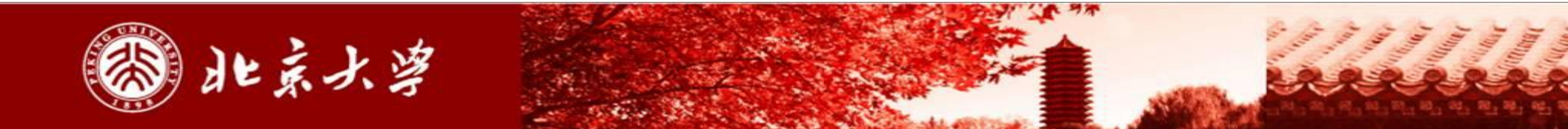
《物理与人工智能》

16. 对抗搜索

授课教师：马滟青

2026/10/26（第七周）

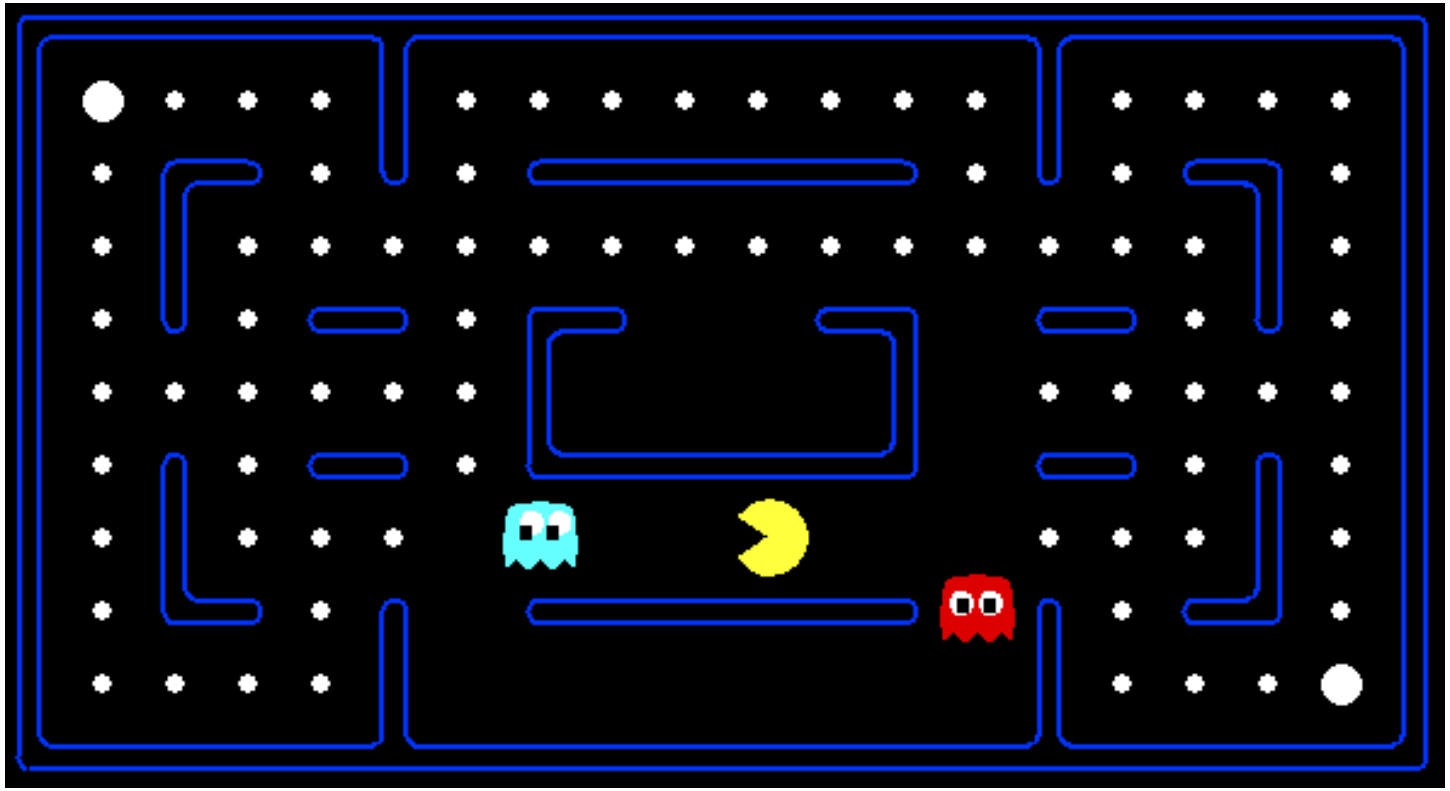
鸣谢：基于计算机学院《人工智能引论》课程组幻灯片



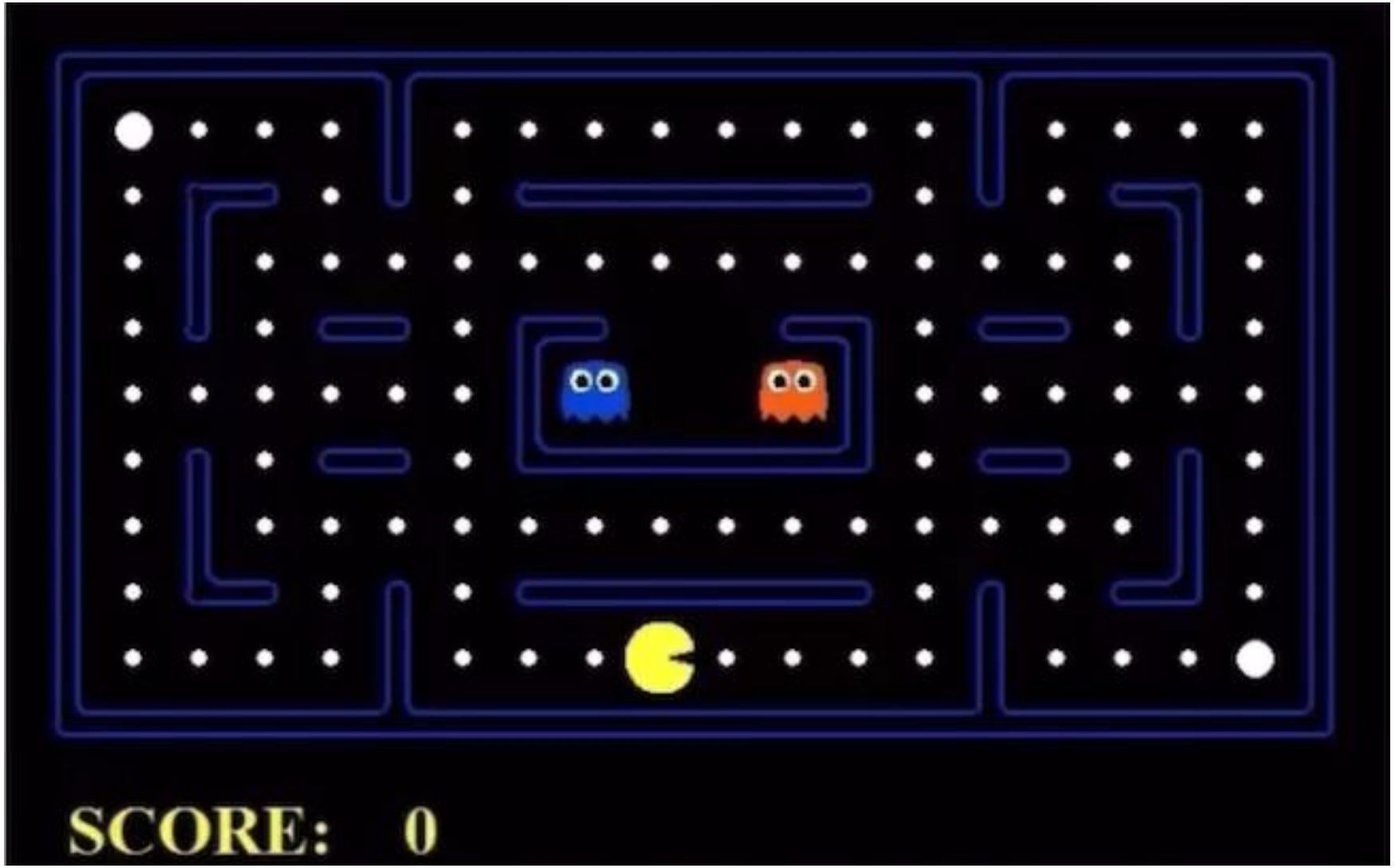
目录

- **什么是博弈搜索**
  - MiniMax --- 博弈的通用解法
- **状态空间过大?**
  - 剪枝 --- AlphaBeta
- **不确定?**
  - 转移方程有不确定性、对手的动作有不确定性
  - 期望

计算 -> 行为（Pacman）



Pacman演示

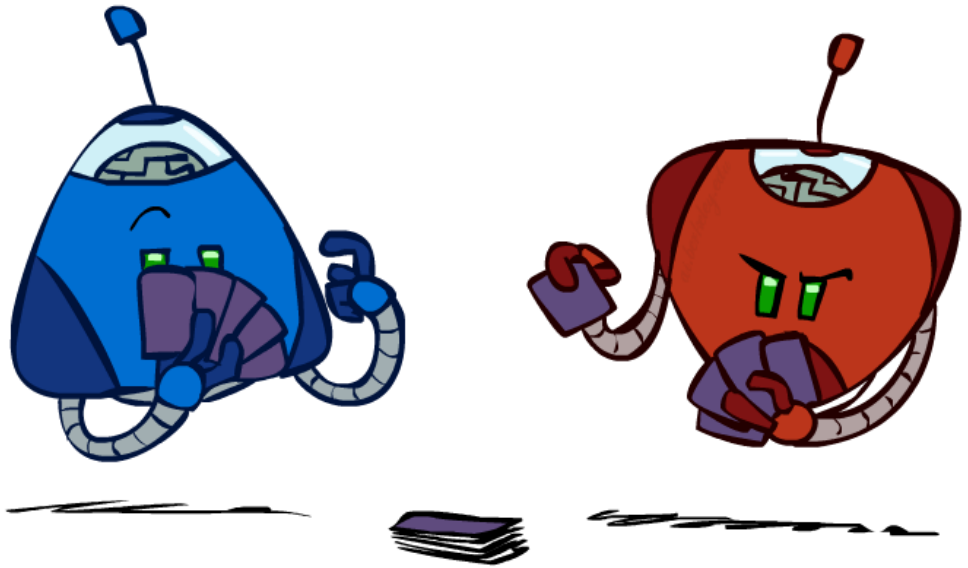


游戏类型

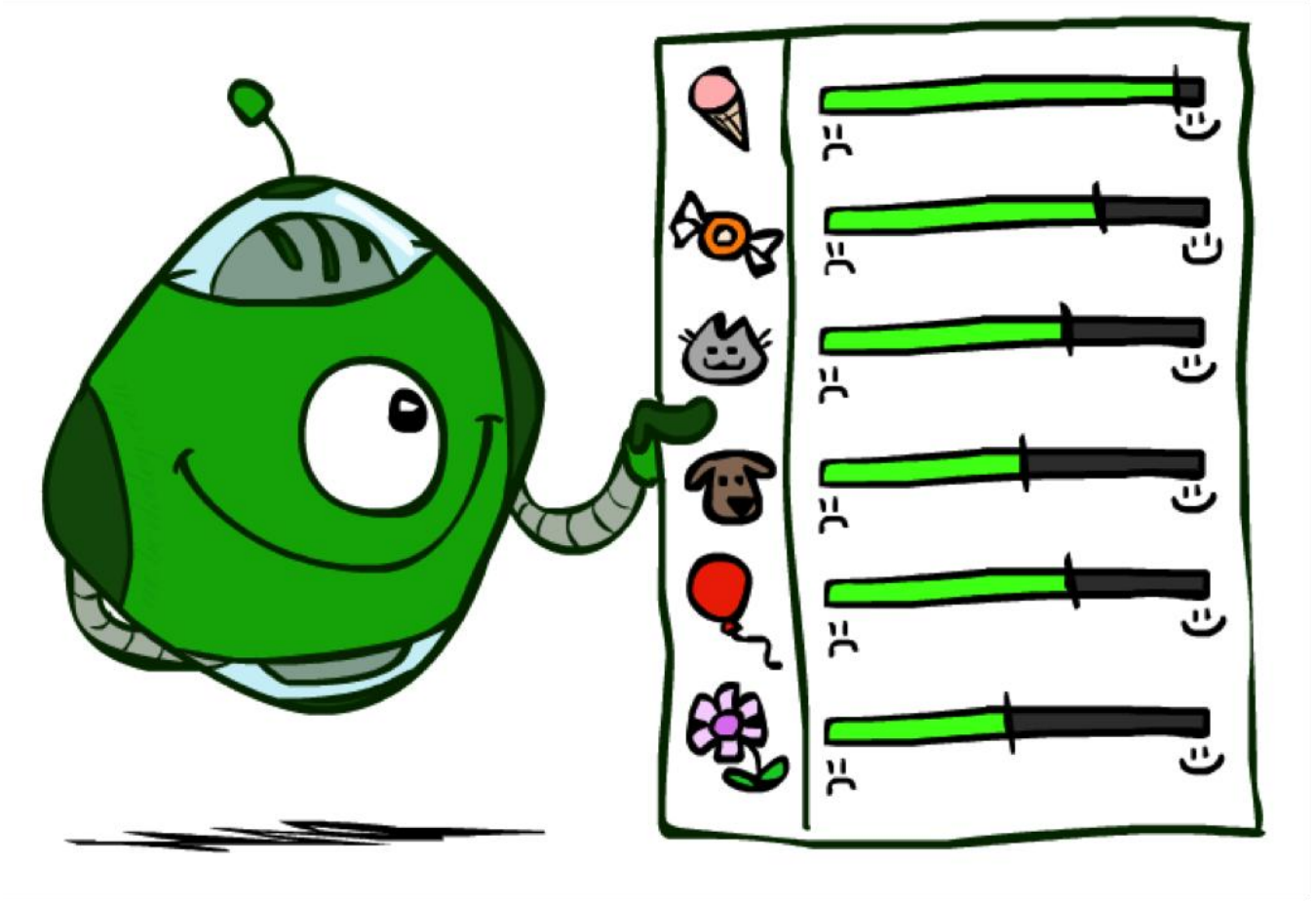
北京大学 PEKING UNIVERSITY

分类方式:

- 确定性的(deterministic) 还是随机性的(stochastic)?
- 单一智能体、2个还是多个?
- 是否是零和博弈(zero sum)?
- 是否是完美信息博弈 (perfect information)?



