《物理与人工智能》

16. 对抗搜索

授课教师：马滟青

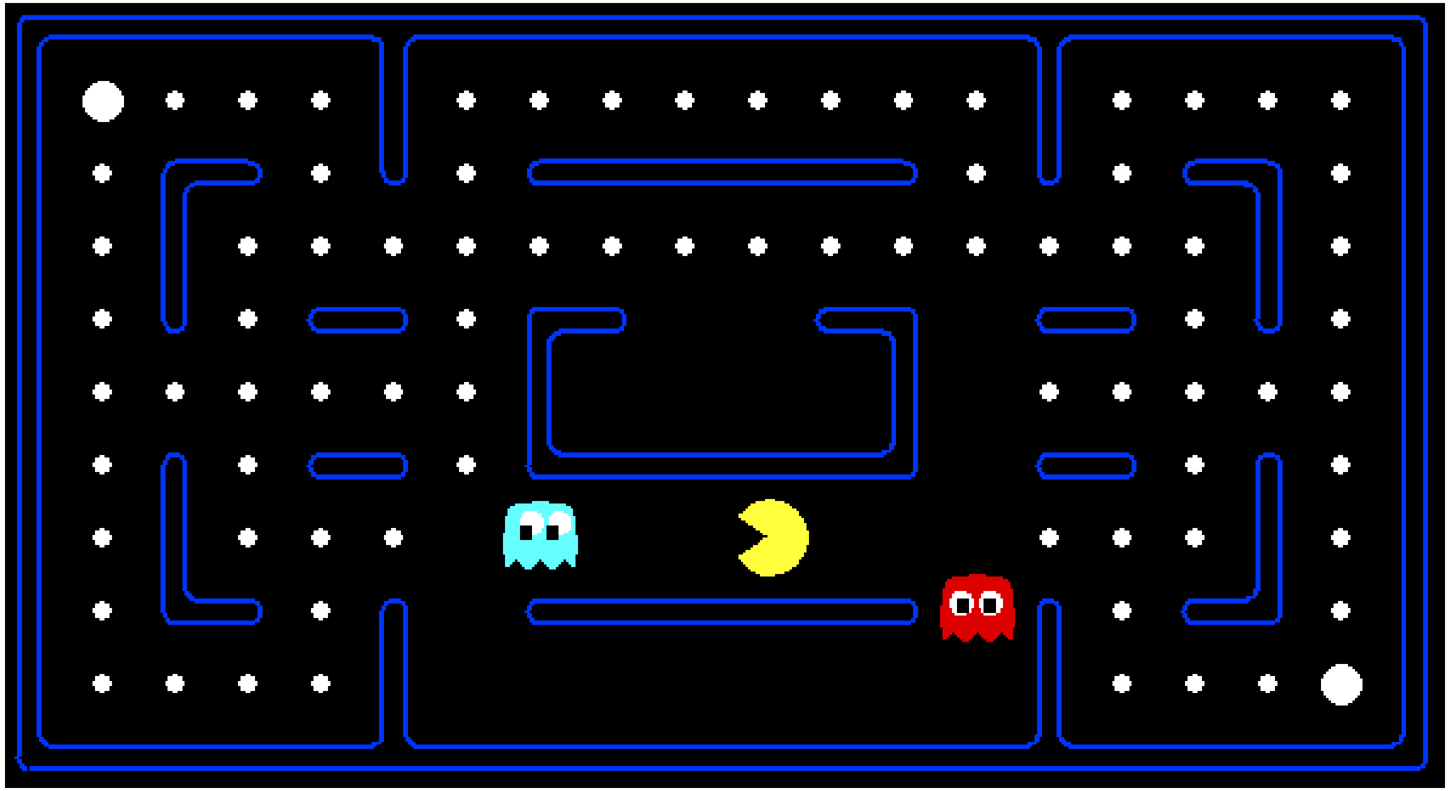
2025/10/27（第七周）

鸣谢：基于计算机学院《人工智能引论》课程组幻灯片

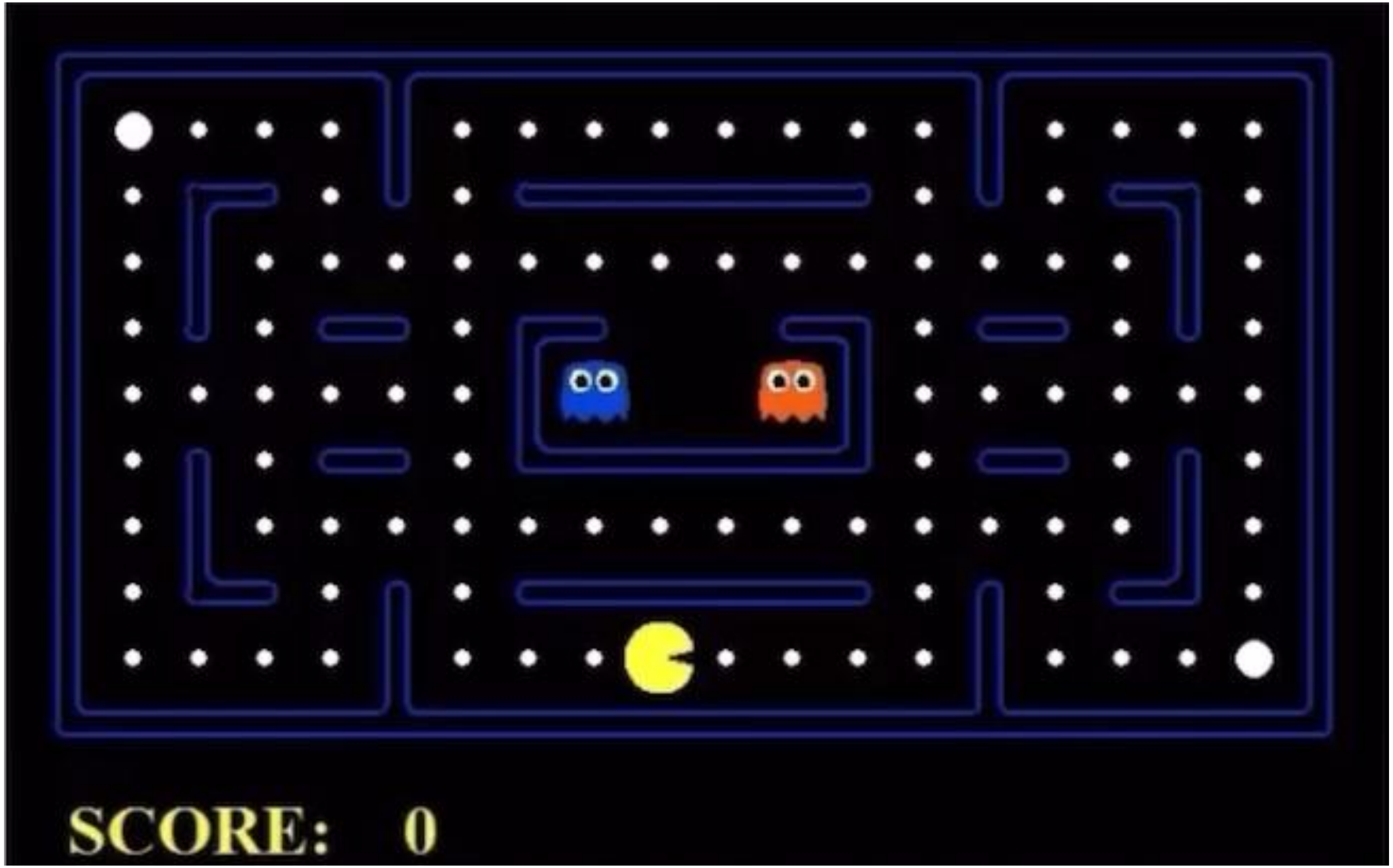
目录

- **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

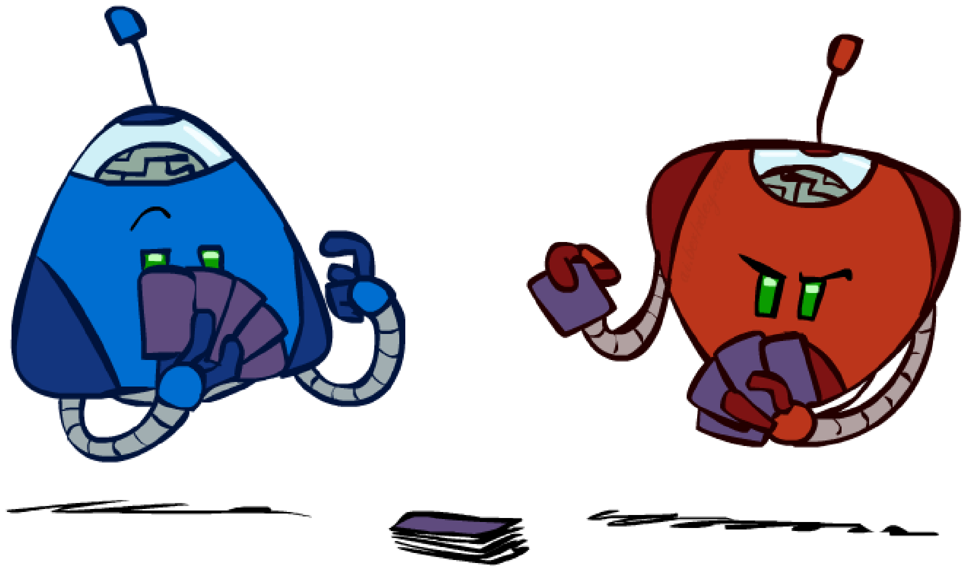


游戏类型

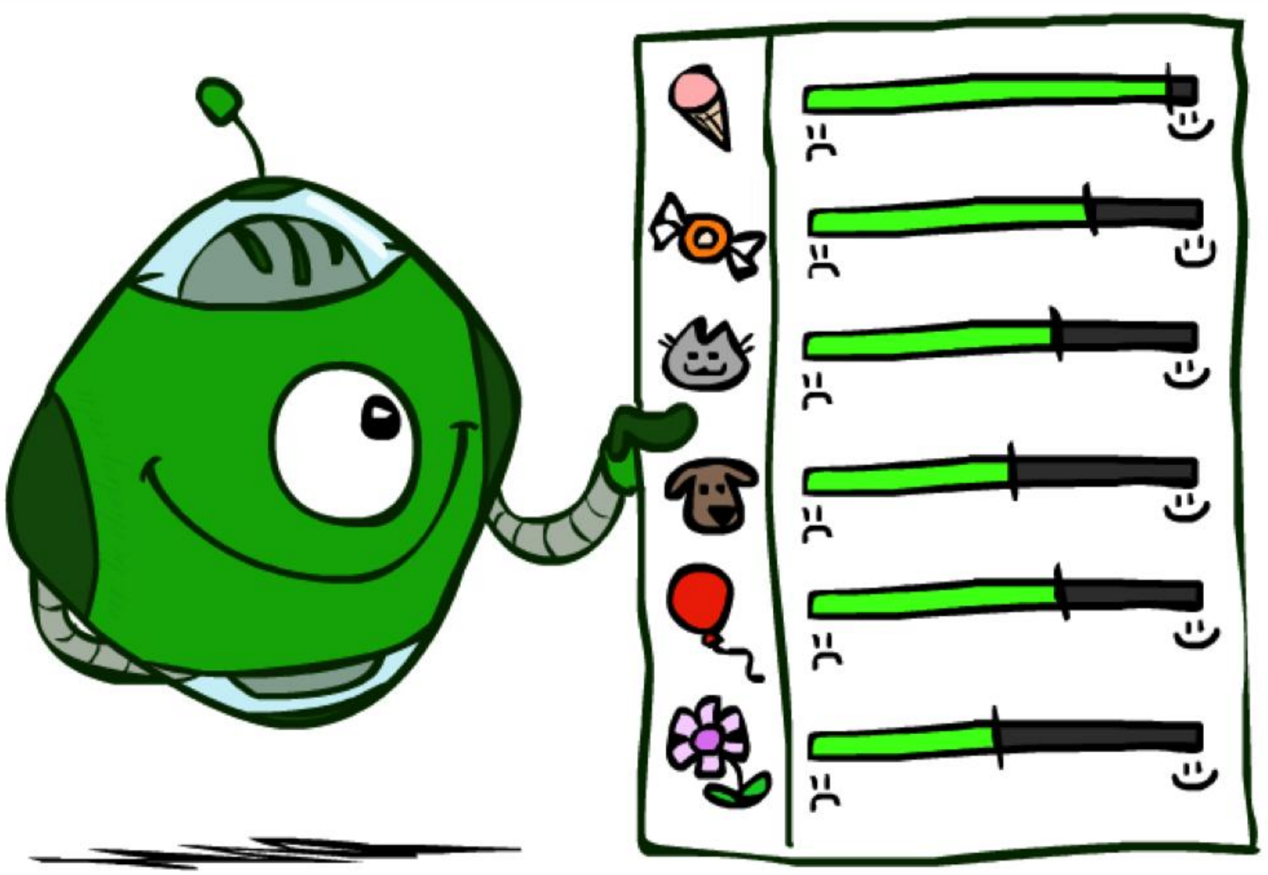
PEKING UNIVERSITY 北京大学

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

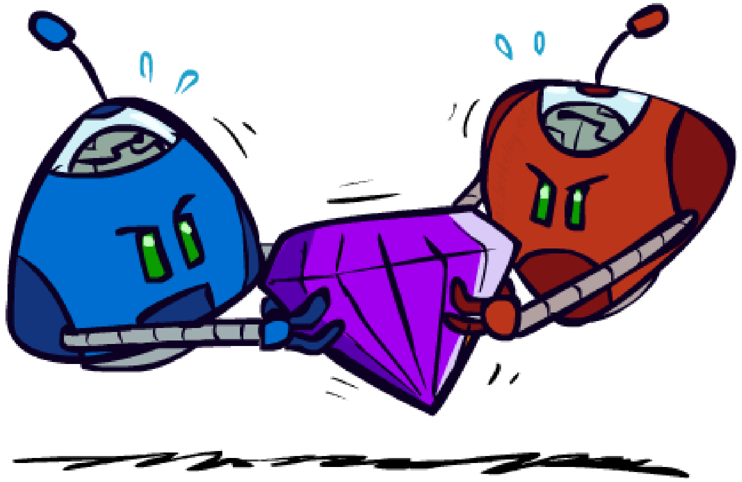
- **效用函数是将结果（世界的状态）映射到实数的函数，用于描述智能体的偏好**

- 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

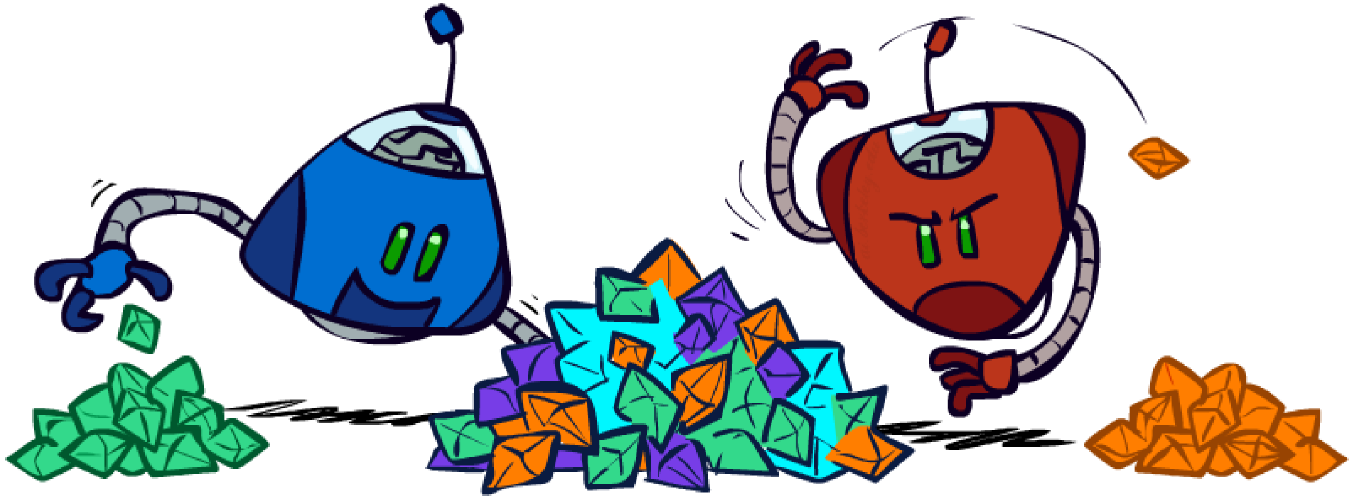
- 效用：关于回报大小的度量



- **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



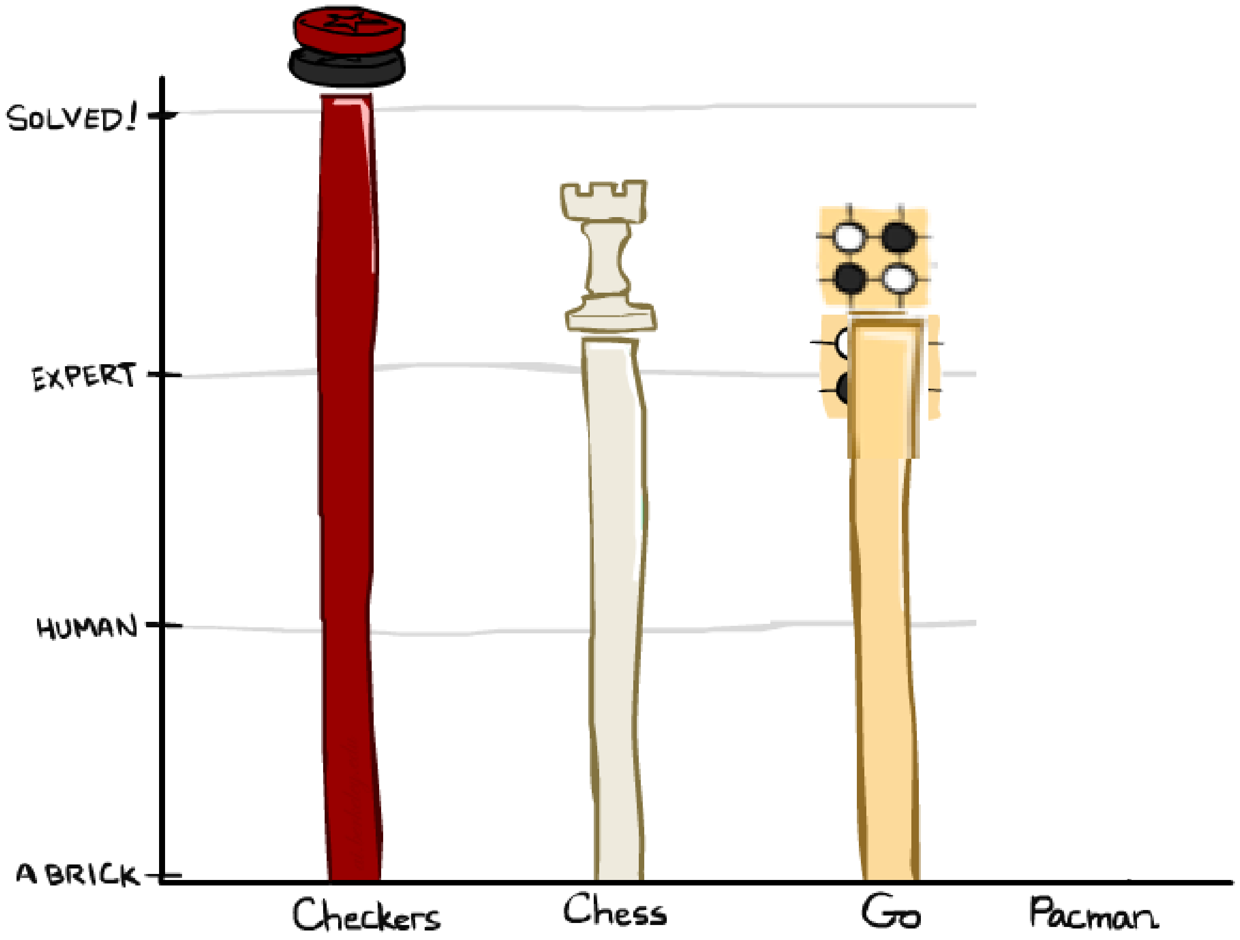
零和博弈

- **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。

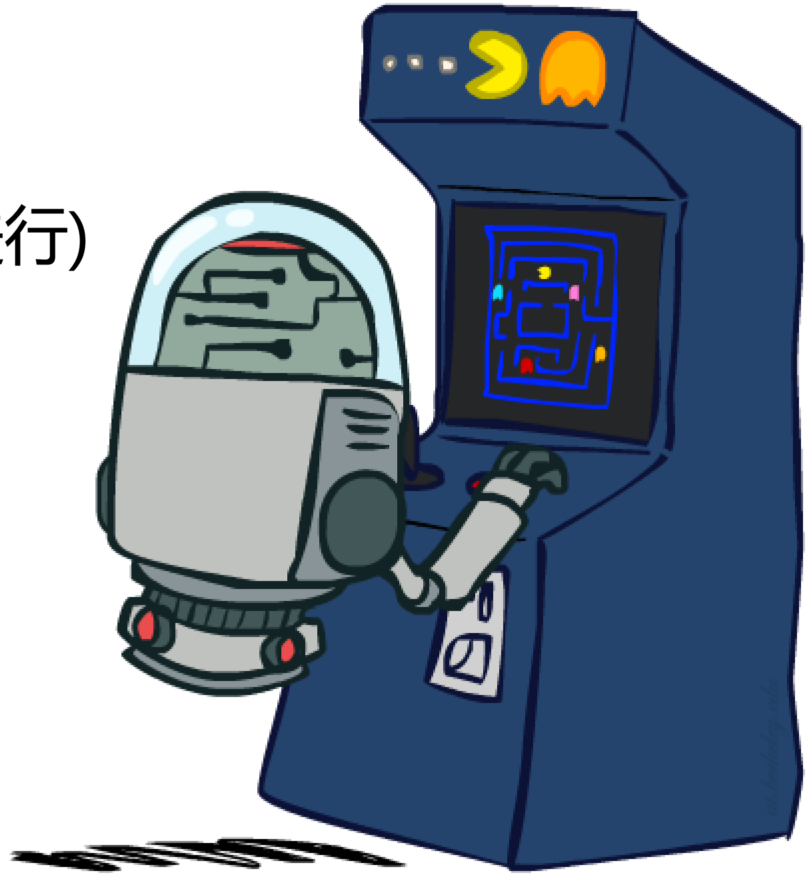
- **Go：2016、 Alpha GO** 打败了传奇李世石。**2017**年击败当时第一人柯洁。**Alpha GO**使用蒙特卡罗树搜索并通过训练学习评估函数。

- **Pacman? 2024?**



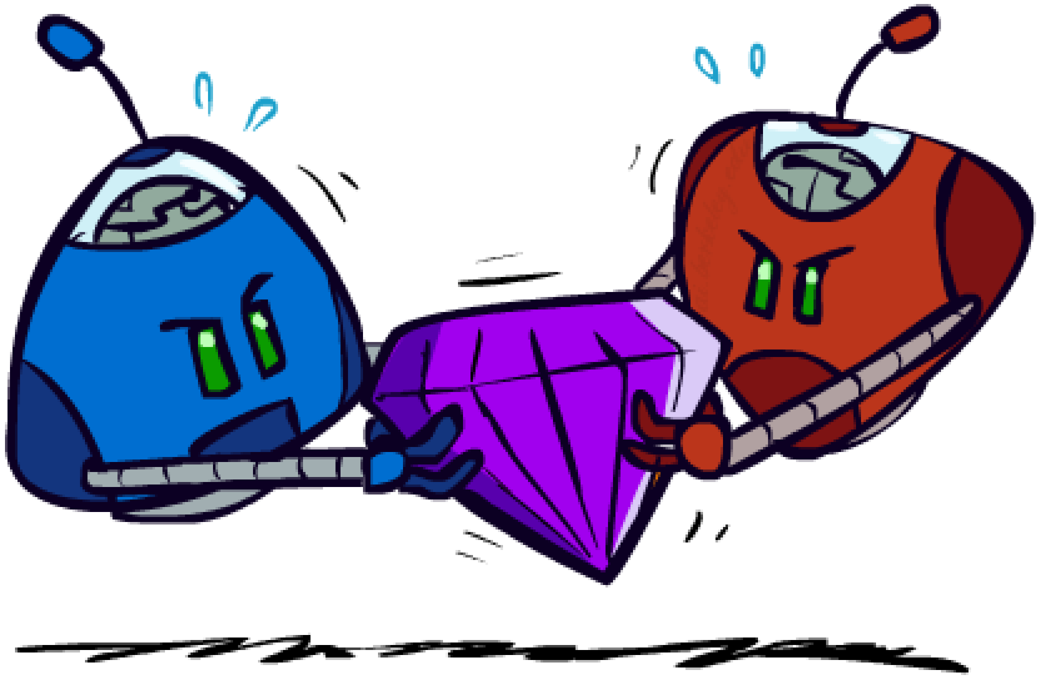
具有最终效用的确定性博弈

- 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): $P = \{1 \ldots N\}$ (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): $S \times A \to S$
  - 结束测试(Terminal Test): $S \to \{t, f\}$
  - 最终**效用**(Terminal Utilities): $S \times P \to R$

