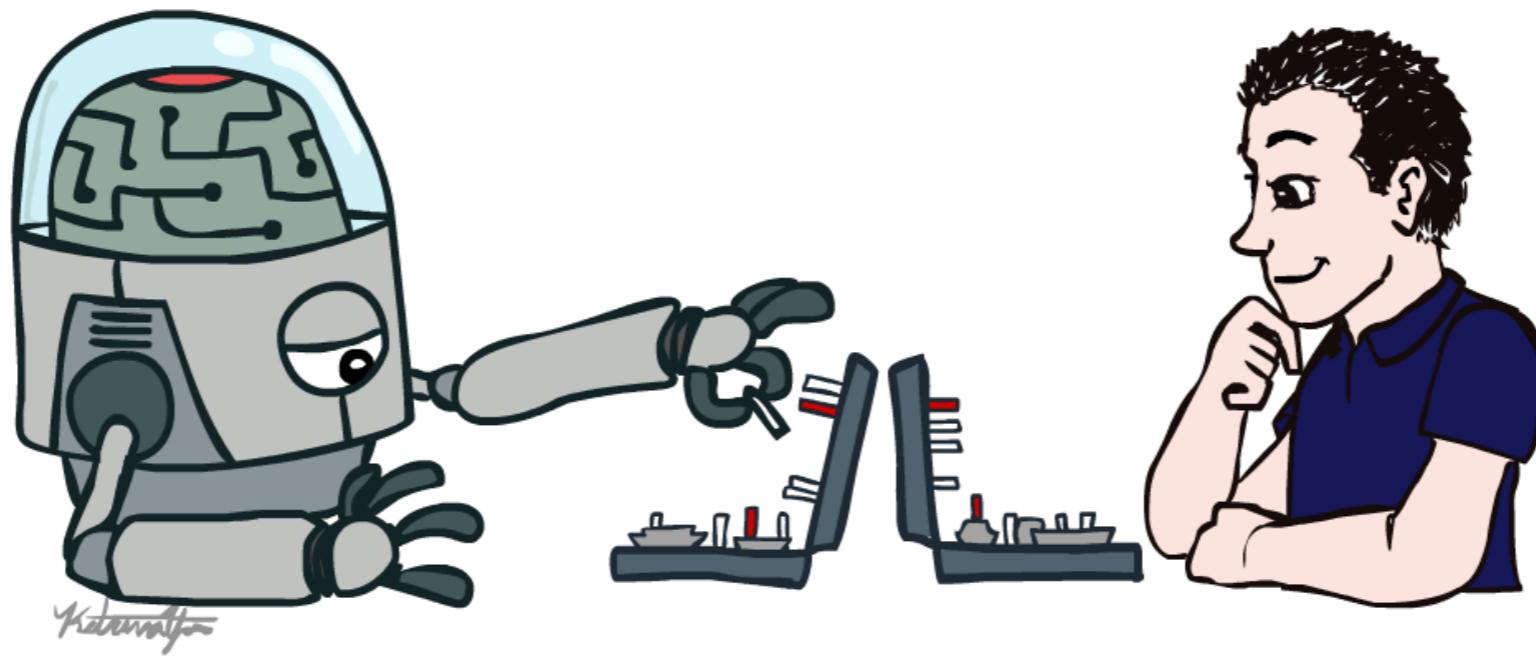
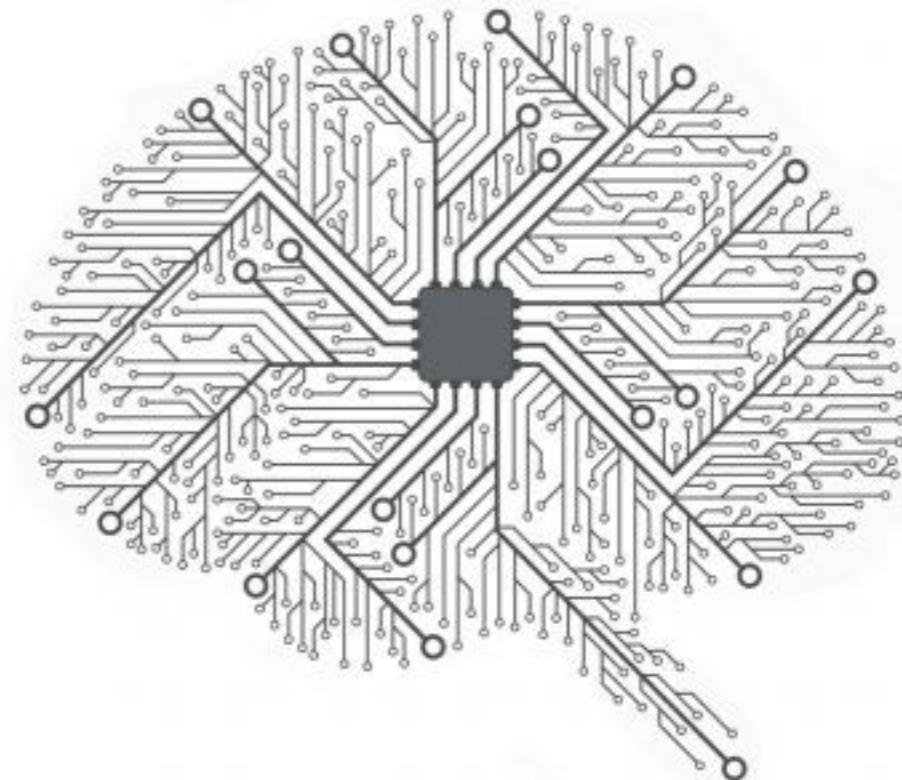


人工智能



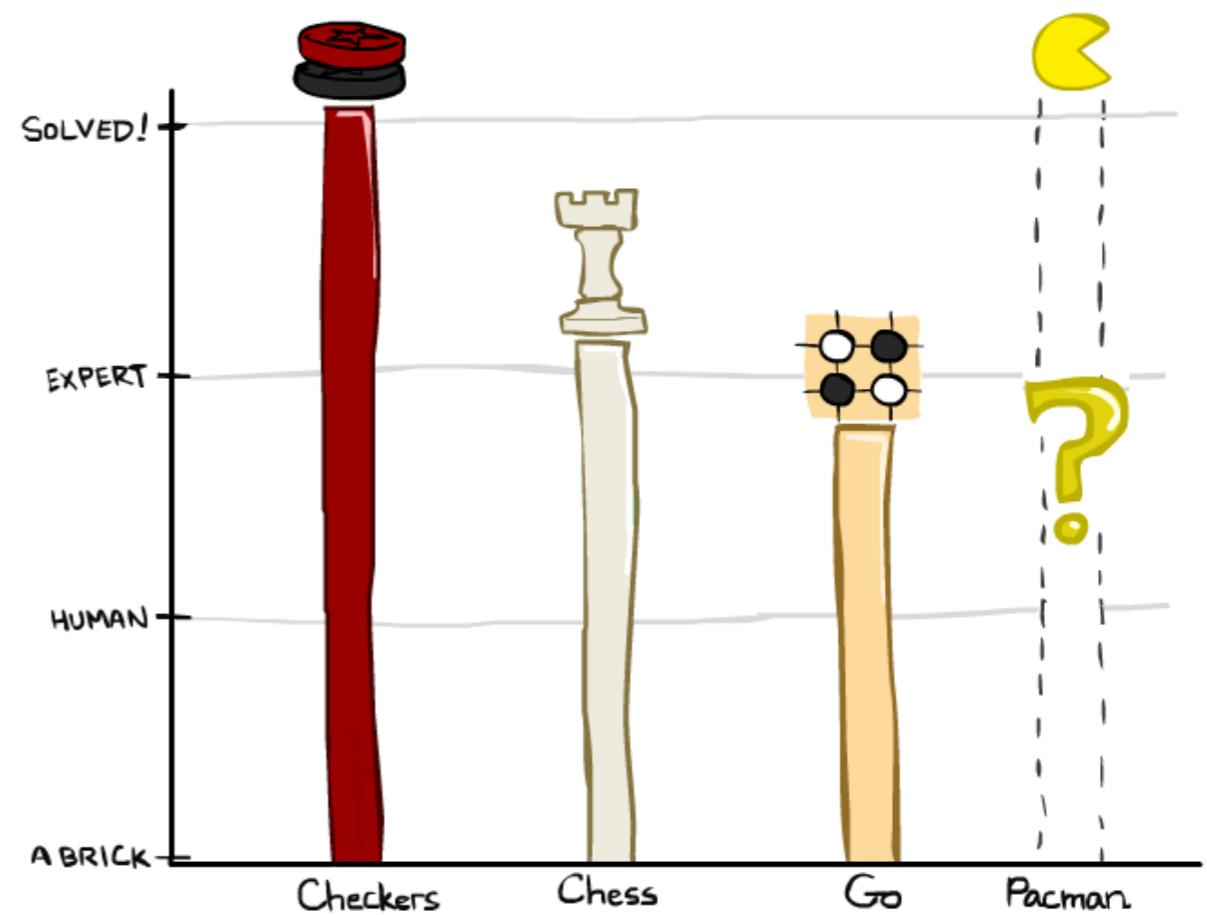
第五章·对抗性搜索

- 零和游戏 (最小最大值)
- 搜索效率 (α - β 剪枝)
- 机遇博弈 (期望最大值)
- 评估函数 (效用)

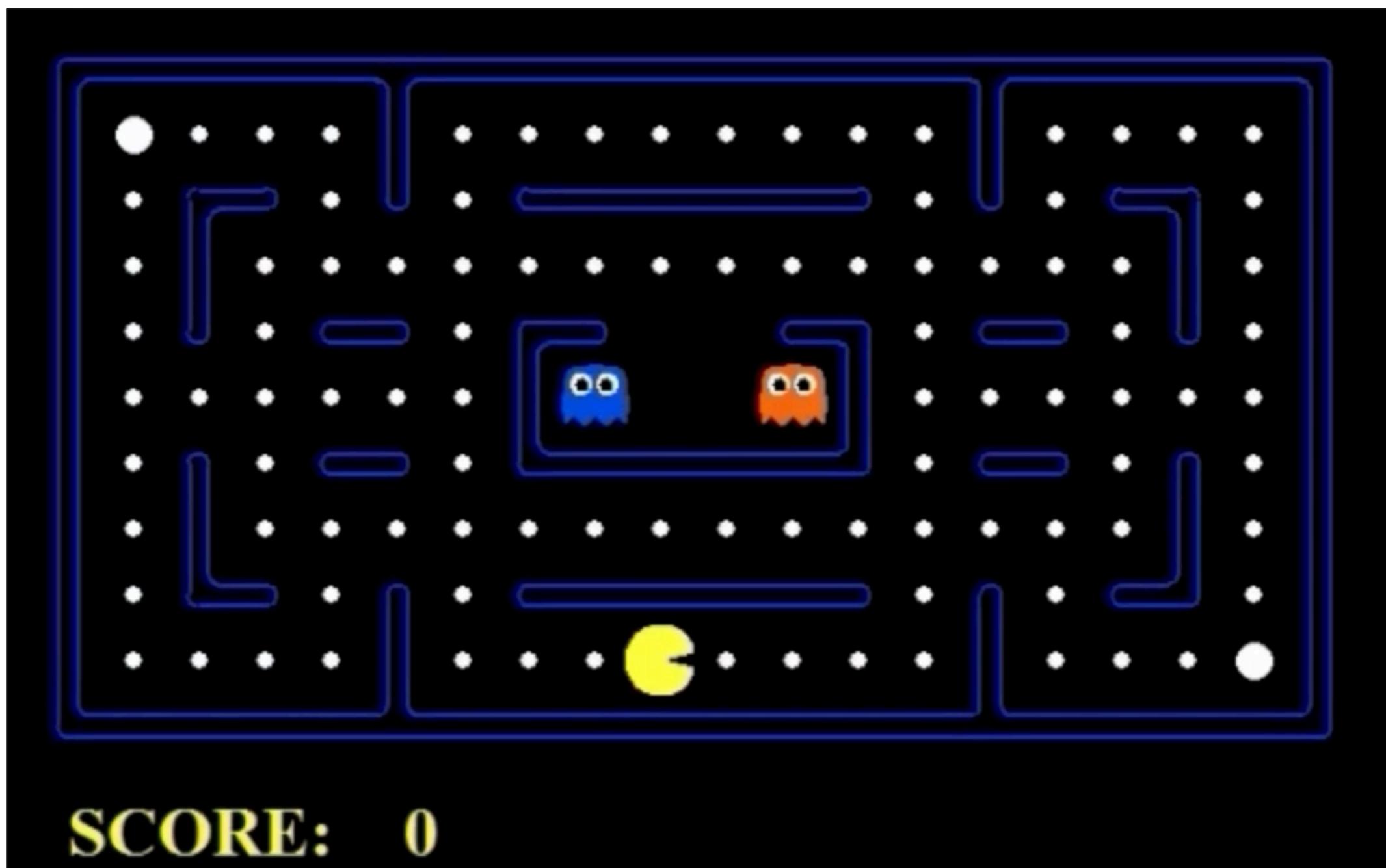


计算机游戏/博弈的当前水平

- 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
 - 2017年，围棋程序AlphaGo击败人类冠军
- Pacman ?