效用

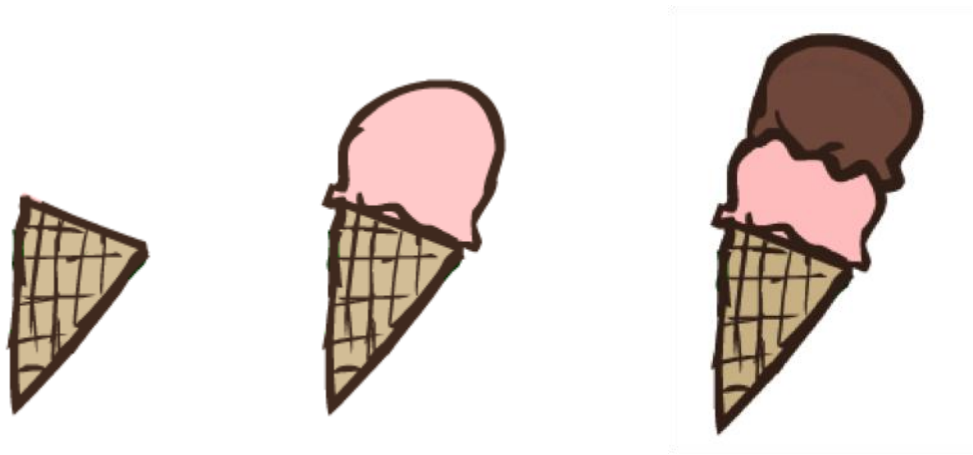


效用

- **效用函数是将结果（世界的状态）映射到实数的函数，用于描述智能体的偏好**

- 效用函数从哪里来?
  - 在游戏中可能很简单，即（+1/-1）
  - 效用函数总结了智能体的目标
  - 定理：任何“理性”的偏好都能用效用函数表达

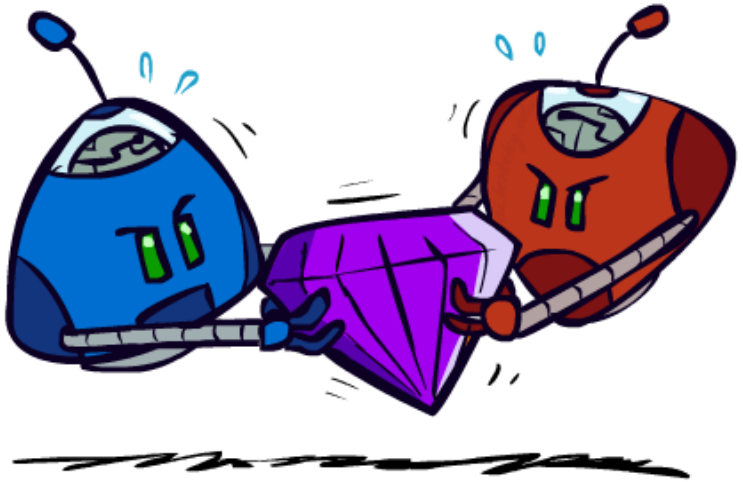
- 我们通过定义效用函数来影响智能体的行为
  - 效用函数（通常）是环境的一部分



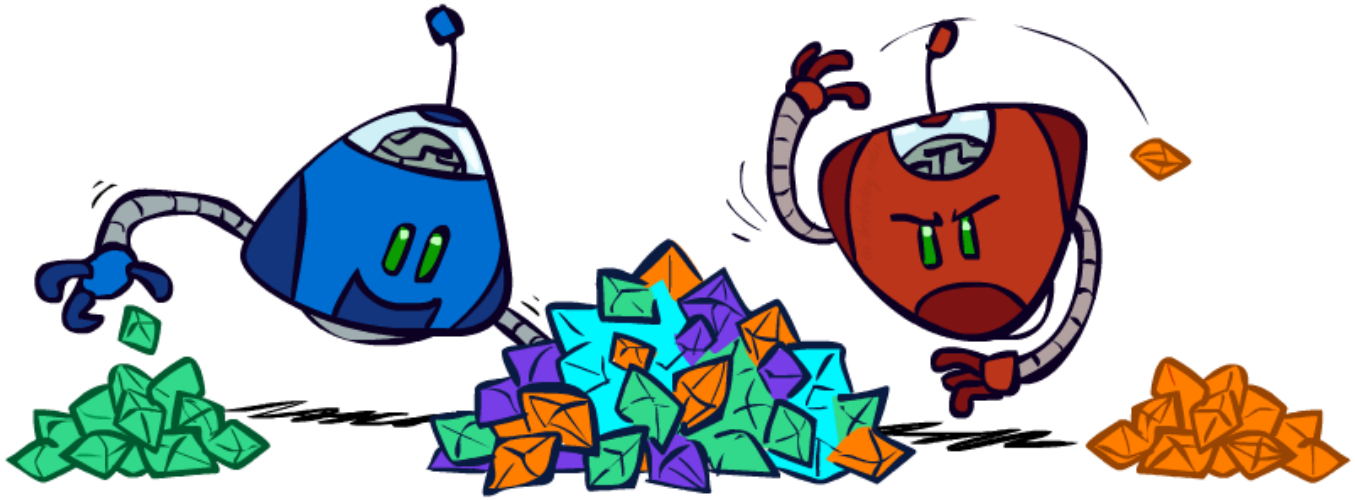
博弈范式

- 效用：关于回报大小的度量

- **零和博弈(Zero-Sum Games)**
  - 智能体的效用是对立的，比如：当一个智能体最大化其效用时，另一智能体效用为其最小值
  - 特点：对抗的**(adversarial)**, 完全竞争**(pure competition)**



- **一般博弈(General Games)**
  - 智能体获得的效用是独立的
  - 智能体之间可能的关系：合作、无关、竞争...
  - 我们希望AI不是孤立地行动，而是与人类共事或是协助人类，因此每个AI智能体都需要面对和处理博弈的情况



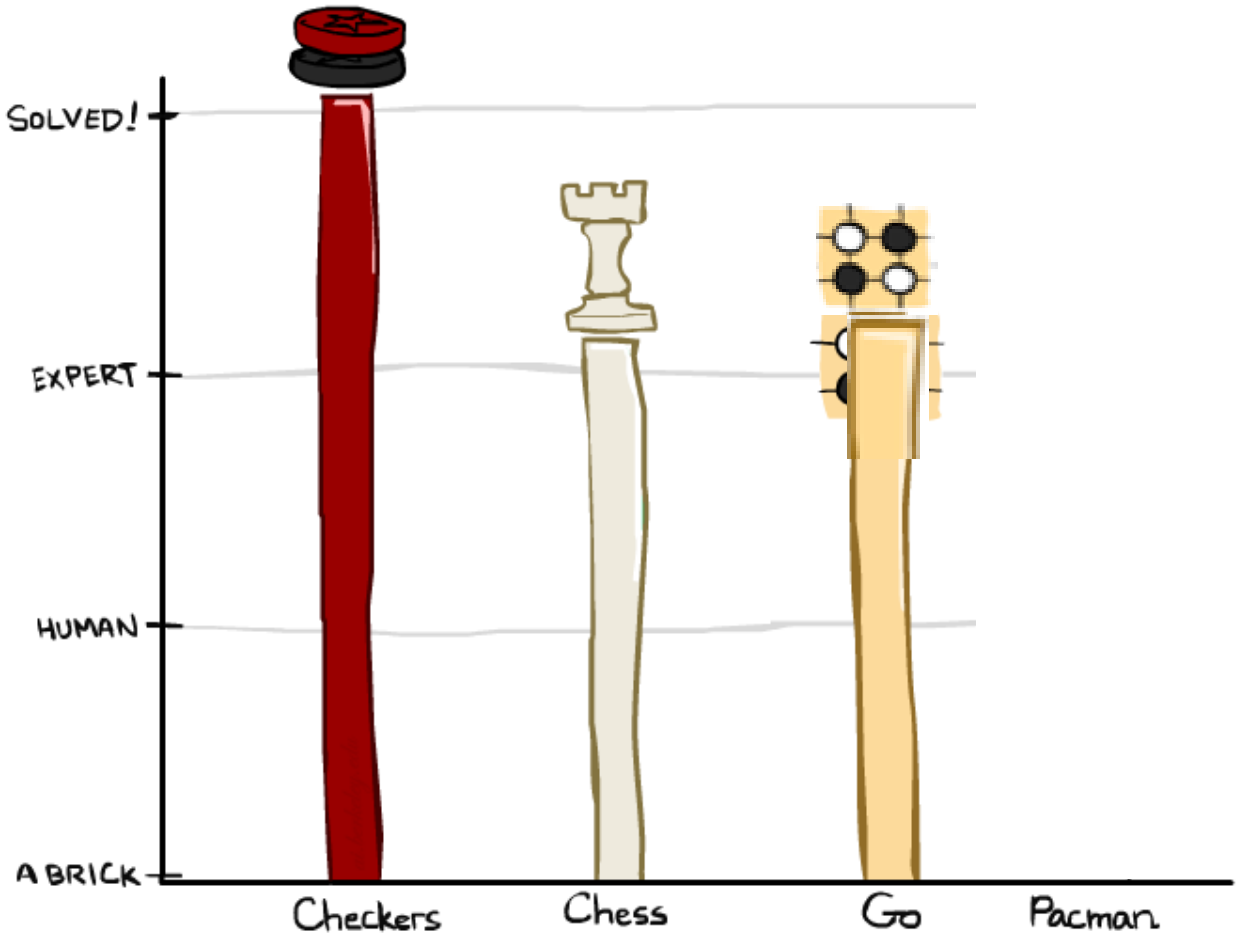
零和博弈

- **Checkers：1950、 第一个计算机玩家. 1994、计算机玩家取得第一个冠军， Chinook结束了人类冠军玩家Marion Tinsley长达四十年的霸榜(使用 complete 8-piece endgame). 2007、Checkers solved!**

- **Chess：1997、 Deep Blue 在一场持续六局的比赛中打败了人类冠军玩家Gary Kasparo。 Deep Blue 每秒能搜索200M个盘面，使用了非常复杂且未公开的计算评估方法将一些搜索线路扩展到40个棋局。目前的程序甚至能做到更好。**

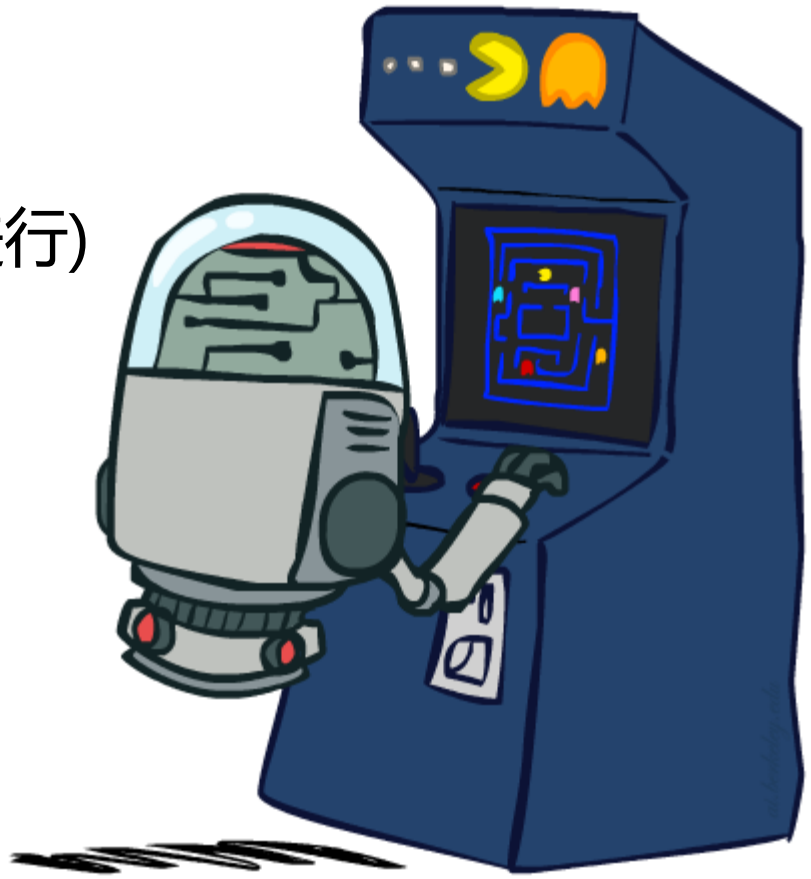
- **Go：2016、 Alpha GO 打败了传奇李世石。2017年击败当时第一人柯洁。Alpha GO使用蒙特卡罗树搜索并通过训练学习评估函数。**

- **Pacman？2026?**



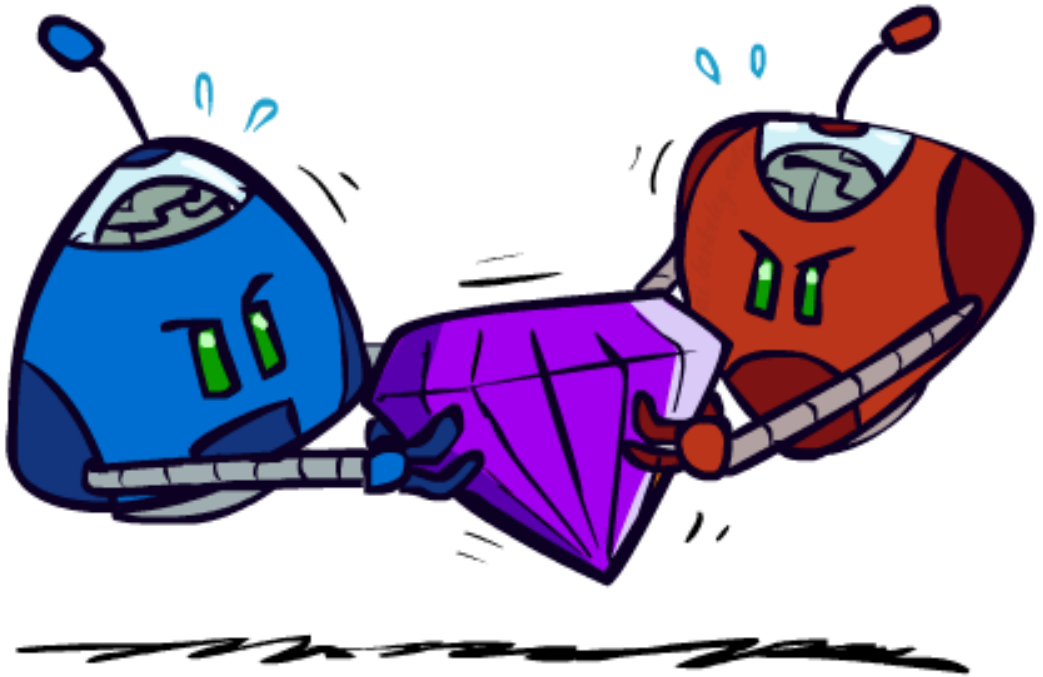
具有最终效用的确定性博弈

- 一个可行的问题建模:
  - 状态(States): S (起始状态为 s_0)
  - 博弈主体(Players): $P = \{1 \dots N\}$ (通常轮流进行)
  - 行动(Actions): A (可能取决于 玩家 / 状态)
  - 转移方程(Transition Function): $S \times A \rightarrow S$
  - 结束测试(Terminal Test): $S \rightarrow \{t, f\}$
  - 最终**效用**(Terminal Utilities): $S \times P \rightarrow R$