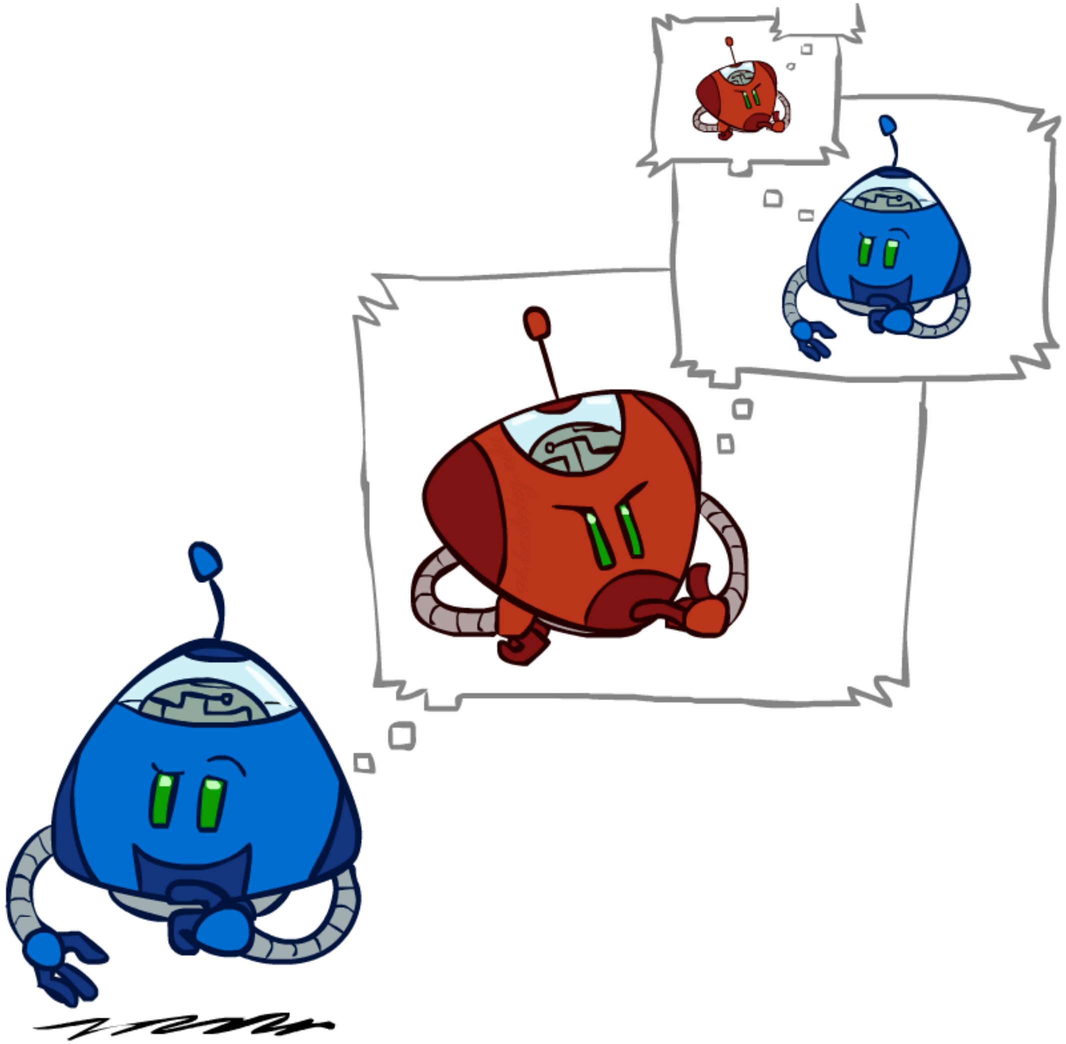


- 应对方案称为策略(policy): $S \to A$

对抗游戏/博弈



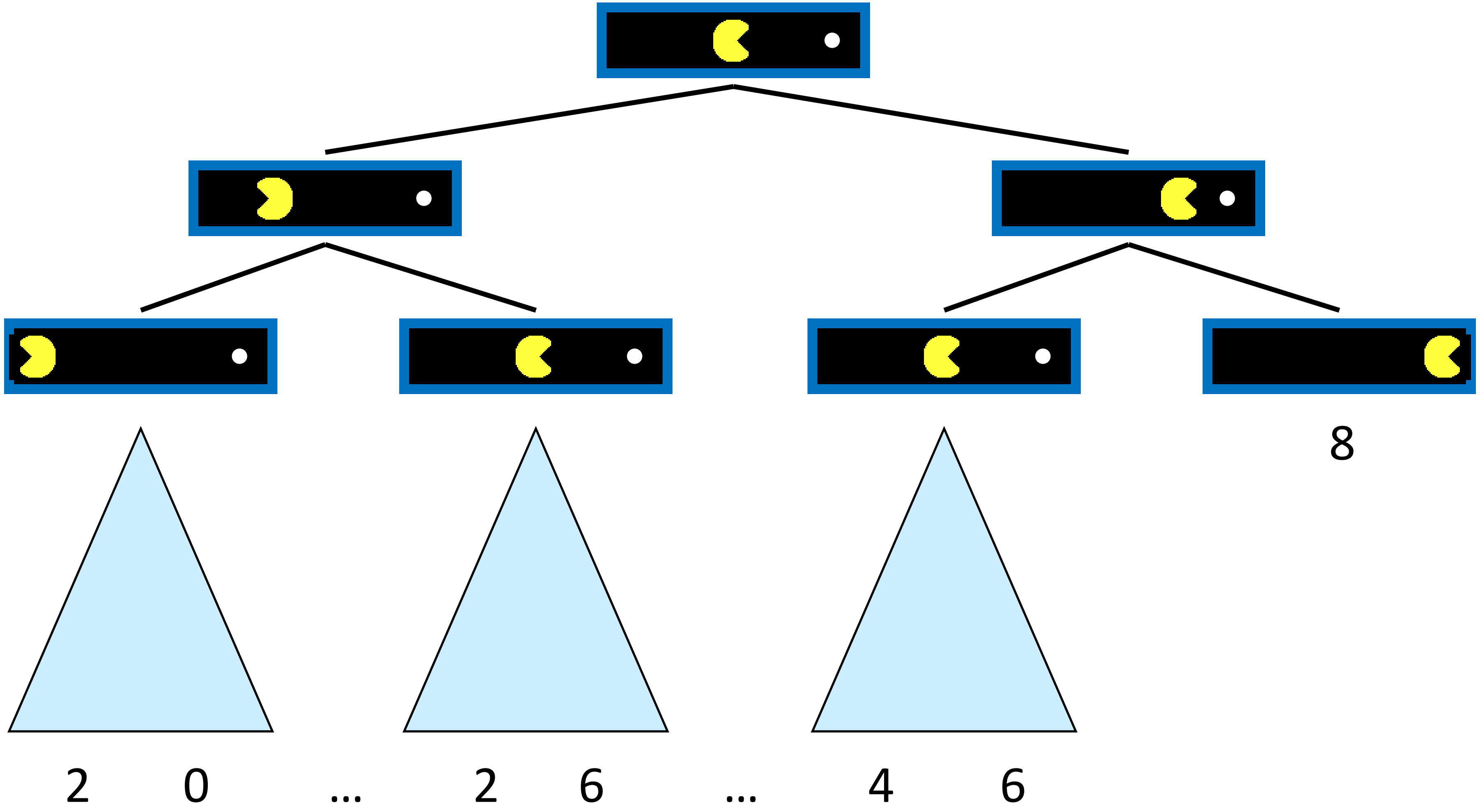
对抗搜索



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

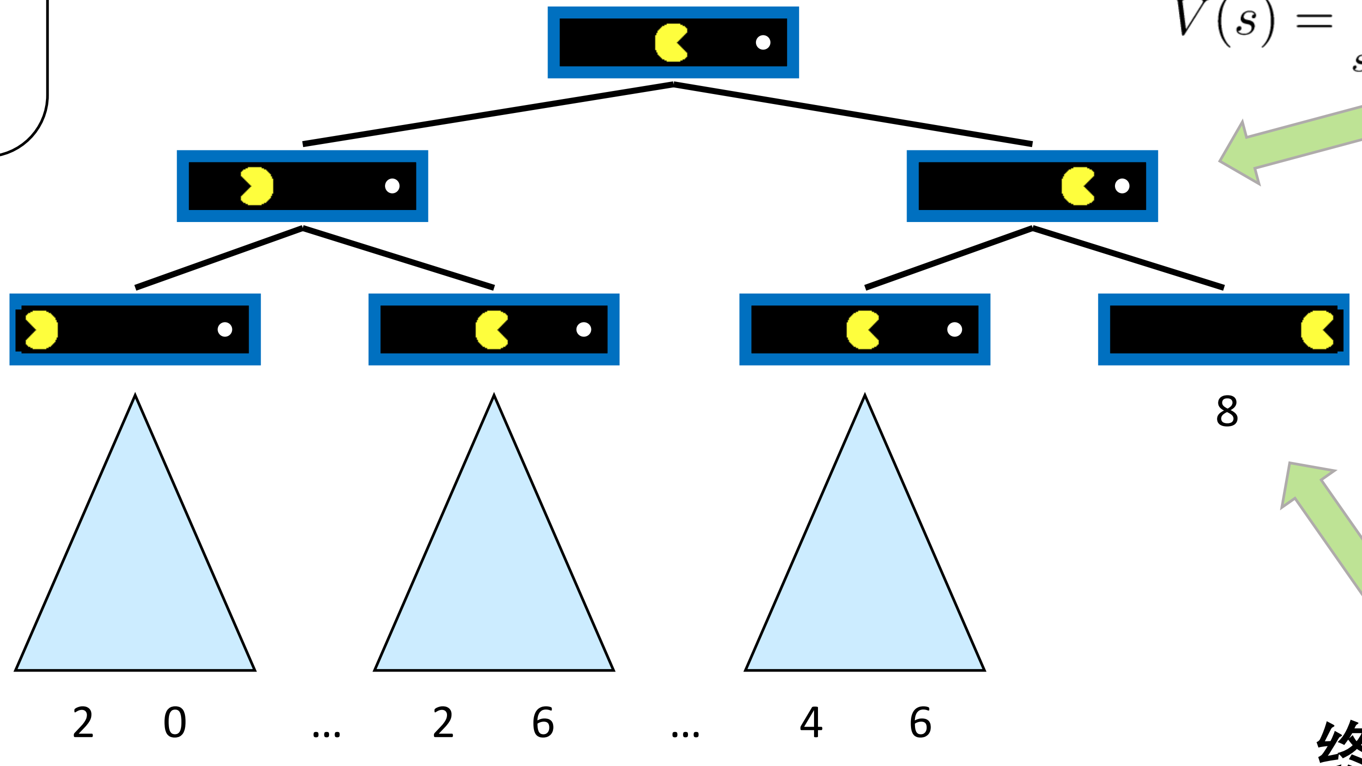


状态的价值(Value)

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:

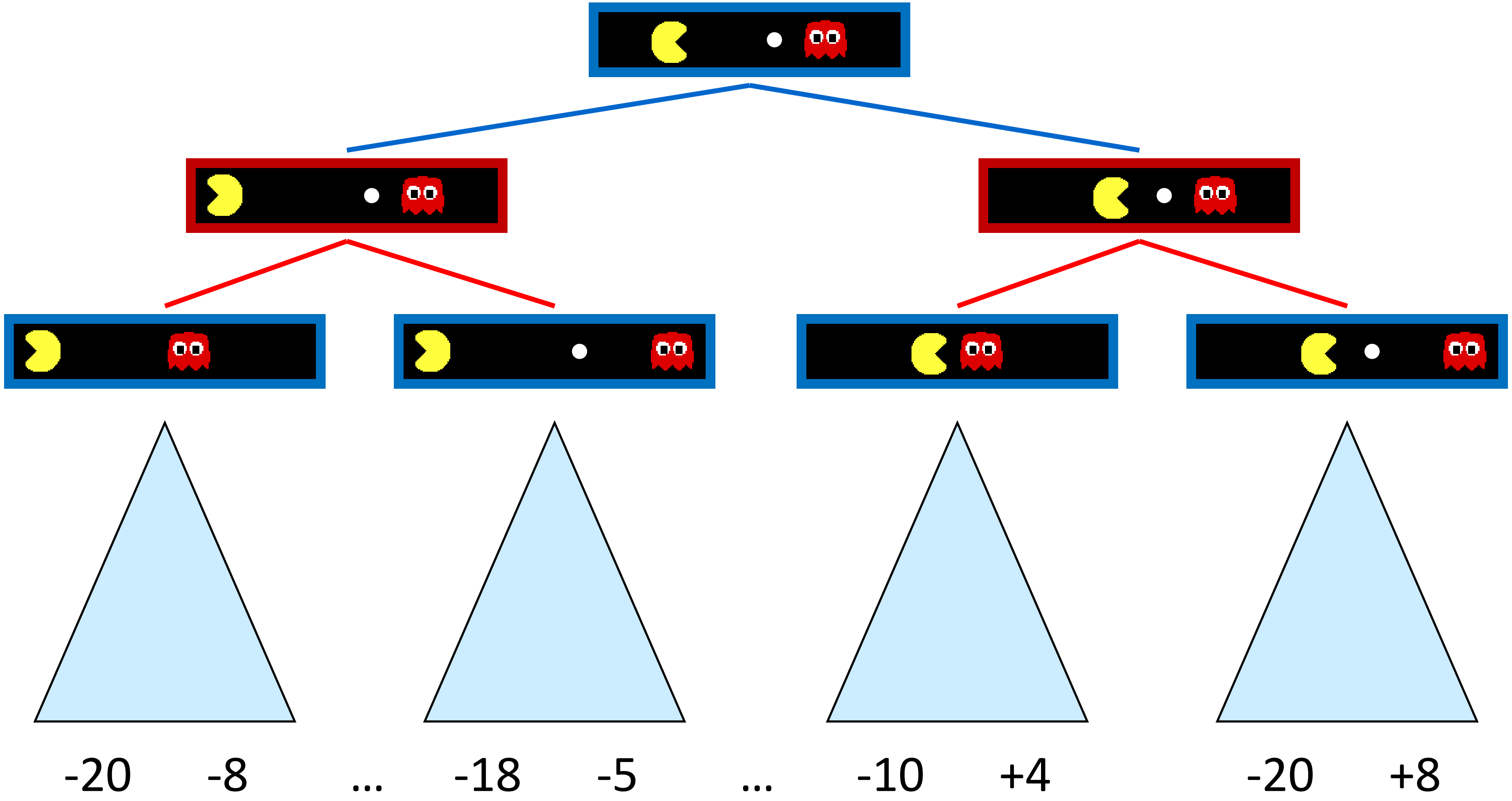
$$V(s) = \max_{s' \in \text{children}(s)} V(s')$$



终止状态:

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

对抗博弈的搜索树



极小极大搜索(Minimax Values)

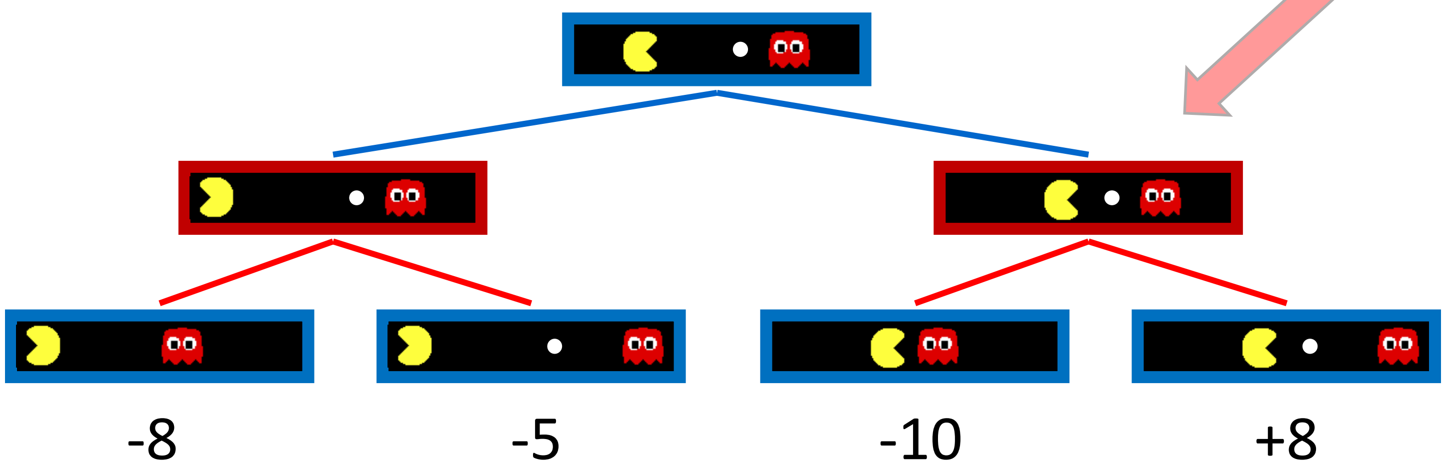
北京大学 PEKING UNIVERSITY

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:

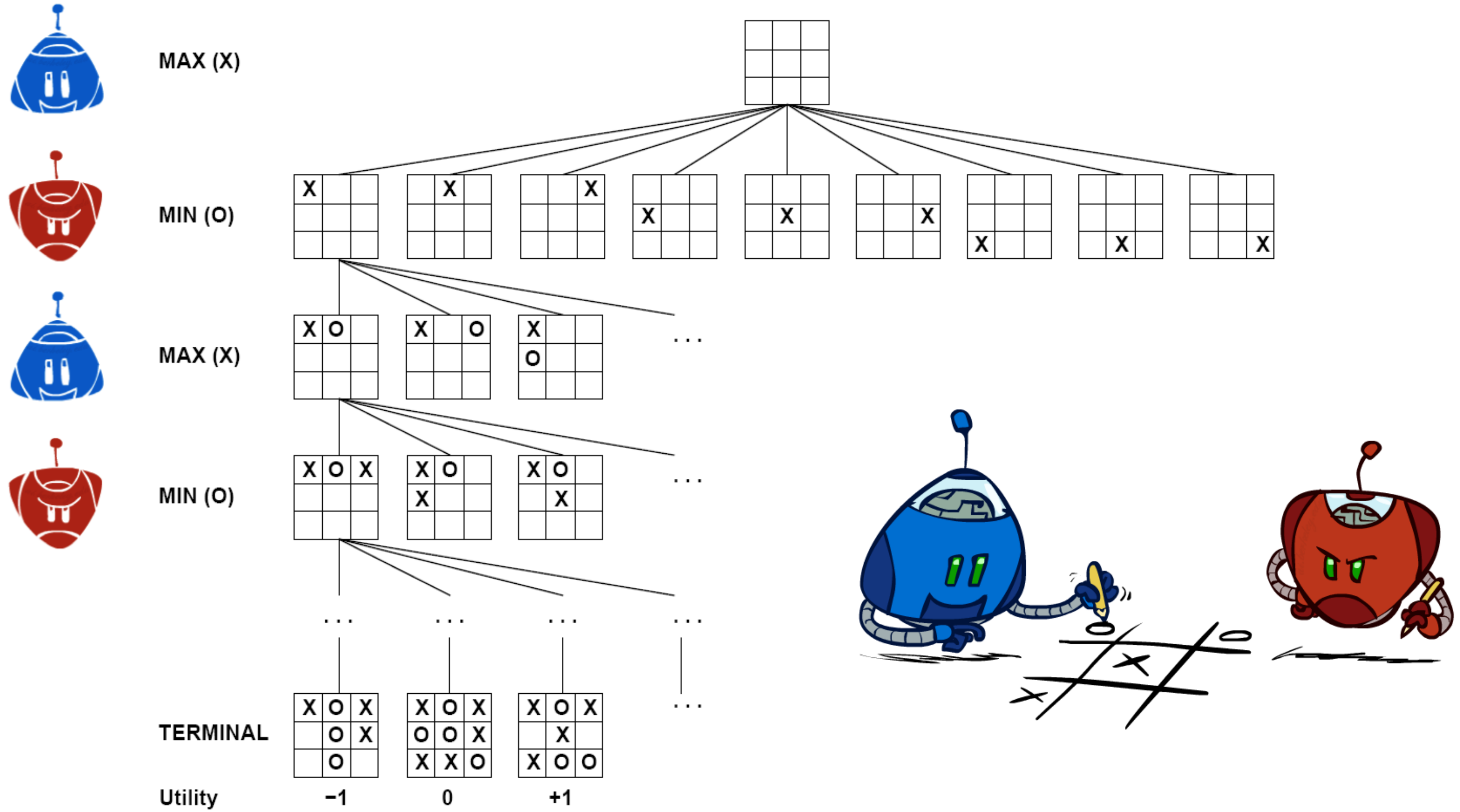
$$V(s') = \min_{s \in \text{successors}(s')} V(s)$$



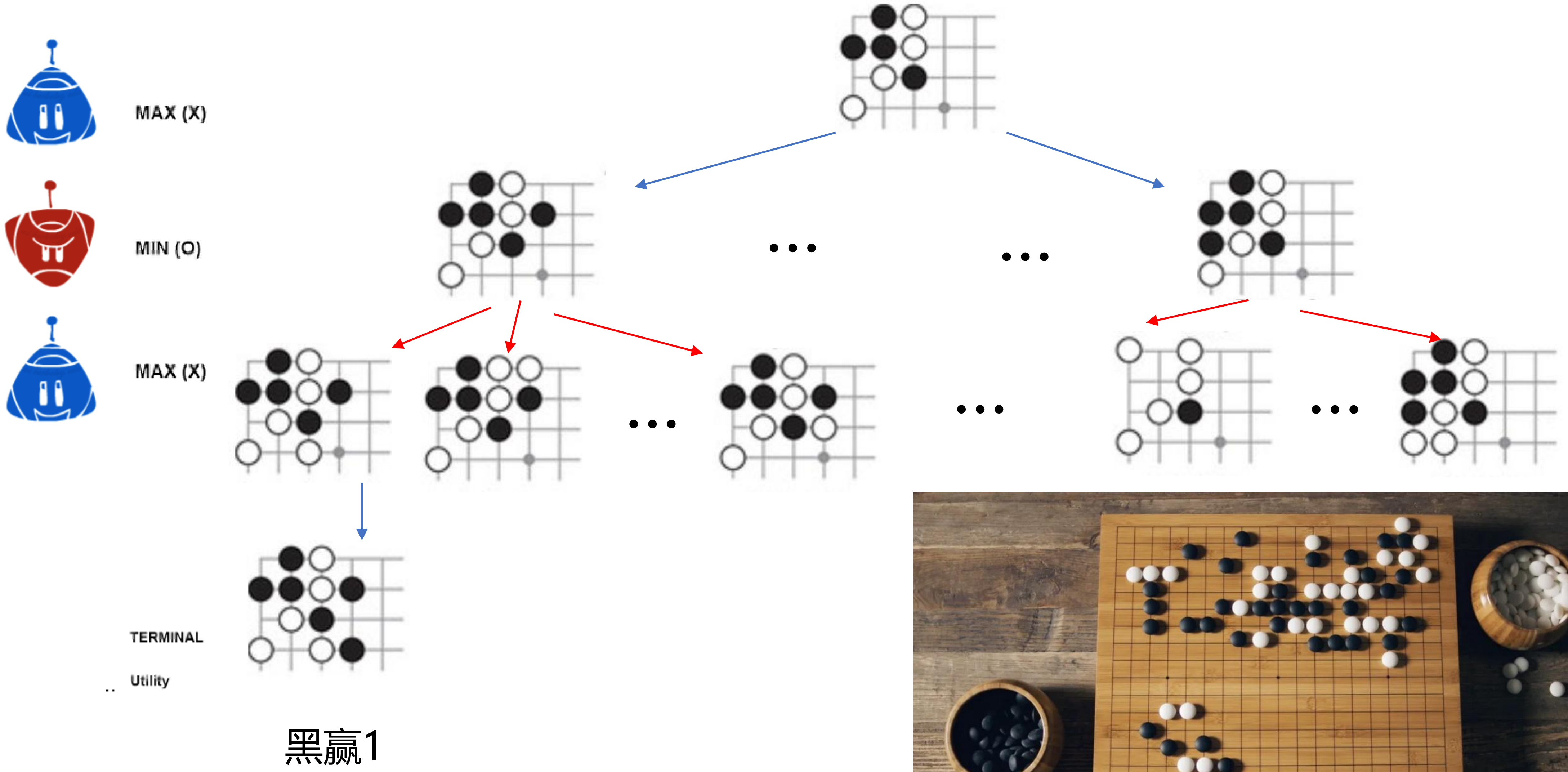
终止状态:

