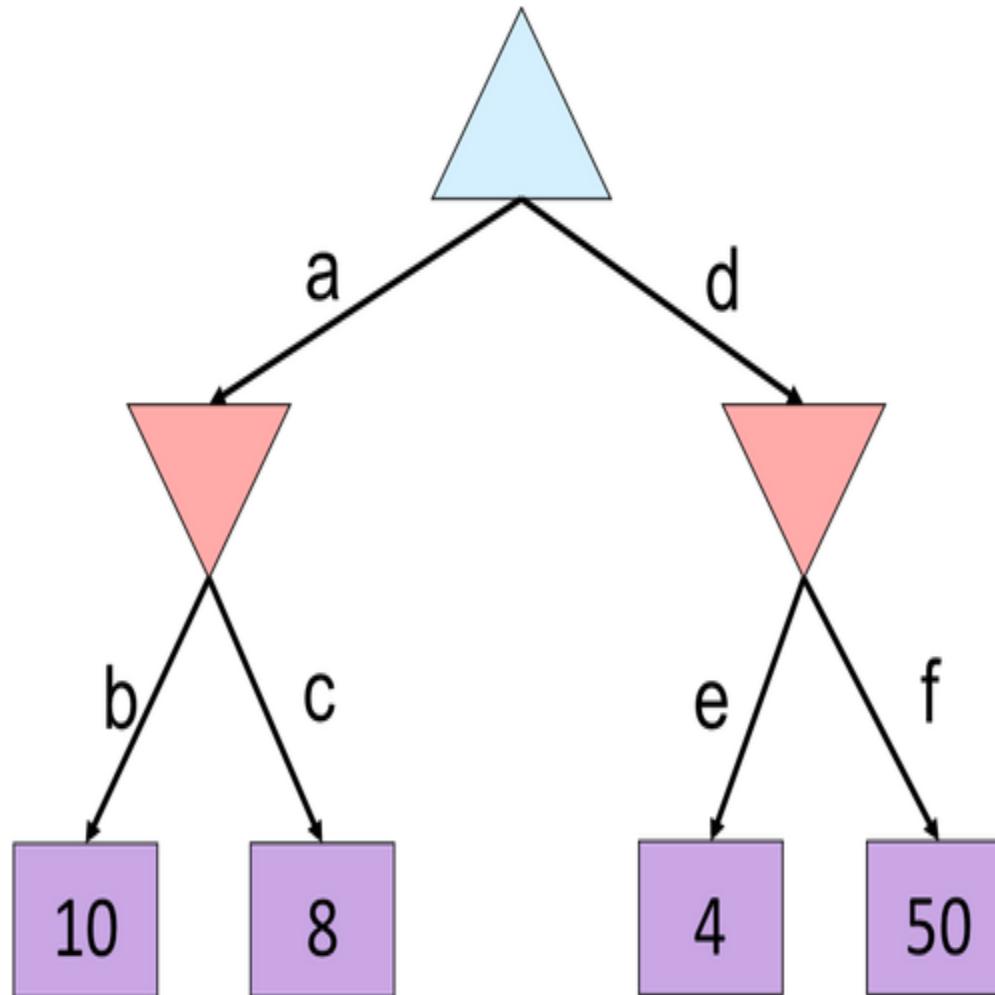
```
def min-value(state , α, β):  
if 终局-检测(state) return Utility(state)  
初始化 v = +∞  
for each a in ACTIONS(state) do  
    v = min(v, max-value(Result(s,a),  
                           α, β))  
    if v ≤ α return v  
    β = min(β, v)  
return v
```