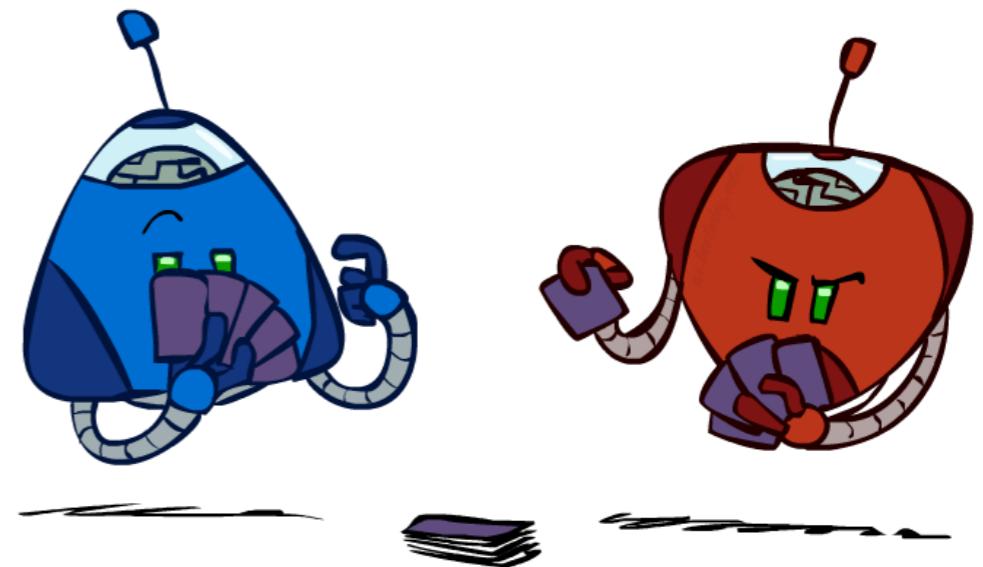


计算机控制Pacman吃豆



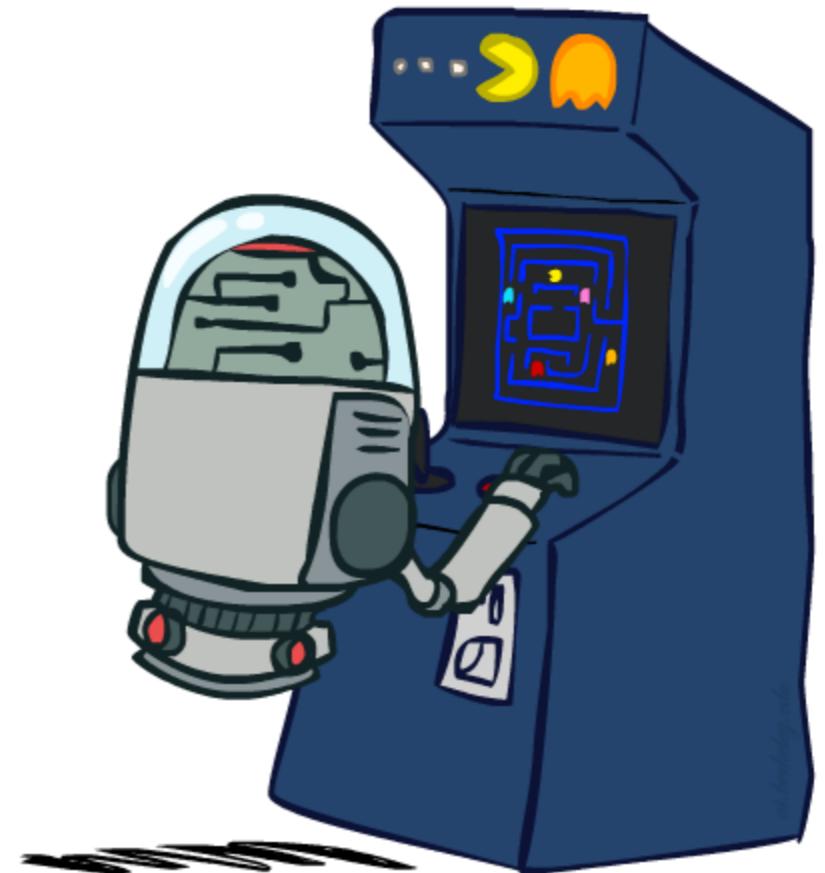
游戏/博弈类型

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

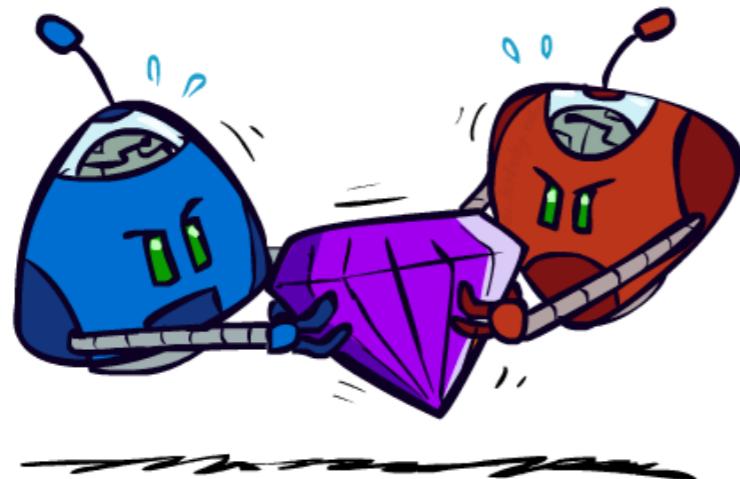


人工智能所研究的“标准的”游戏比赛(games)

- 游戏问题的模型建立:
 - 状态集: S , 其中初始状态 s_0
 - 玩家: $P=(1\dots N)$ 通常轮流行动
 - 行动: Action 玩家可能的移动
 - 状态转换模型: $S \times A \rightarrow S$
 - 终局状态检测: $S \rightarrow \{t, f\}$
 - 终局结果/效用(Utility): $S \times P \rightarrow R$
- 解决方案是一个**行动策略** $S \rightarrow A$
- 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。