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

井字棋博弈树



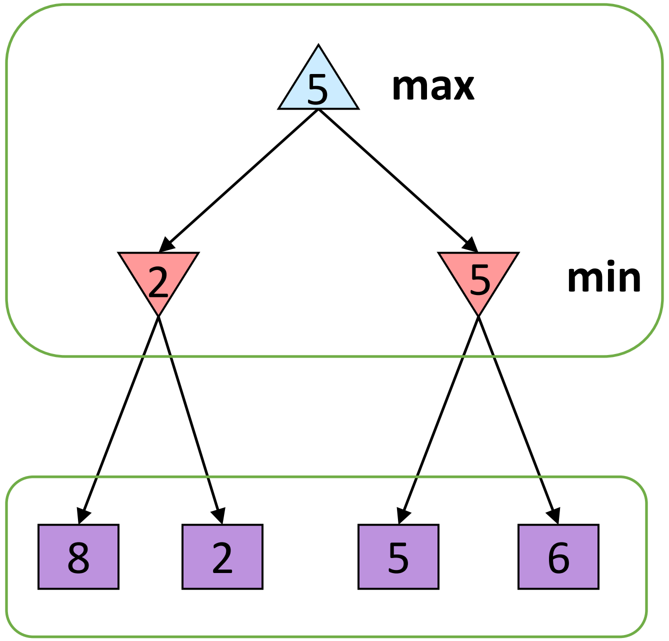
围棋的博弈树



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

极大极小价值:
递归计算



终态价值

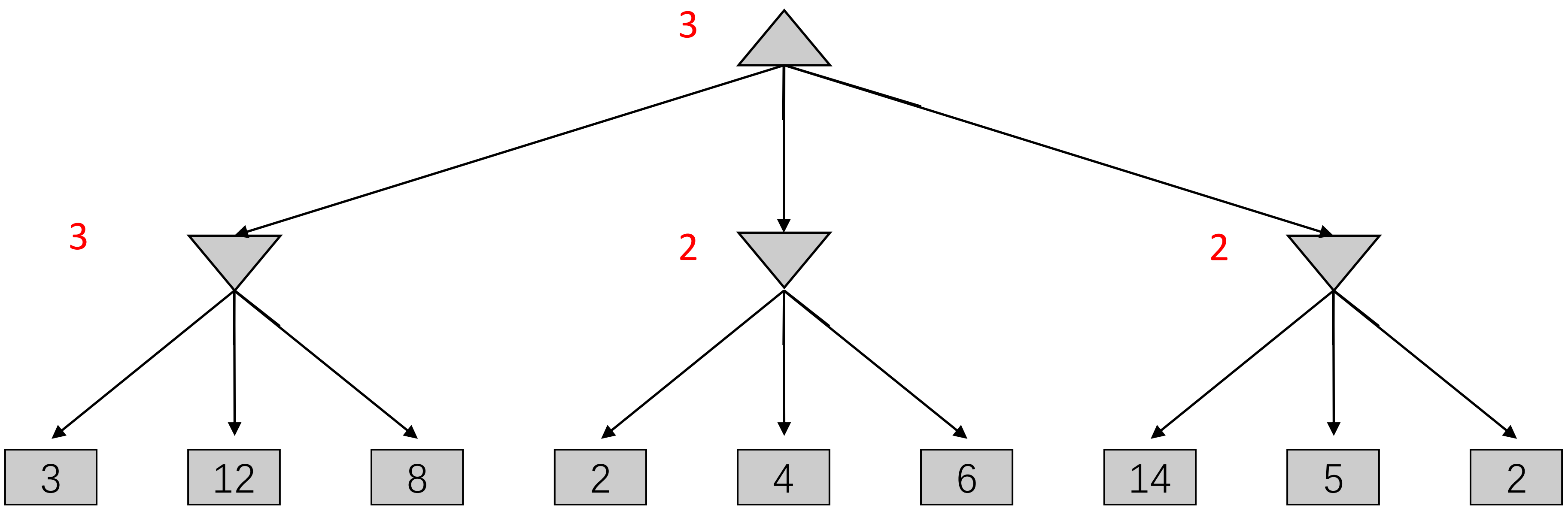
极大极小搜索算法实现

```
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



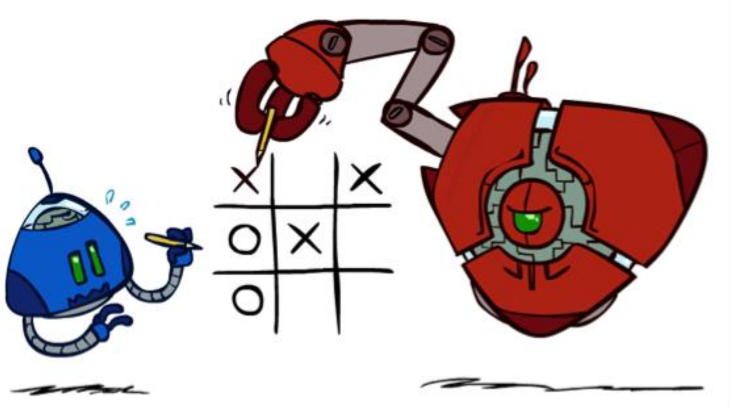
max

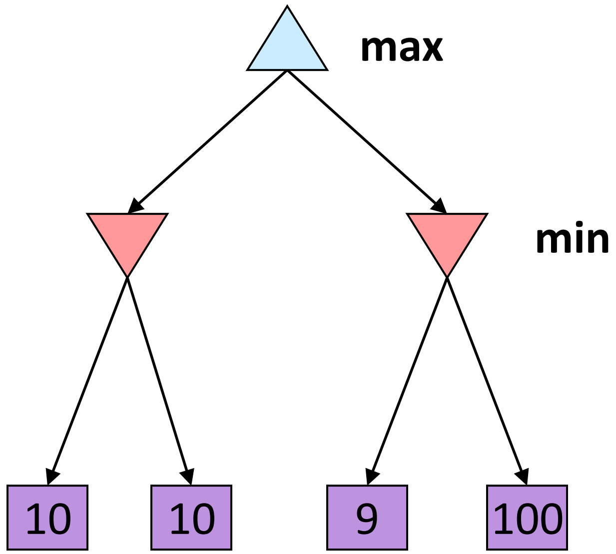
min

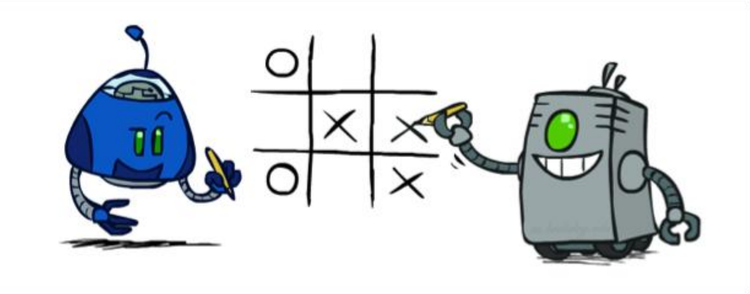
terminal

极大极小搜索算法的性质

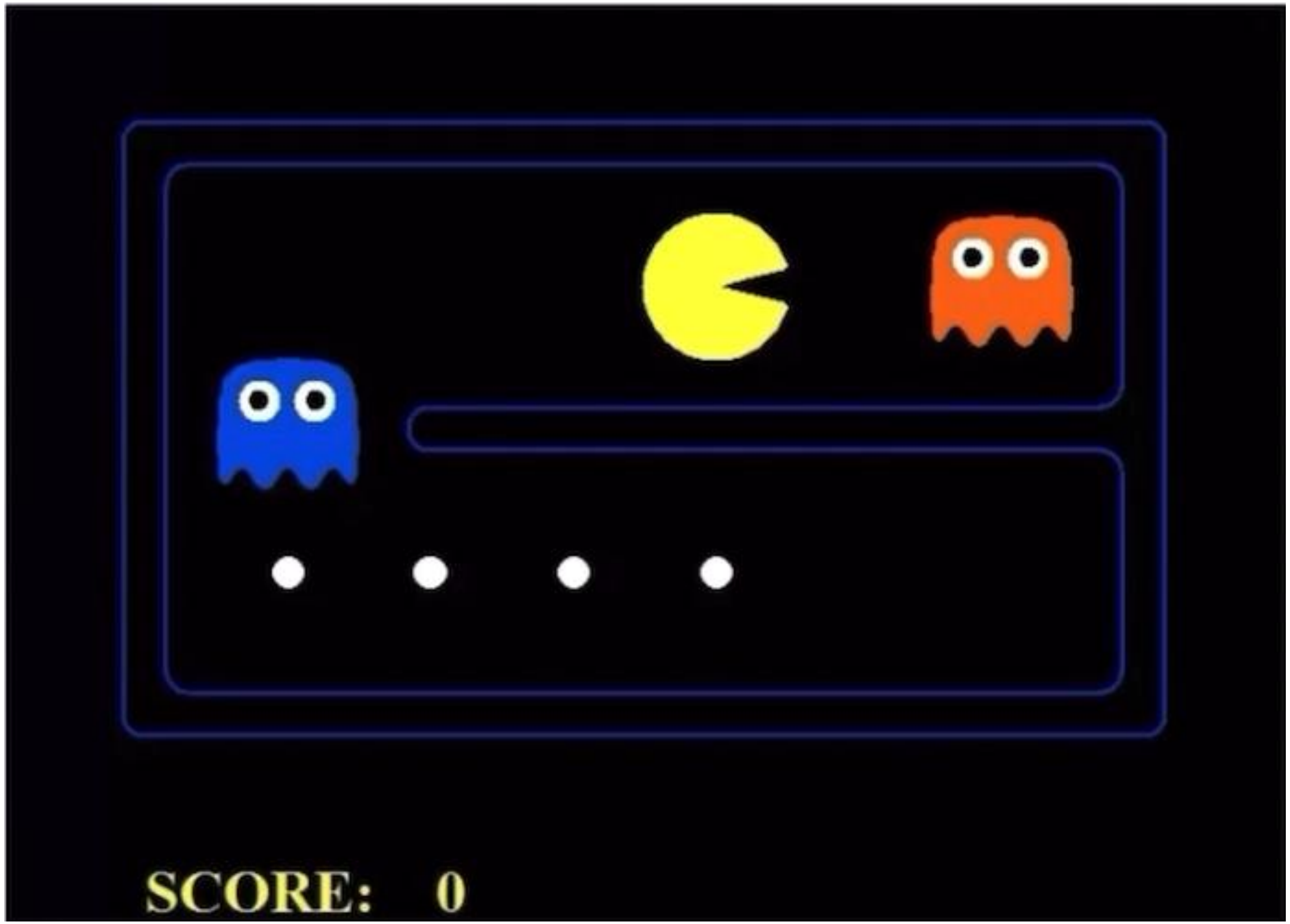
- 对于完美理性玩家是最优的，否则呢?



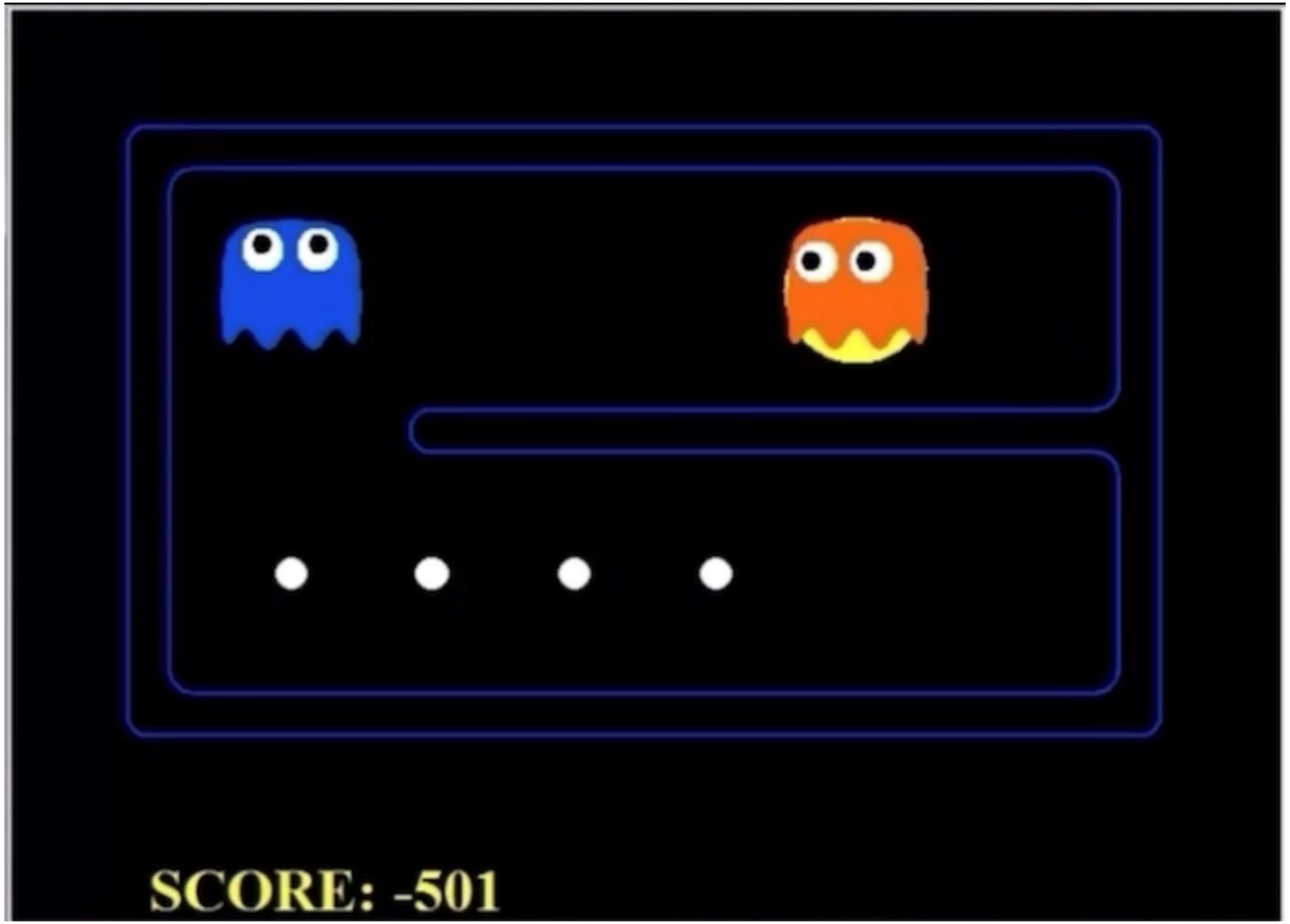




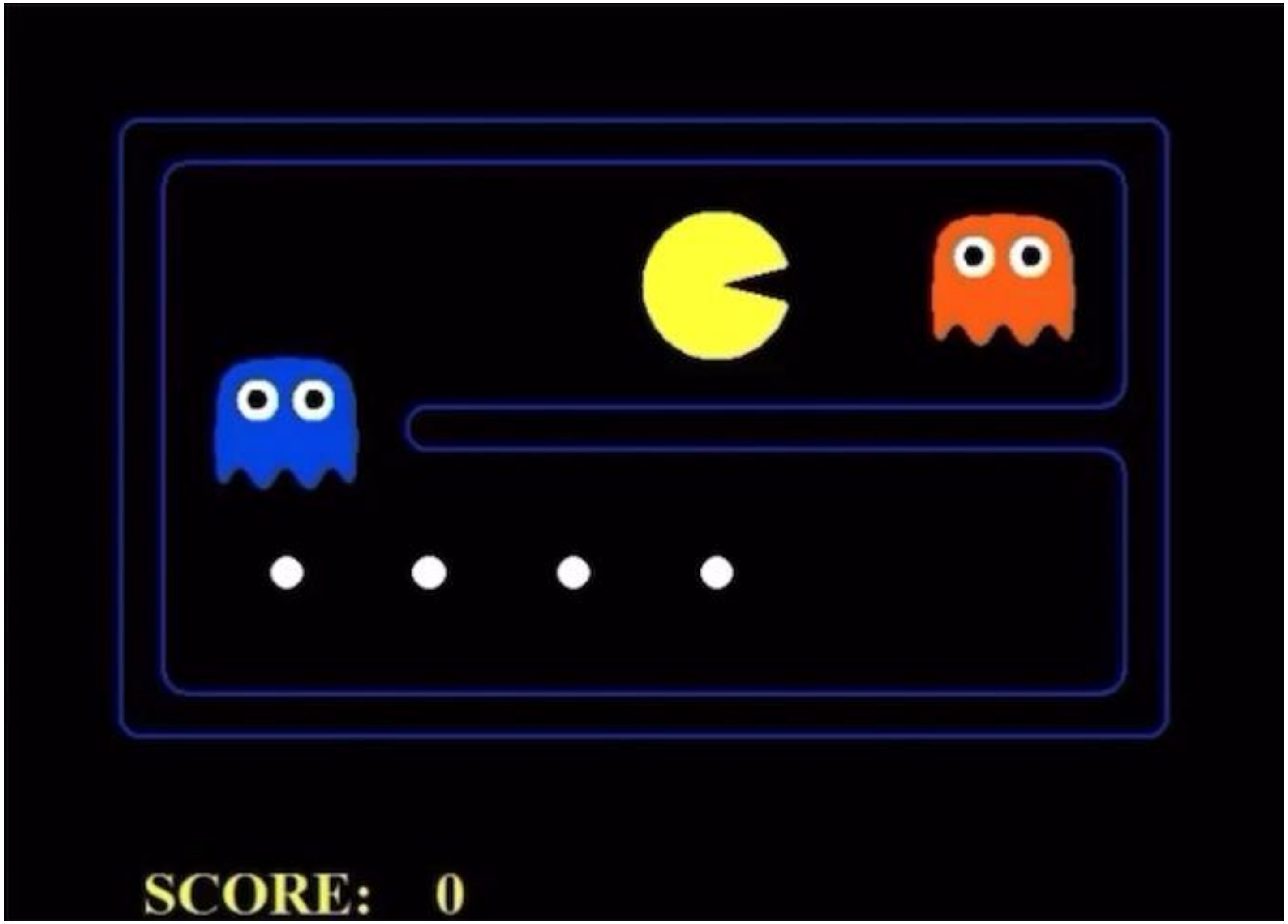
Pacman：假定鬼是理智的



Pacman：假定鬼是理智的



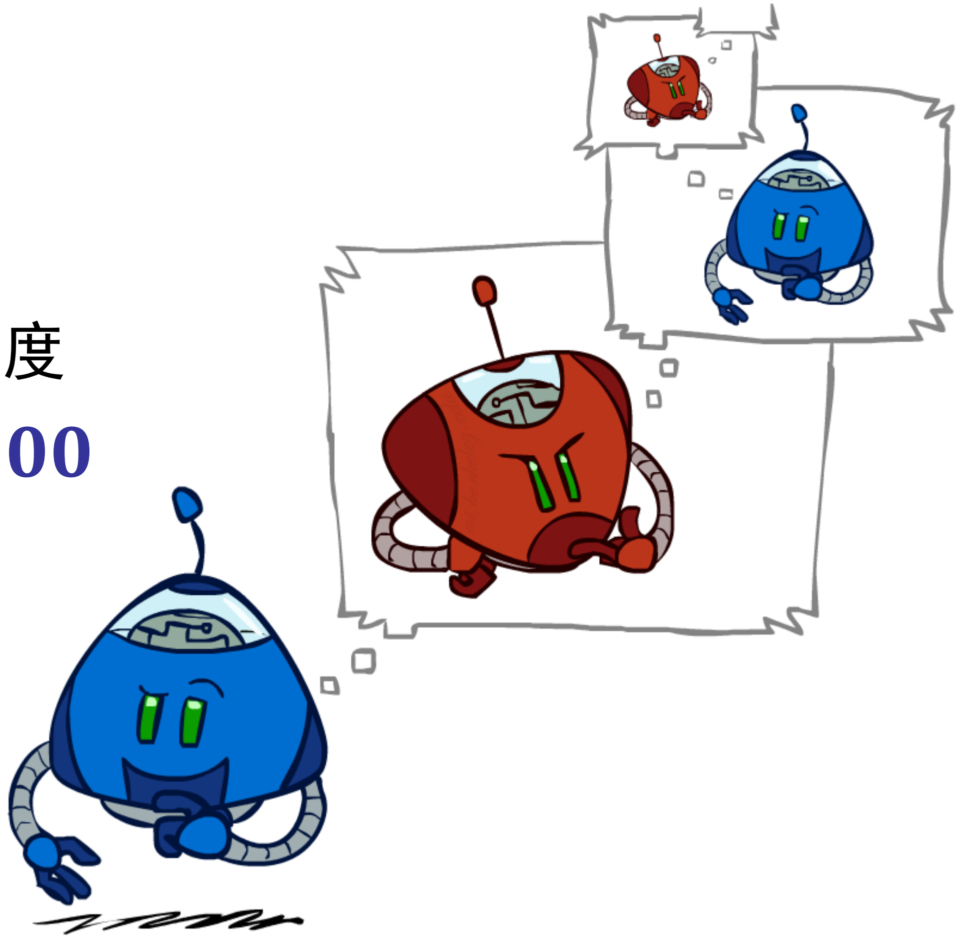
Pacman：鬼不大聪明



Pacman：鬼不大聪明

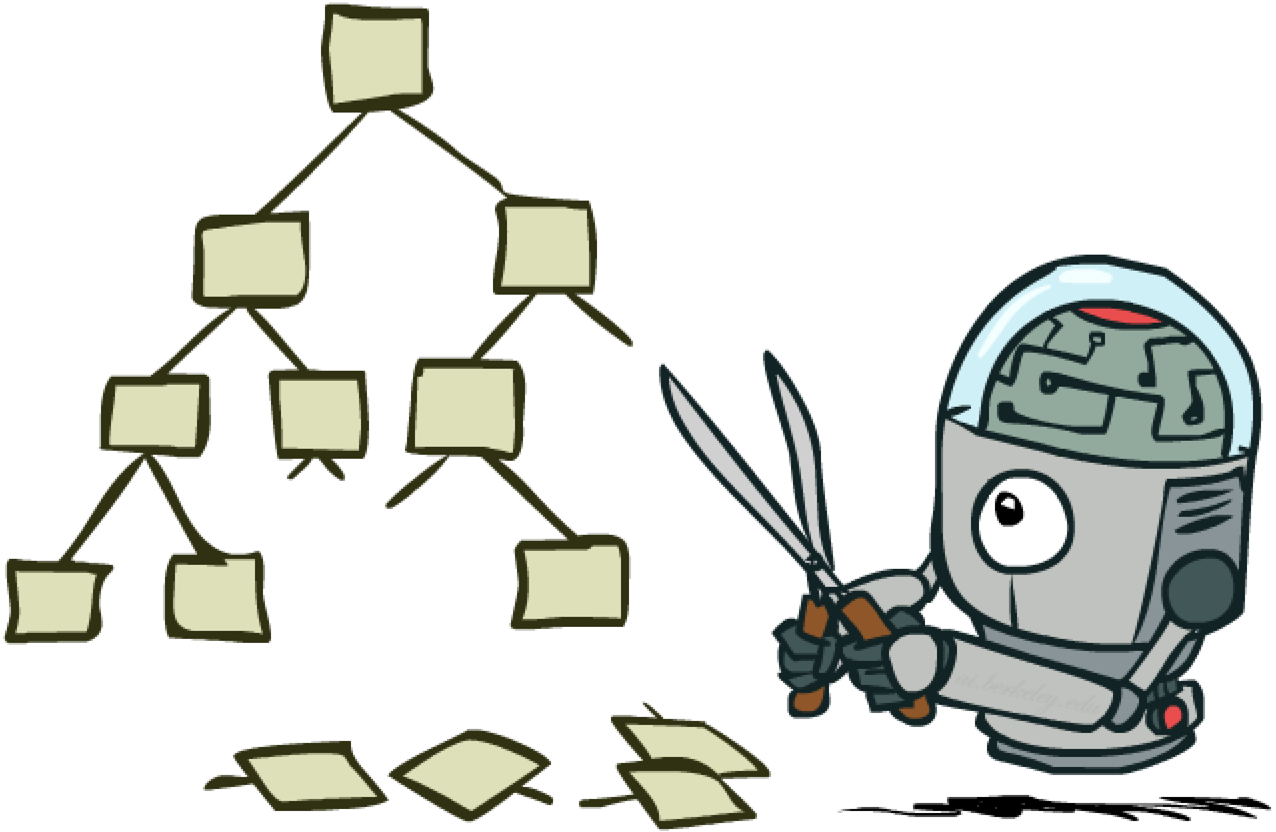
极大极小搜索的效率

- **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？

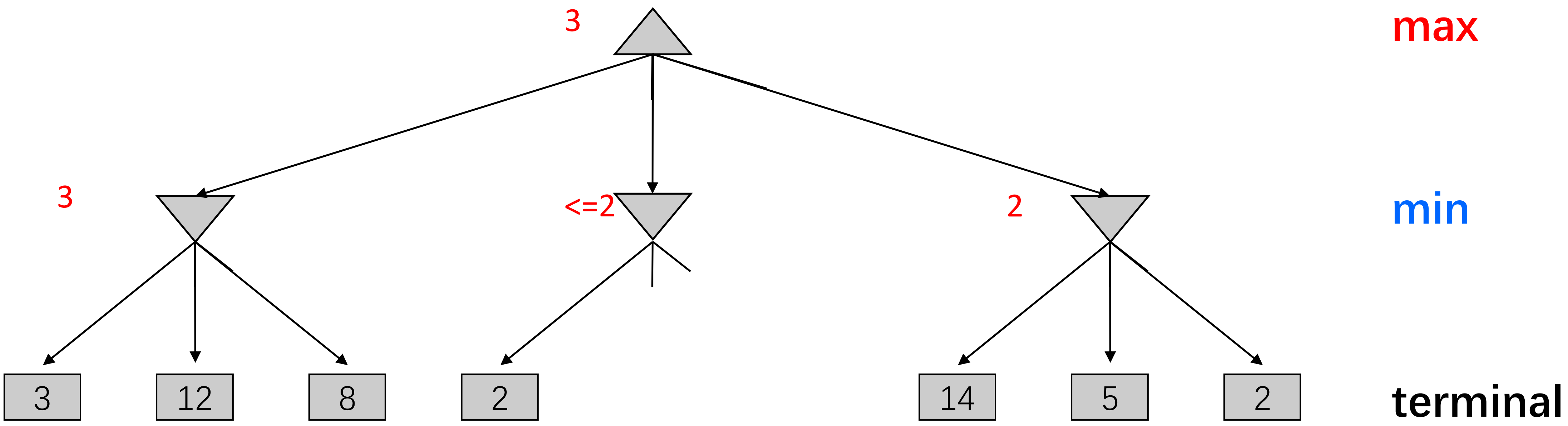


资源限制

博弈树剪枝



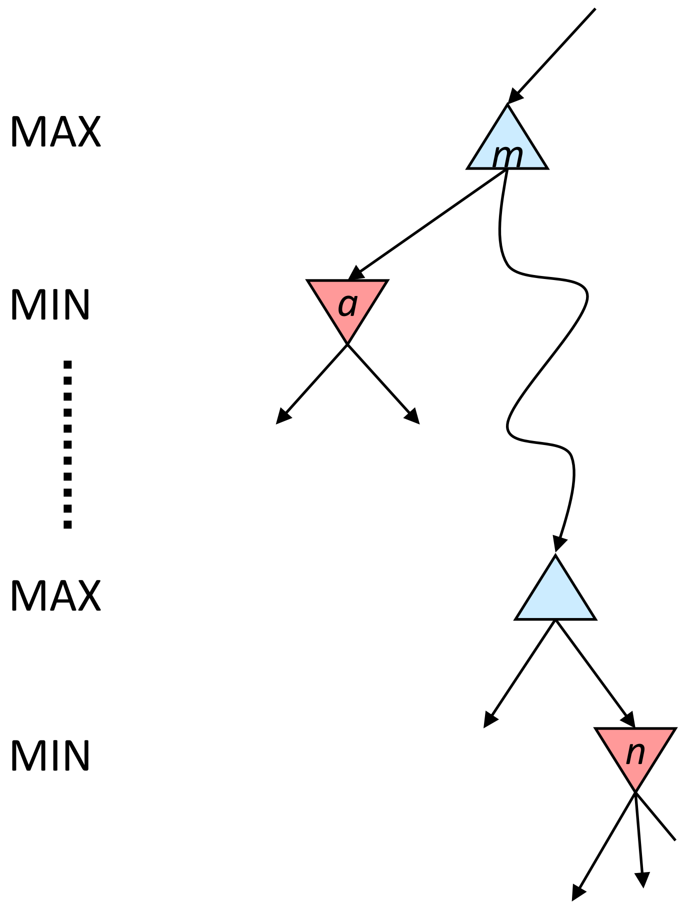
极小极大搜索的示例



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

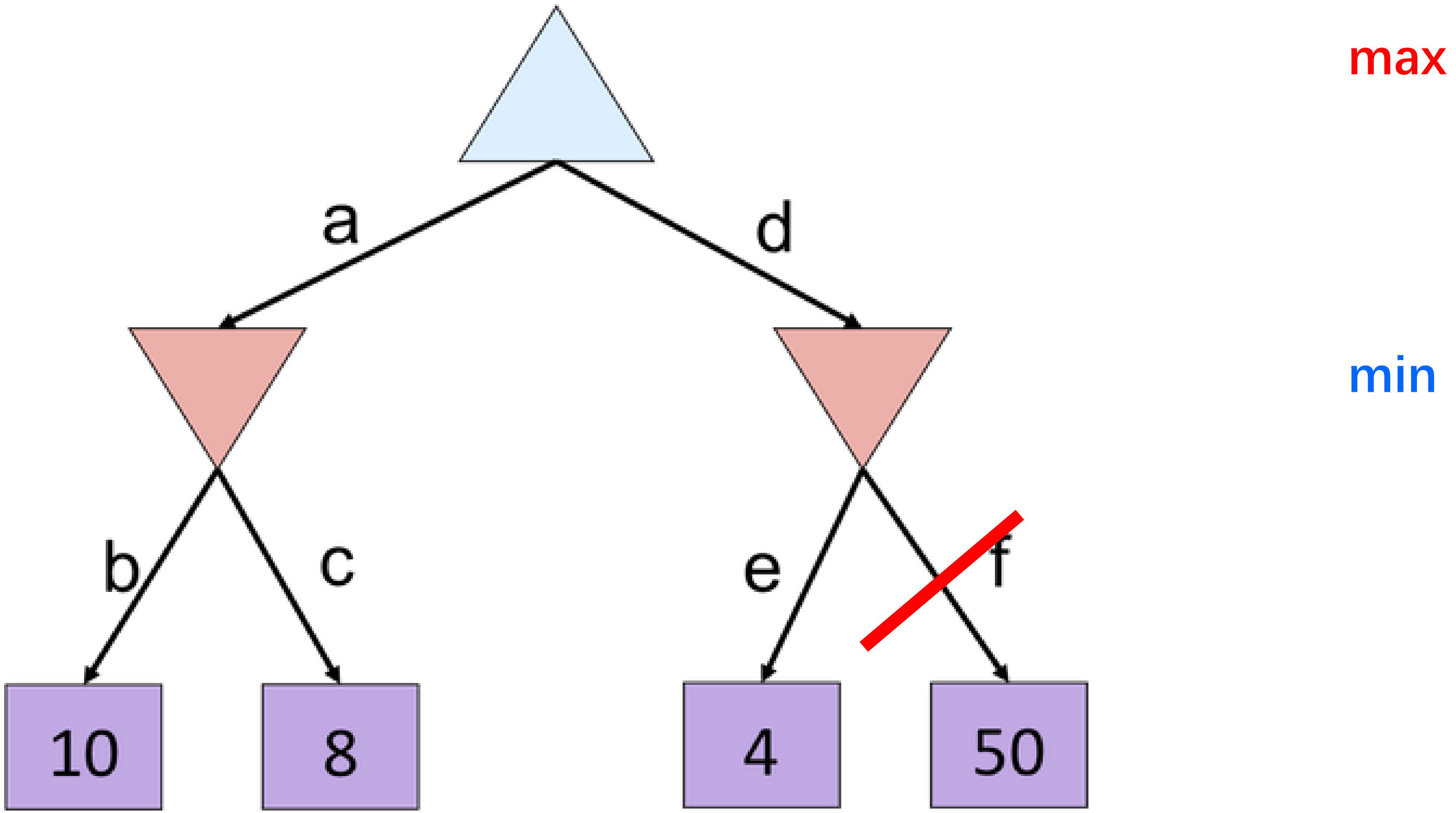
β: MIN's best option on path to root