# Alpha-Beta 剪枝属性

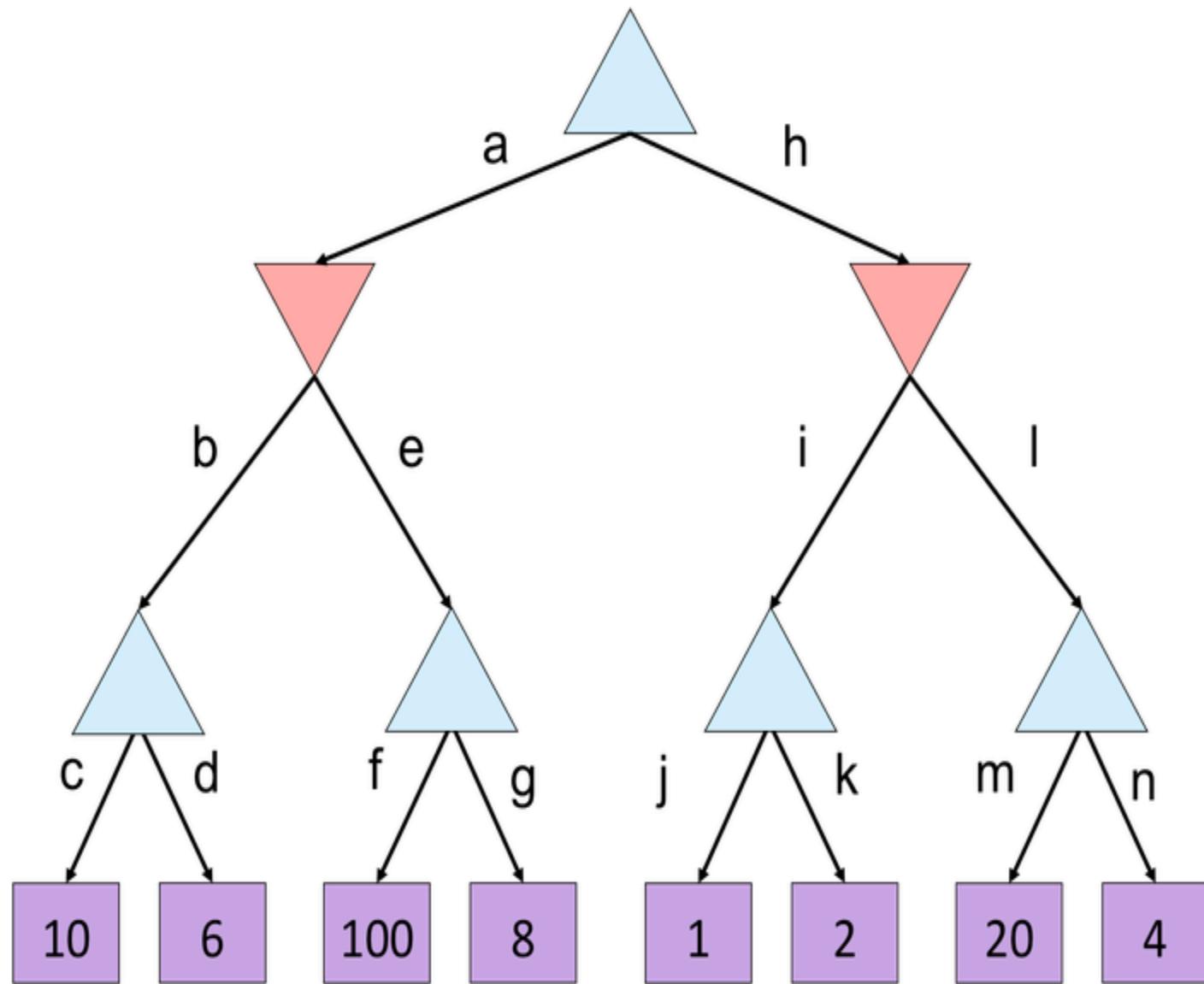
- 剪枝**不影响** 根节点的最小最大值的计算！
- 好的子节点的排序能提高剪枝的有效性
- 假定是“完美的排序”：
  - 时间复杂度能降到  $O(b^{m/2})$
  - 搜索深度能提高一倍！