零和游戏



零和游戏

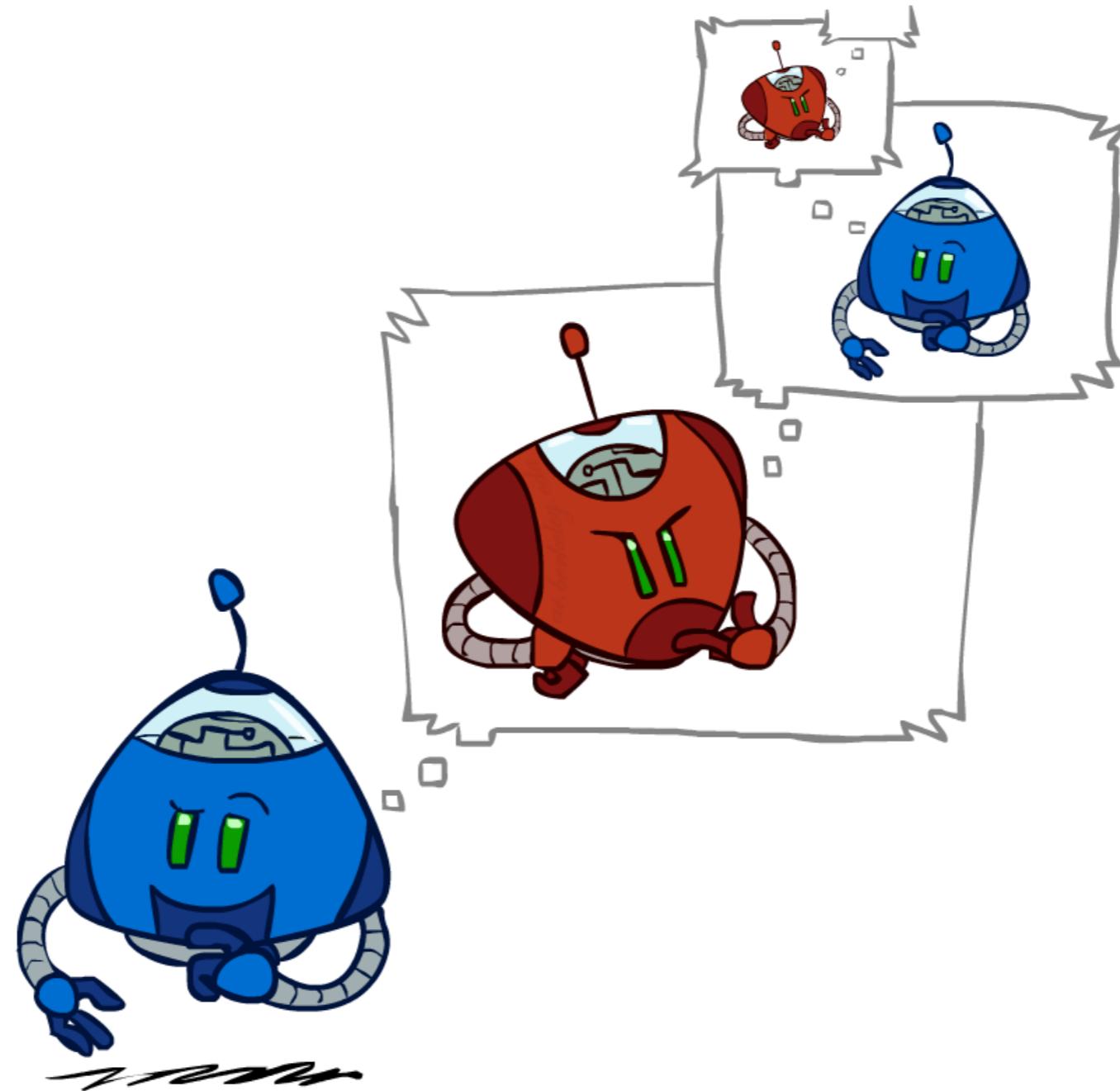
- 智能体竞争实现相反的利益
- 一方最大化这个利益,另一方最小化它



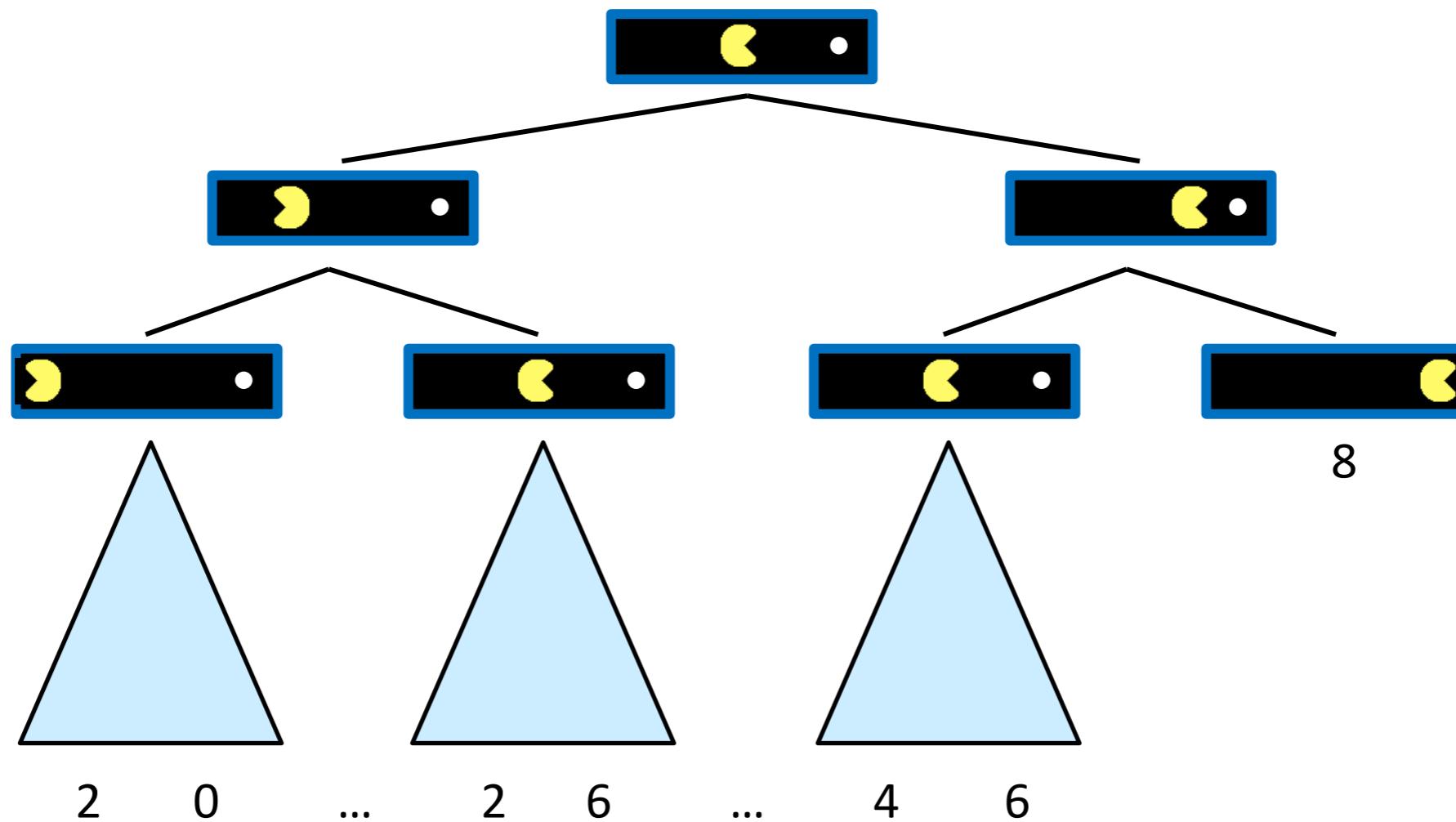
通常游戏

- 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



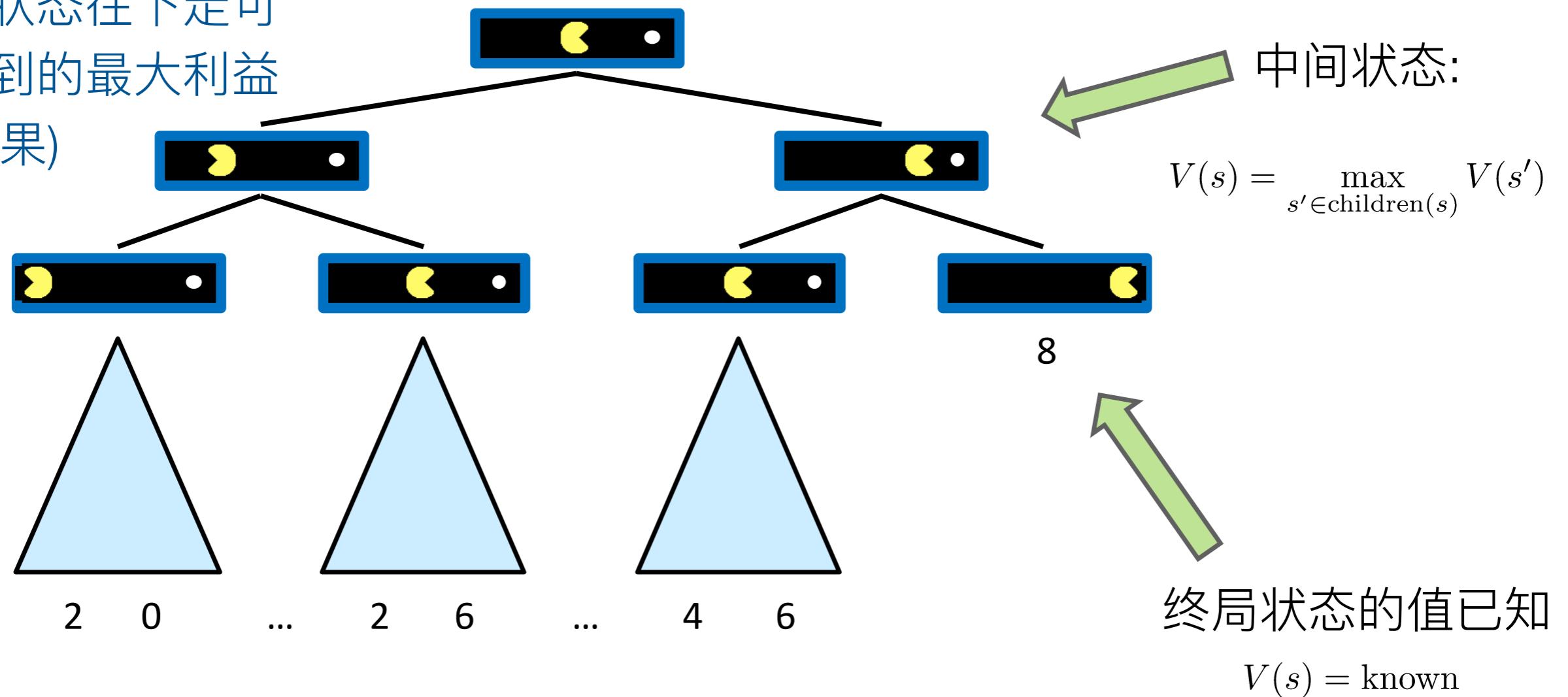
单一智能体搜索树



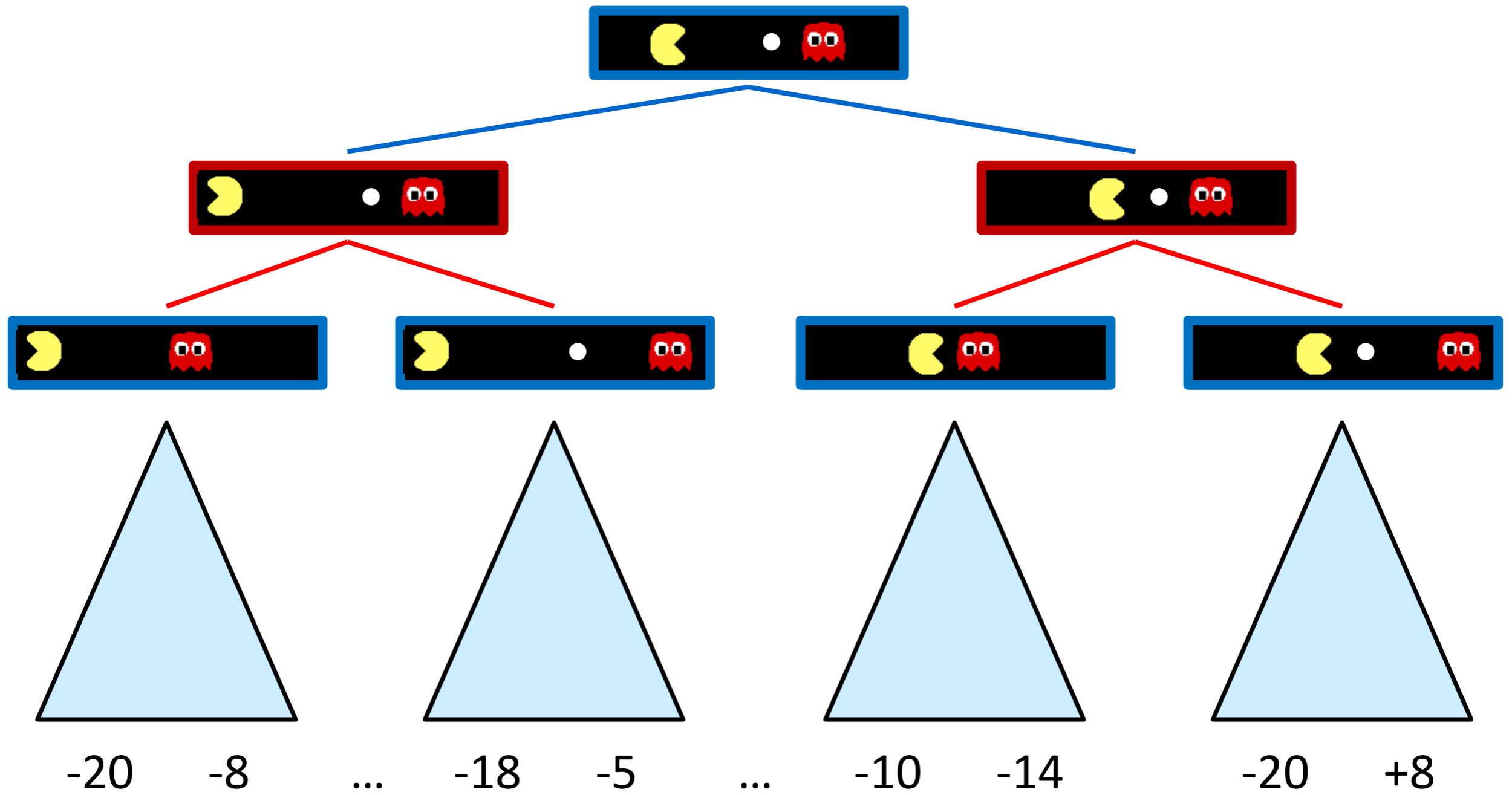
效用值 = 吃豆数 \times 10 - 步数

状态值(中间和终局状态值)

一个状态的值：从这个状态往下走可能达到的最大利益值(结果)



对抗性搜索树



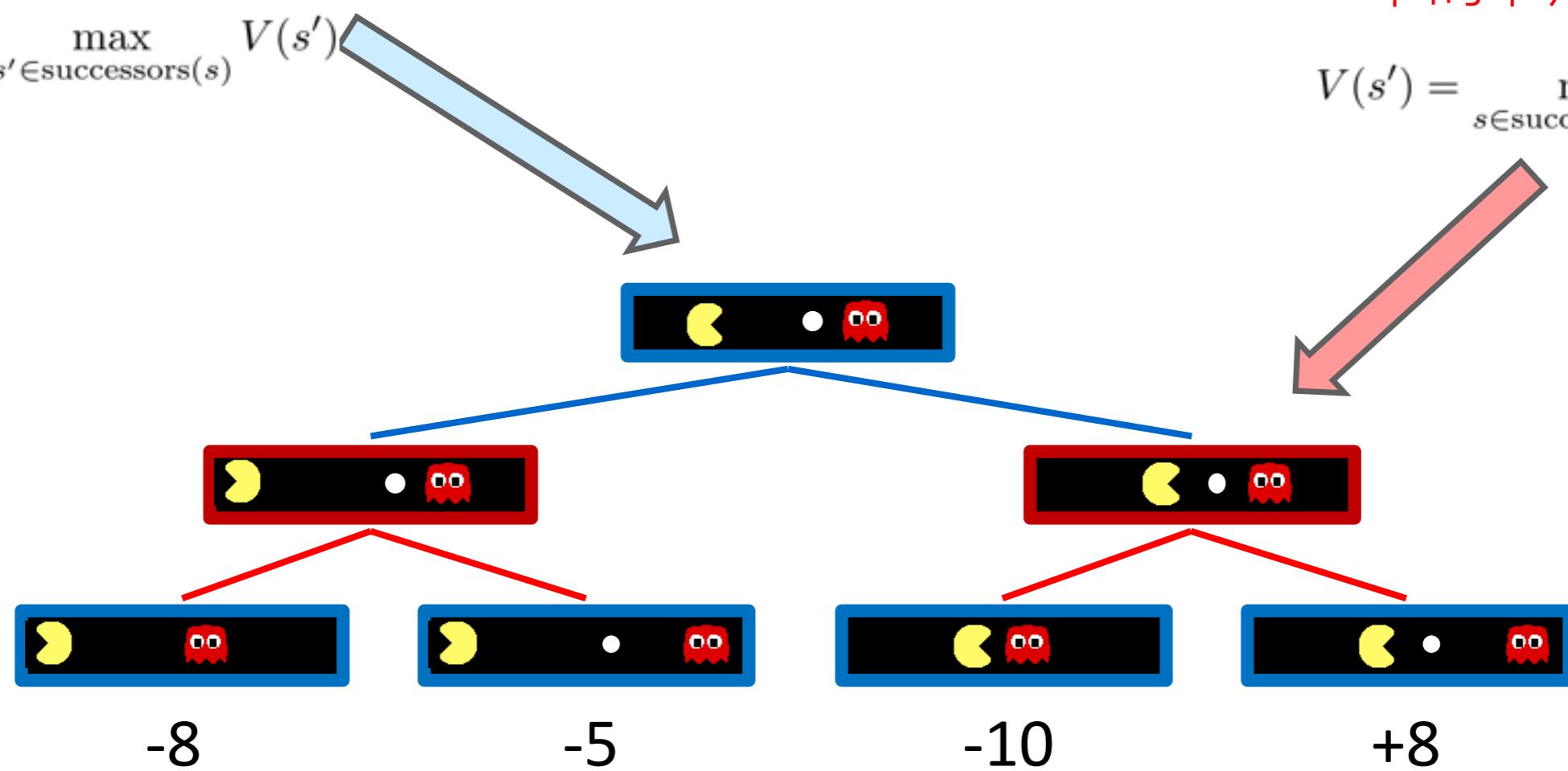
极大极小值(Minimax values)