```
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)
```

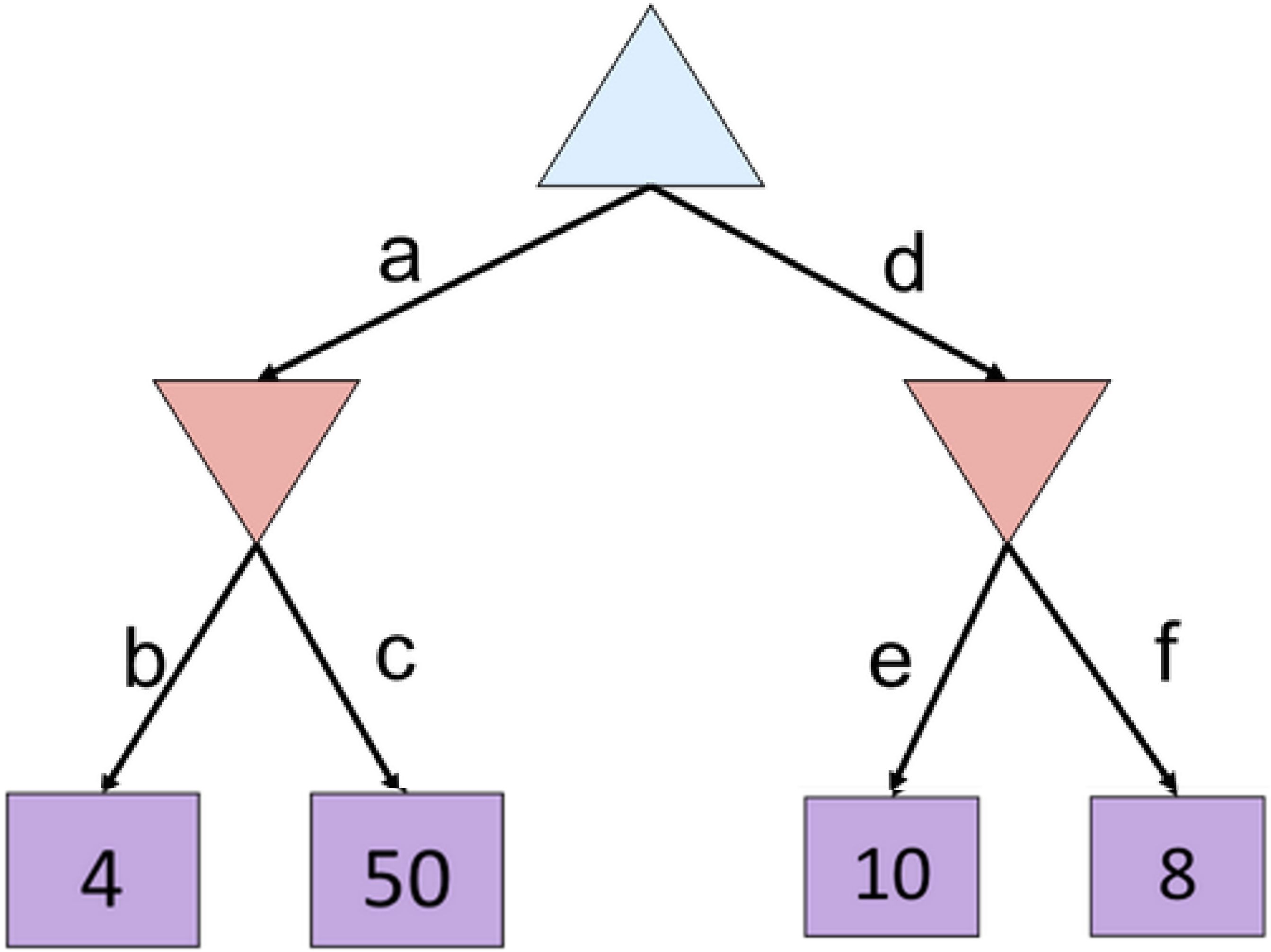
```
def max-value(state, α, β):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor, α, β))
        if v ≥ β return v
        α = max(α, v)
    return v
```

```
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



Alpha-Beta 测验

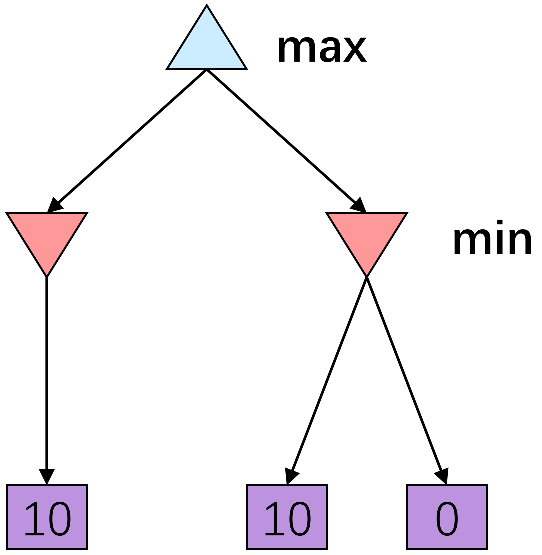


max

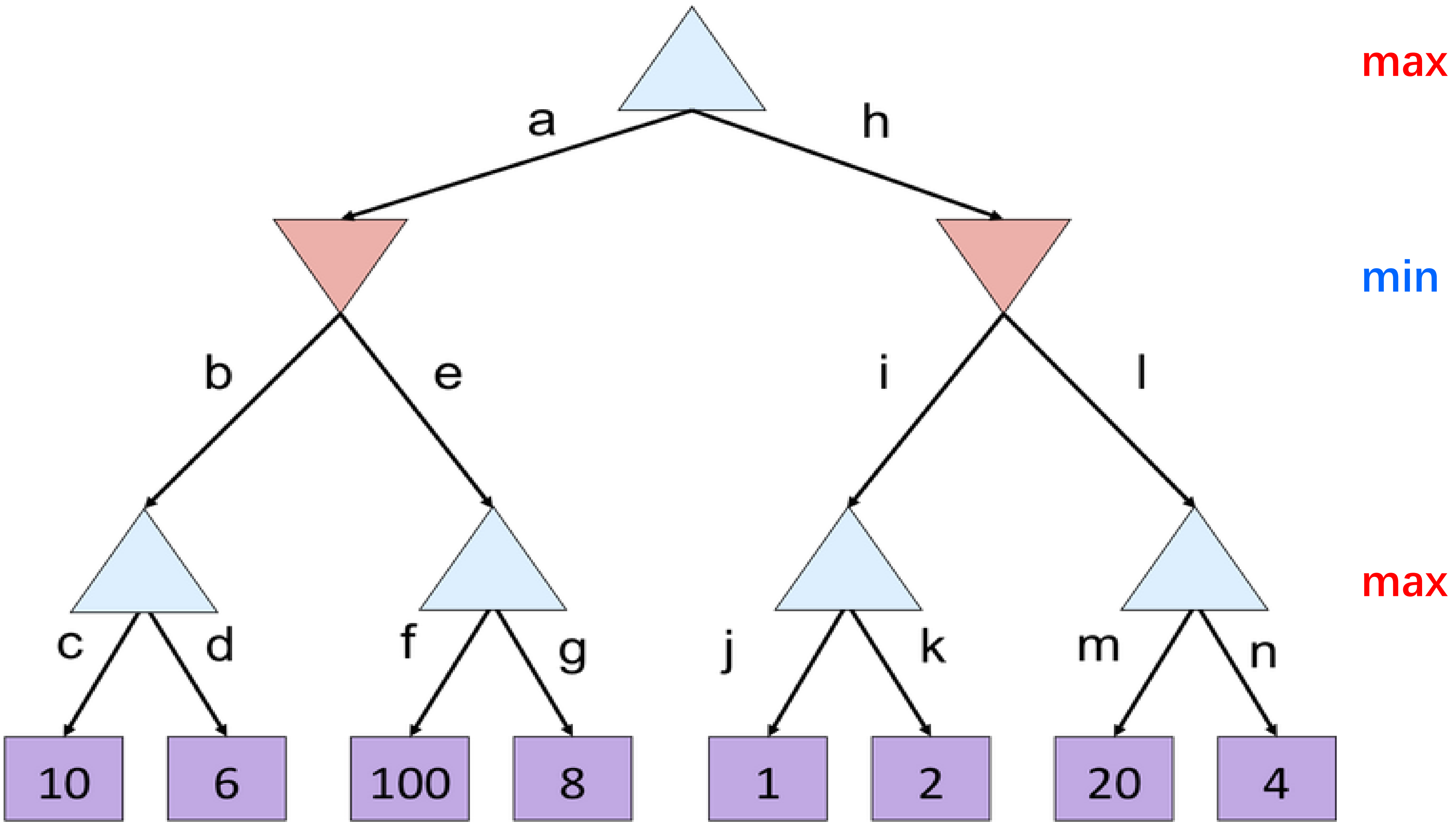
min

Alpha-Beta 剪枝的性质

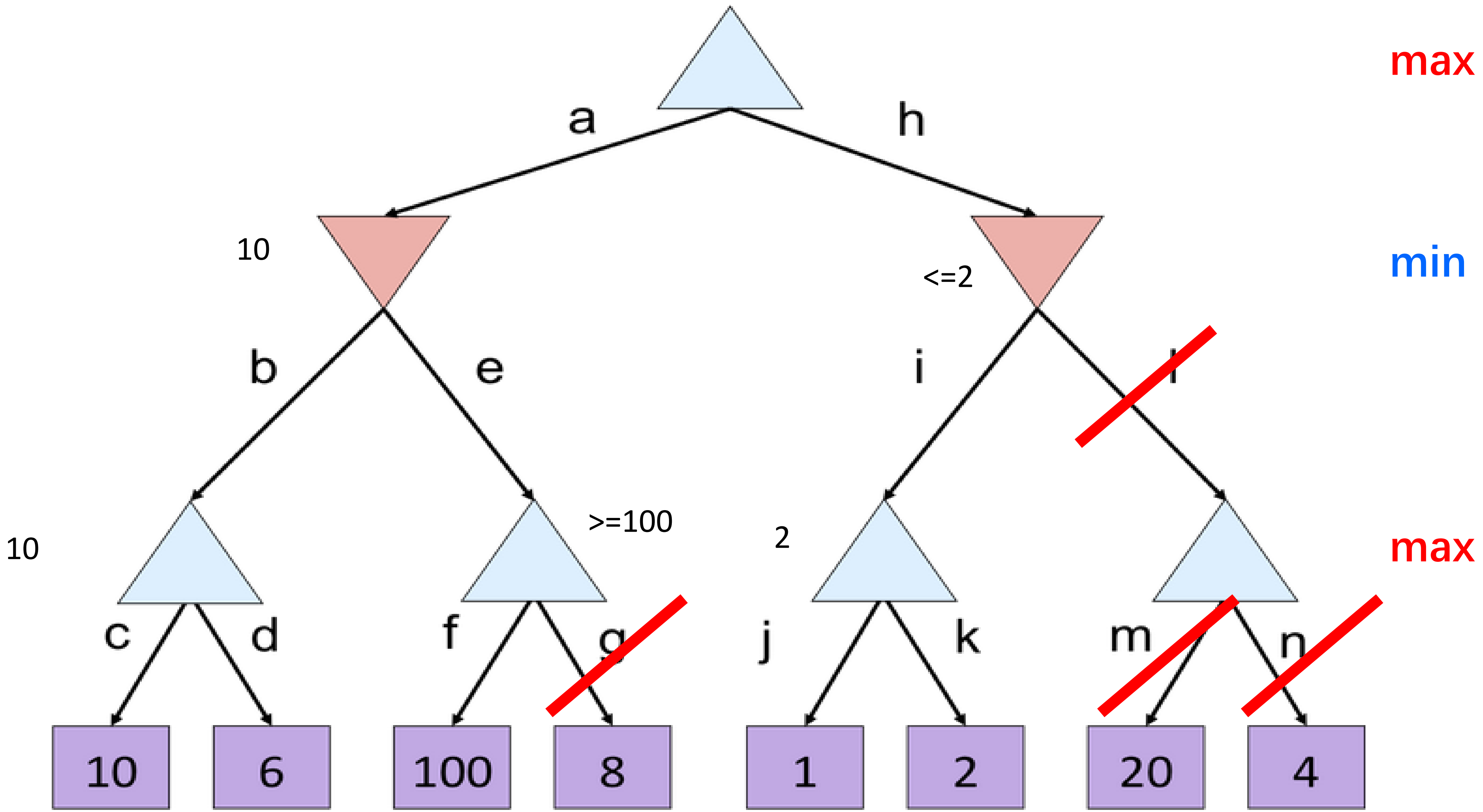
- 这种剪枝方式不会对计算根节点的极小极大值产生影响！
- 中间节点的值可能是**错误**的
  - 重要提示：根节点的子节点可能具有错误的值
  - 因此，最朴素的版本无法进行动作选择
- 良好的子节点排序可以提高剪枝的效率
- 通过“完美排序”：
  - 时间复杂度降至 $O(bm/2)$
  - 双倍可解深度！
  - 然而，对于国际象棋等任务进行完备搜索仍然是不可行的…



Alpha-Beta 测验2

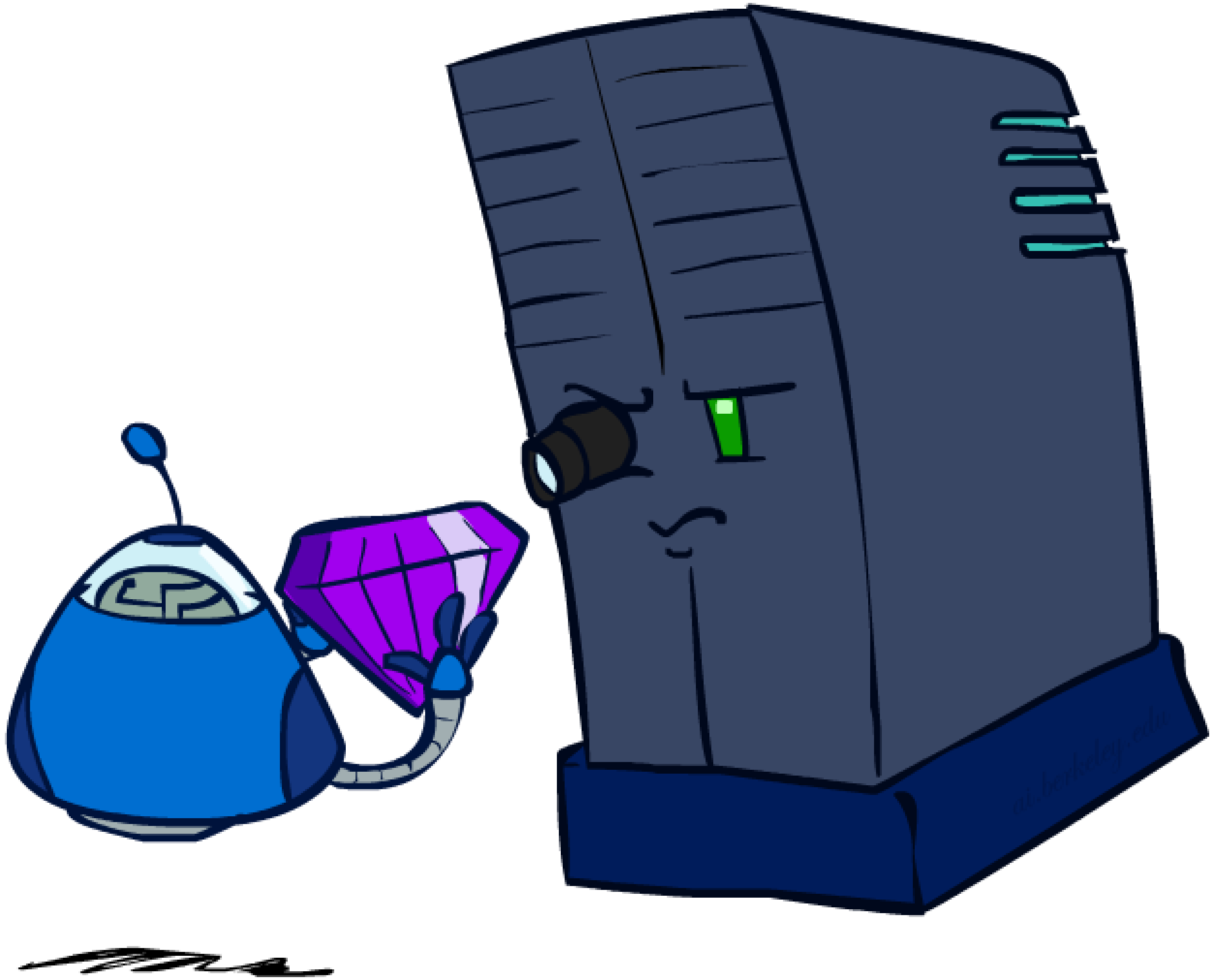


Alpha-Beta 测验2



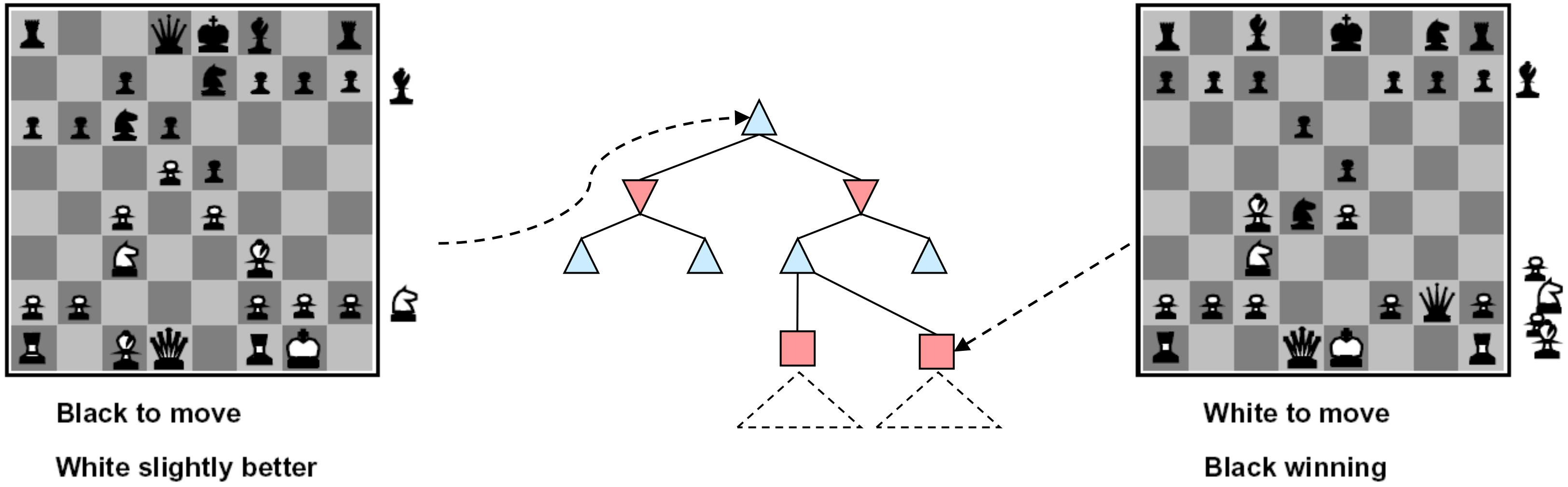
资源限制

估值函数



估值函数

- 估值函数为非终止状态打分：



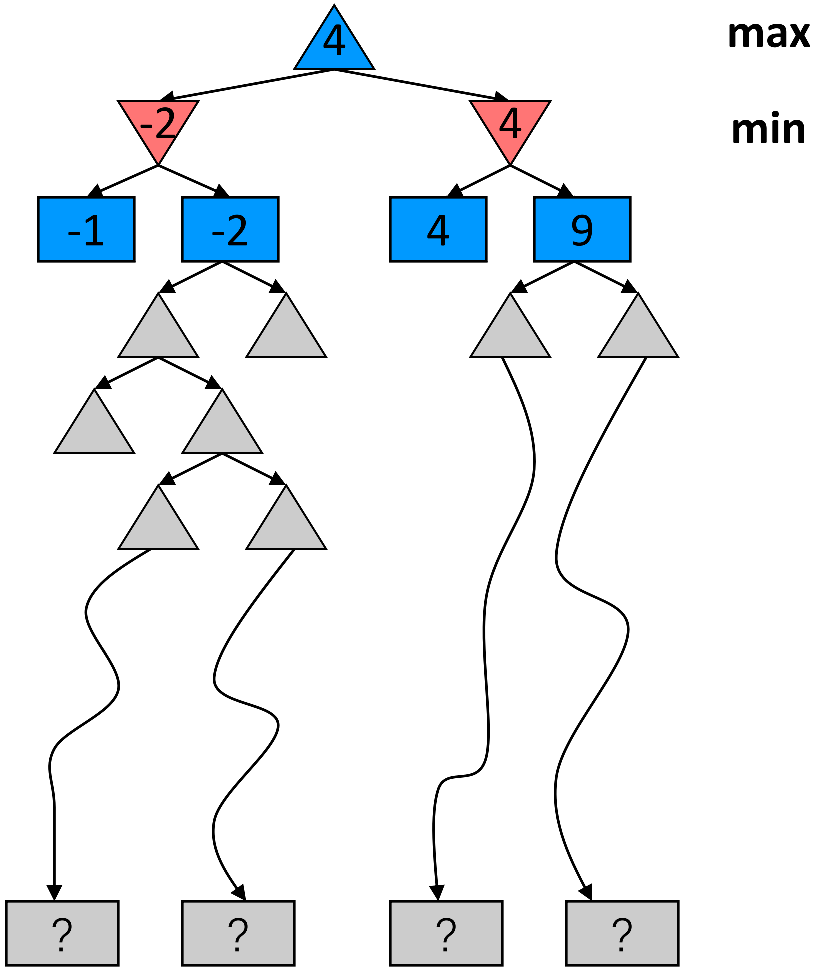
- **理想的估值函数**：返回当前局面的实际极大极小值
- **实际的估值函数**：通常是特征的加权线性和