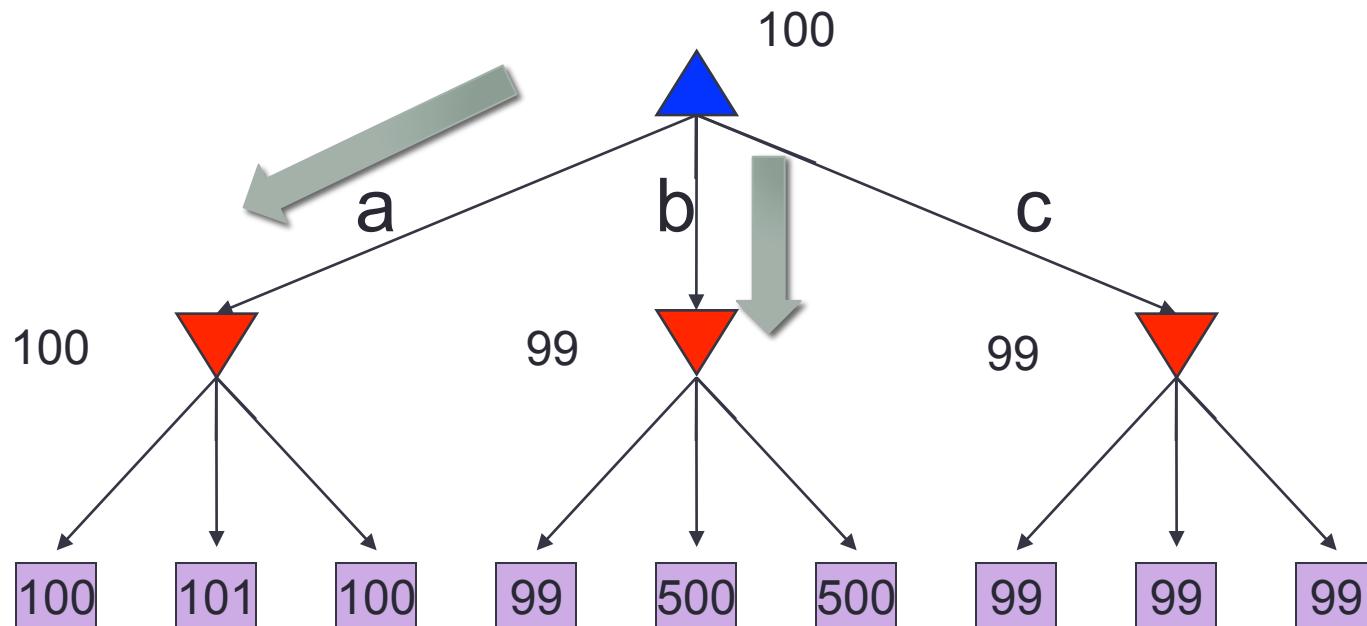
# Alpha-Beta 剪枝测试



# Alpha-Beta 剪枝测试2



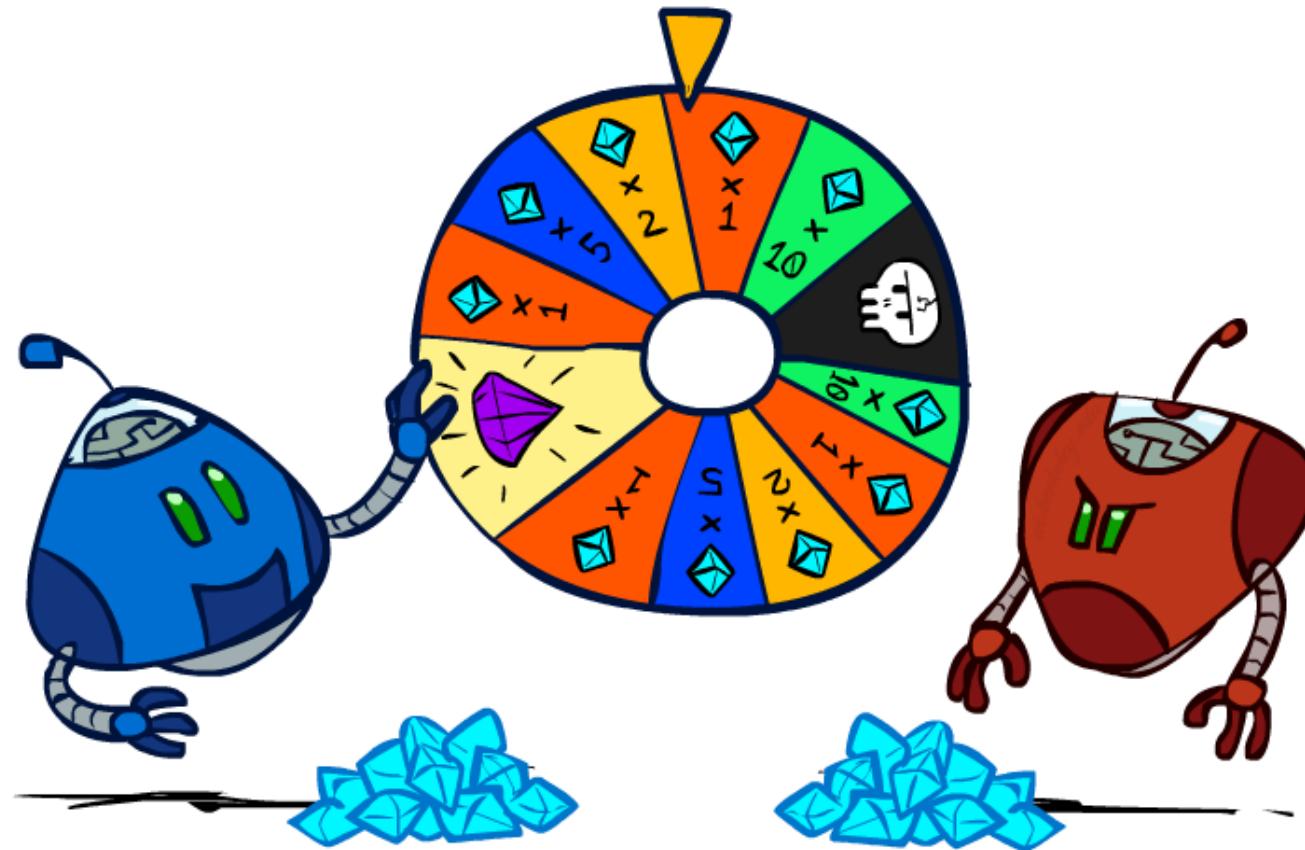