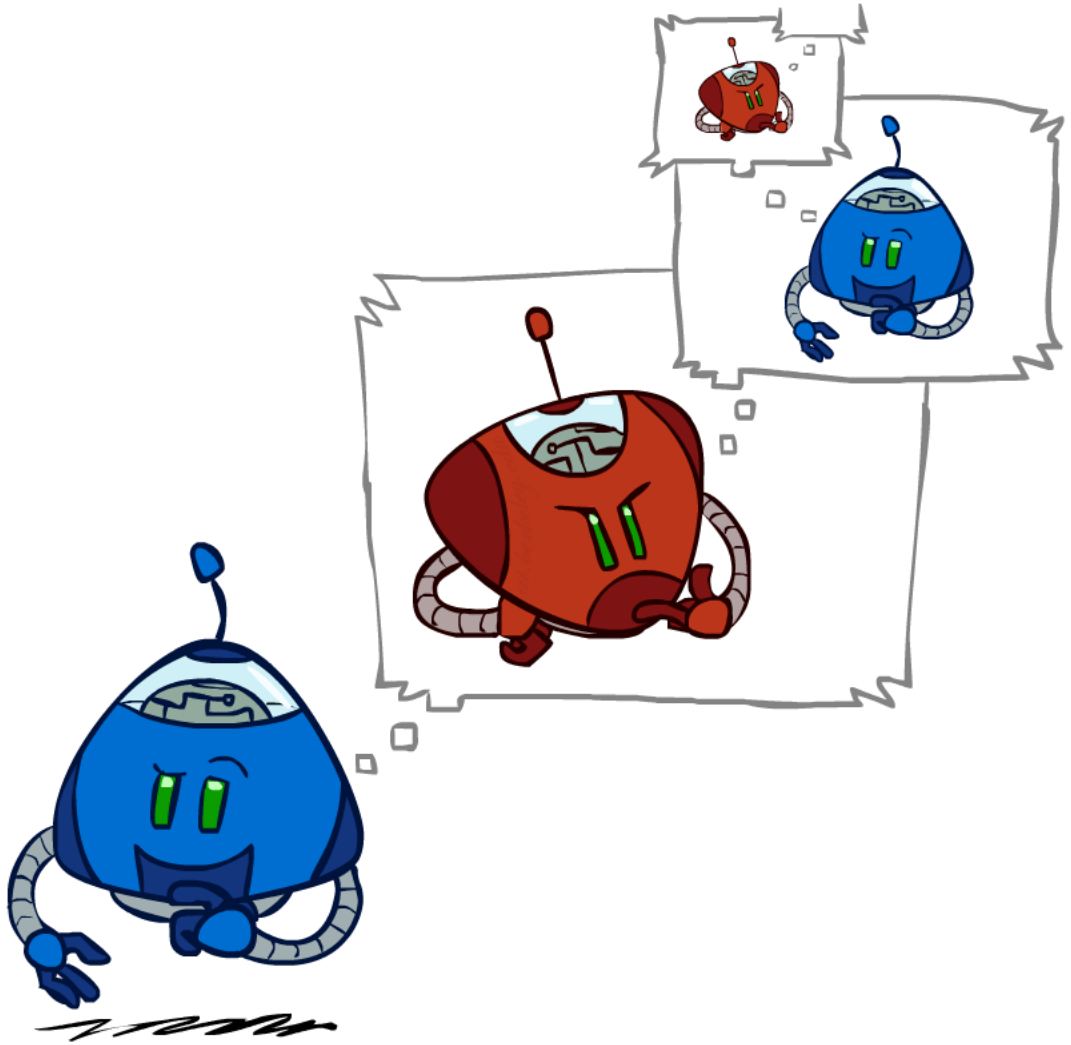


- 应对方案称为策略(policy): $S \rightarrow A$

对抗游戏/博弈



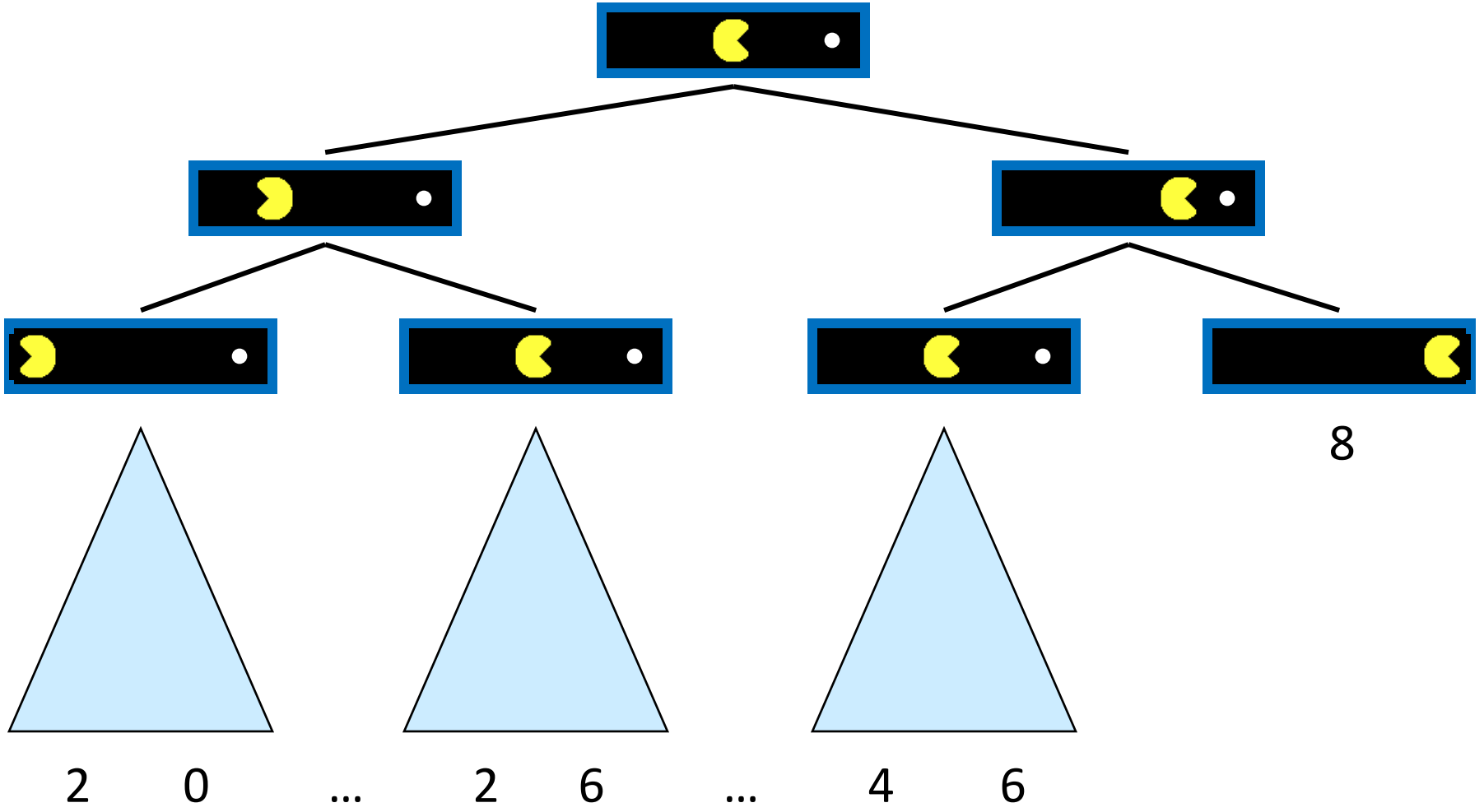
对抗搜索



搜索：代价(cost) -> 效用(utility)!

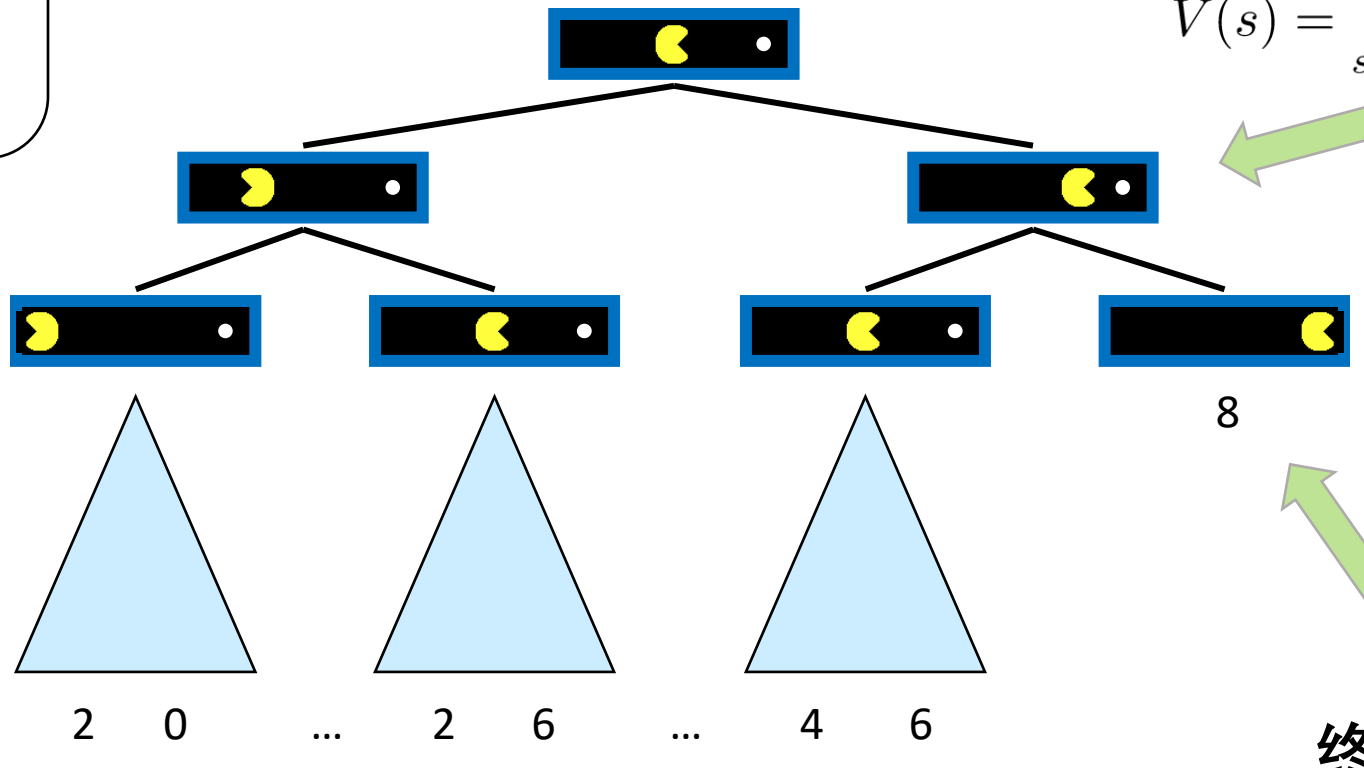
- 我们不再最小化代价，而是最大化智能体所得的**分数/效用!**

单一智能体的搜索树



状态的价值(Value)

状态价值: 从该状态出发可能获得的最大最终效用



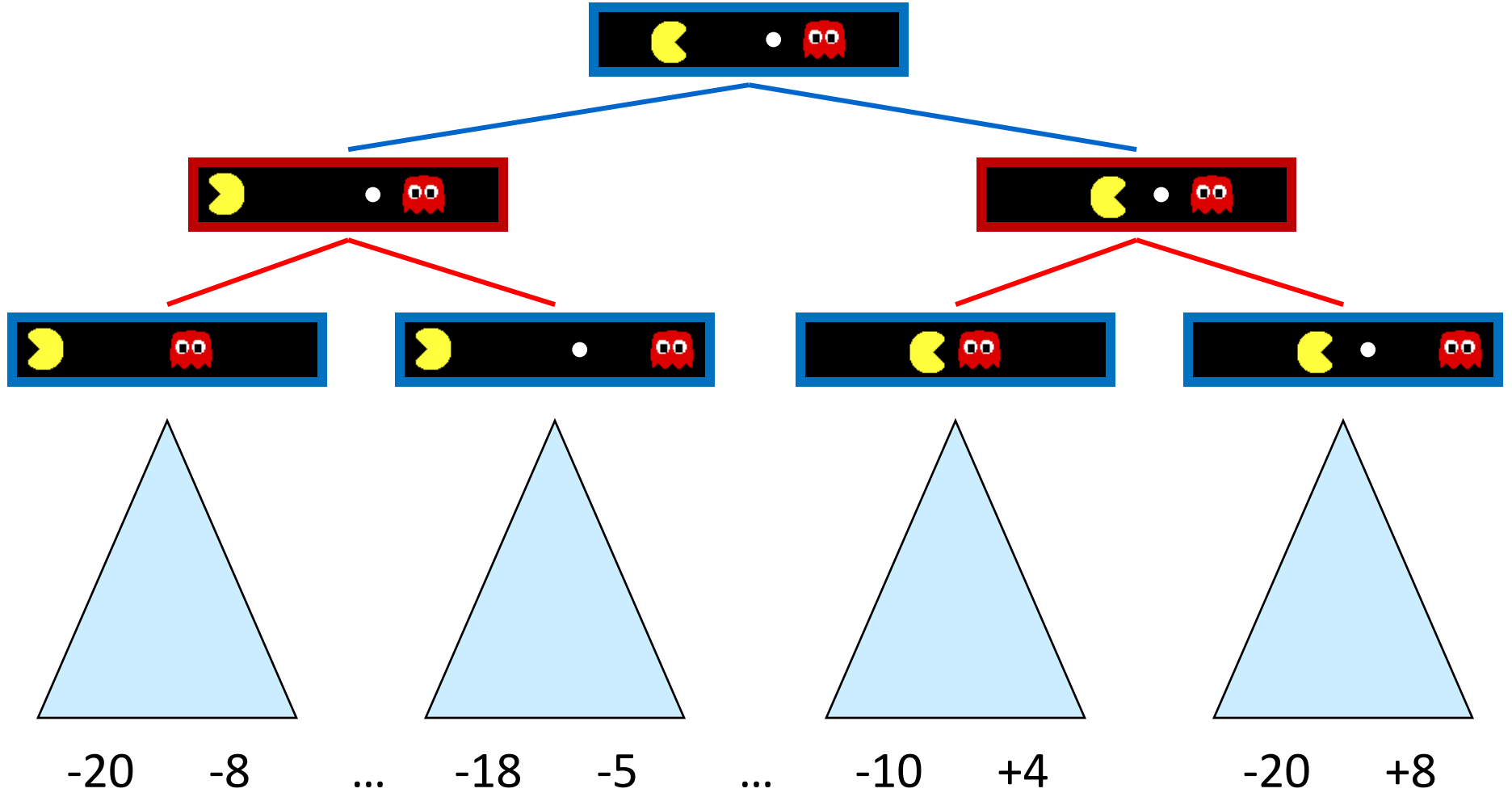
中间状态:

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

终止状态:

$$V(s) = \text{known}$$

对抗博弈的搜索树



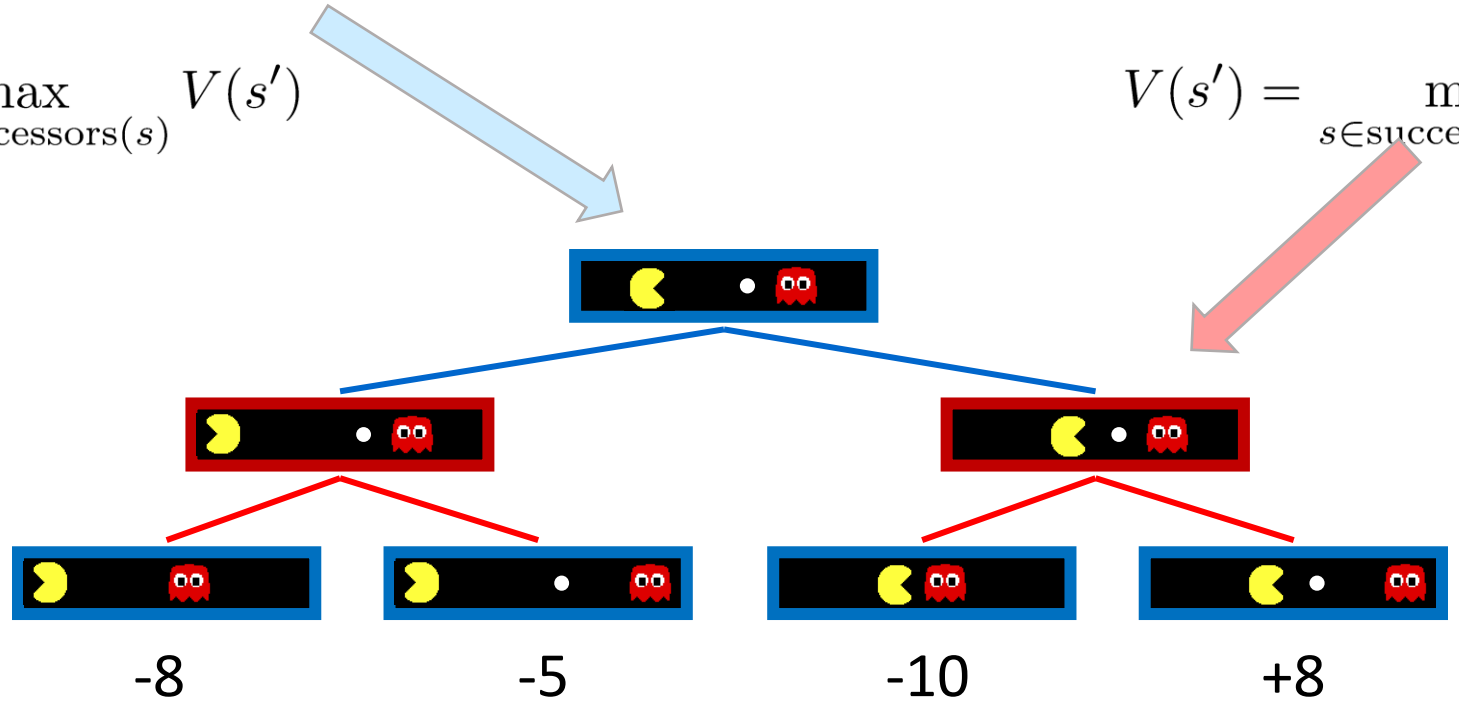
极小极大搜索(Minimax Values)

在智能体(我方)控制下的状态：

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

在对手控制下的状态：

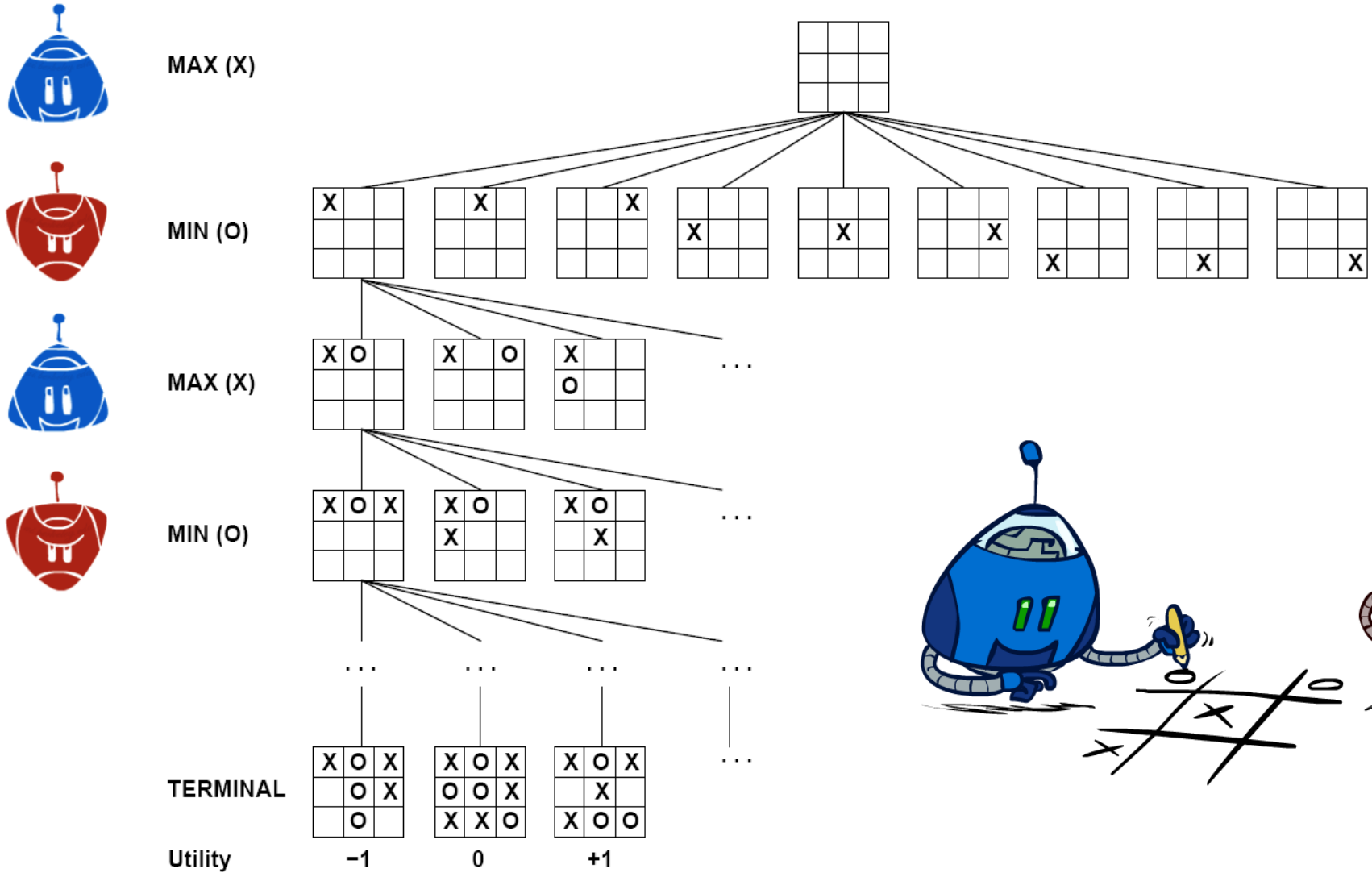
$$V(s') = \min_{s \in \text{successors}(s')} V(s)$$



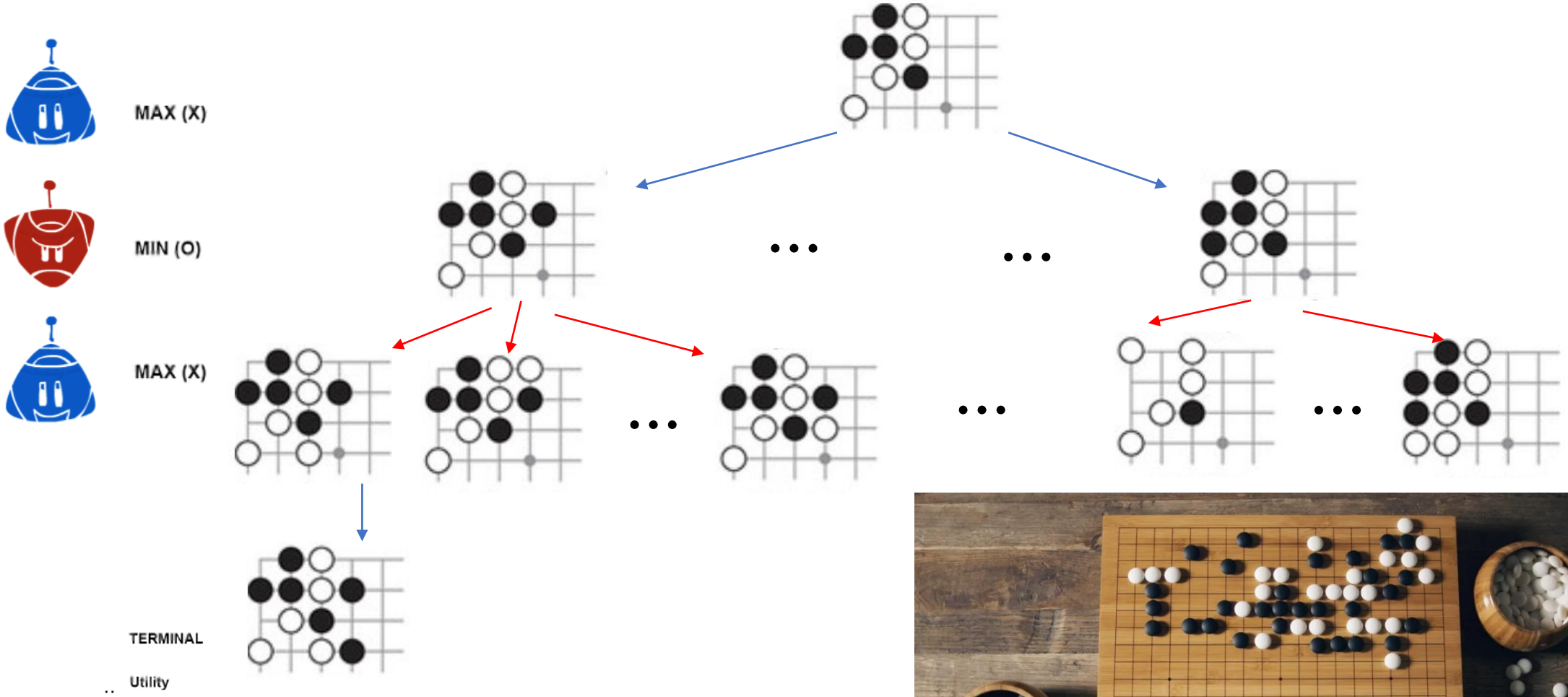
终止状态：

$$V(s) = \text{known}$$

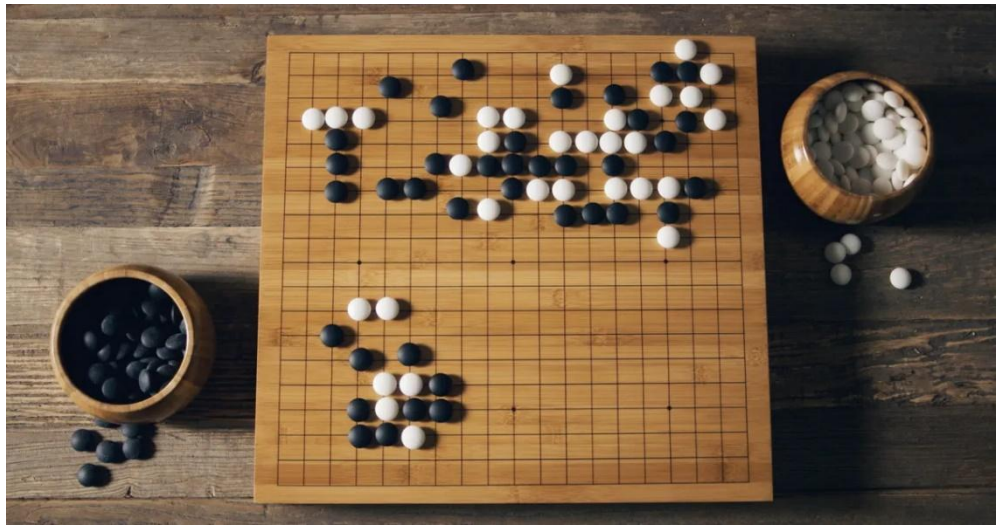
井字棋博弈树



围棋的博弈树

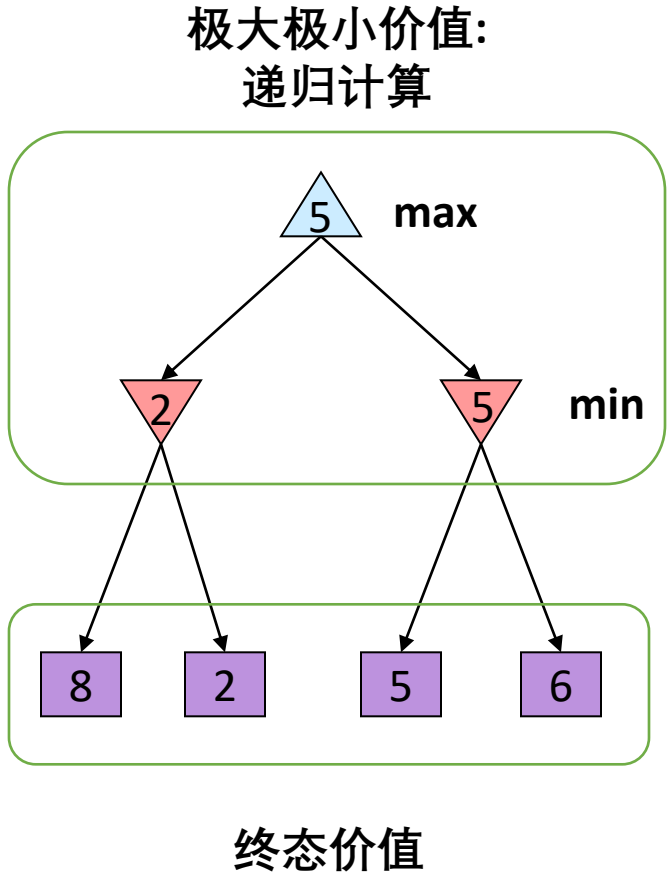


黑赢1



对抗搜索 (极小极大策略)

- **确定性零和博弈:**
  - 井字棋, 国际象棋, 跳棋、围棋
  - 其中一个玩家最大化价值，对手玩家最小化价值
- **极小极大搜索:**
  - 状态空间的搜索树
  - 玩家轮流行动
  - 计算每个节点极大极小价值: 假设对手是理性的情况下（即对方总是努力达到最优），自己可以达到的最优效用值