$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

- e.g. $f_1(s)$ = (num white queens – num black queens).
- 如何构建**更好的估值函数**：机器学习！

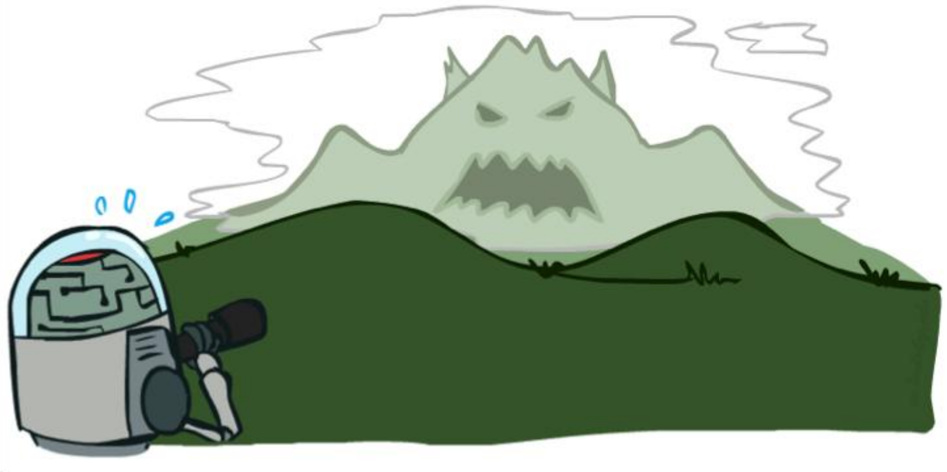
资源限制

- **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

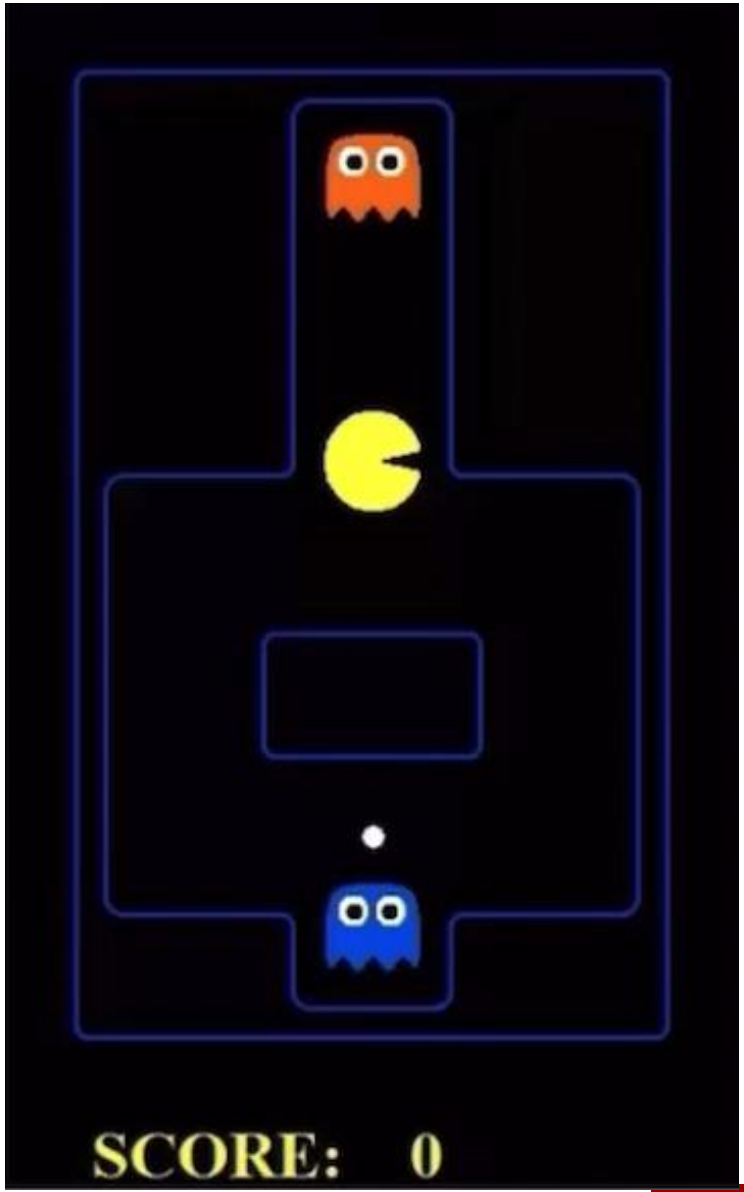


谢谢

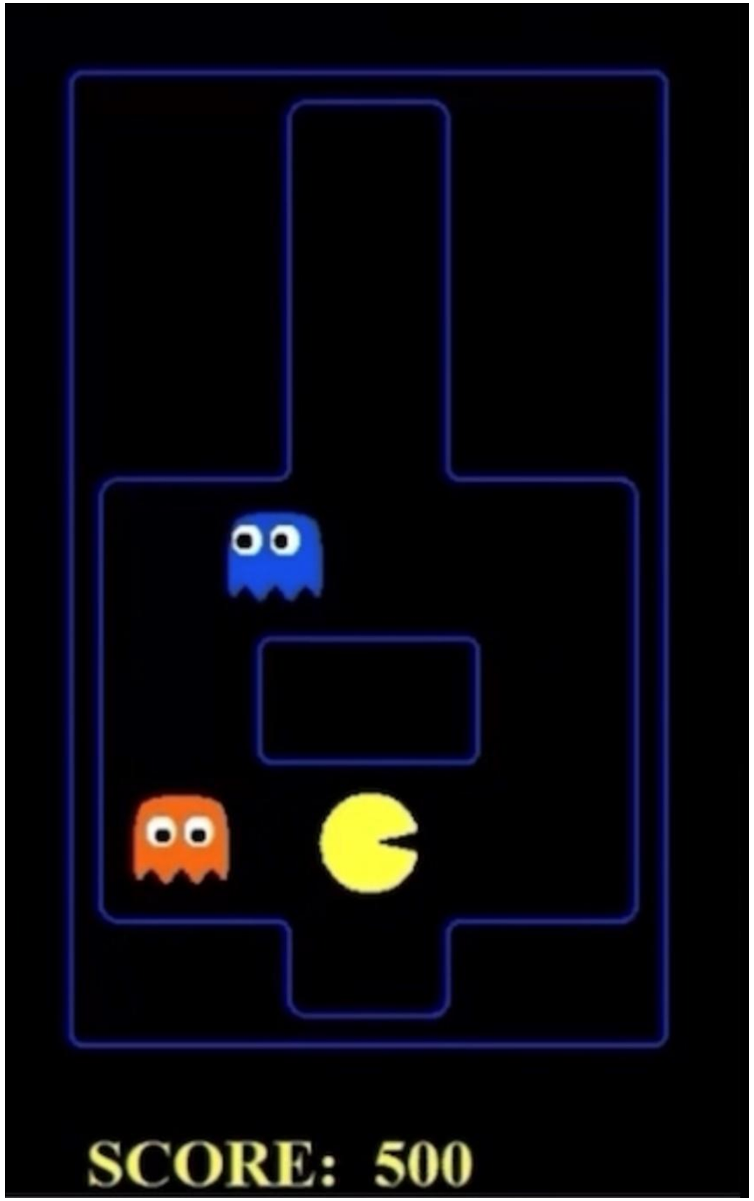
Pacman：搜索深度=2

Pacman：搜索深度=2

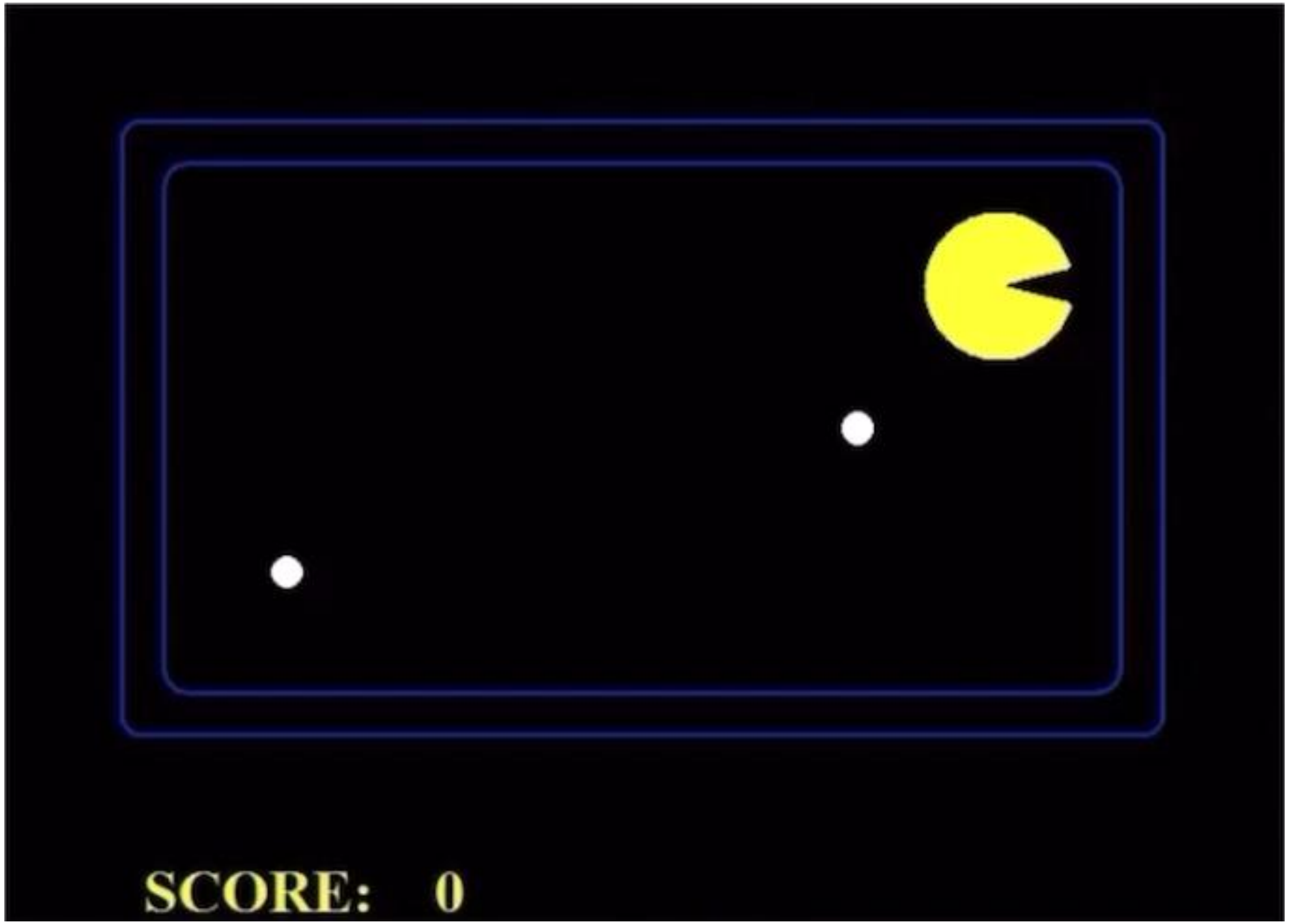
Pacman：搜索深度不受限



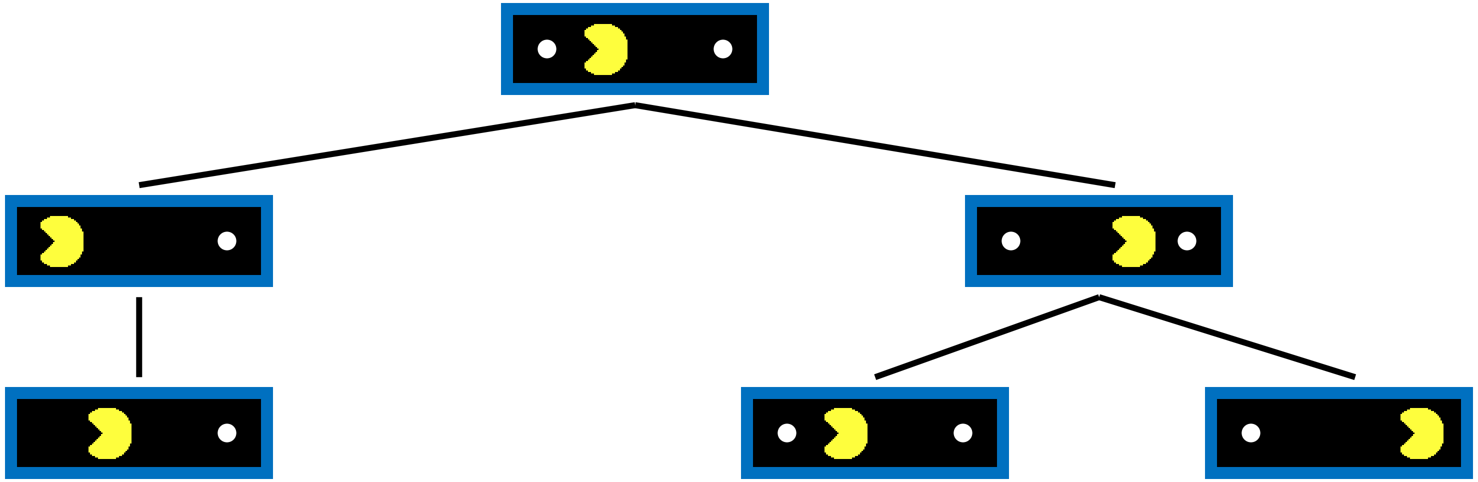
Pacman：搜索深度不受限



Pacman：不好的估值函数：左右抖动 (d=2)



为什么Pacman不吃豆子?



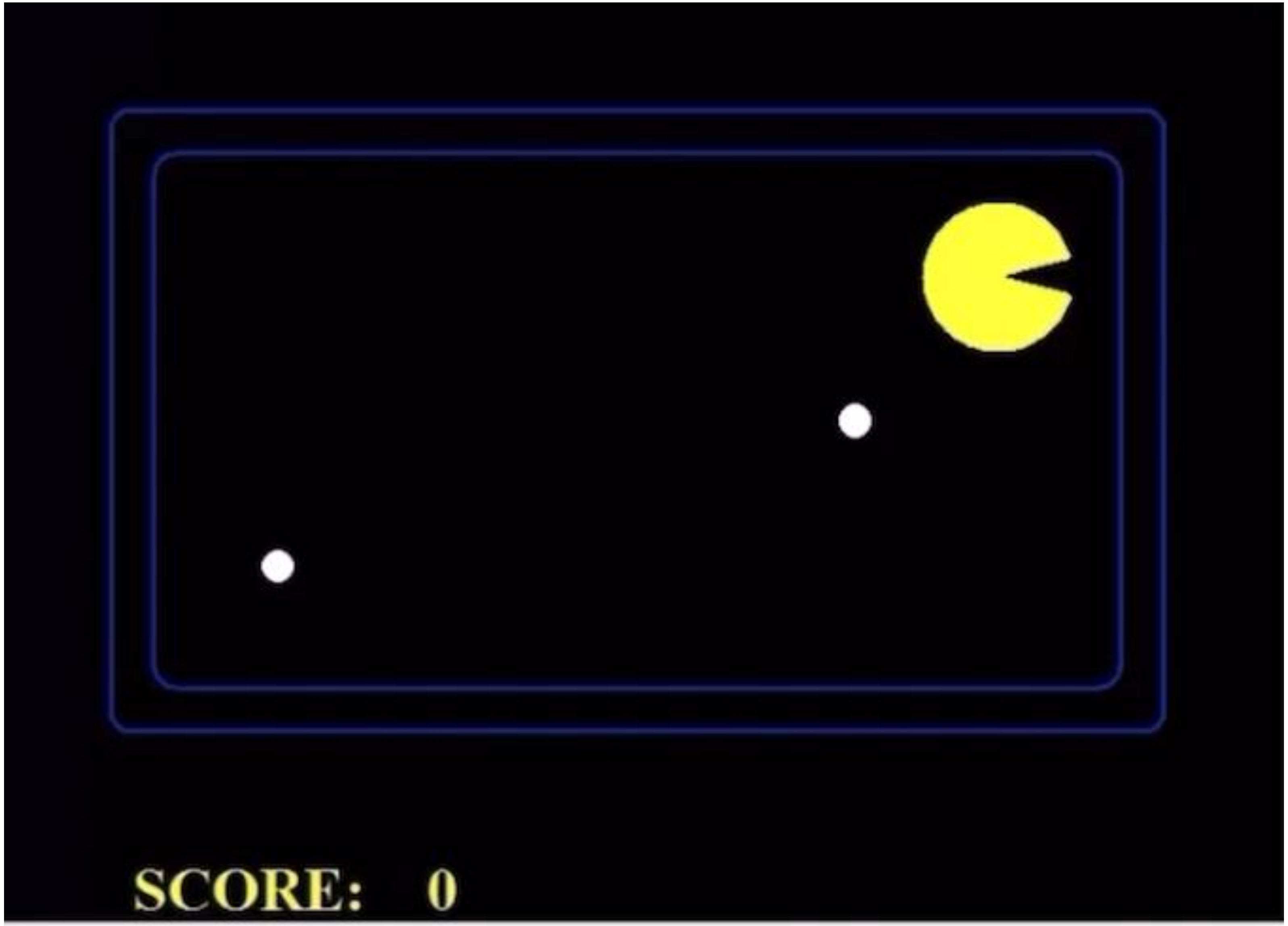
他知道现在吃豆子可以提高得分 (向西或向东)

他也知道他可以在稍后吃豆子同样提高得分 (向西或向东)

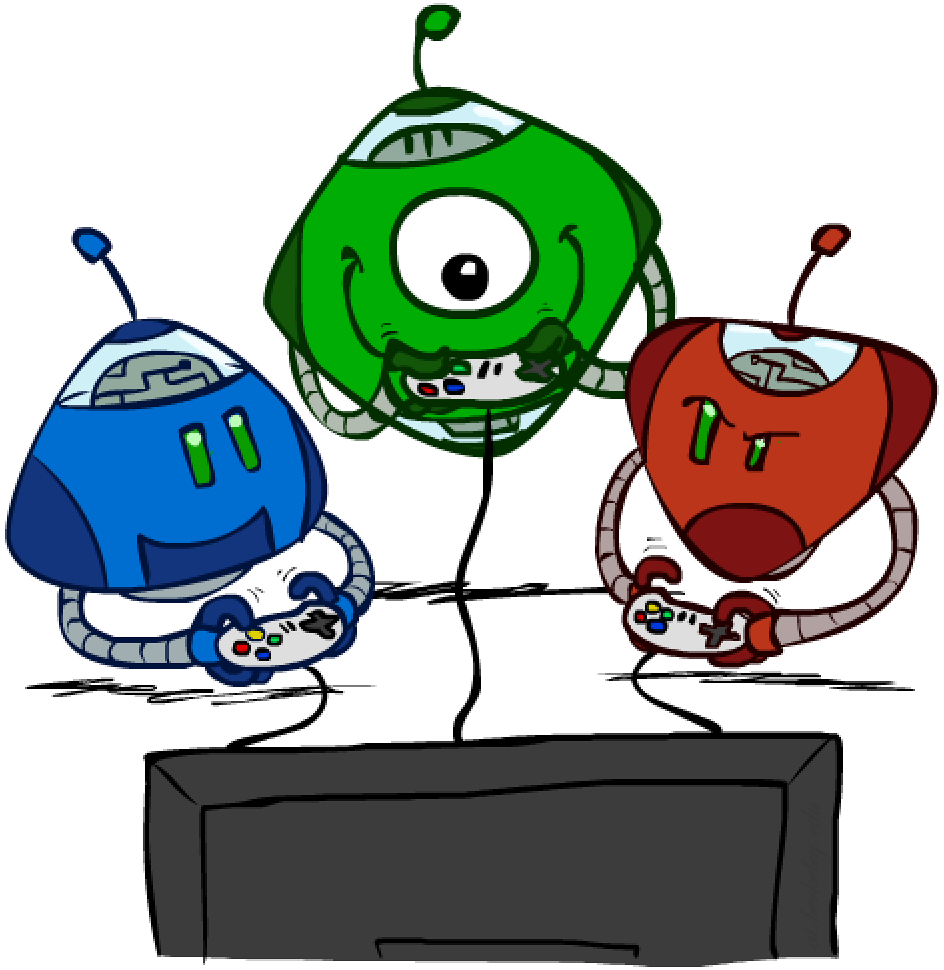
在吃豆子后，不会有任何得分机会 (在视野范围内，有两个点)

因此，等待似乎跟进食一样好：可在下一轮重新规划时向西移动，然后再回到向东的方向！

Pacman：解决抖动(d=2)



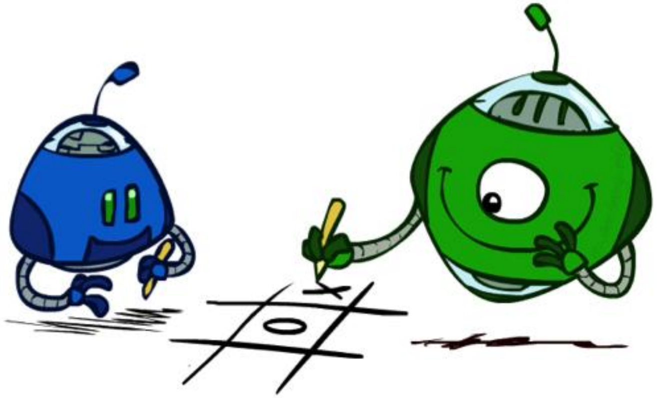
其它博弈类型

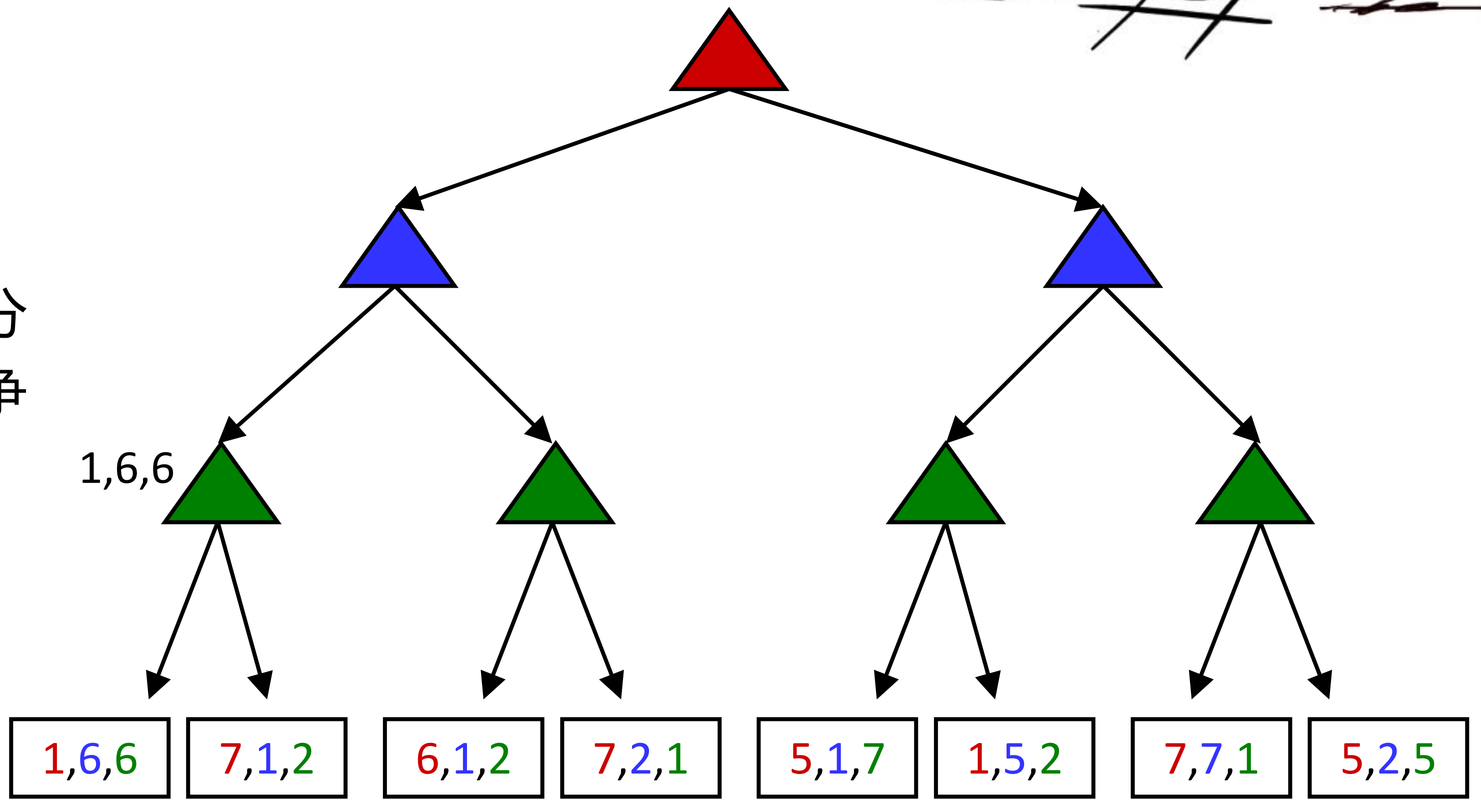


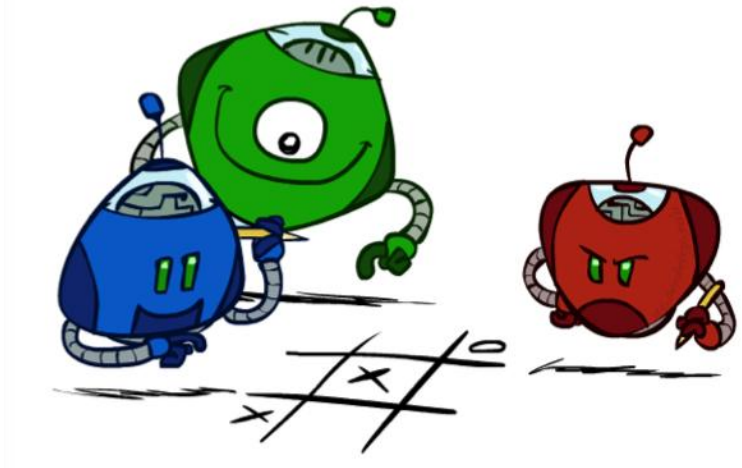
多智能体效用

- 如果博弈不是零和的，或者有多个智能体参与呢?