# 最小最大值 重新思考



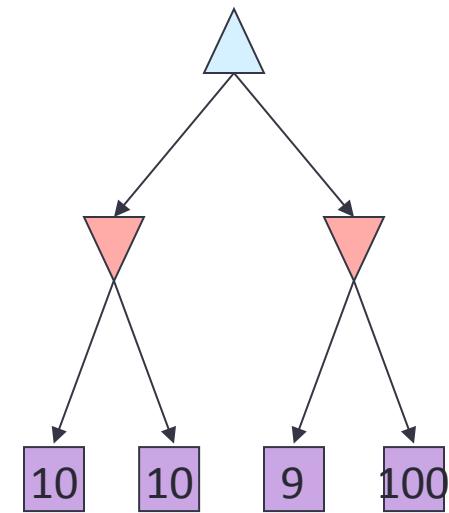
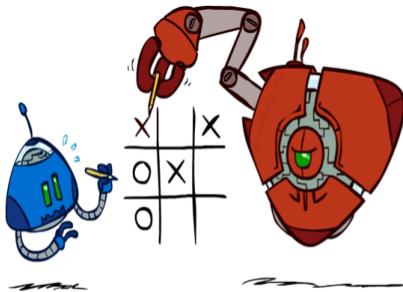
把叶结点值当成准确值。