MAX节点:自己控制
下的节点状态

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

MIN节点:对手控制
下的节点状态

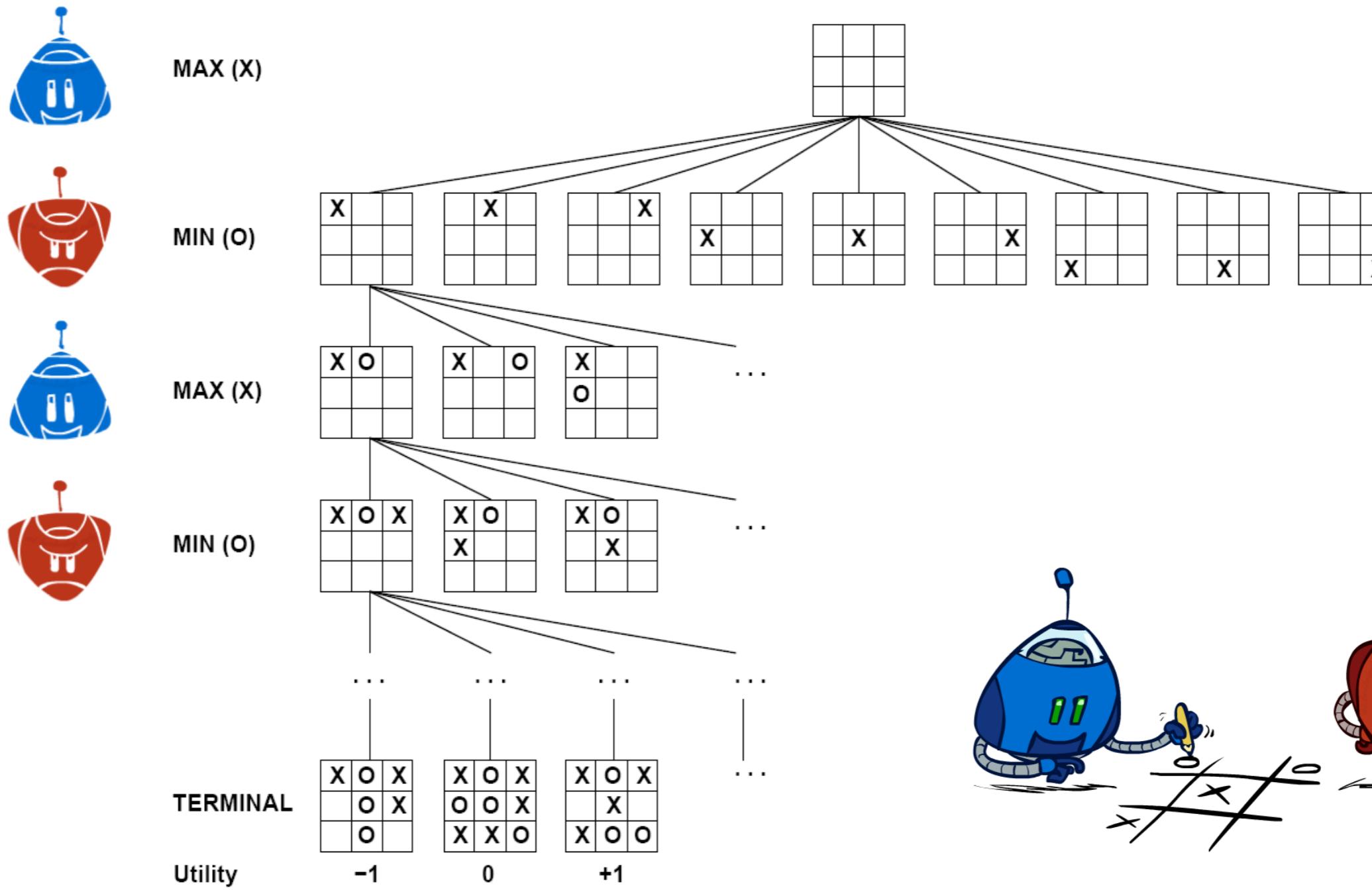
$$V(s') = \min_{s \in \text{successors}(s')} V(s)$$



终局状态的值是已知的

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

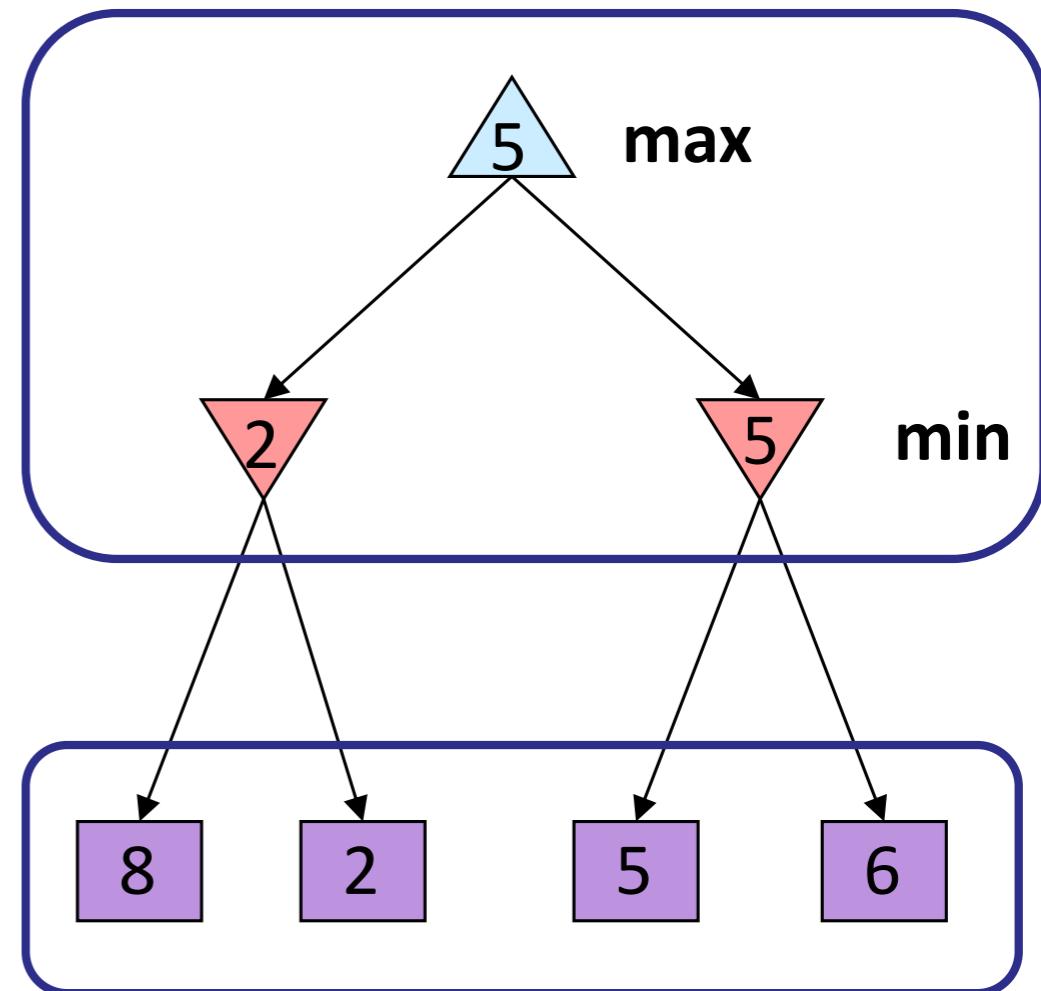
井字棋搜索树



对抗搜索 (Adversarial Search)

- 确定性, 零和游戏:
 - 井字棋、国际象棋、围棋
 - 一个玩家使结果最大化
 - 另一个玩家使结果最小化
- 极大极小搜索(Minimax):
 - 基于状态空间搜索树
 - 模拟玩家交替进行游戏
 - 计算每个节点的**极大极小值**: 针对一个合理(最优)对手的最佳可实现效用