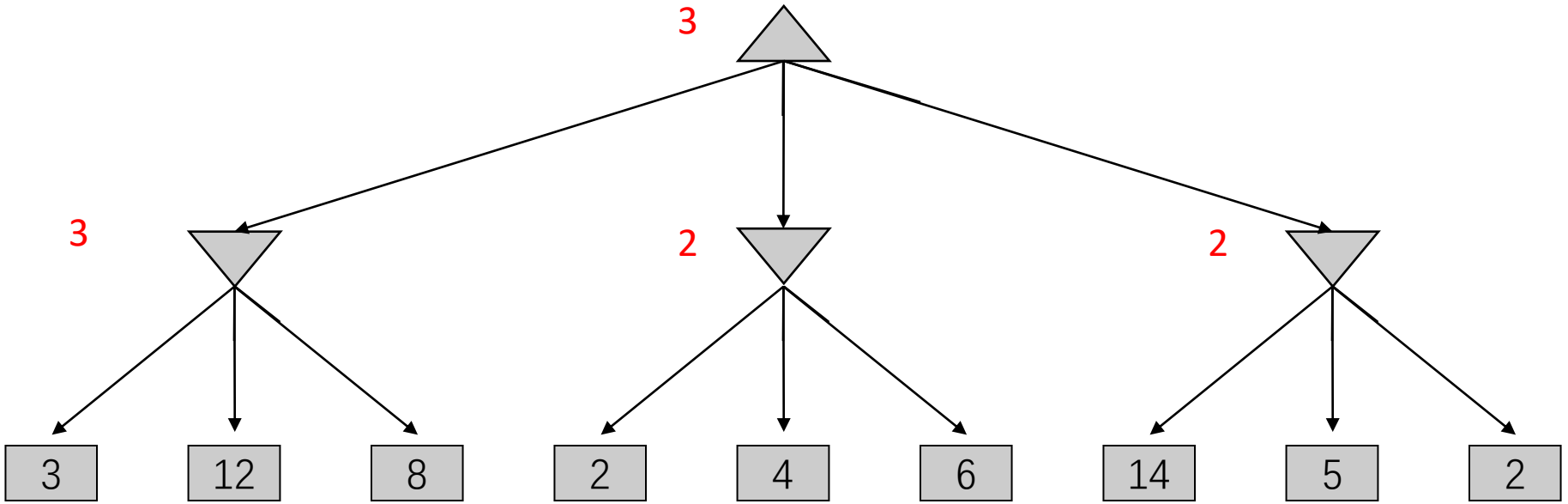
极大极小搜索算法实现

```
def value(state):
    if the state is a terminal state: return the state's
      utility
    if the next agent is MAX: return max-value(state)
    if the next agent is MIN: return min-value(state)
```

```
def max-value(state):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor))
    return v
```

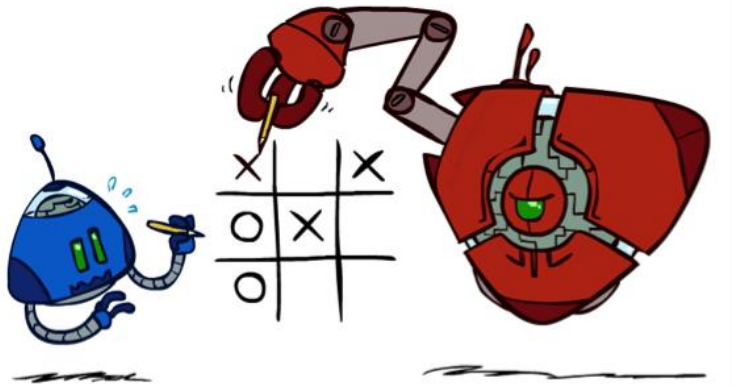
```
def min-value(state):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor))
    return v
```

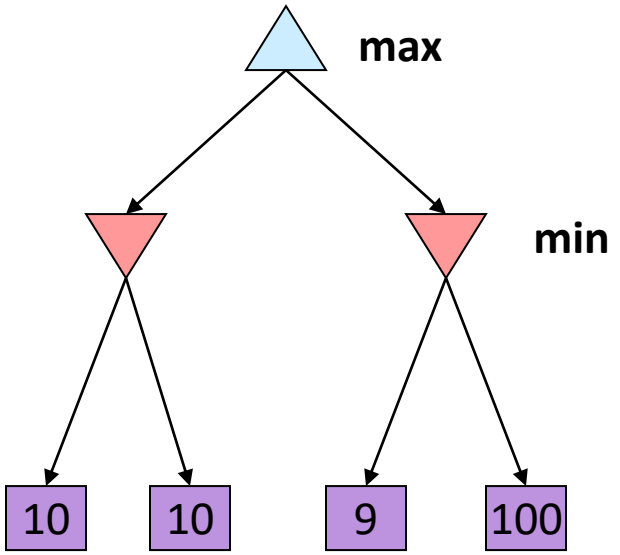
极大极小搜索示例

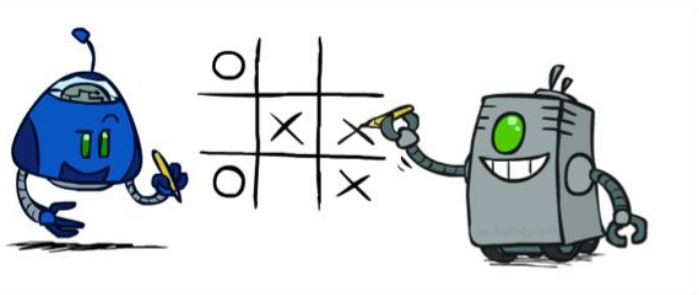


极大极小搜索算法的性质

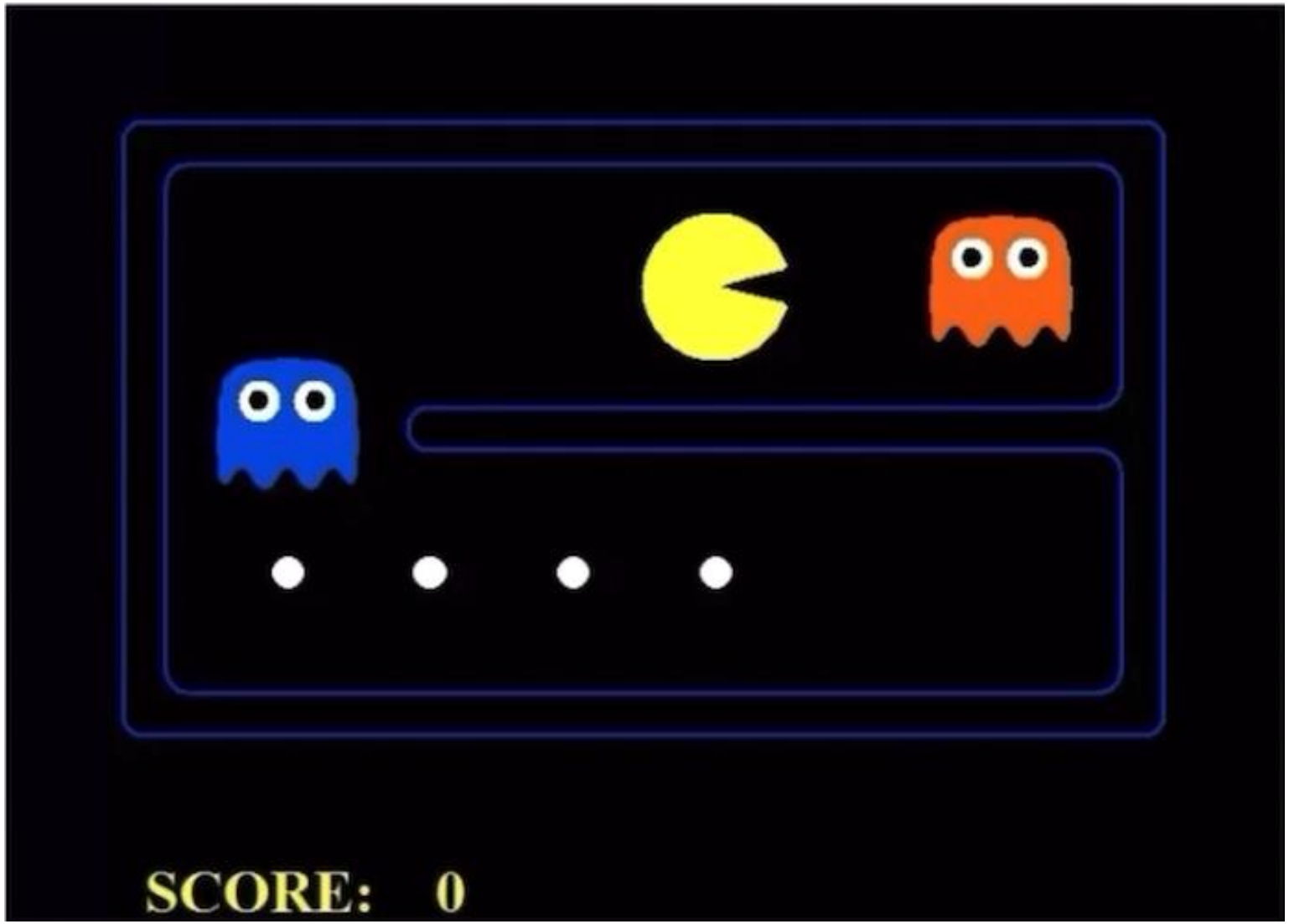
- 对于完美理性玩家是最优的，否则呢?