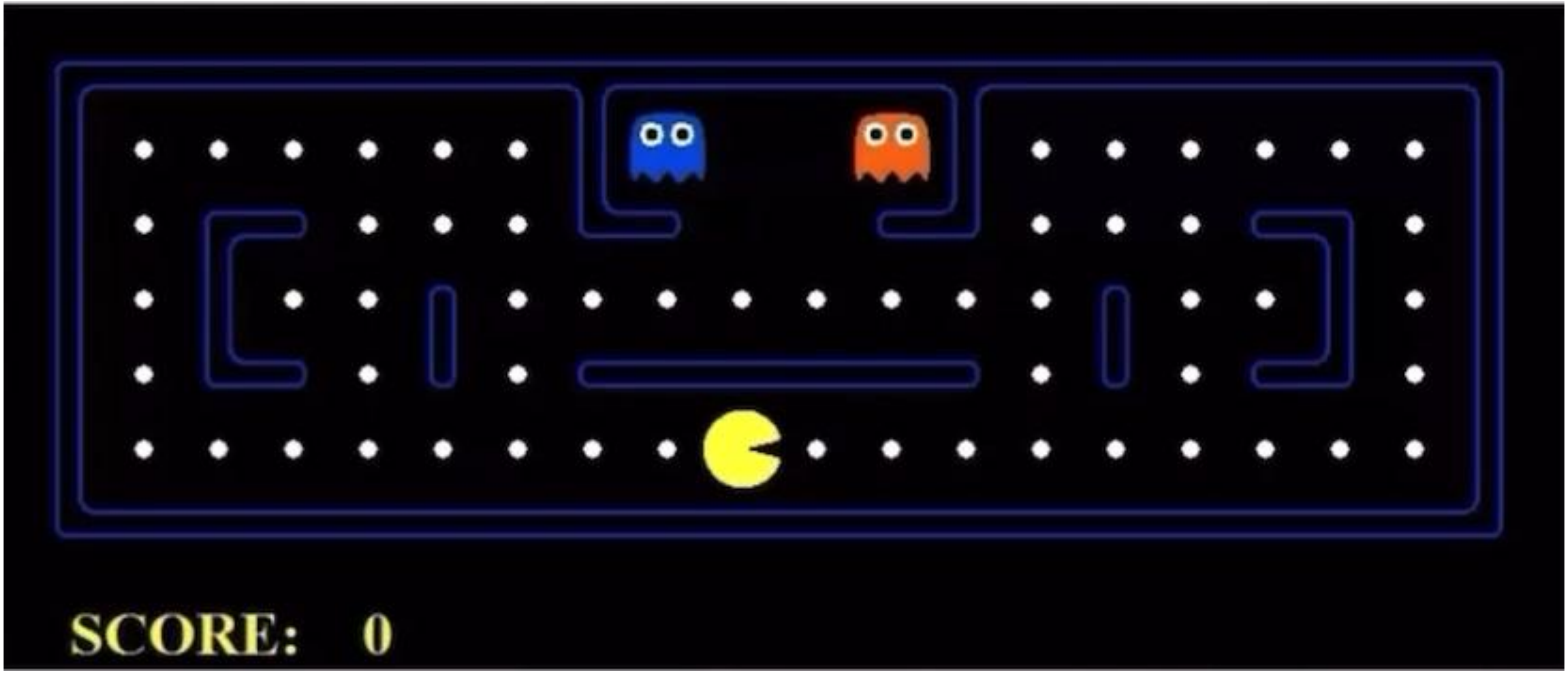
- 极大极小策略的推广:
  - 终止状态会给出效用元组
  - 节点的价值也是效用元组
  - 每个玩家最大化自己的部分
  - 可以动态地产生合作和竞争



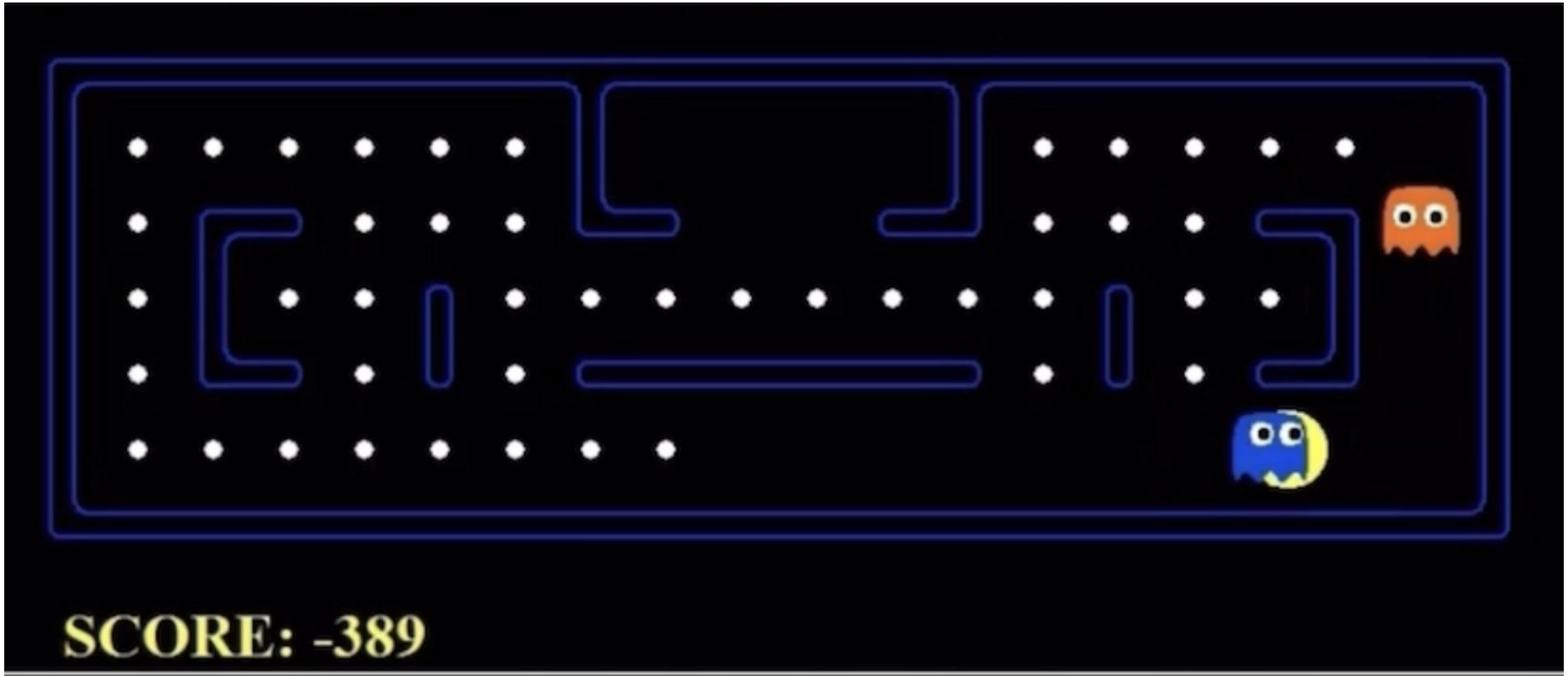




鬼的演示视频(有估值函数)

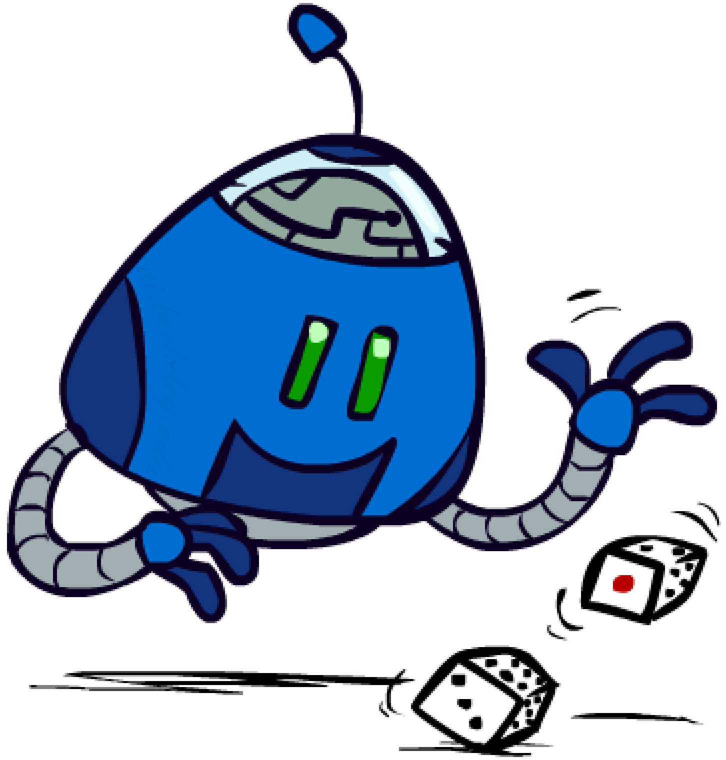


鬼的演示视频(有估值函数)

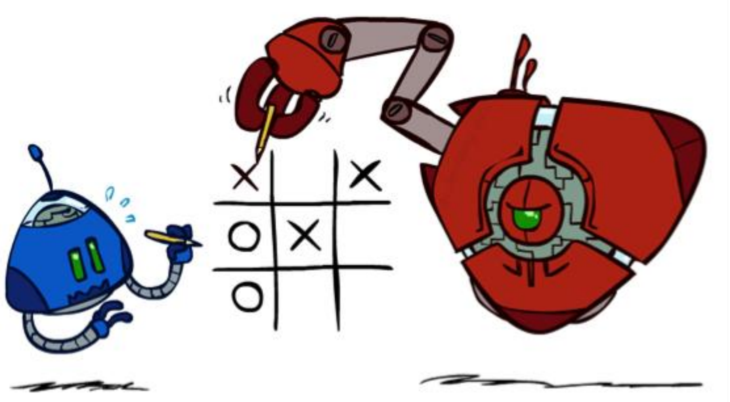


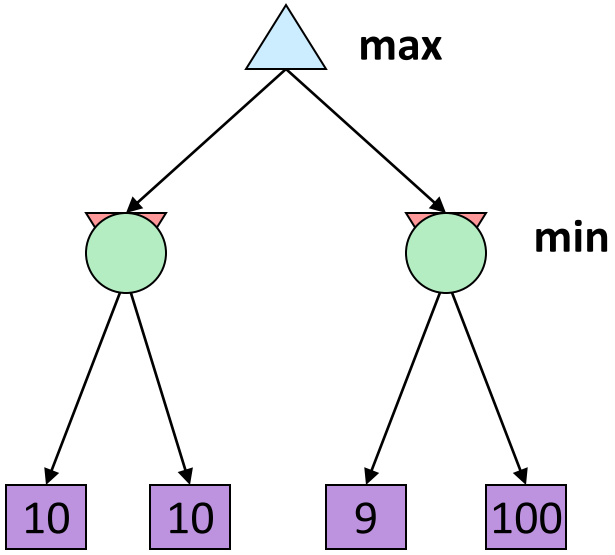
鬼的演示视频(有估值函数)——局部

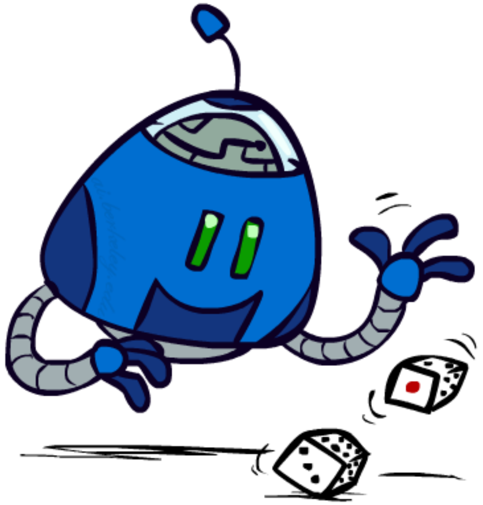
不确定的结果



最差情况 vs. 平均情况



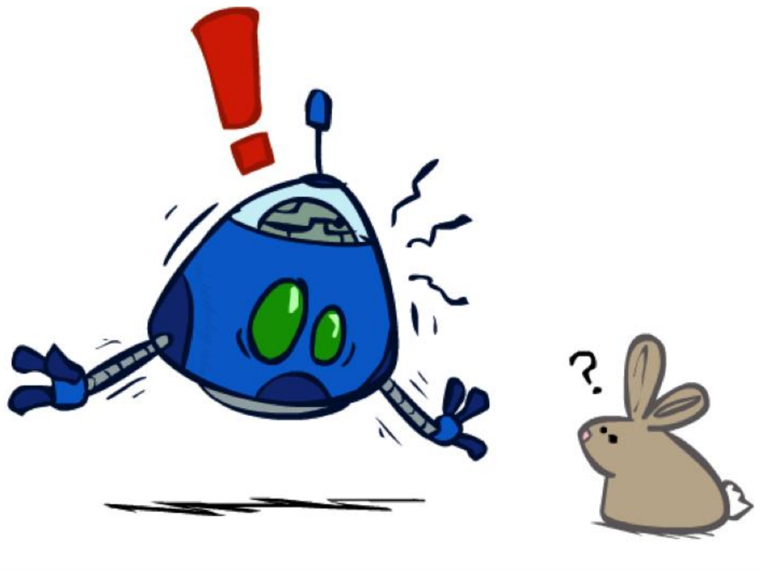




不确定的结果由偶然性控制，而不是对手！

为什么不使用极大极小策略?

- 最坏情况的推理过于保守，因此需要平均情况的推理。

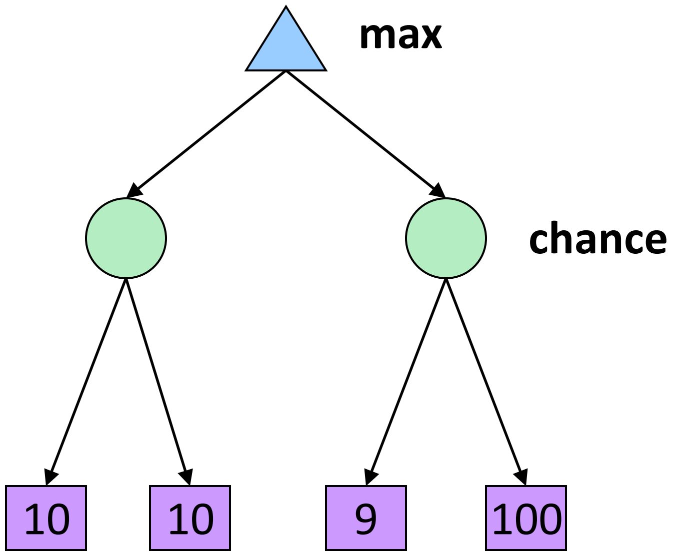


期望最大化搜索

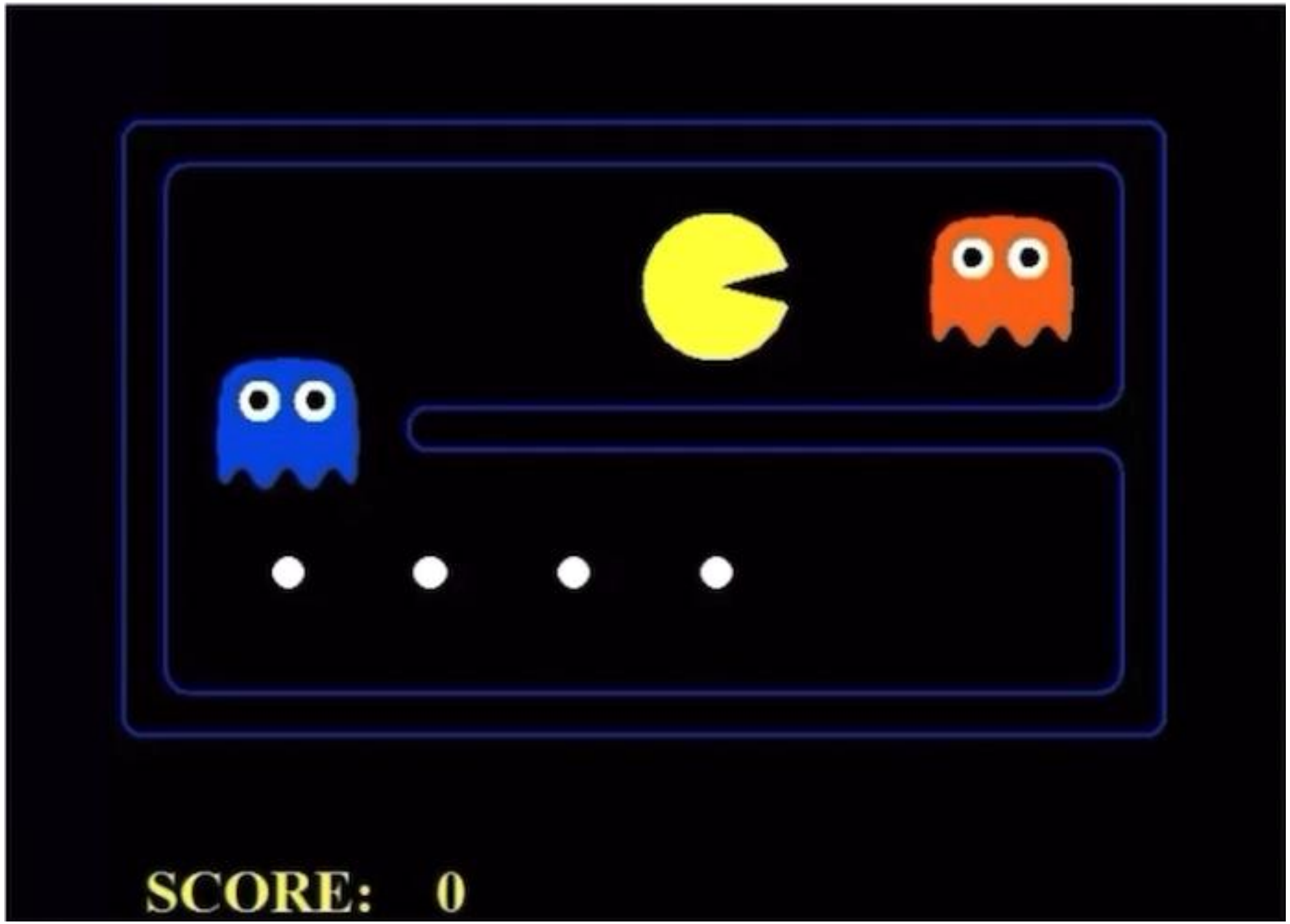
- 为什么我们无法知道给定行动的结果是什么?
  - 显式的随机性：掷骰子
  - 无法预测的对手：鬼的反应是随机的
  - 无法预测的人类：人类是不完美的
  - 行动可能失败：当移动机器人时，轮子可能会打滑

- 因此，价值应该反映平均情况（**期望最大化**），而不是最坏情况（极小）的结果。

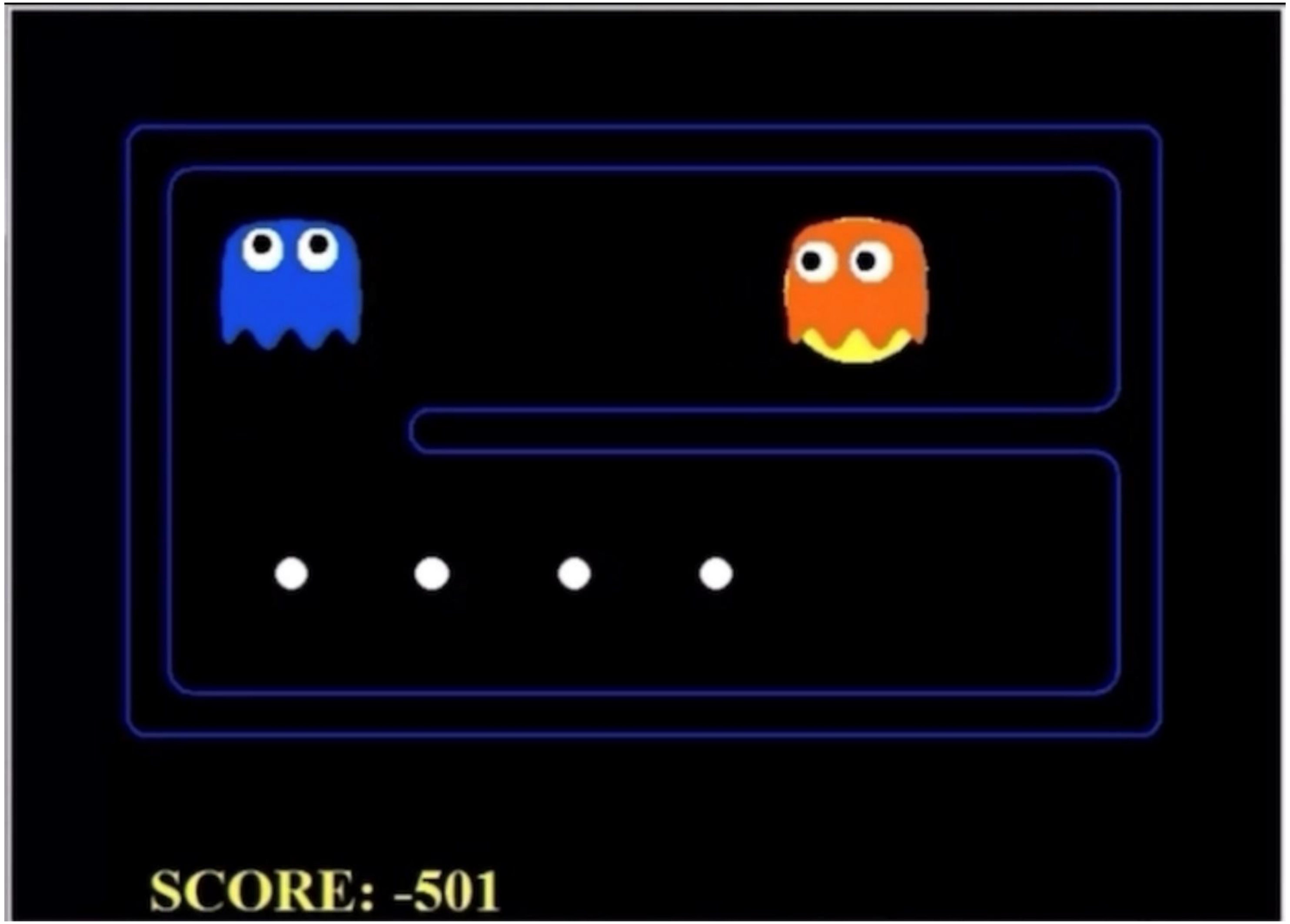
- **期望最大化搜索：计算在最优游戏过程下的平均得分**
  - Max节点与极大极小搜索相同
  - 概率节点类似于Min节点，但结果不确定
    - 计算它们的预期效用
    - 即，对它们的子节点进行加权平均（预期值）