**Minimax values:
computed recursively**



**Terminal values:
part of the game**

极大极小搜索的实现

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

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

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

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

极大极小搜索的实现（调度）

```
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, min-
 value(successor))

 return v

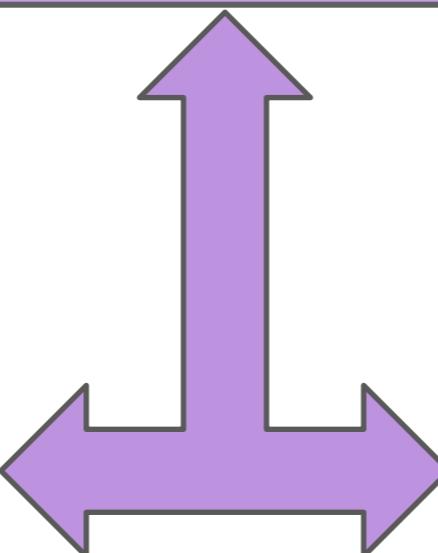
```
def min-value(state):
```

 initialize v = +∞

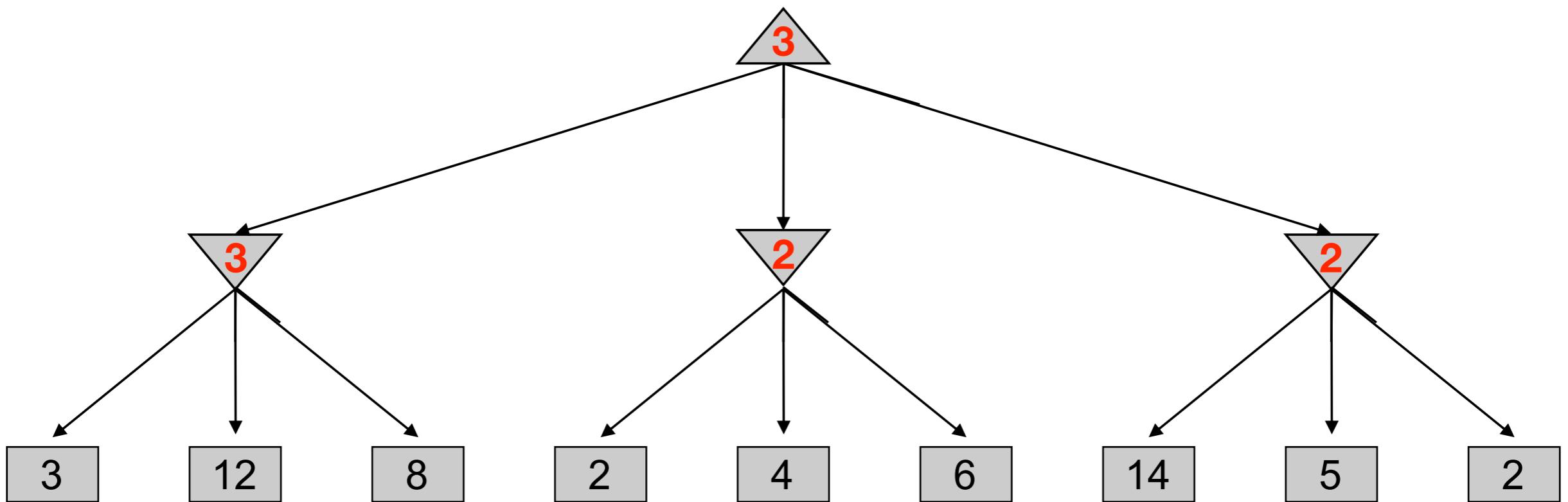
 for each successor of
 state:

 v = min(v, max-
 value(successor))

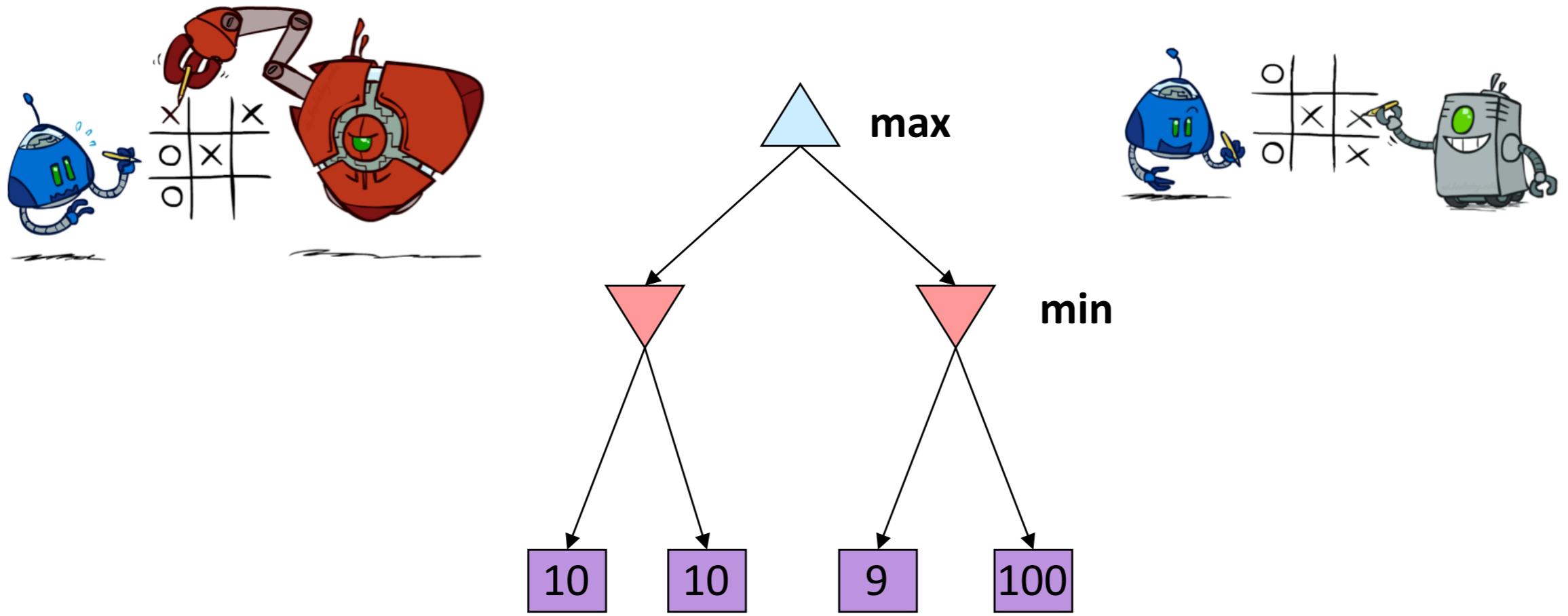