实际它们是不确定情况下的估计值，也许实际情况中，走b路径比走a更好。

# 不确定结果的游戏

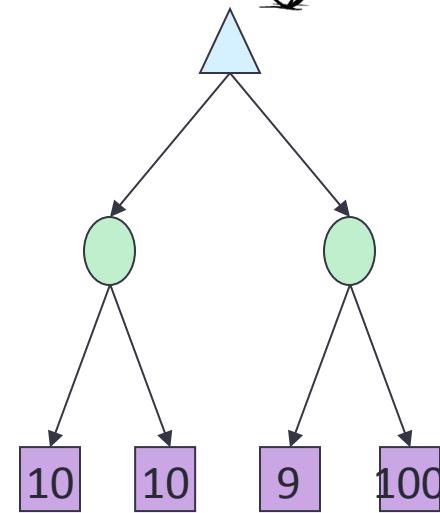
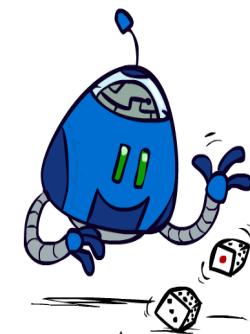


# 搜索树中可能有机遇节点



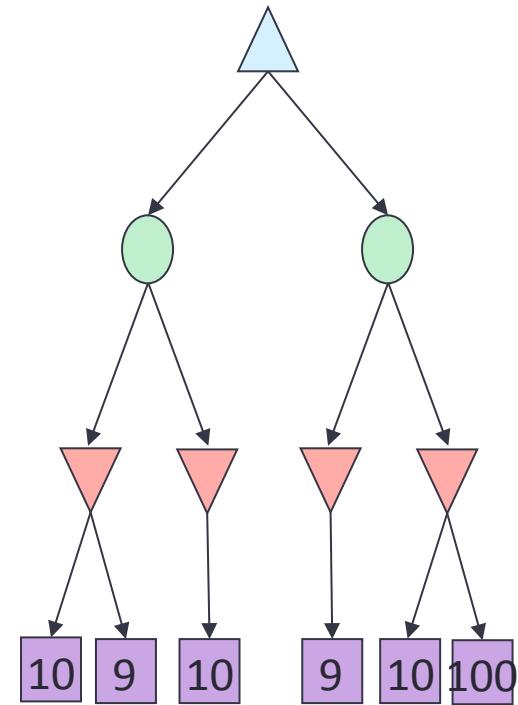
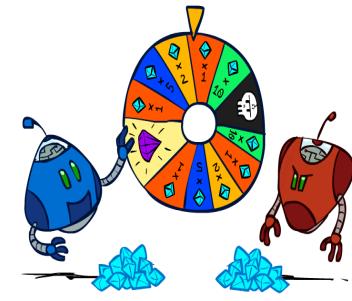
井字游戏

**最小最大值(Minimax)**



投资游戏

**期望最大值  
(Expectimax)**



大富翁游戏

**期望最小最大值(Expectiminimax)**

# 最小最大值

Function 决策(s) returns 一个行动  
return Actions(s) 中的行动  $a$ , 它的  
value(Result(s,a)) 最大



function value(s) returns a value  
If 终局-检测(s) then return Utility(s)  
if Player(s) = MAX then return  $\max_{a \text{ in Actions}(s)}$  value(Result(s,a))  
if Player(s) = MIN then return  $\min_{a \text{ in Actions}(s)}$  value(Result(s,a))

# 期望最小最大值(Expectiminimax)

Function 决策(s) returns 一个行动

return Actions(s) 里的一个行动  $a$ ，  
它的 value(Result(s,a)) 是最大的



function value(s) returns a value

if 终局-检测(s) then return Utility(s)