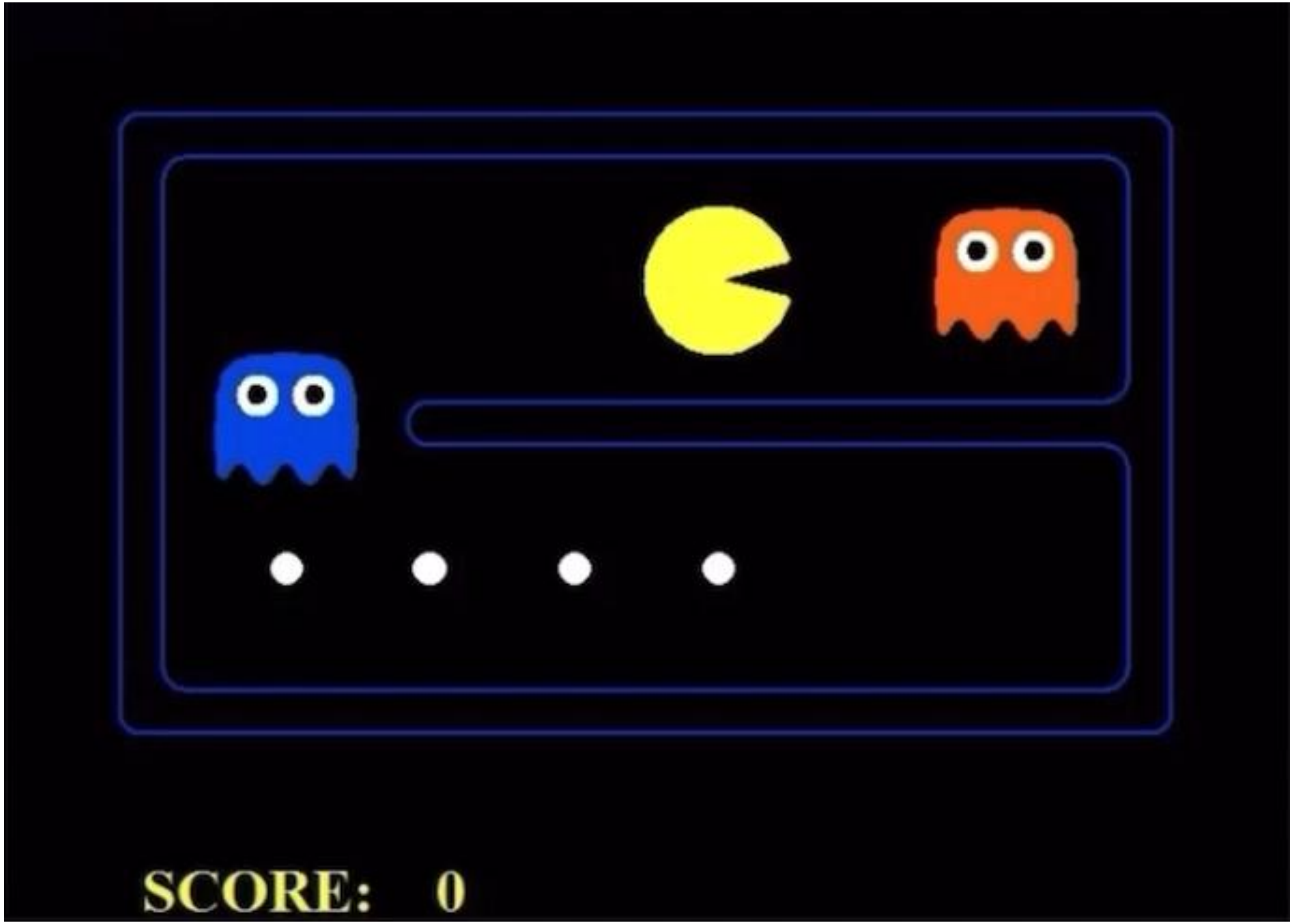
对比：极小极大搜索



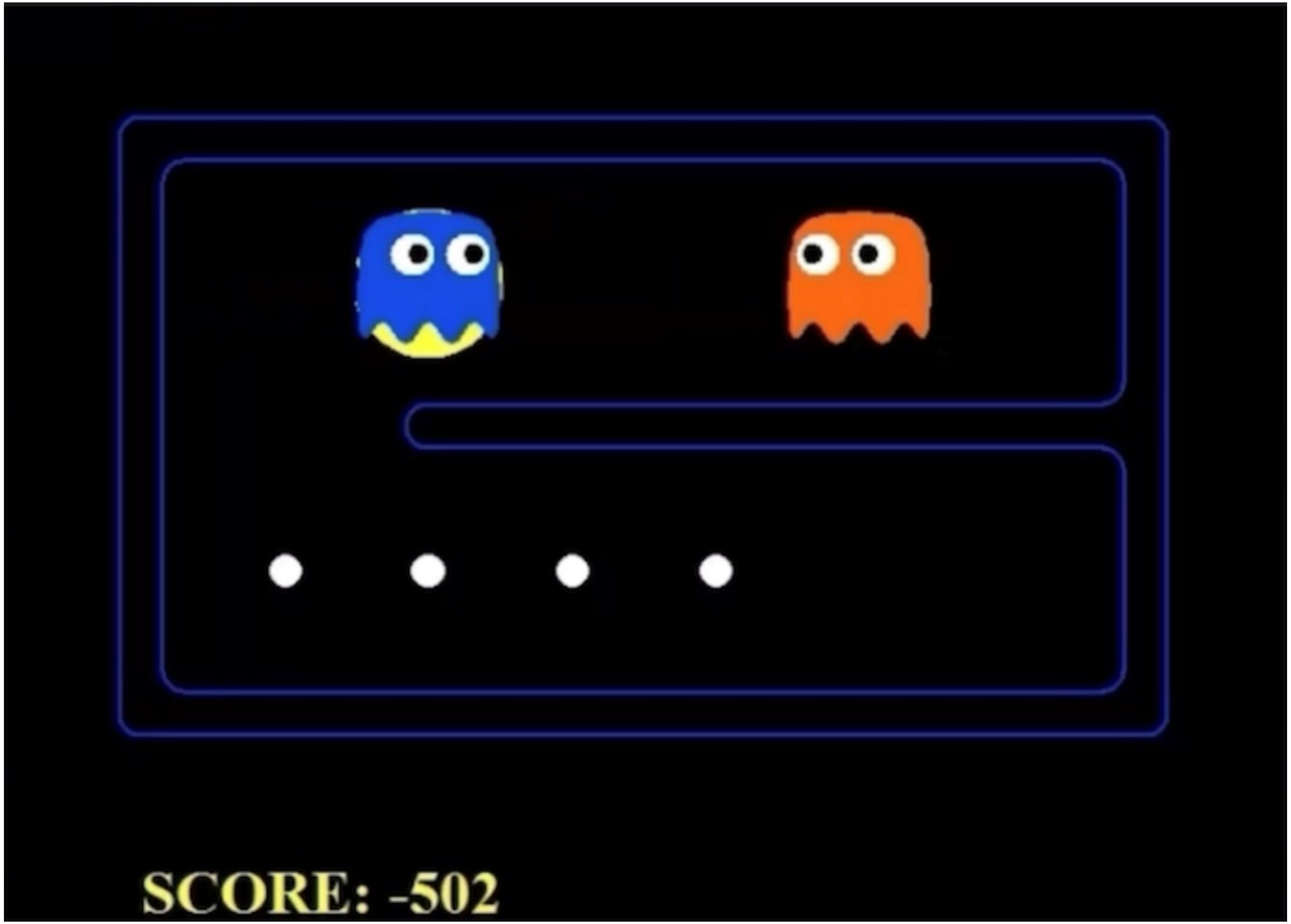
对比：极小极大搜索



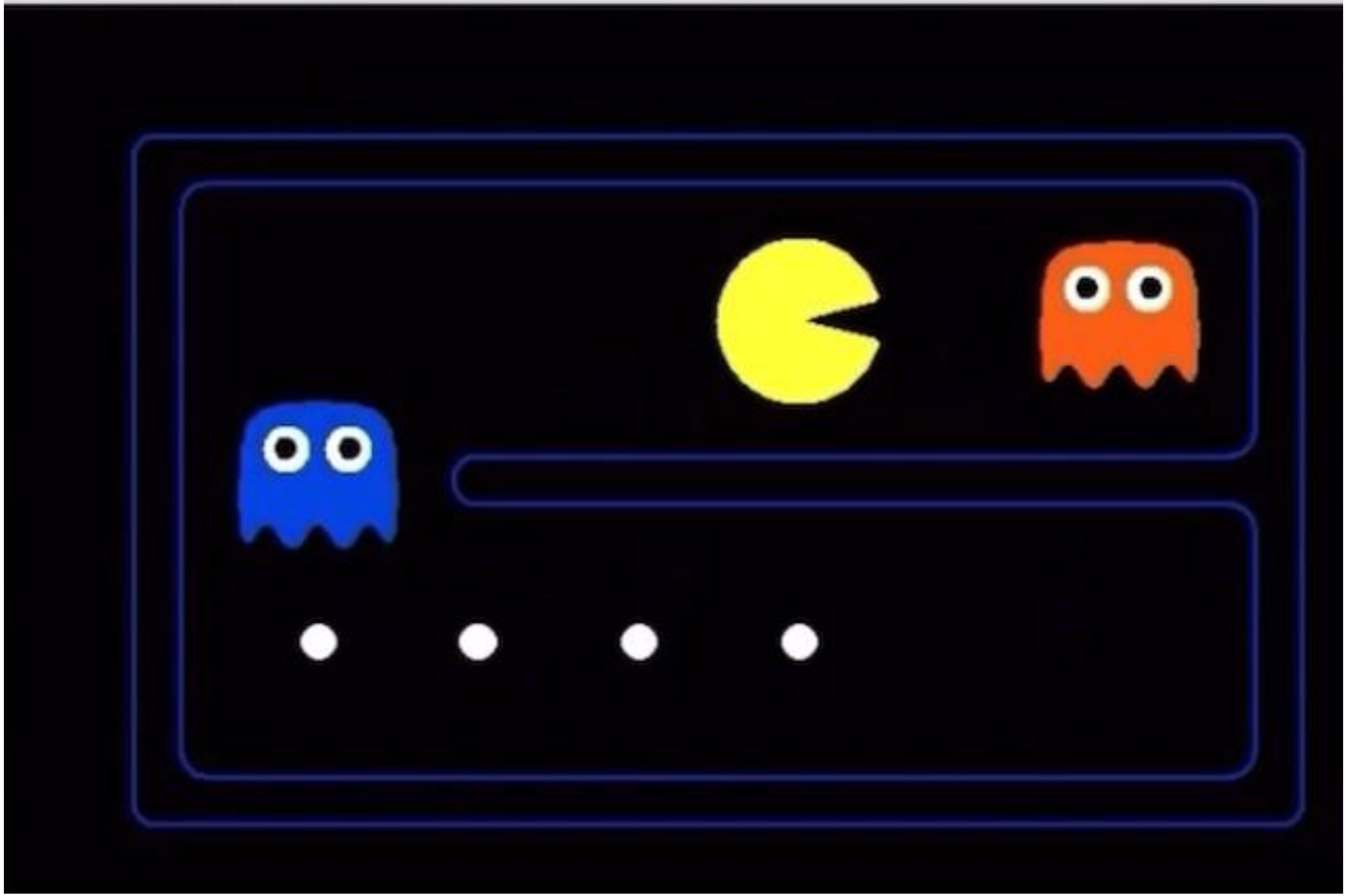
对比：期望最大（小）化搜索



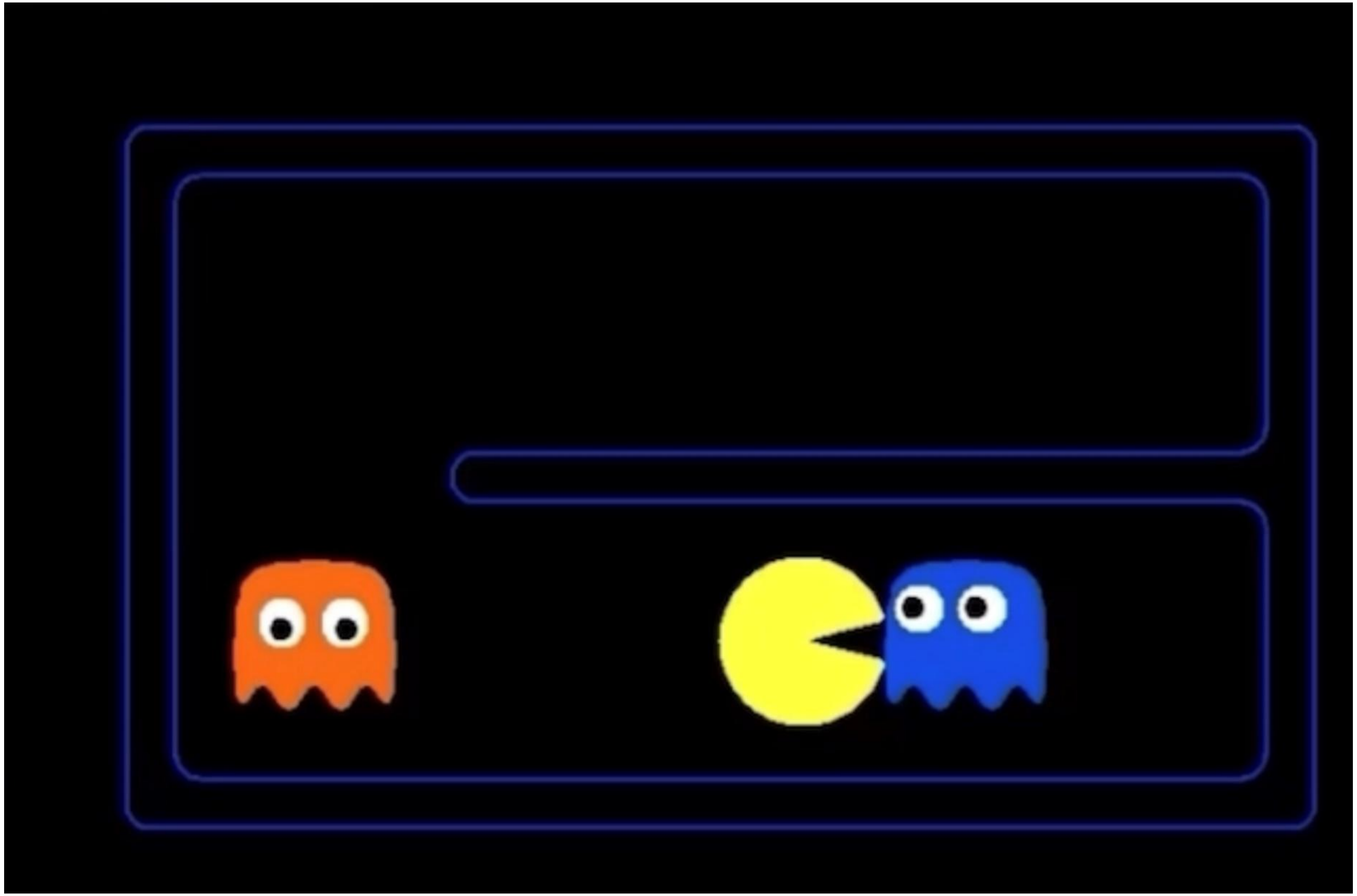
对比：期望最大（小）化搜索



对比：期望最大（小）化搜索



对比：期望最大（小）化搜索



期望最大化算法流程图

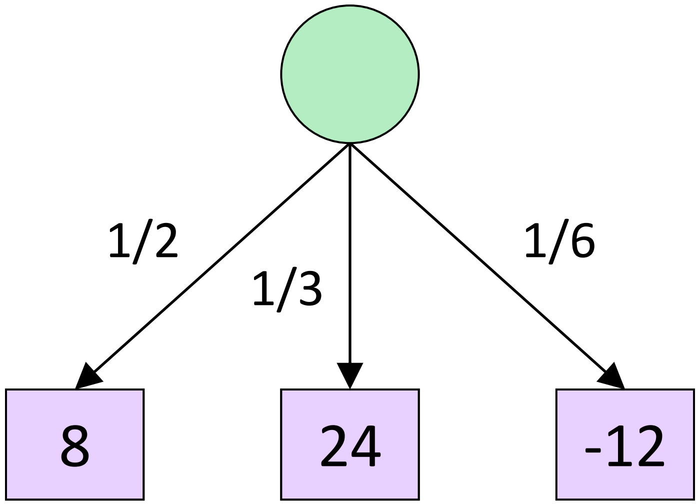
```
def value(state):
    if the state is a terminal state: return the state's utility
    if the next agent is MAX: return max-value(state)
    if the next agent is EXP: return exp-value(state)
```

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

```
def exp-value(state):
    initialize v = 0
    for each successor of state:
        p = probability(successor)
        v += p * value(successor)
    return v
```

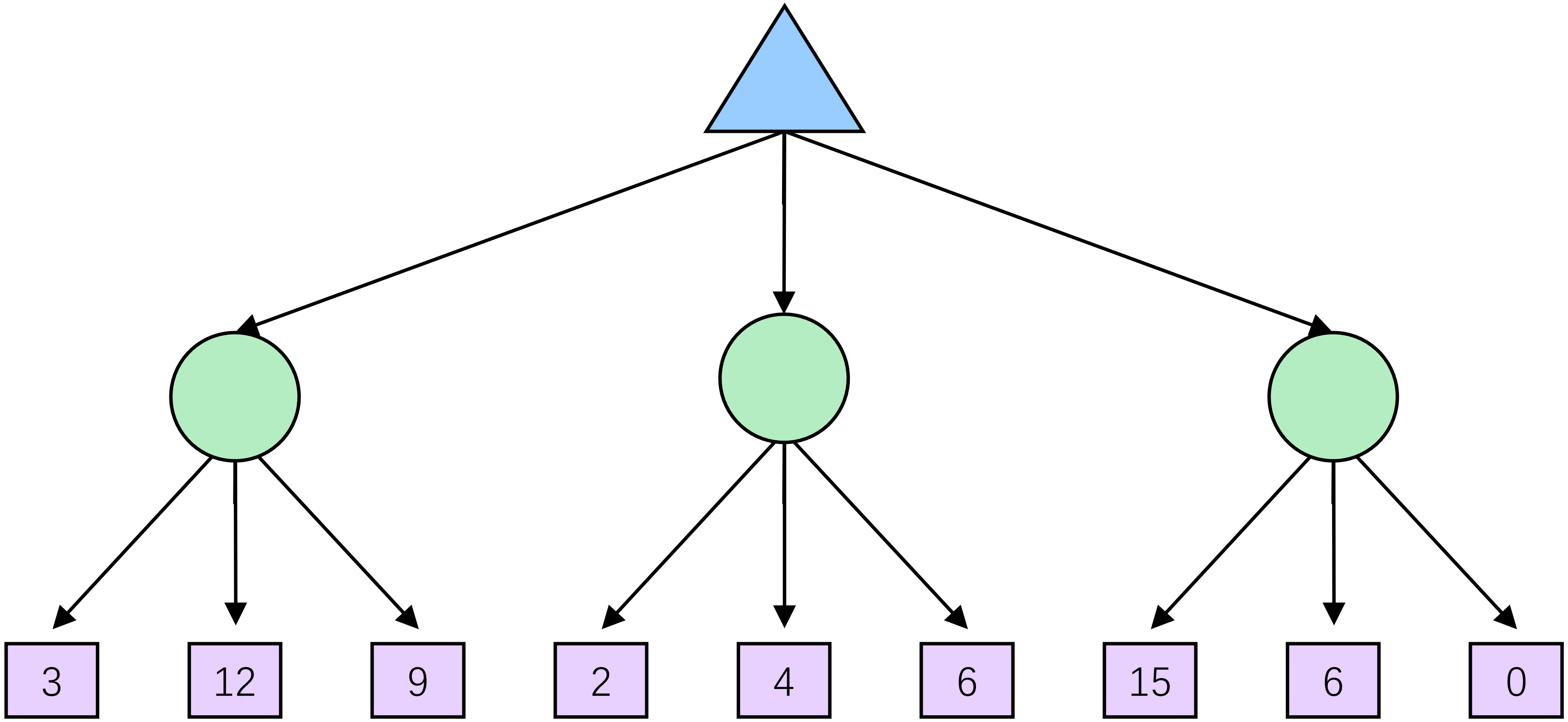
期望最大化算法

```
def exp-value(state):
    initialize v = 0
    for each successor of state:
        p = probability(successor)
        v += p * value(successor)
    return v
```

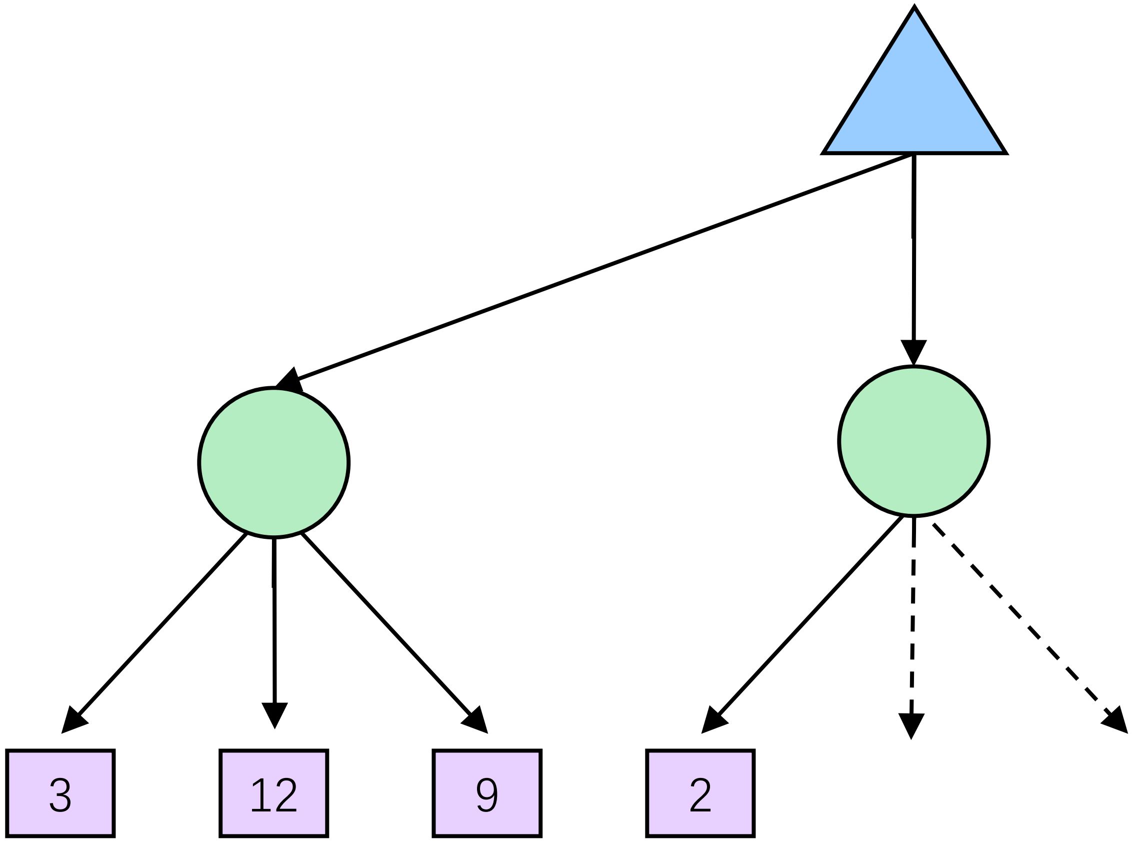


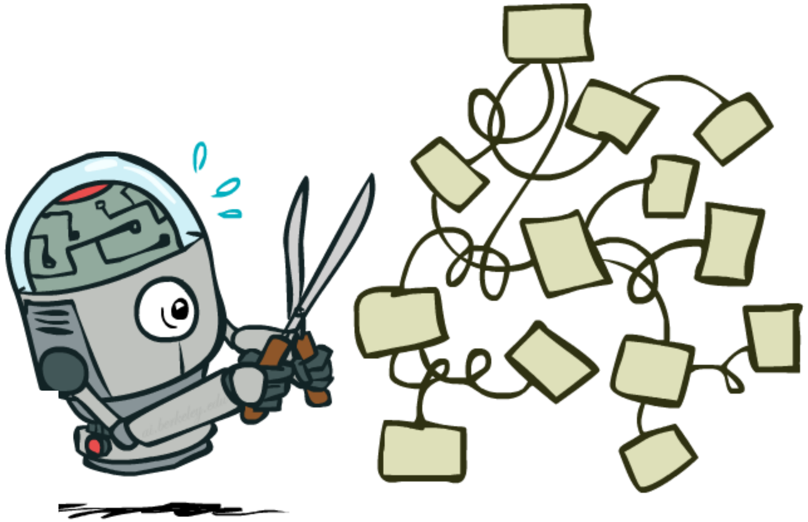
$$v = (1/2)\,(8) + (1/3)\,(24) + (1/6)\,(-12) = 10$$

期望最大化算法示例

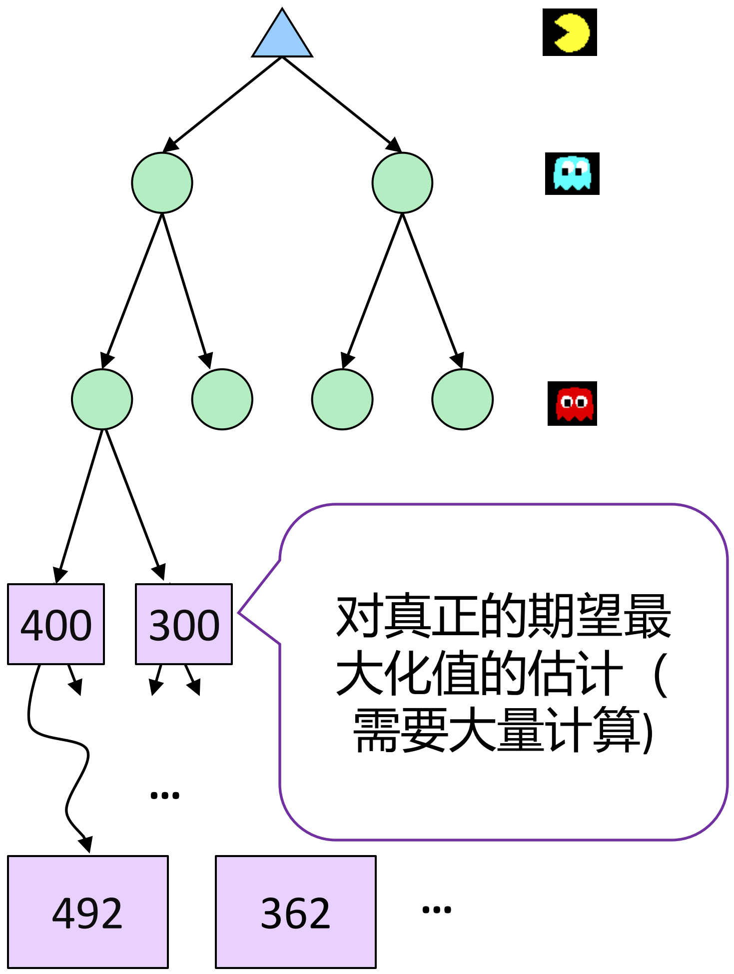


期望最大化剪枝?