 return v



极大极小搜索示例

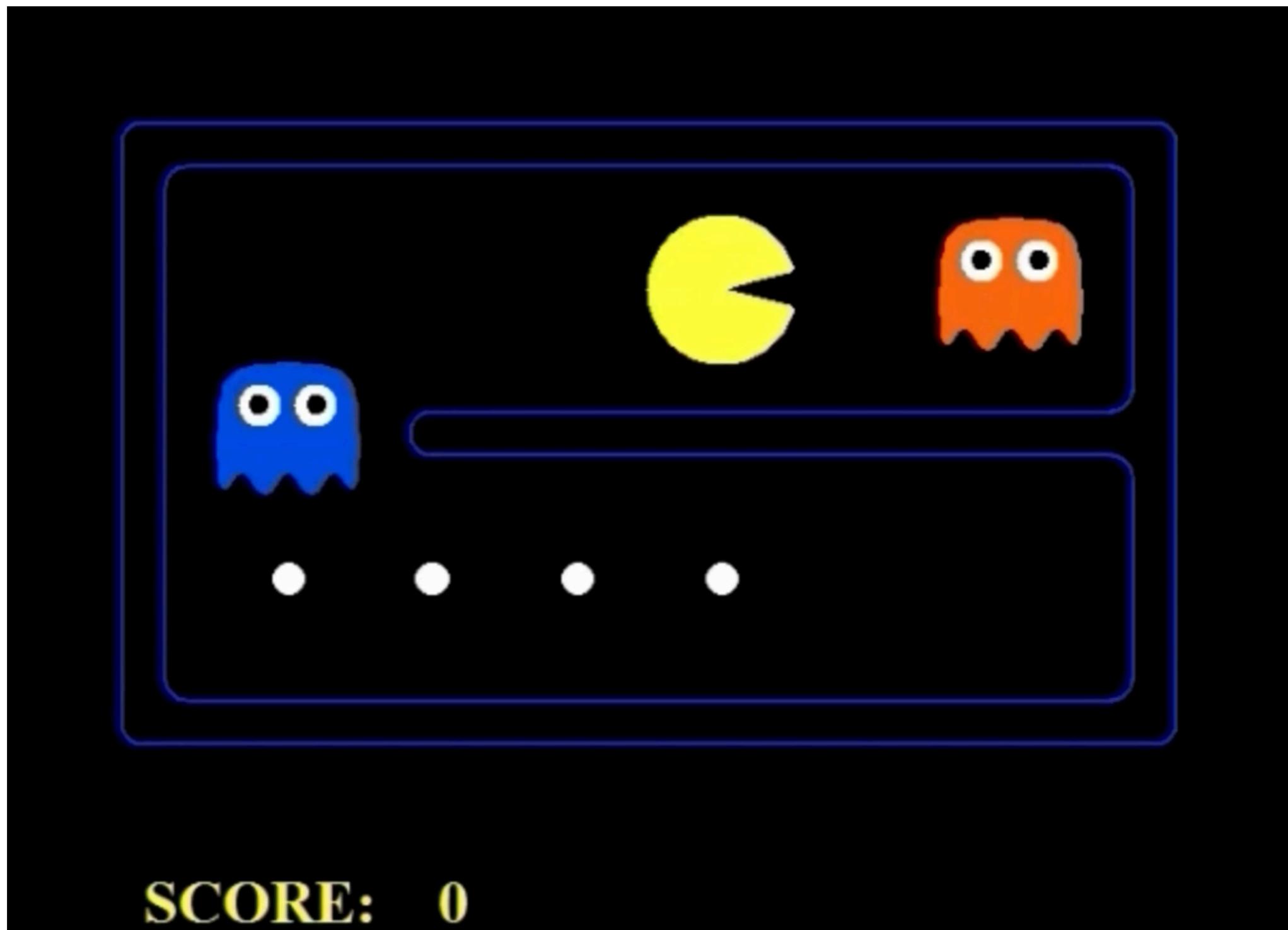


极大极小搜索的性质

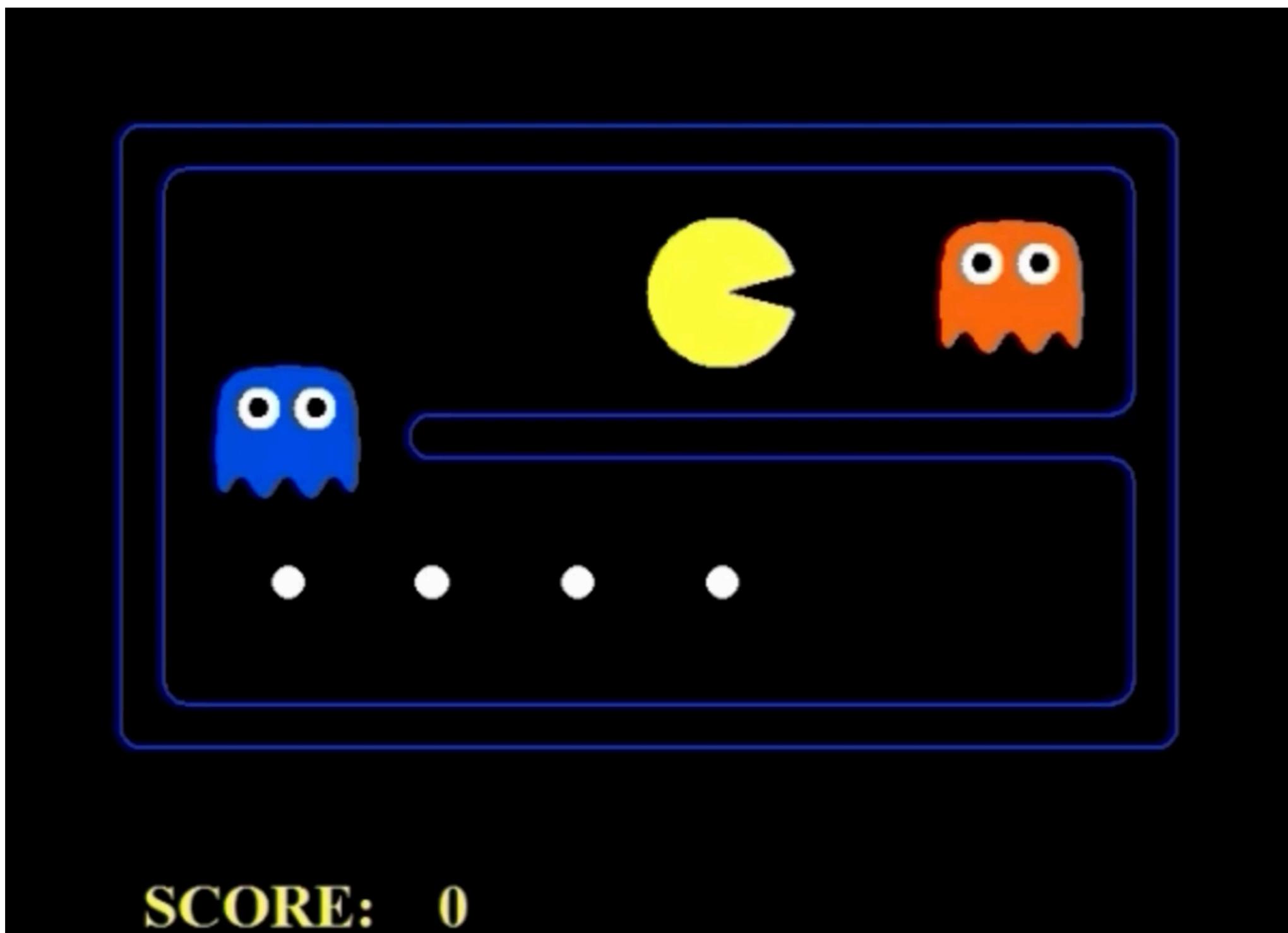


在对手也是最优的时候，表现得最
佳。但是.....？

假设对手是最优的

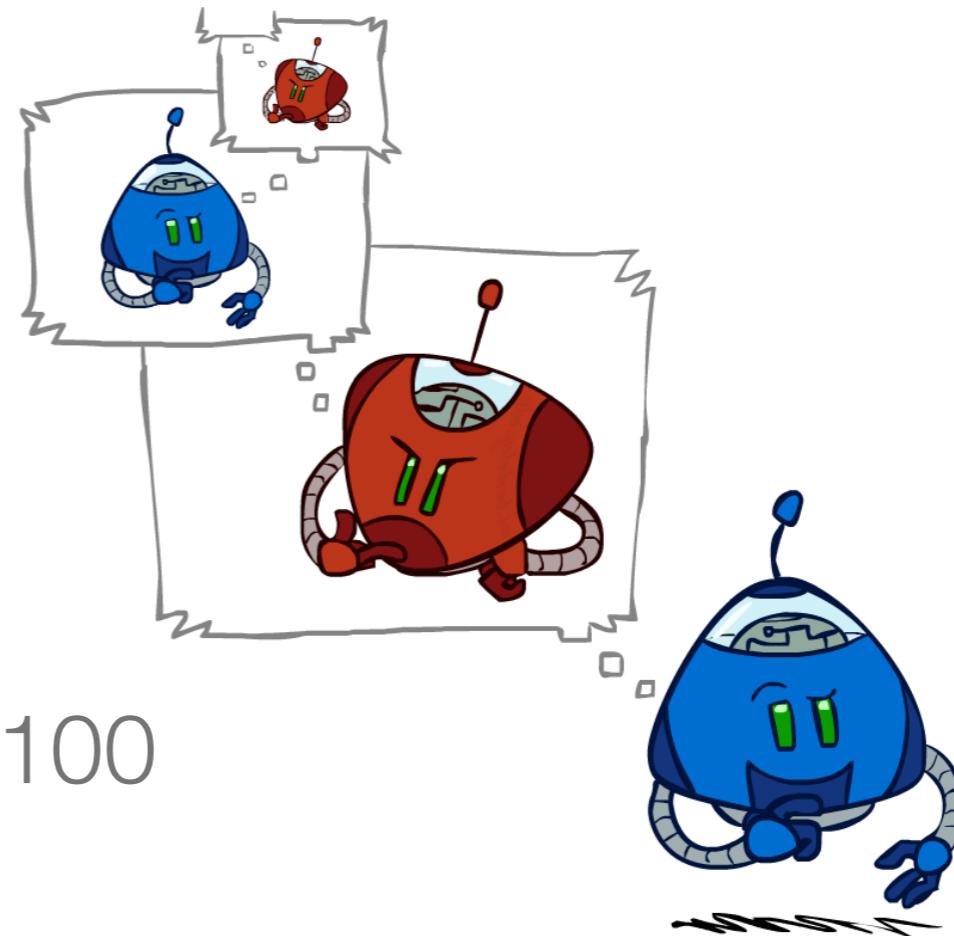


不假设对手是最优的



极大极小搜索的效率

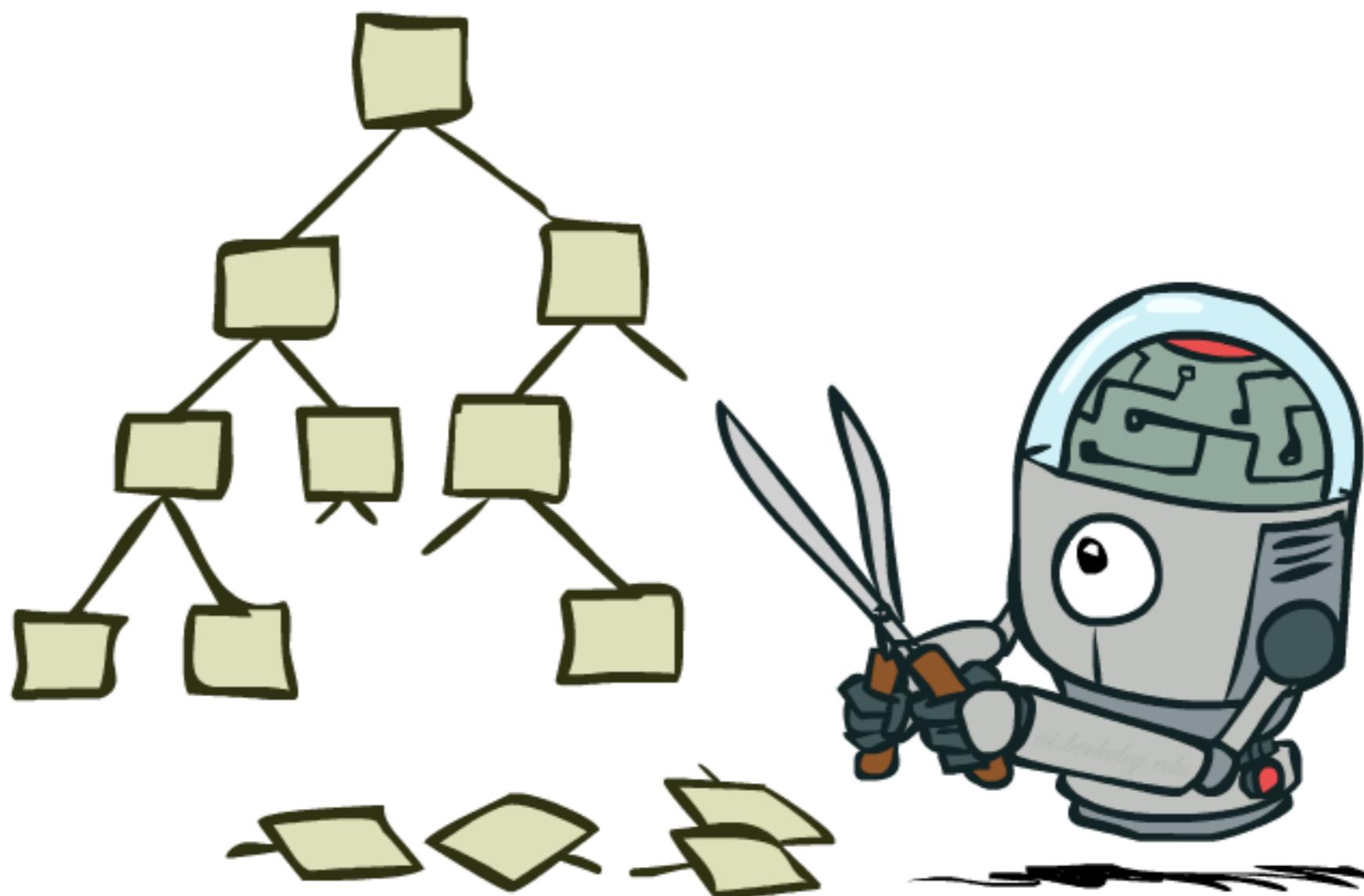
- Minimax的效率?
 - 深度优先穷尽搜索
 - 时间复杂度: $O(b^m)$
 - 空间复杂度: $O(bm)$
- 举例：国际象棋， $b \approx 35$, $m \approx 100$
 - 找到准确解是完全不可行的
 - 但是，有必要探索整棵树吗？人是如何下象棋的？



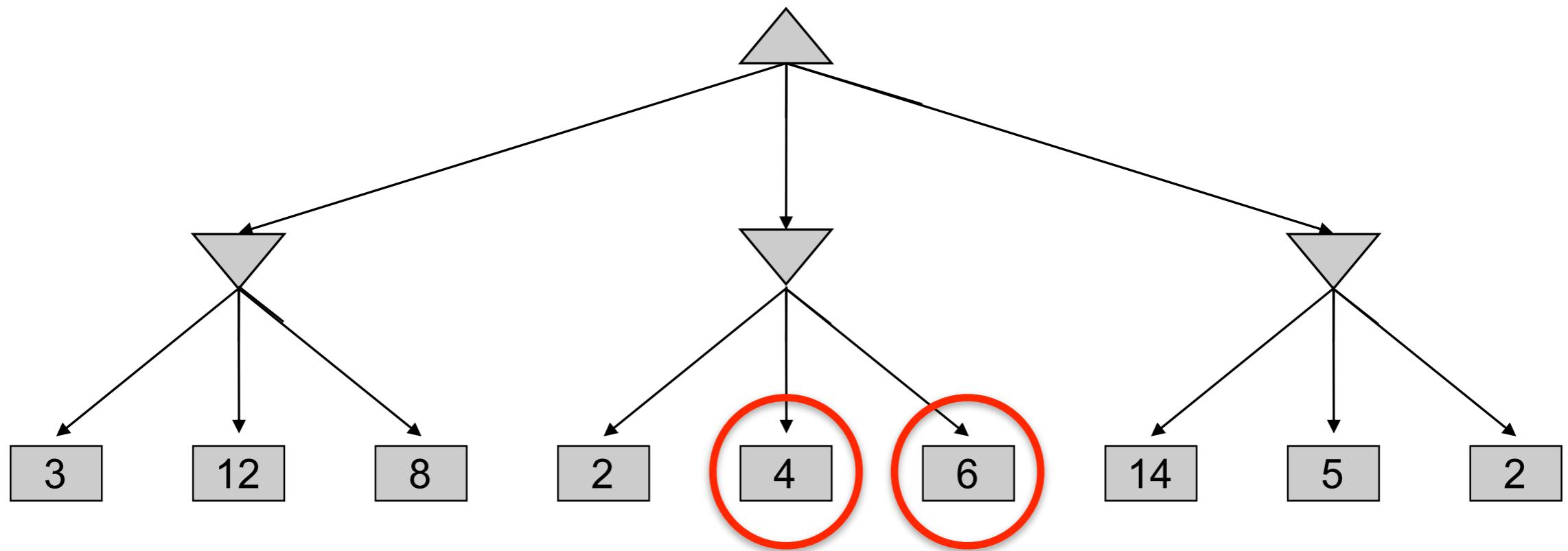
资源有限 vs 庞大的搜索空间



搜索树的修剪

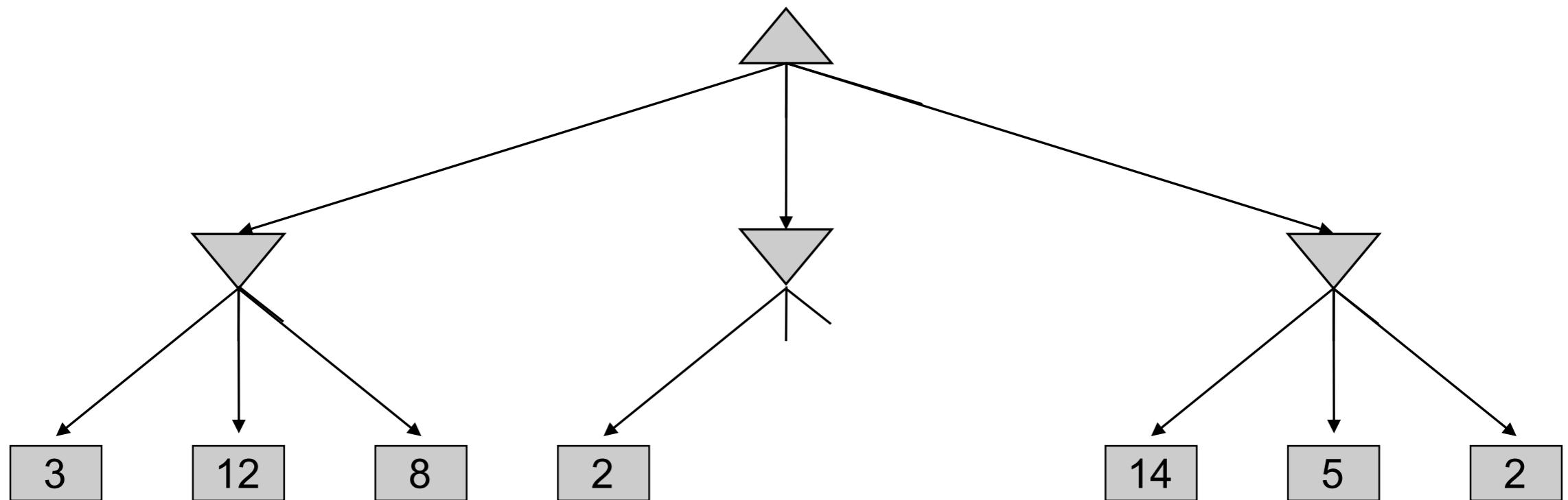


极大极小搜索树示例



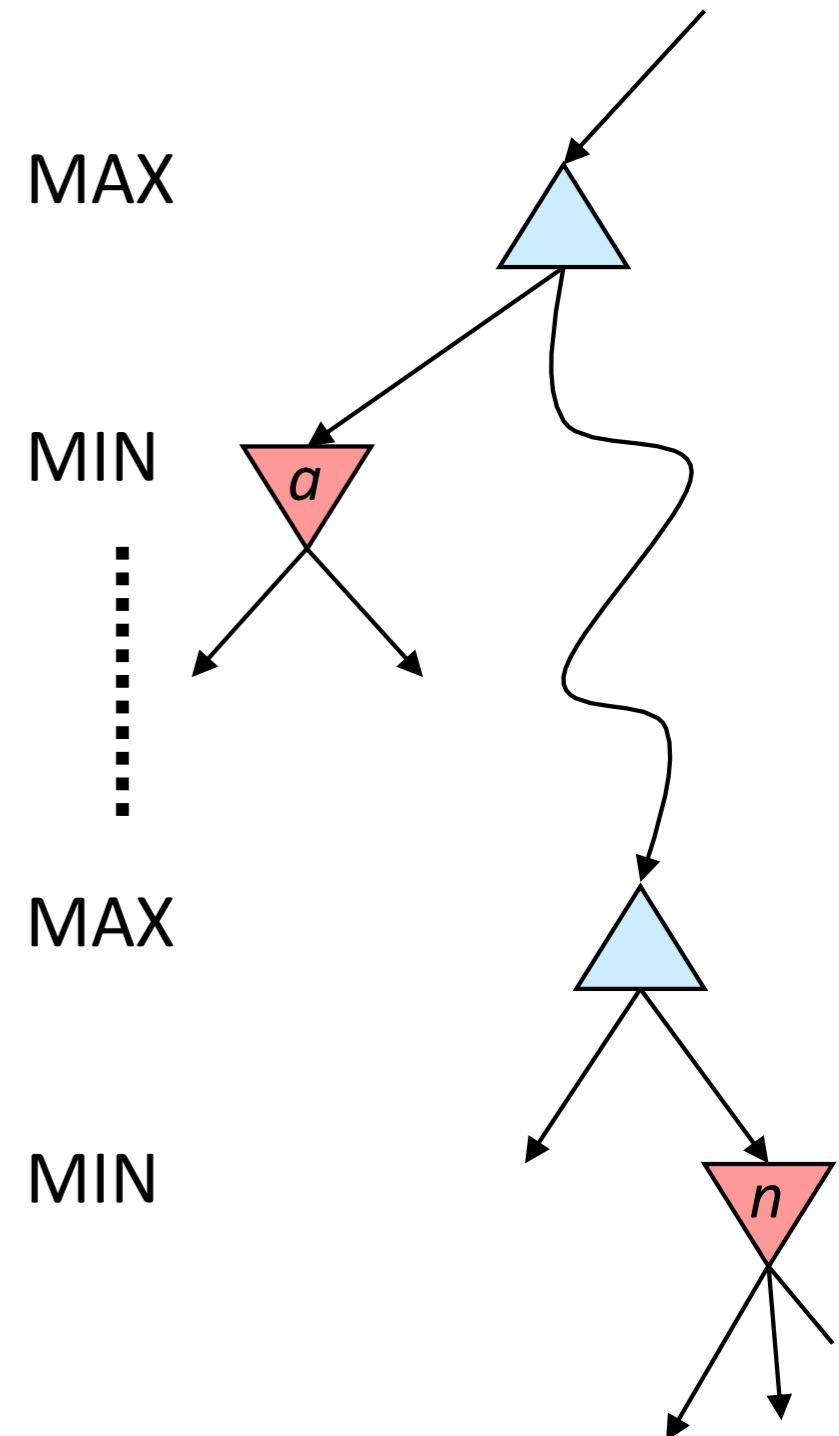
哪些节点取任意值都不改变根节点的值？

极大极小搜索树的修剪



Alpha-Beta 剪枝算法