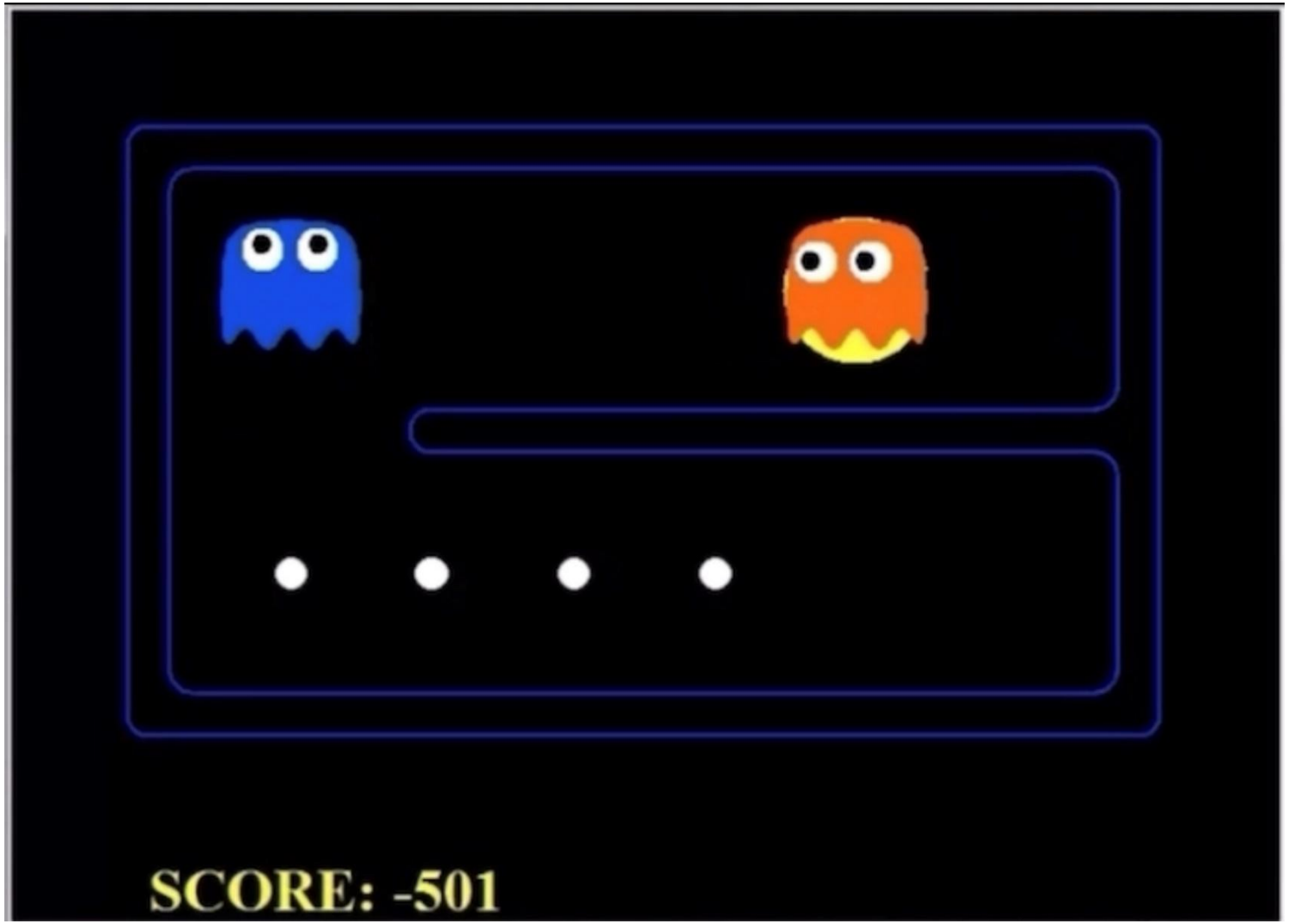




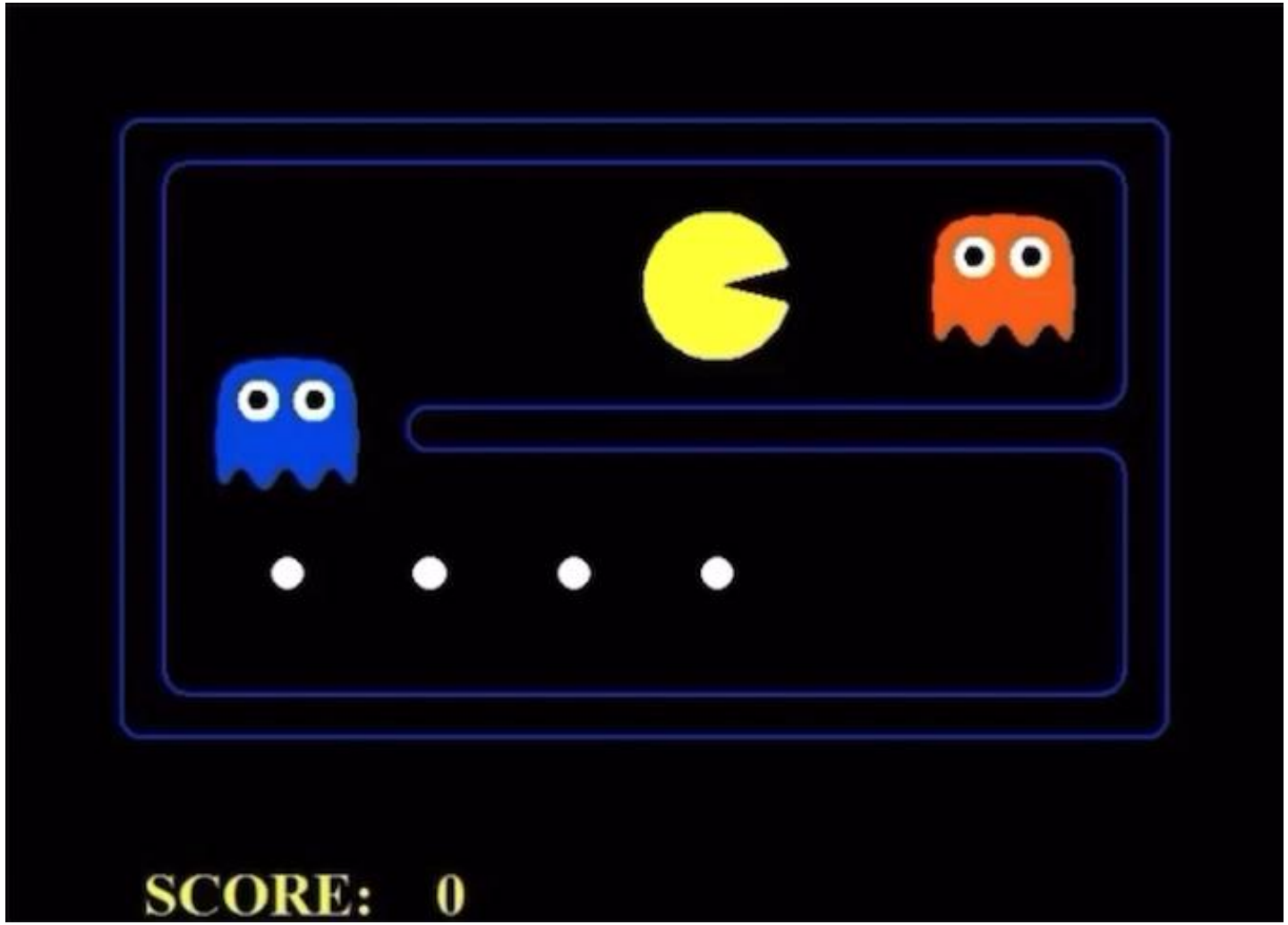
Pacman：假定鬼是理智的



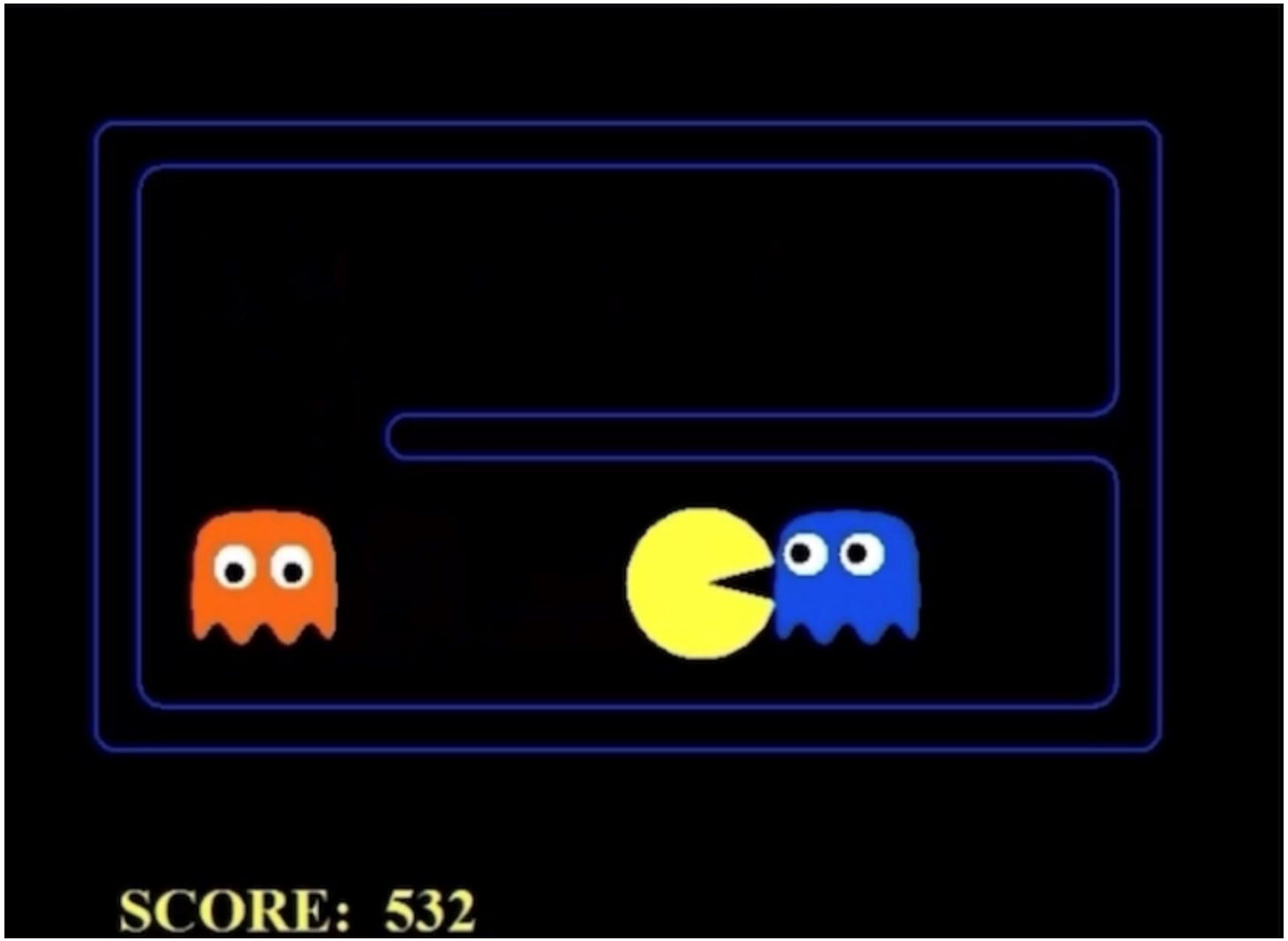
Pacman：假定鬼是理智的



Pacman：鬼不大聪明

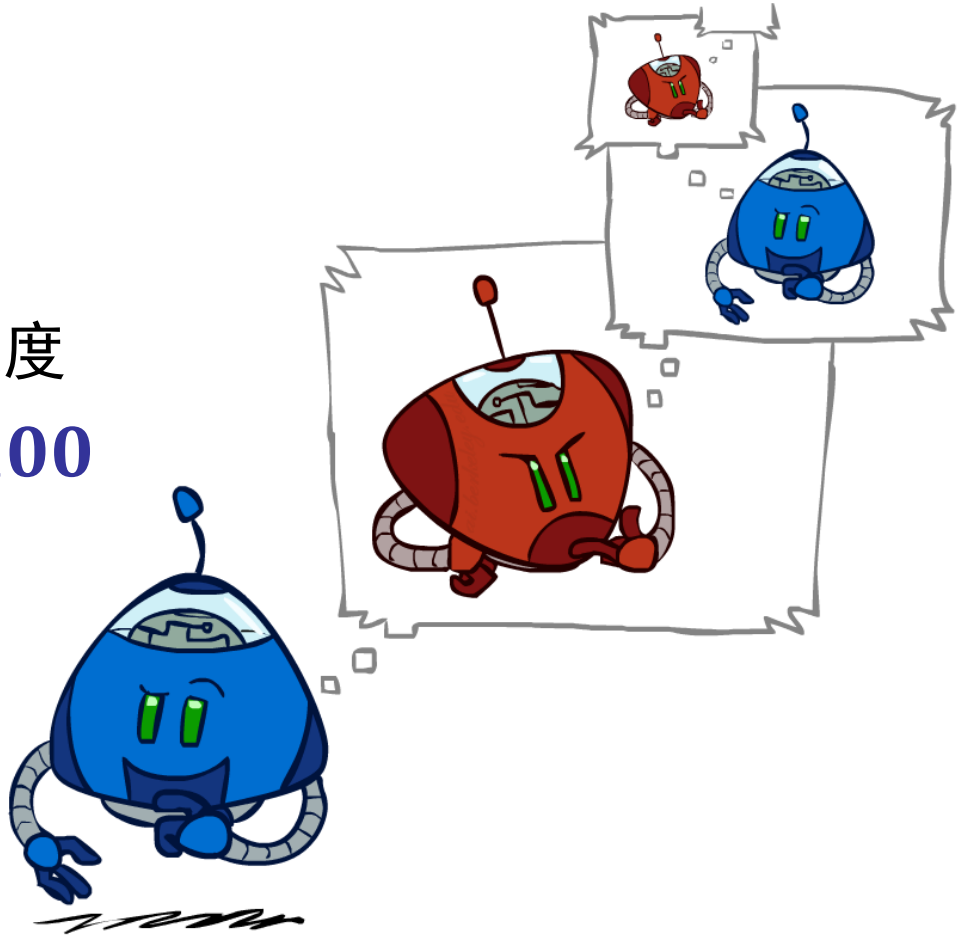


Pacman：鬼不大聪明



极大极小搜索的效率

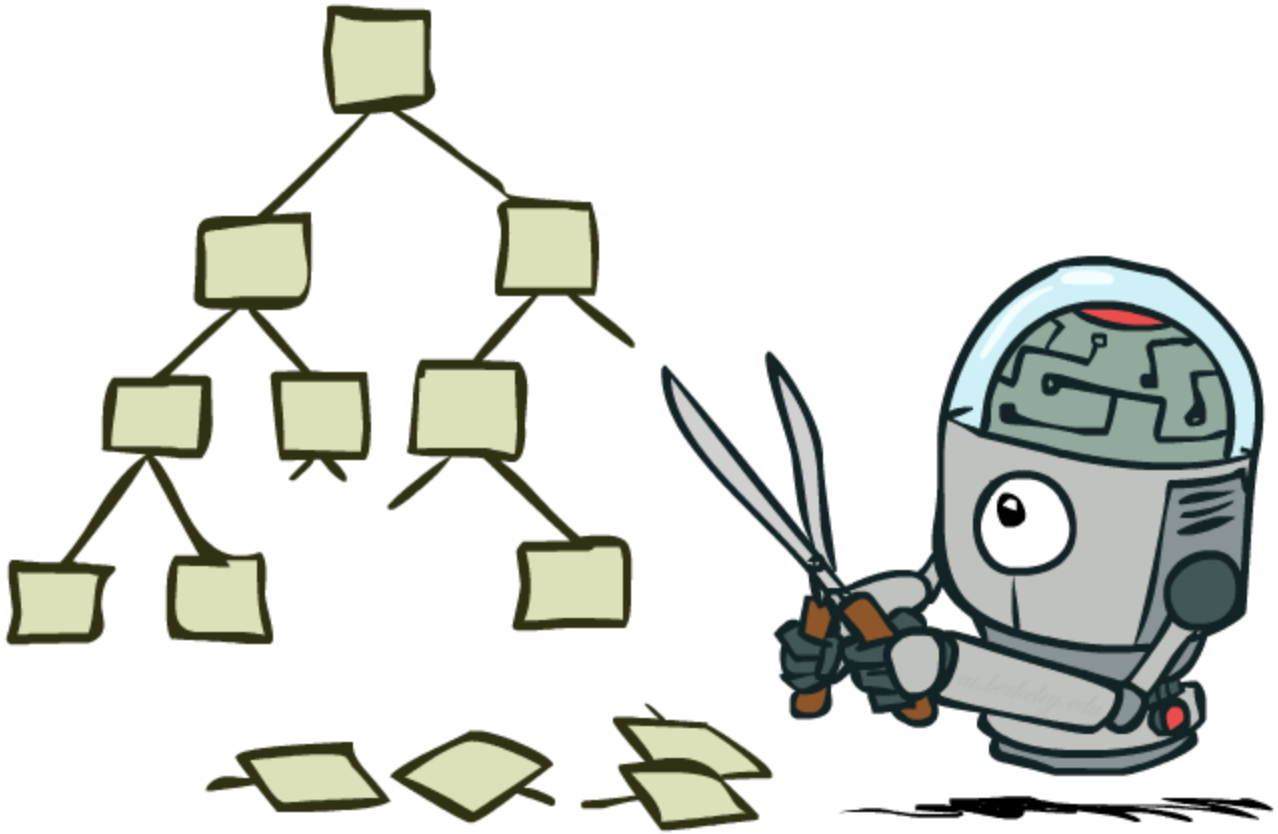
- **效率如何?**
  - **与DFS一致**
  - 时间复杂度: $O(b^m)$
  - 空间复杂度: $O(bm)$
  - b:(估计)branching数量，m：(估计)深度
- **举例：对于国际象棋, $b \approx 30, m \approx 100$**
- **举例：对于围棋，$b \approx 30, m \approx 170$**
  - 搜索完整的博弈树是不可行的，
  - 但是，我们需要这样做吗？