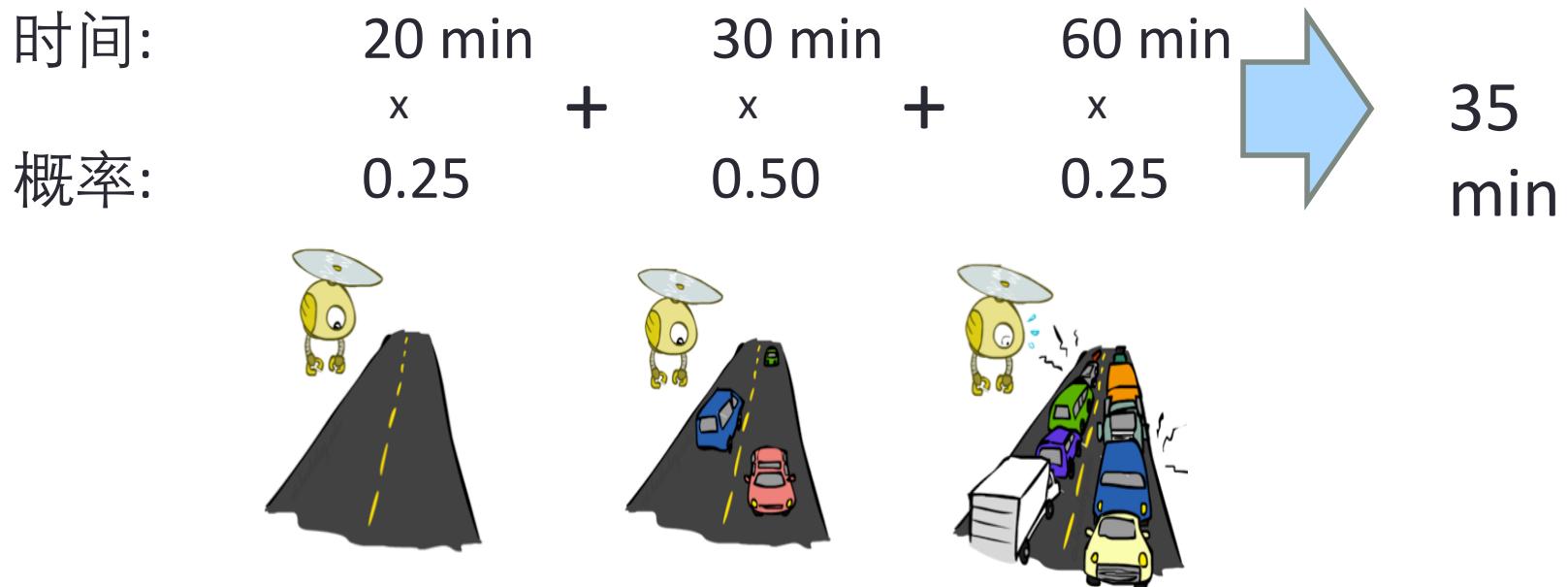
if Player(s) = MAX then return  $\max_{a \text{ in } \text{Actions}(s)}$   
value(Result(s,a))

if Player(s) = MIN then return  $\min_{a \text{ in } \text{Actions}(s)}$   
value(Result(s,a))

if Player(s) = CHANCE then return  $\sum_{a \text{ in } \text{Actions}(s)} \text{Pr}(a) * \text{value}(\text{Result}(s,a))$

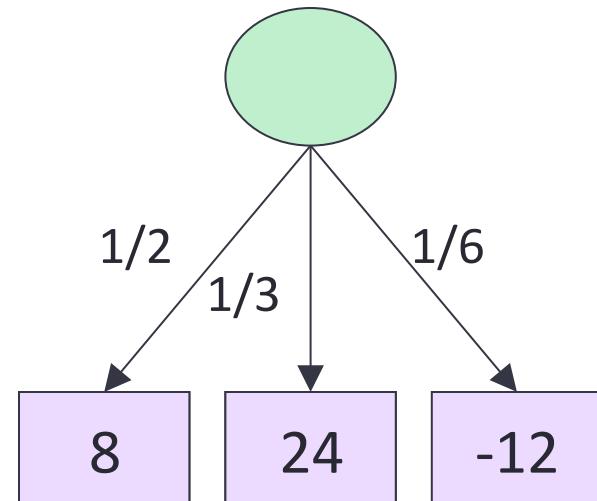
# 提示: 期望值

- 一个随机变量的期值是概率分布到对应输出结果的加权平均值
- 举例: 去机场路上花费的时间?