- 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 α 是从根节点到当前MIN节点路径上的(任何一个)MAX节点所能取到的最大值
 - 如果n的当前值比 α 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节点的子节点剪枝操作是对称的
 - 令 β 是从根到当前MAX节点路径中的MIN节点MIN所能达到的最小值



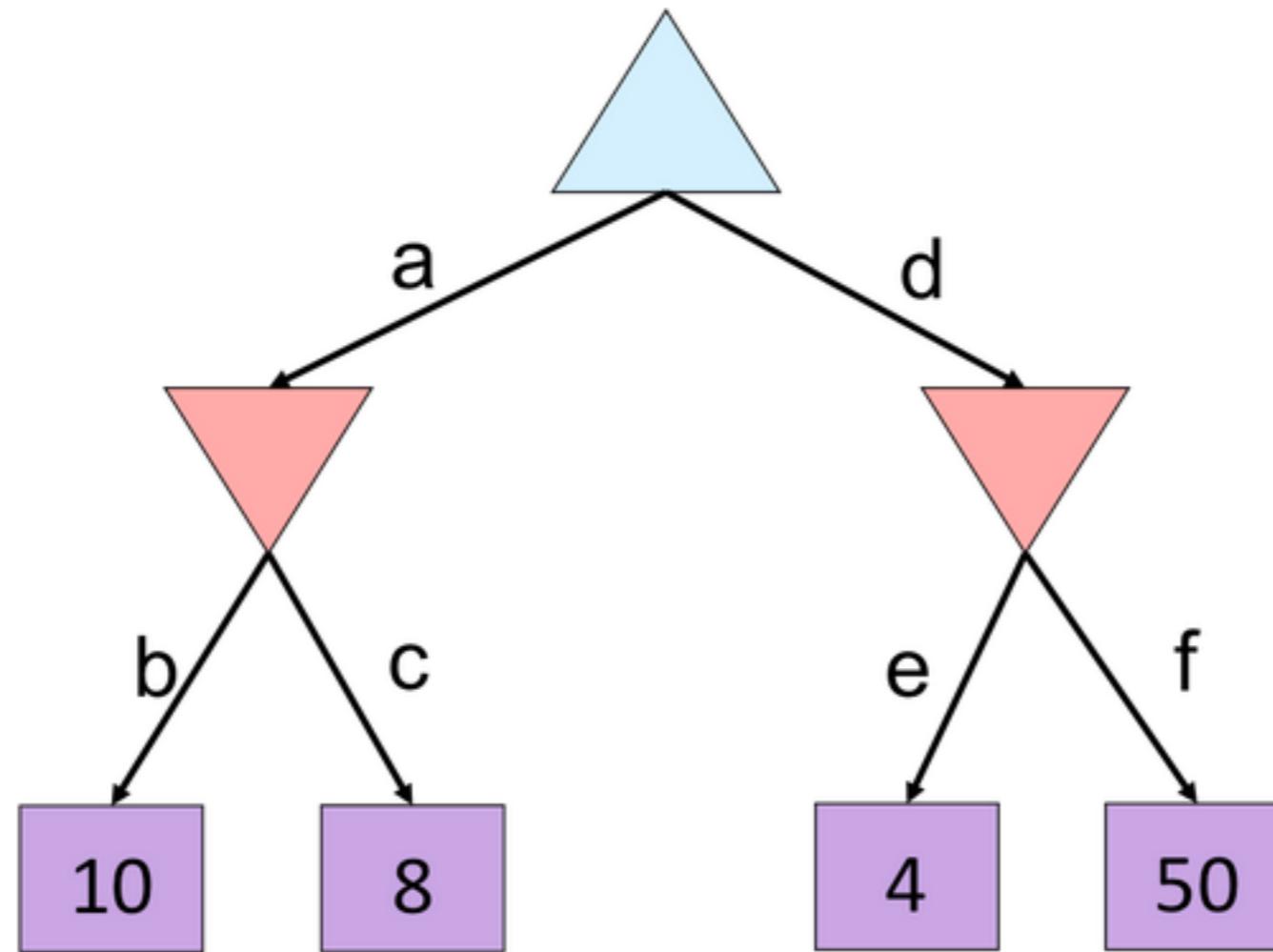
Alpha-Beta 剪枝算法的实现

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

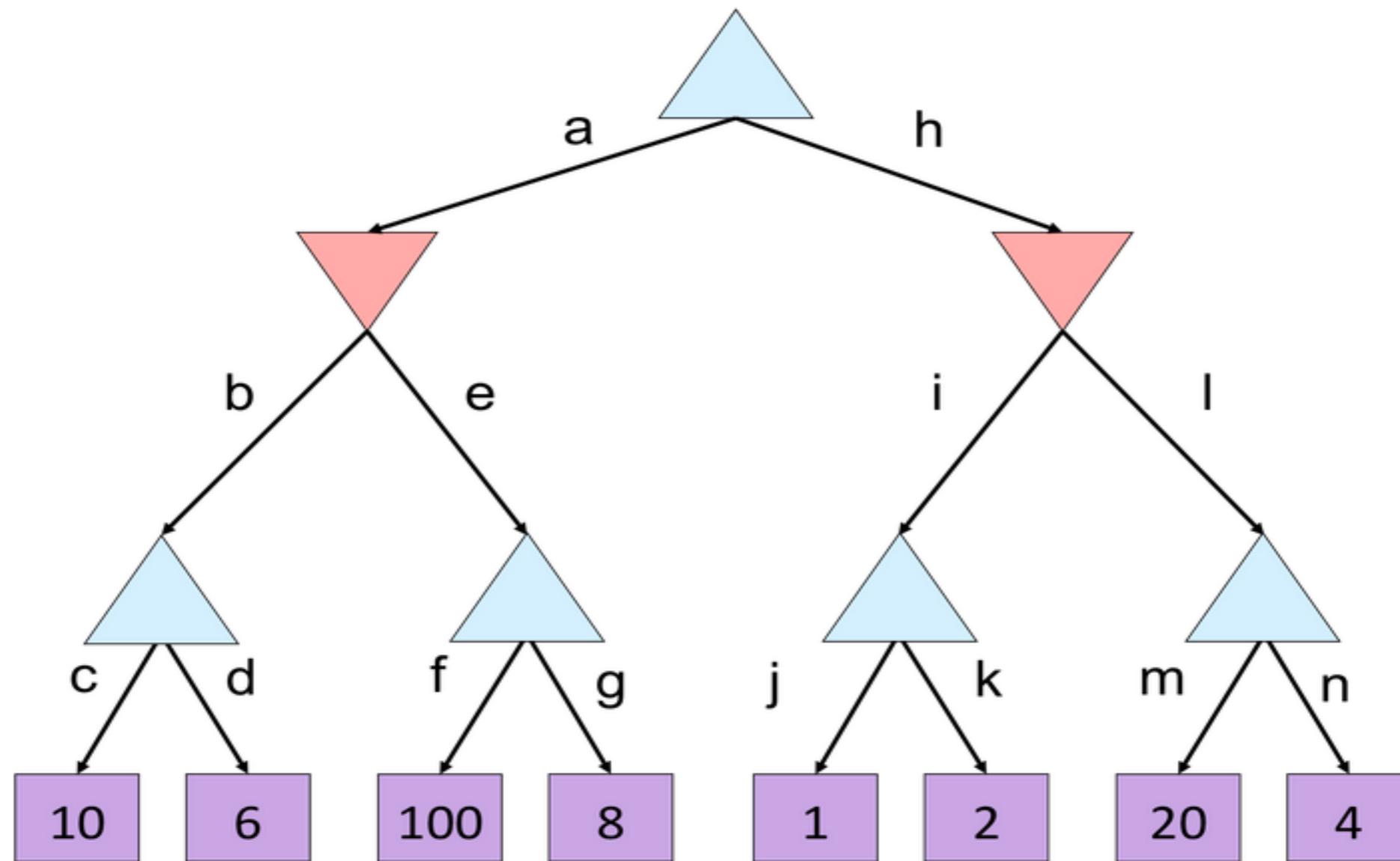
```
def max-value(state,  $\alpha$ ,  $\beta$ ):  
    initialize v = -∞  
    for each successor of state:  
        v = max(v, value(successor,  $\alpha$ ,  $\beta$ ))  
        if v  $\geq \beta$  return v  
         $\alpha$  = max( $\alpha$ , v)  
    return v
```