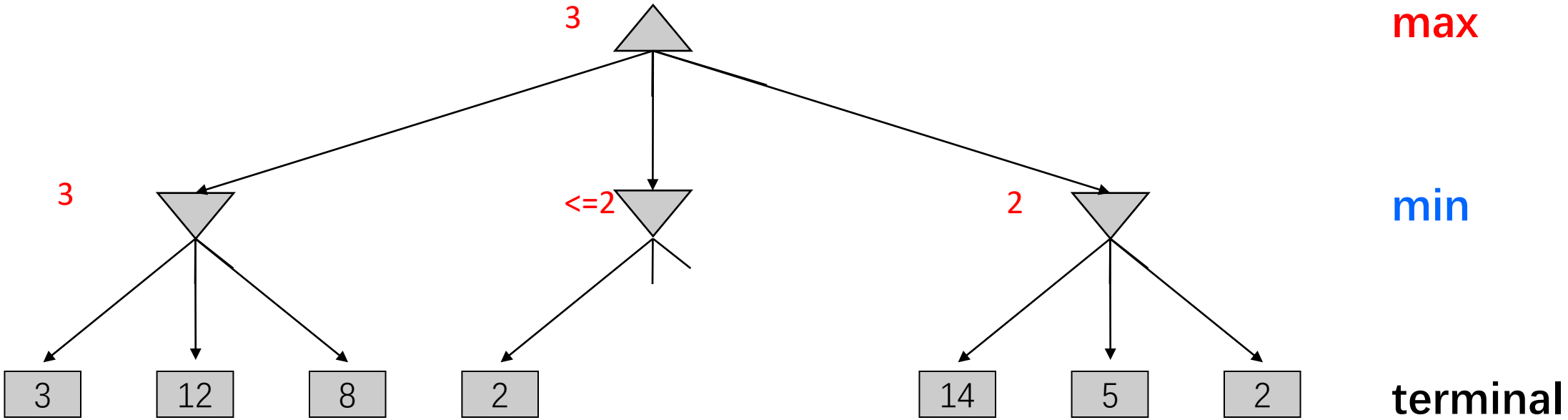
资源限制



博弈树剪枝

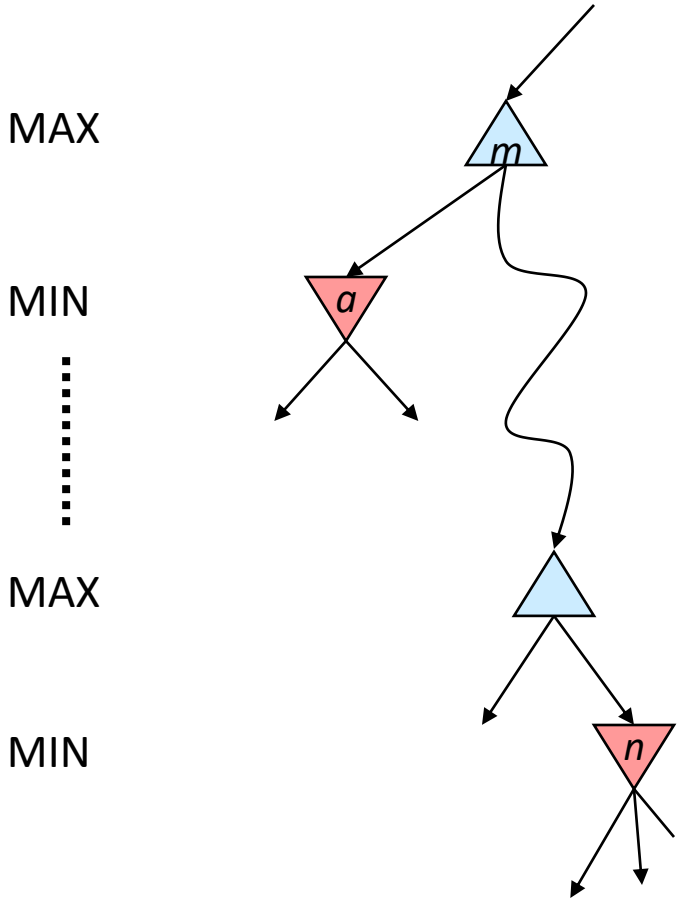


极小极大搜索的示例



Alpha-Beta 剪枝

- 一般流程 (**MIN** 版本)
  - 假设我们正在计算节点***n***的 ***MIN-VALUE***
  - 我们需要遍历***n***的子节点
  - ***n***对于子节点最小值的估计会不断下降
  - 谁会在意***n***节点的价值***?*** ***MAX***
  - 设***a***是***n***到达根节点路径上各种***MAX***可以得到的最大值
  - 如果***n***变得比***a***更差，***MAX***将避免它，从而我们可以不再考虑***n***的其他子节点
- **MAX** 版本可以对称地考虑



Alpha-Beta 算法实现

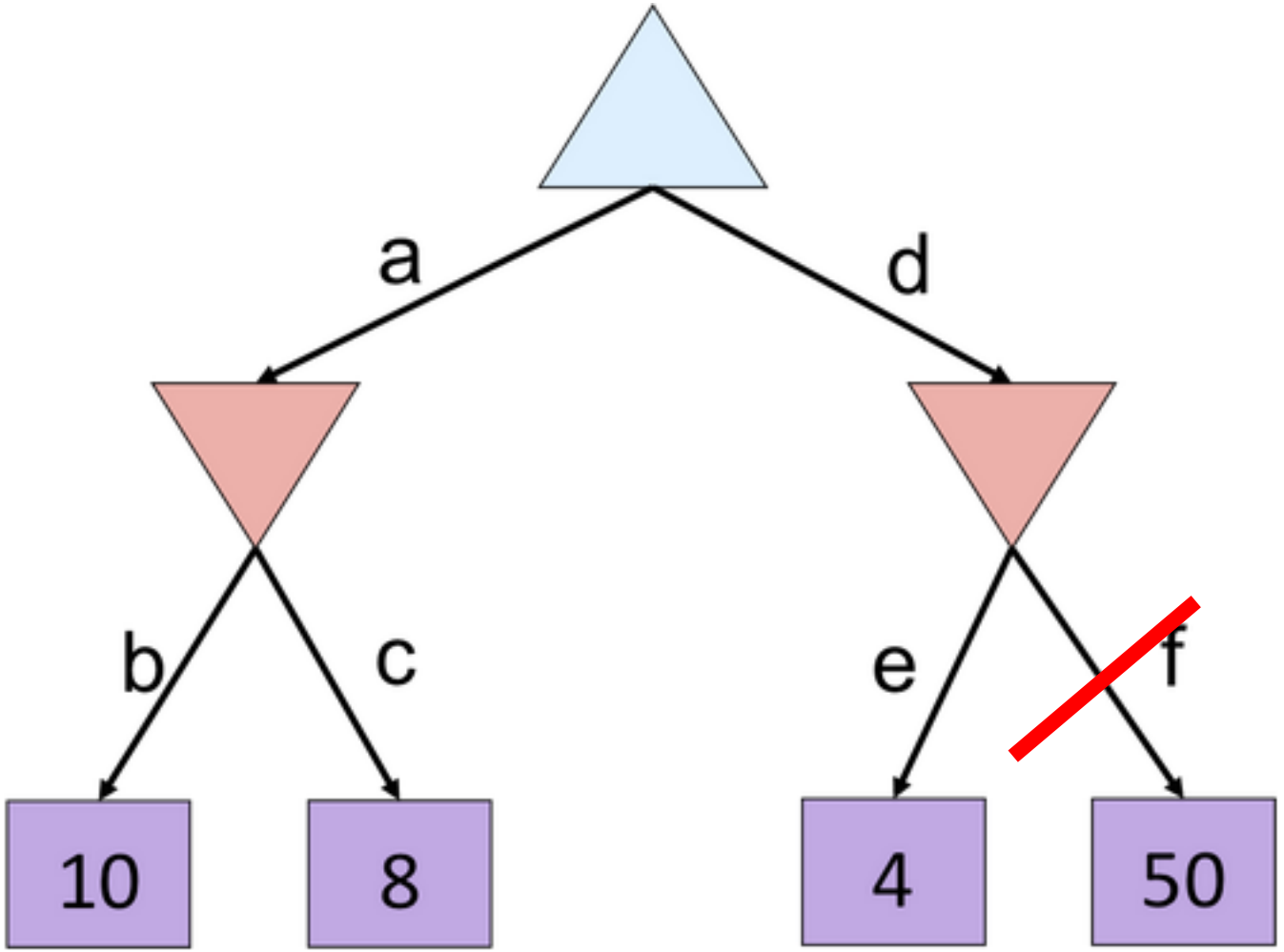
α: MAX's best option on path to root
β: MIN's best option on path to root

```
def value(state):
    if the state is a terminal state: return the state's utility
    if the next agent is MAX: return max-value(state)
    if the next agent is MIN: return min-value(state)
```

```
def max-value(state, α, β):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor, α, β))
        if v ≥ β return v
        α = max(α, v)
    return v
```

```
def min-value(state , α, β):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor, α, β))
        if v ≤ α return v
        β = min(β, v)
    return v
```

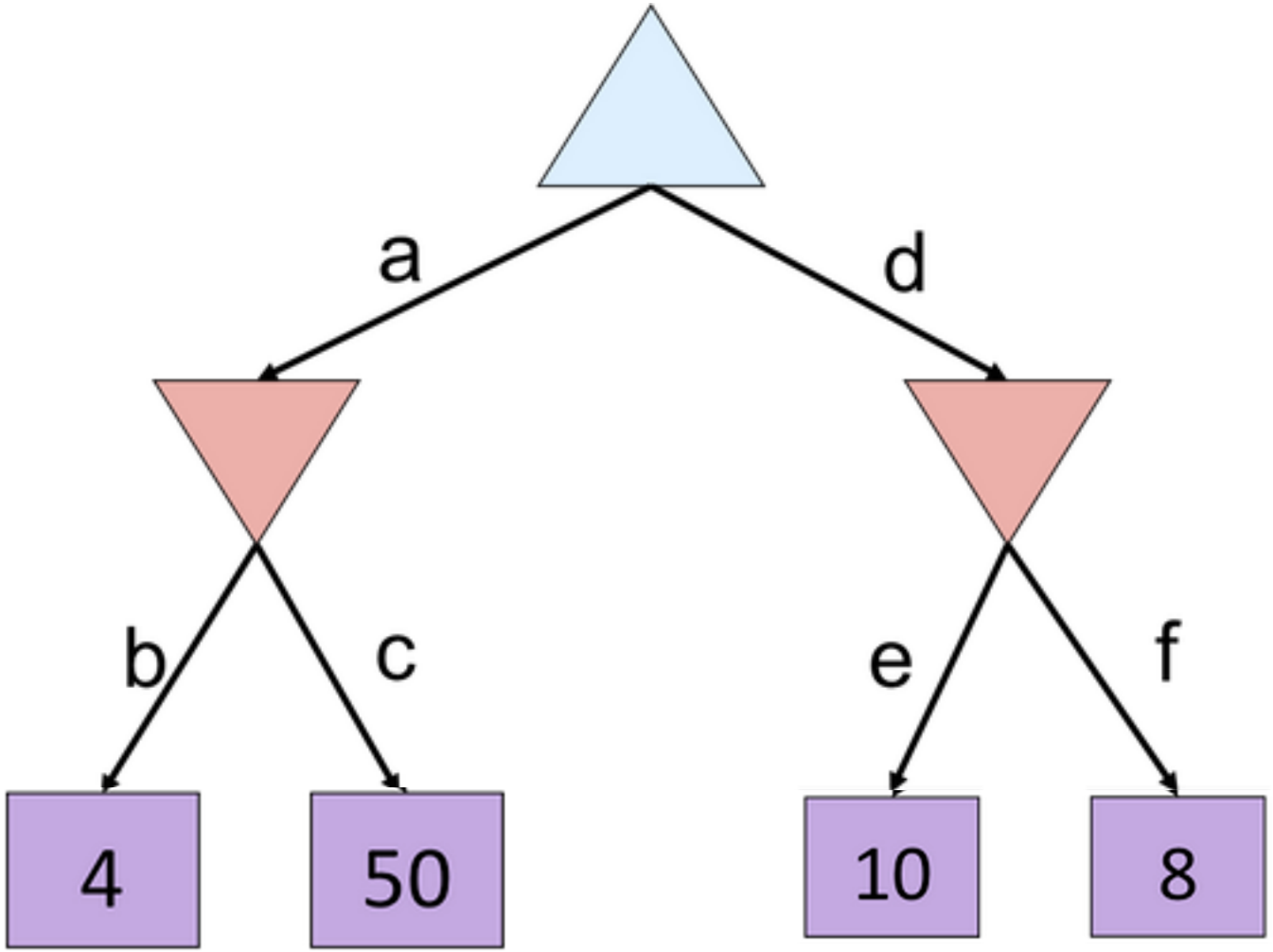
Alpha-Beta 测验



max

min

Alpha-Beta 测验

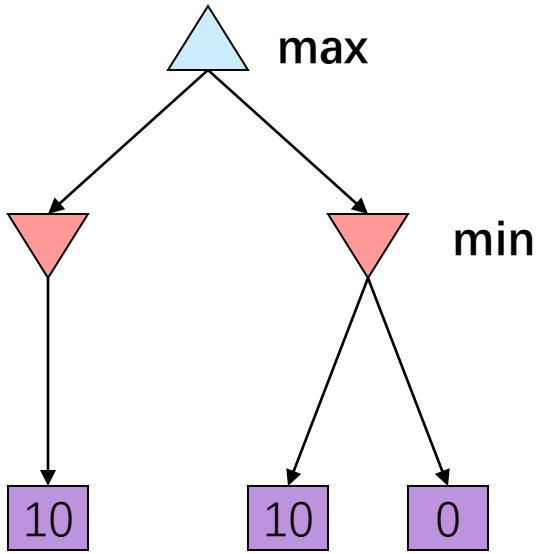


max

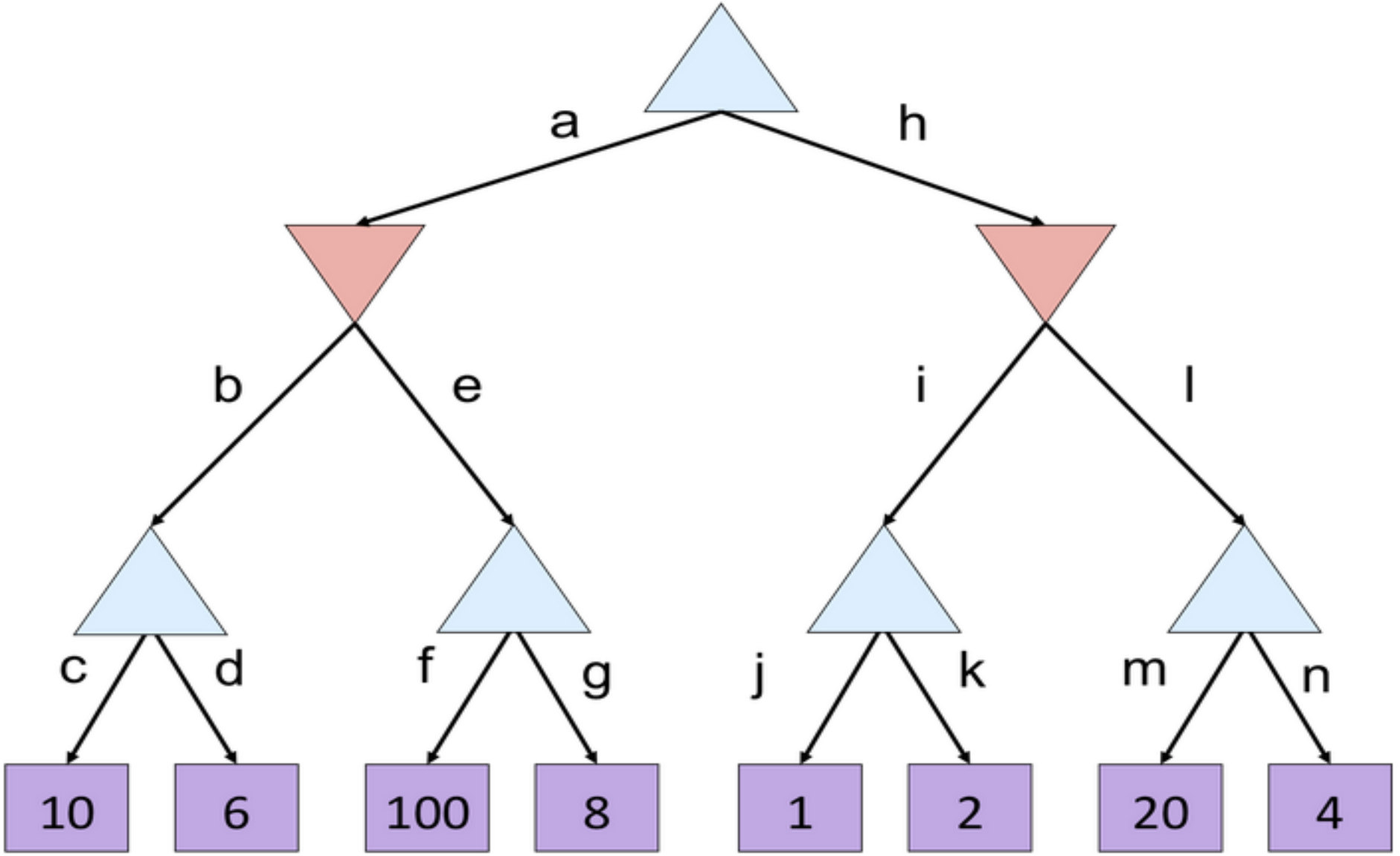
min

Alpha-Beta 剪枝的性质

- 这种剪枝方式不会对计算根节点的极小极大值产生影响！
- 中间节点的值可能是**错误**的
  - 重要提示：根节点的子节点可能具有错误的值
  - 因此，最朴素的版本无法进行动作选择
- 良好的子节点排序可以提高剪枝的效率
- 通过“完美排序”：
  - 时间复杂度降至 $O(bm/2)$
  - 双倍可解深度！
  - 然而，对于国际象棋等任务进行完备搜索仍然是不可行的…