深度受限的期望最大化算法

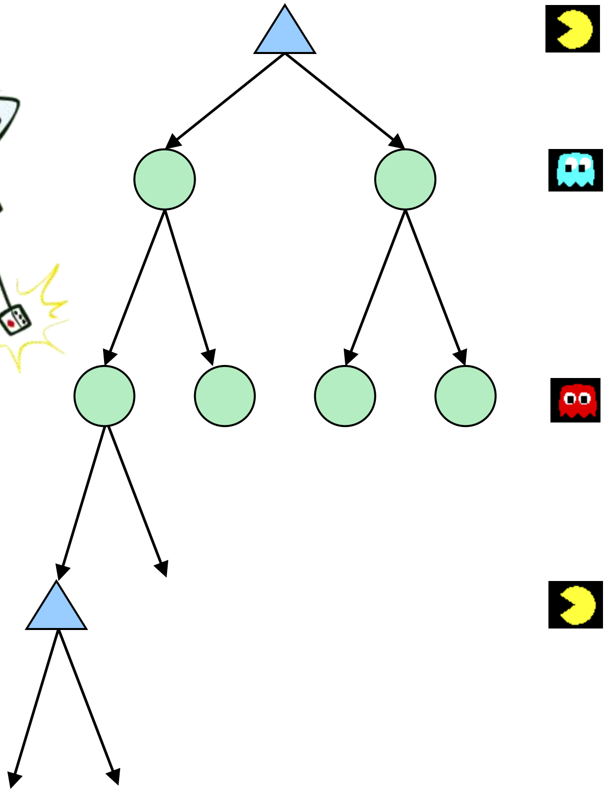


用什么概率?

在期望最大化搜索中，我们有一个概率模型，它能预测对手（或环境）在给定状态下的行为

- 这种模型可以是简单的均匀分布（比如掷骰子）
- 也可以是复杂的需要大量计算的模型
- 由于我们无法控制对手或环境的某些结果，因此需要为这些结果设置一个概率节点
- 这个模型可能表明，对手的行动很有可能是具有对抗性的！

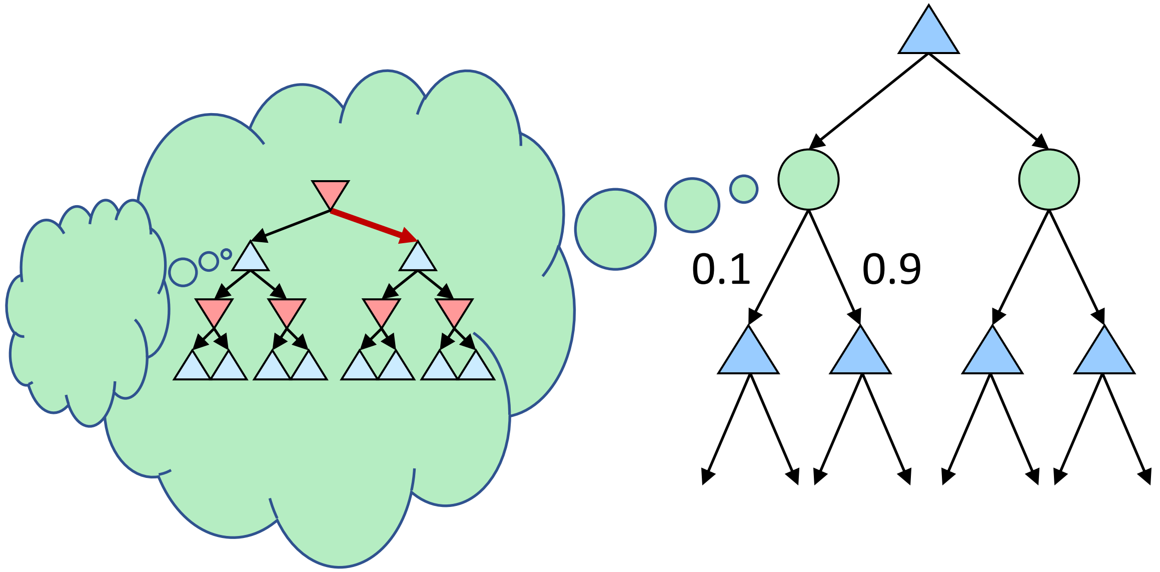
我们暂且假设每个概率节点都自动附带一个概率值，用以参数化它的结果分布。



认为对手的动作是不确定的，并不代表对手的行动是存粹地抛硬币！

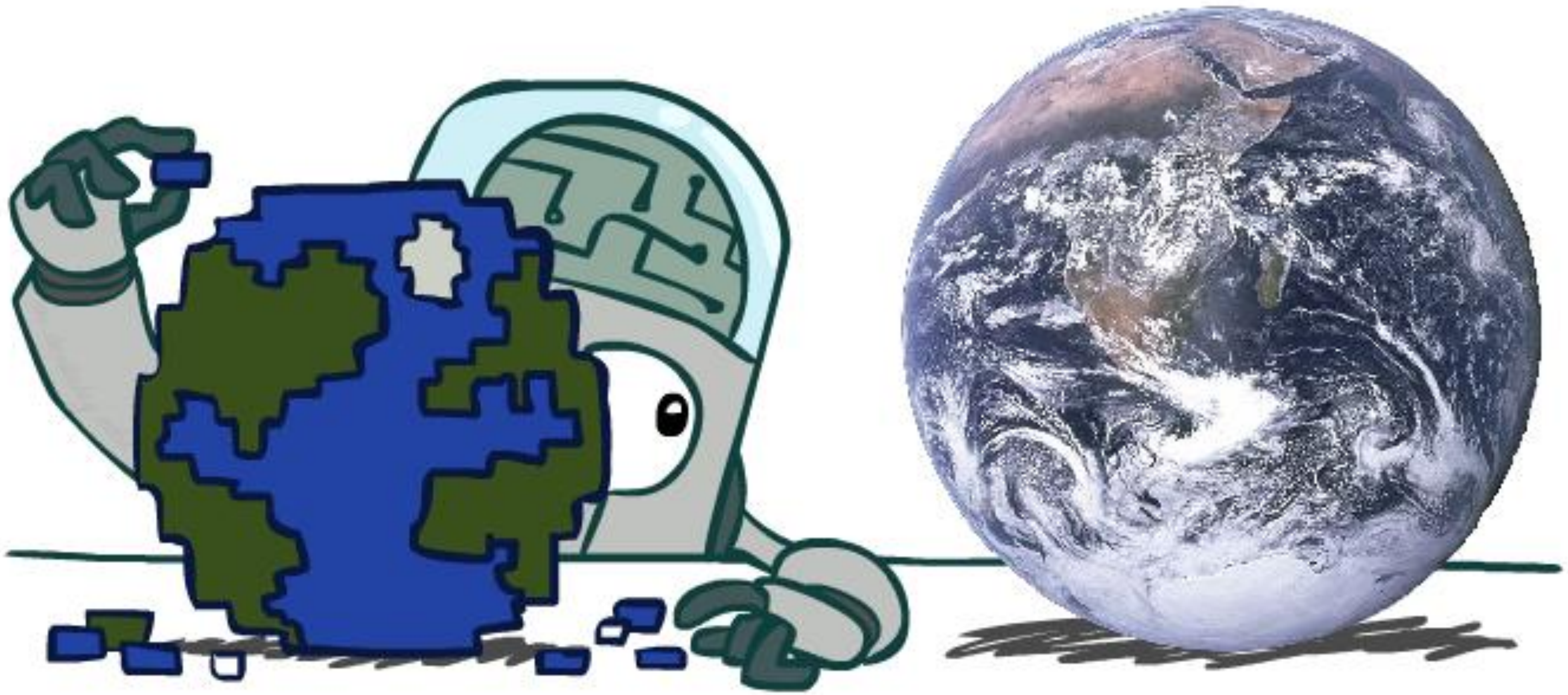
测验: 信息概率

- 假设知道对手正在运行深度为2的极小化搜索算法，80%的时间按照搜索结果进行行动，其余20%的时间随机行动，
- 问题: 你应该使用什么树搜索算法?



- **答案: 期望最大化!**
  - 要确定每个概率节点的概率，你需要模拟仿真你对手的决策过程
  - 这种方法会很快变得非常缓慢
  - 尤其是如果你需要模拟你的对手模拟你的情况时...
  - ... 除非是极小化和极大化，因为它们具有一个好的特性，即它们都能折合成一个博弈树

建模假设



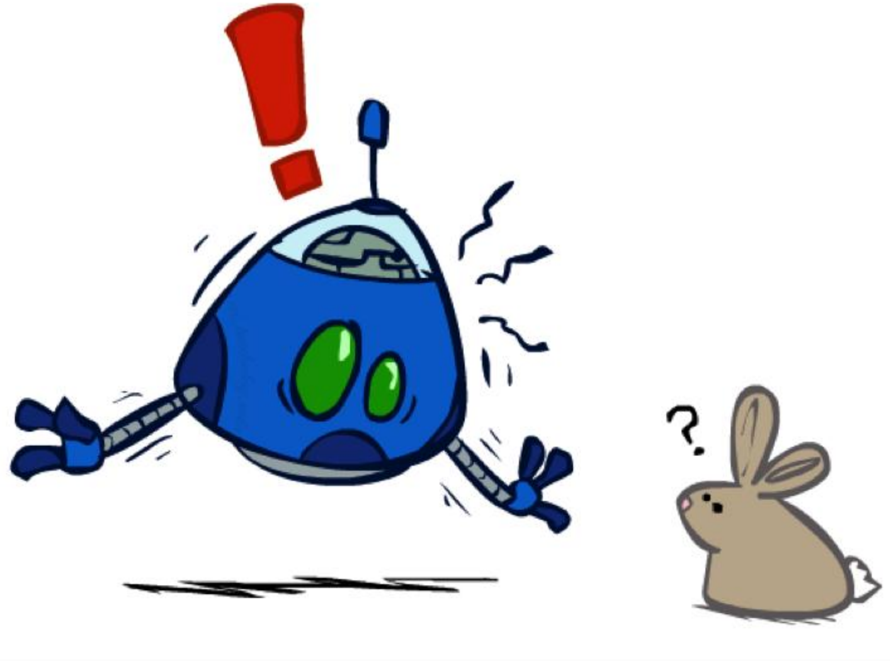
乐观主义和悲观主义的危险性

危险的乐观主义

面对敌对的世界时假定机遇的存在

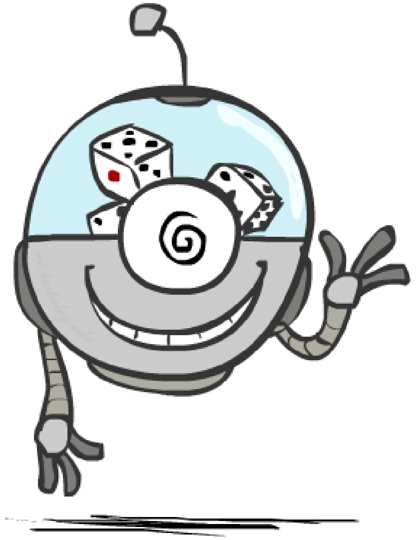
危险的悲观主义

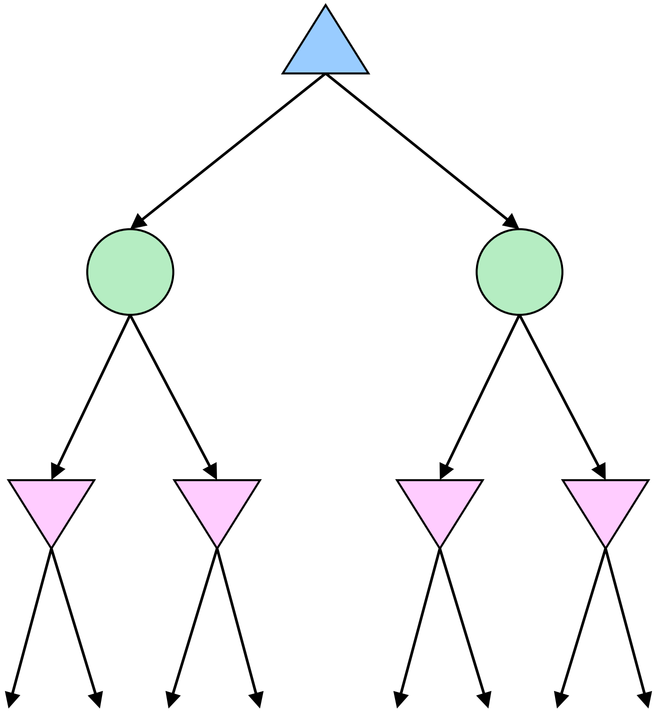
预设不可能发生的最差情况



混合层类型

- 比如：斗地主





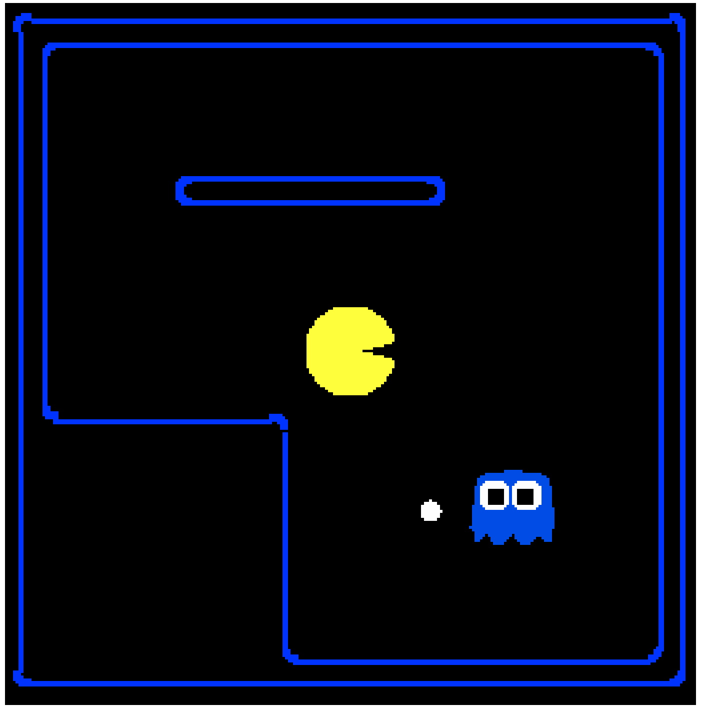
- 期望极小极大(Expectiminimax)：
  - 第三个玩家是“随机”玩家，在每个min/max代理之后移动
  - 每个节点计算其子节点的恰当组合。

总结

- MiniMax是 博弈的通用解法
- Alpha-Beta剪枝减少状态空间
- 用期望来解决不确定性

- 对**特大状态空间和不确定性**有没有更强大的解法？**MCTS**！ （下节）

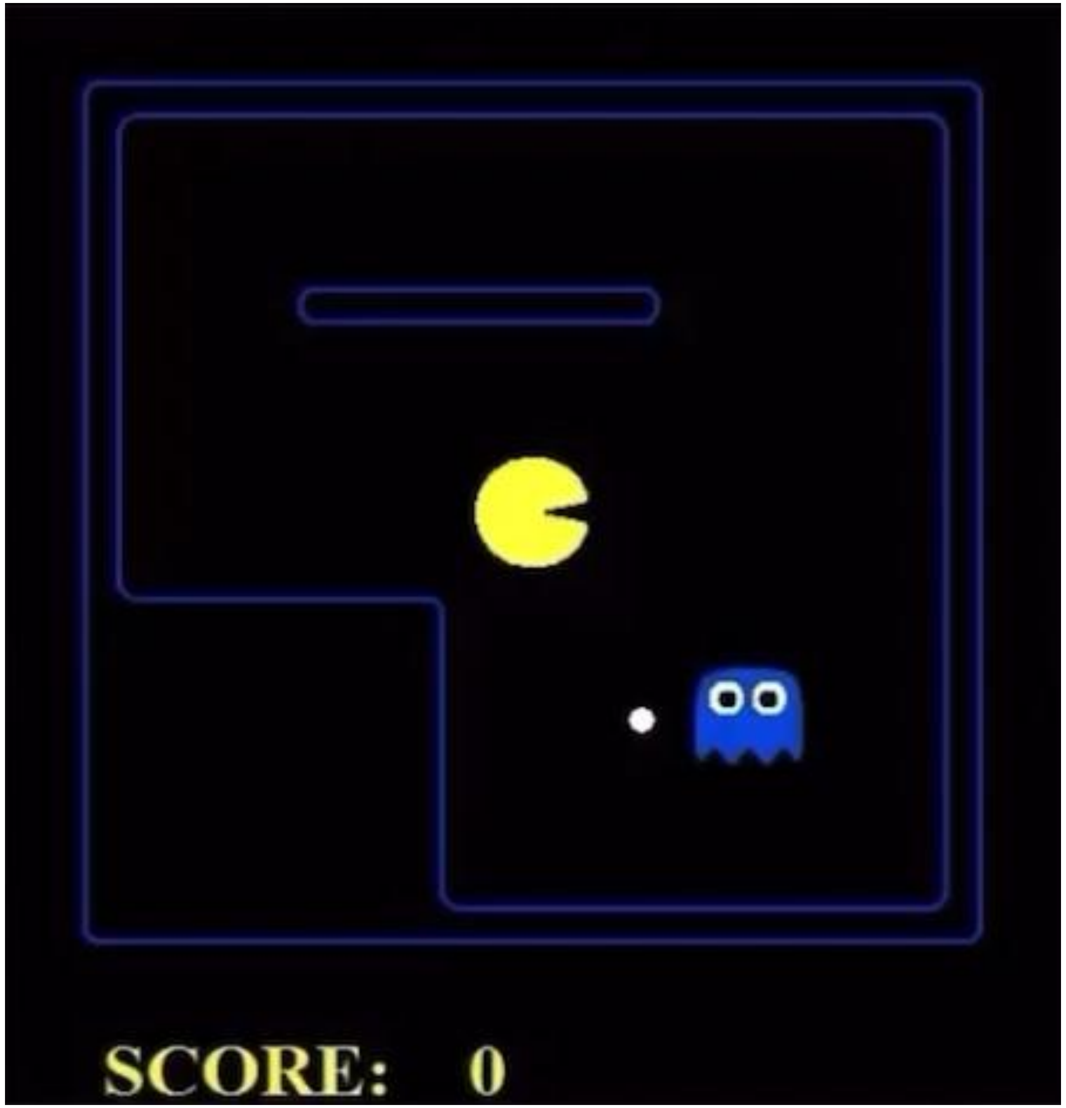
假设 vs. 实际



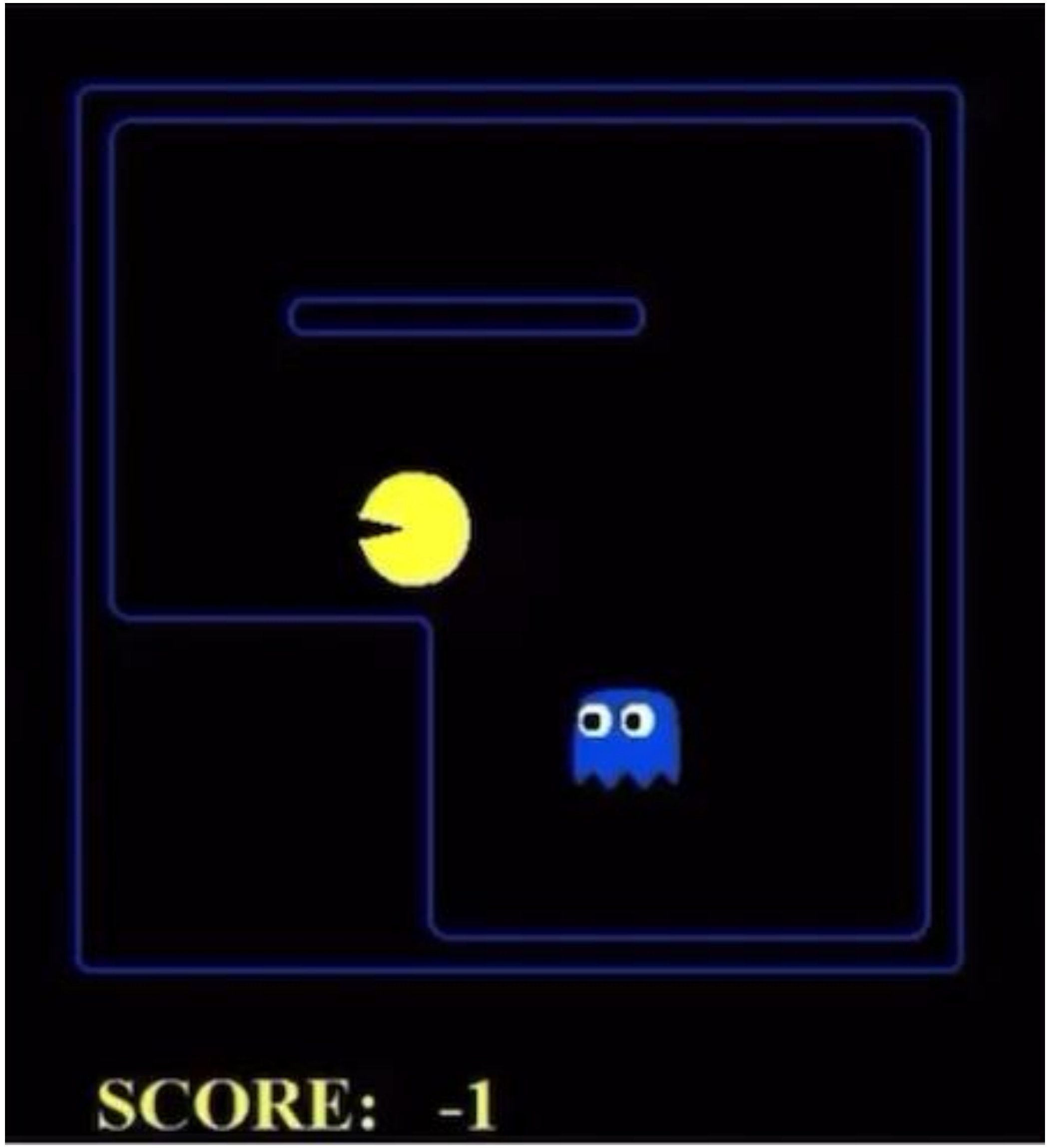
	对抗 Ghost	随机 Ghost
Minimax Pacman	获胜 5/5 平均分数: 483	获胜 5/5 平均分数: 493
Expectimax Pacman	获胜 1/5 平均分数: -303	获胜 5/5 平均分数: 503

Pacman使用深度4搜索，并使用一个“避免危险”的估值函数；
幽灵使用深度2搜索，并使用一个“寻找Pacman”的估值函数。

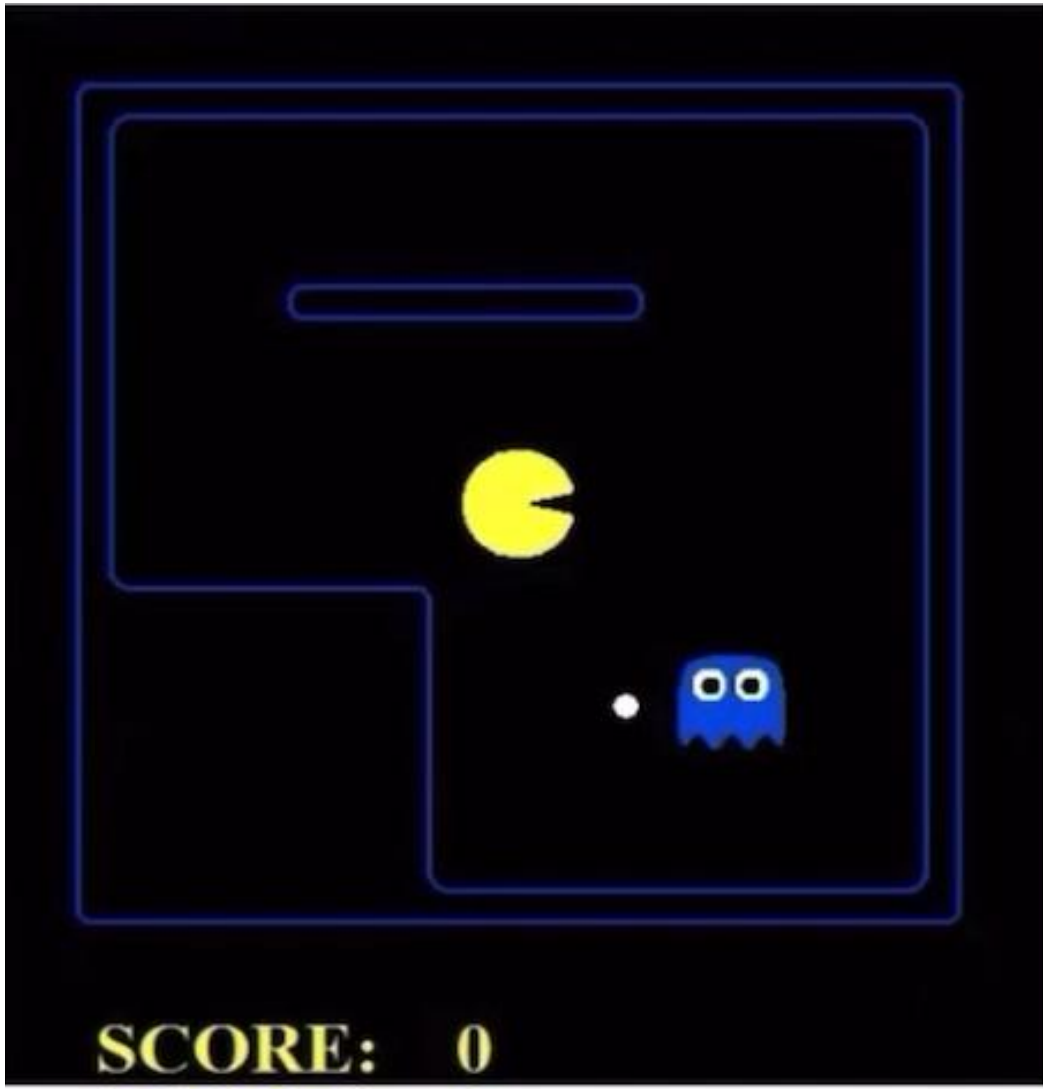
随机鬼，期望最大化Pacman



对抗性鬼，极小极大Pacman



对抗性鬼，期望最大化Pacman



随机鬼，极小极大Pacman