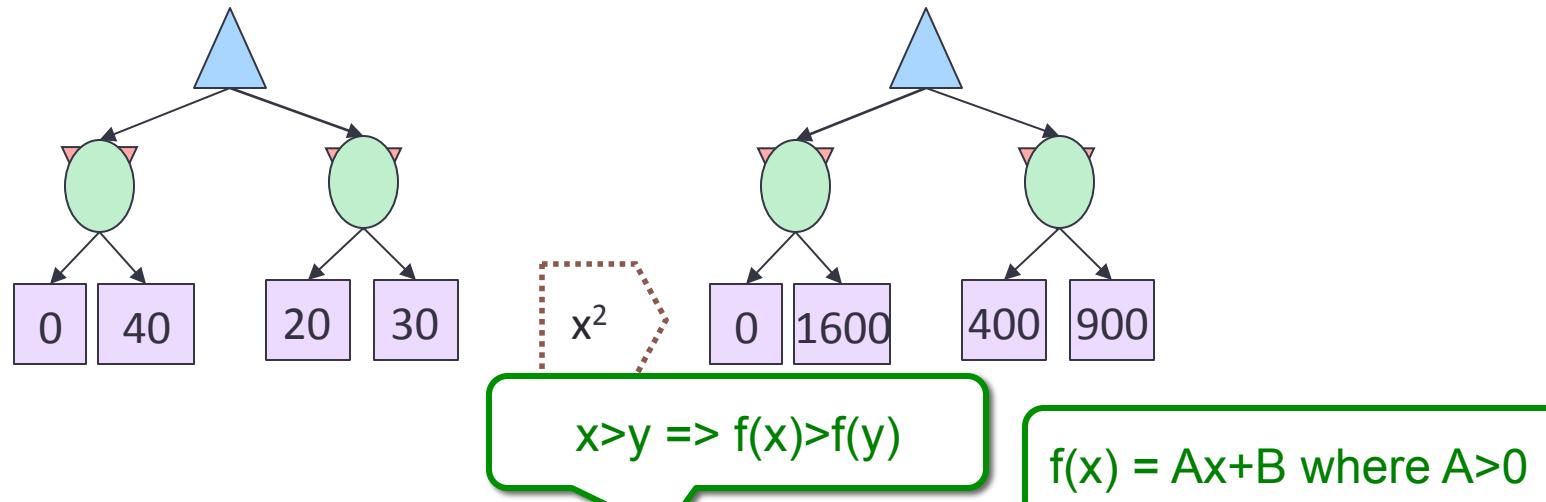
# 计算期望最小最大值的代码

$\text{sum}_{a \in \text{Action}(s)}$   
 $\Pr(a) * \text{value}(\text{Result}(s,a))$



$$v = (1/2)(8) + (1/3)(24) + (1/6)(-12) = 10$$

# Utility值与决策的关系



- 在最小最大值决策中，评估函数单调递增变化不影响决策结果
  - 只要好的状态结果有高的评估值；排序是对的就对了
- 期望最小最大值，决策结果不受评估值的正仿射变换影响
  - 获胜概率高的状态，其对应的评估值也应该大

# 总结

- 博弈（游戏） 当找到最优解是不可能的时候，需要行动上的决策
  - 有界深度搜索，和近似评估函数
- 博弈（游戏） 决策需要有效率的搜索计算
  - Alpha-beta 剪枝
- 在博弈游戏上的探索产生了重要的研究思想
  - 增强式学习(国际跳棋)
  - 迭代加深(国际象棋)
  - 合理的元推理(奥赛罗棋)
  - 蒙特卡罗树搜索(围棋)
  - 经济学中关于部分信息可知的博弈方法 (纸牌)
- 视屏游戏中的决策问题，仍是巨大的挑战！
  - $b = 10^{500}$ ,  $|S| = 10^{4000}$ ,  $m = 10,000$