Alpha-Beta 测验2

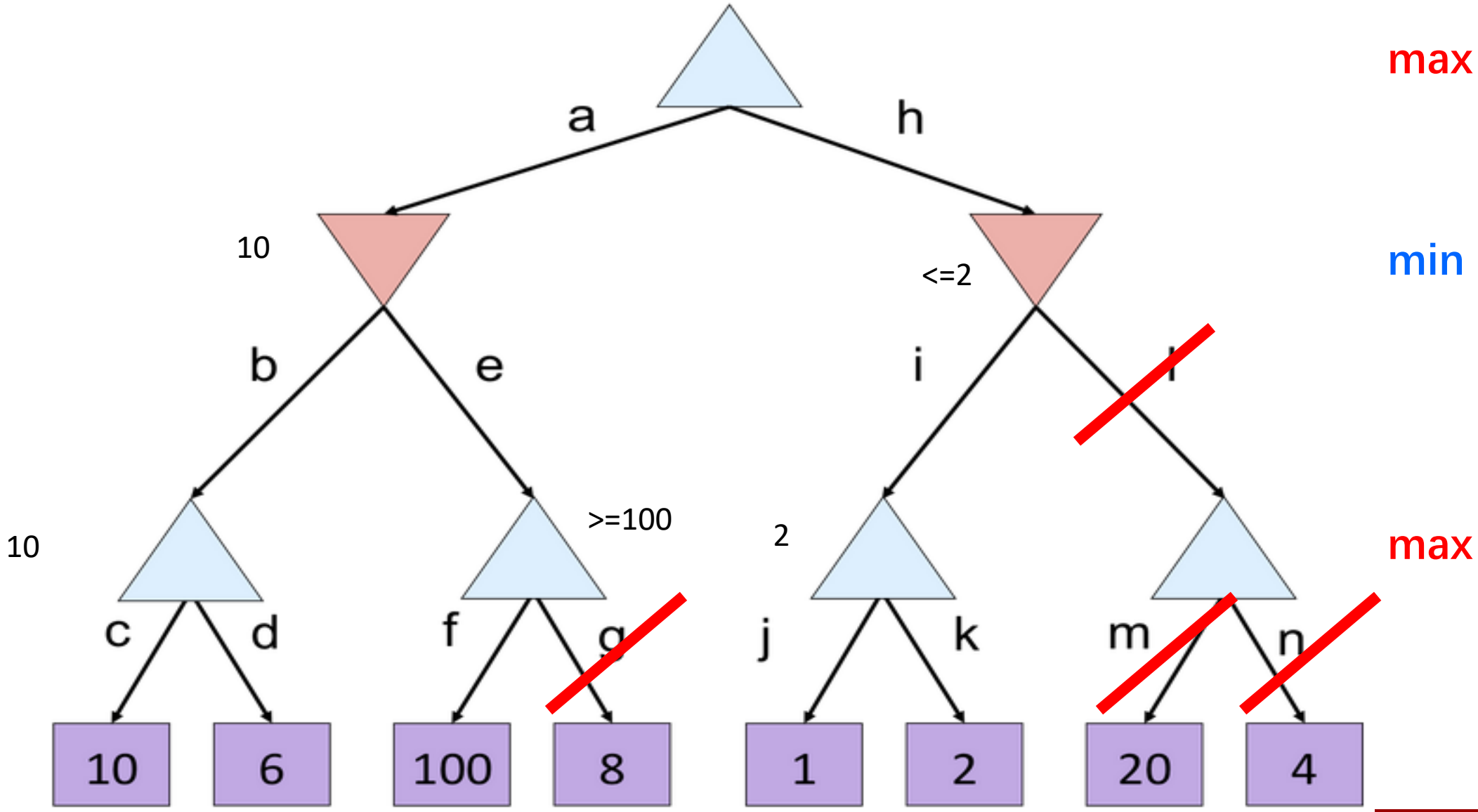


max

min

max

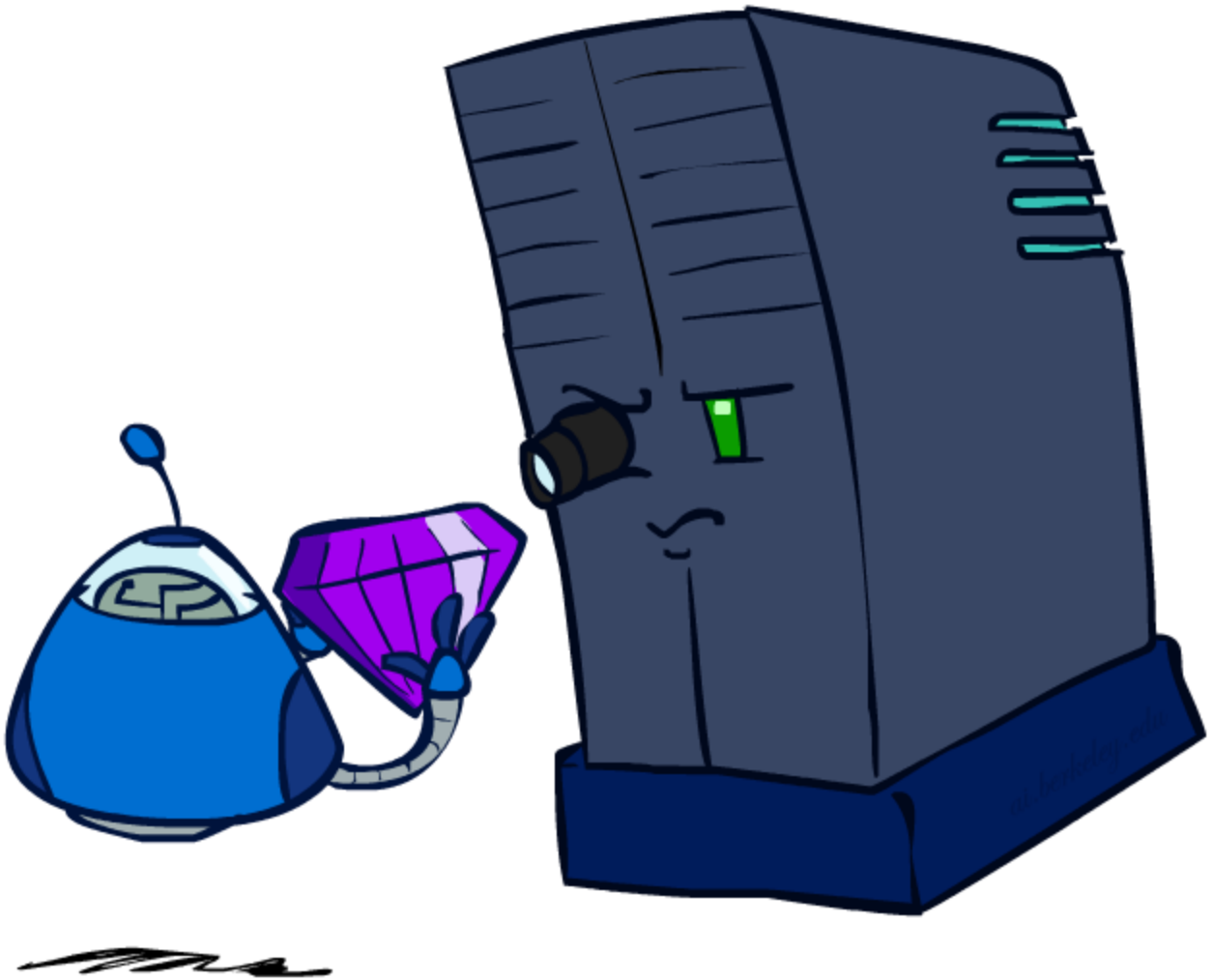
Alpha-Beta 测验2



资源限制

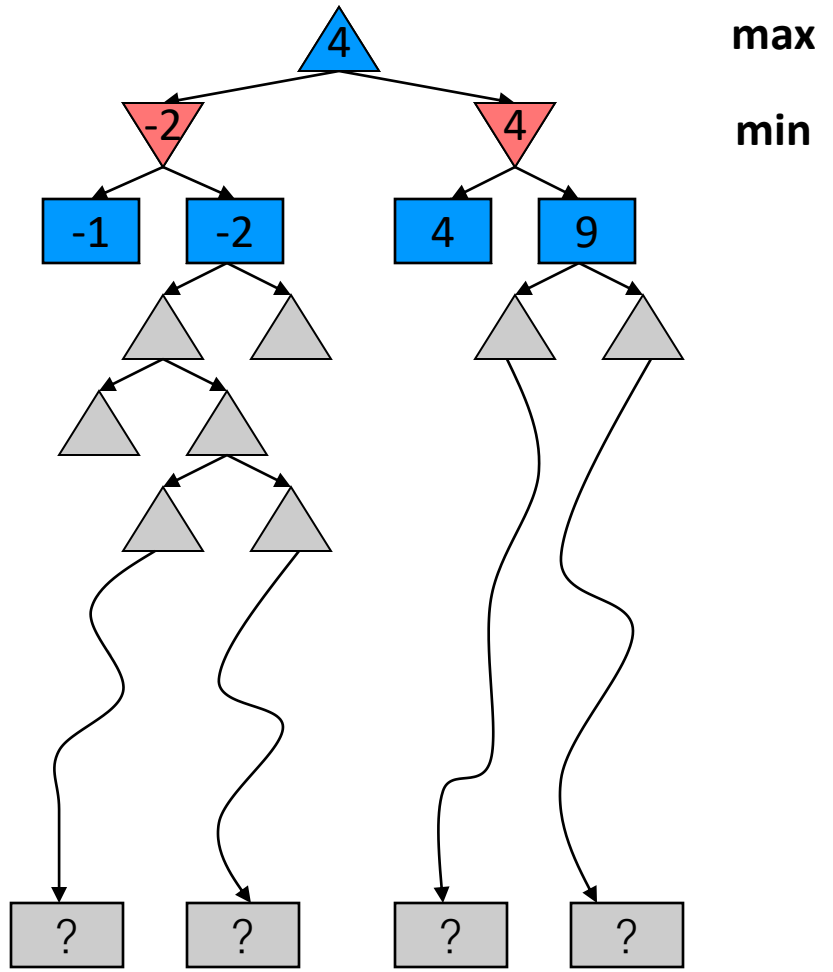


估值函数



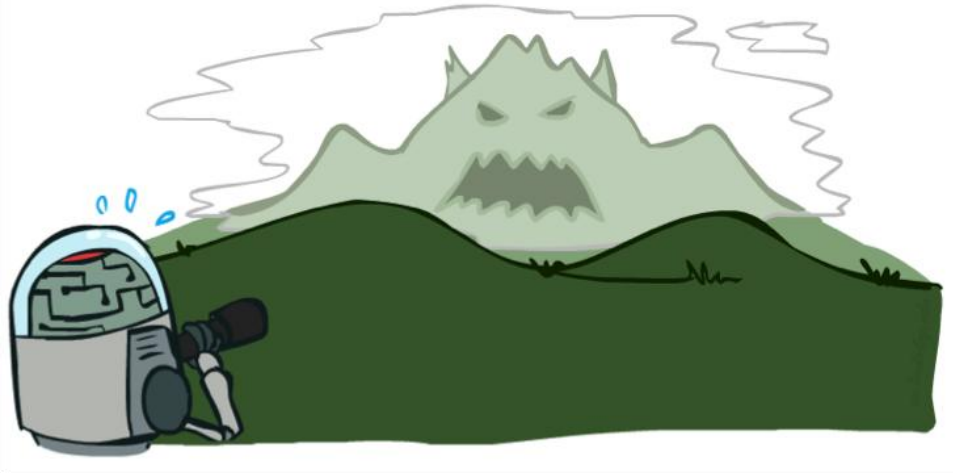
资源限制

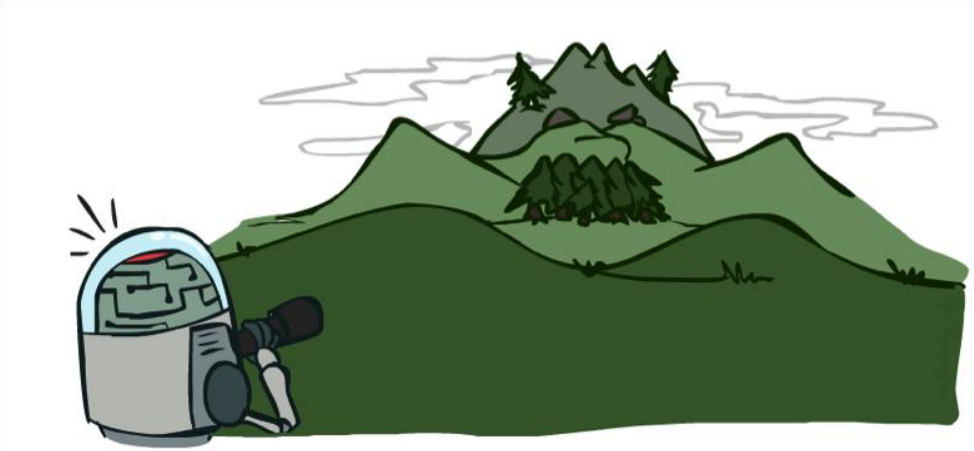
- **问题: 针对现实情况中的博弈，不可能走到叶子节点**
- **解决方案: 深度受限搜索**
  - **搜索只在有限深度中进行**
  - **对于非叶节点，将终态效用替换为估值函数**
- 示例:
  - 假设我们有100秒，每秒可以搜索10k个节点
  - 那我们每次可以搜索1M个节点
  - 因此α-β 可以达到8的搜索深度(一个不错的象棋程序)
- 不再具备最优性保证
- 更大的搜索深度会带来显著的差异



搜索深度的影响

- 估值函数永远是不完美的
- 评估函数嵌入博弈树越深，评估函数的质量对算法结果的影响就越小
- 这是特征复杂度与计算复杂度之间权衡的一个重要示例





谢谢

北京大学
PEKING UNIVERSITY