```
def min-value(state,  $\alpha$ ,  $\beta$ ):  
    initialize v = +∞  
    for each successor of state:  
        v = min(v, value(successor,  $\alpha$ ,  $\beta$ ))  
        if v  $\leq \alpha$  return v  
         $\beta$  = min( $\beta$ , v)  
    return v
```

Alpha-Beta 剪枝算法示例

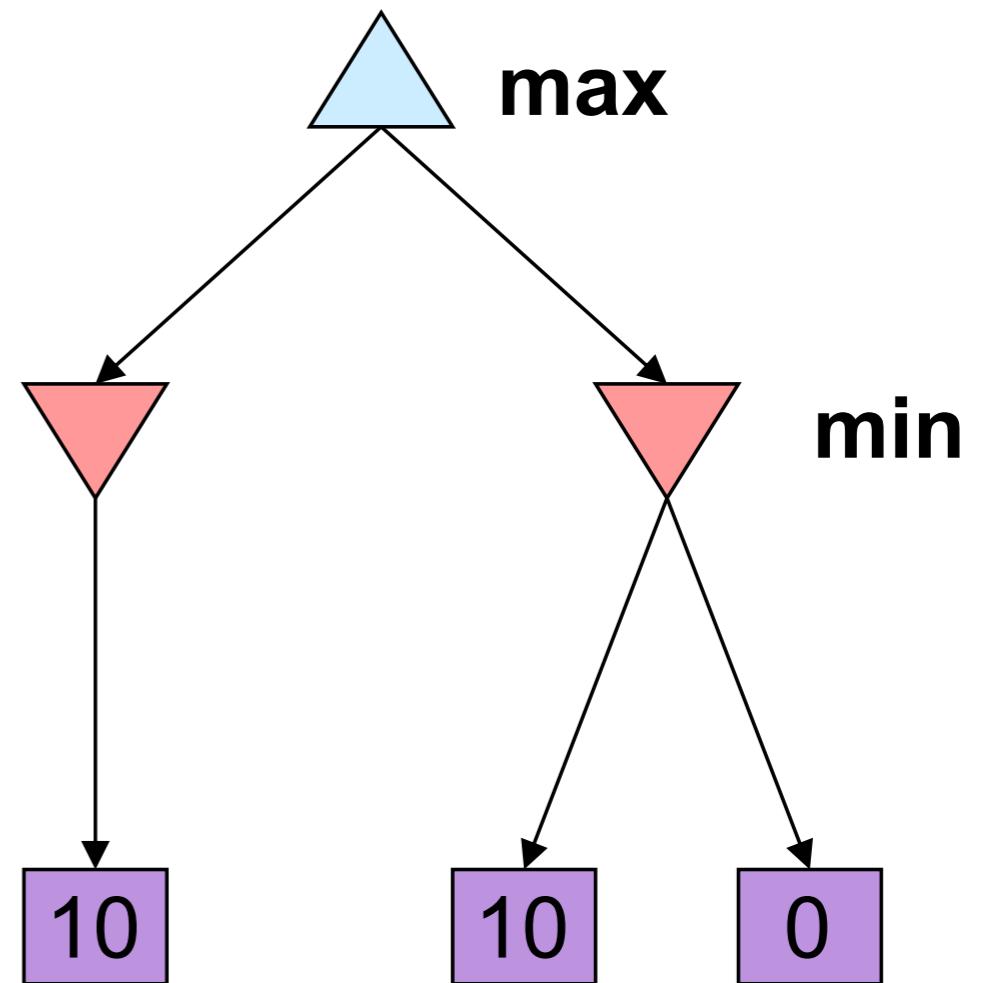


Alpha-Beta 剪枝算法示例2



Alpha-Beta 剪枝算法的性质

- 剪枝对根节点的极大极小值的计算没有影响!
 - 中间节点的值可能错误
- 良好的子节点顺序可以提高修剪的有效性
- 当节点的顺序完美时:
 - 时间复杂度降至 $O(b^{m/2})$
 - 搜索深度提高了一倍
 - 完全搜索仍然是不可能的...



资源受限

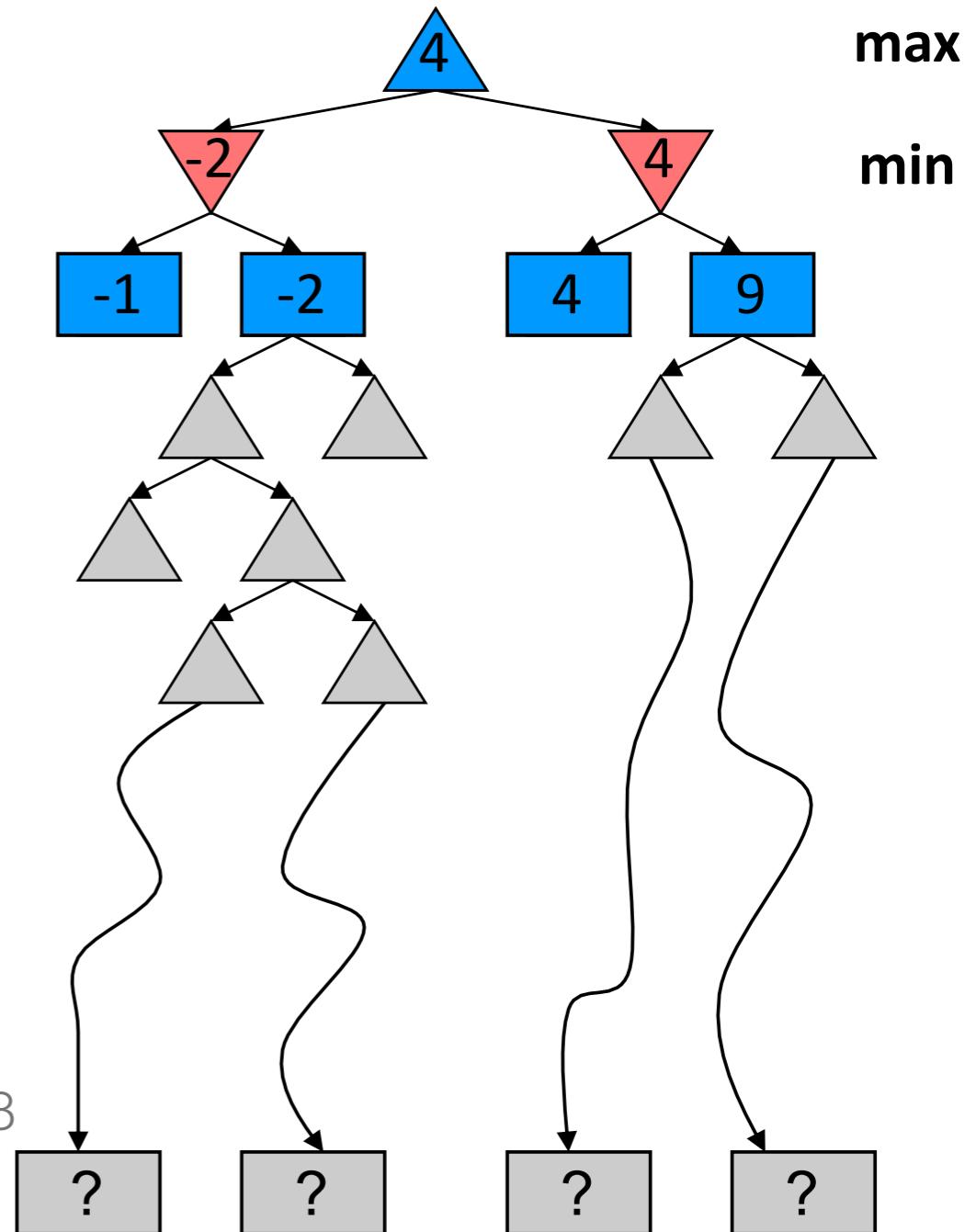
- 问题:现实中,几乎不能搜索到叶节点!

- 一种解决办法:有界搜索加预测

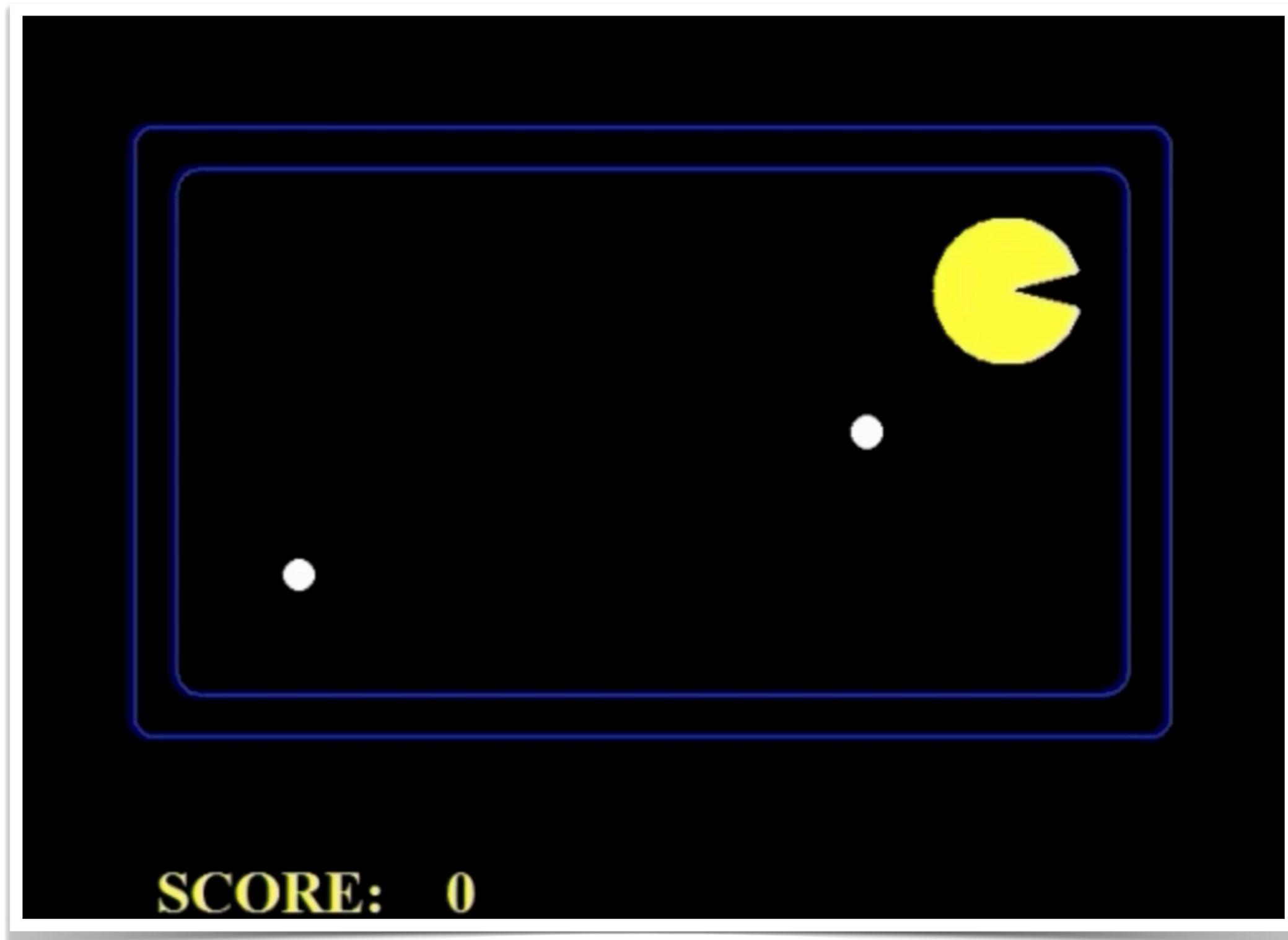
- 搜索只到预定深度层次
 - 使用评估函数预测搜索边界节点的值
 - 失去了最优解的保证
 - 搜索的层次越多结果就越不相同

- 例如:

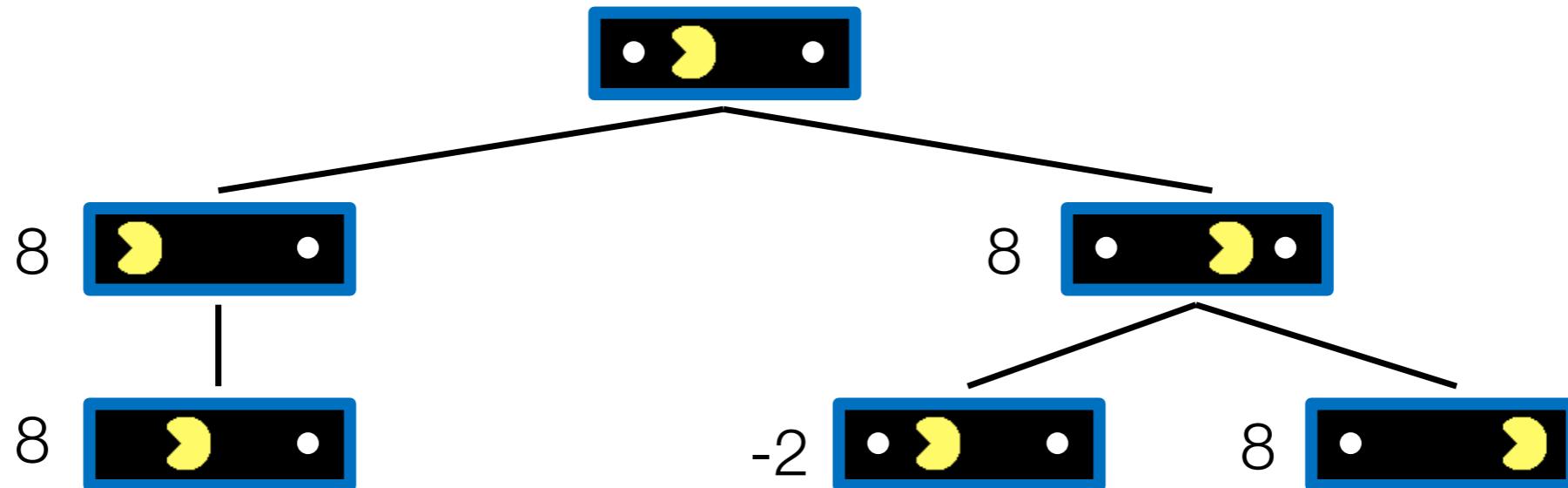
- 假设计算时间为100秒,每秒能探索1万个节点
 - 所以每步能检查1百万个节点
 - 在国际象棋中, $b \approx 35$, 搜索大致能达到搜索树的第8层-还是不够好



有限搜索深度(深度=2)

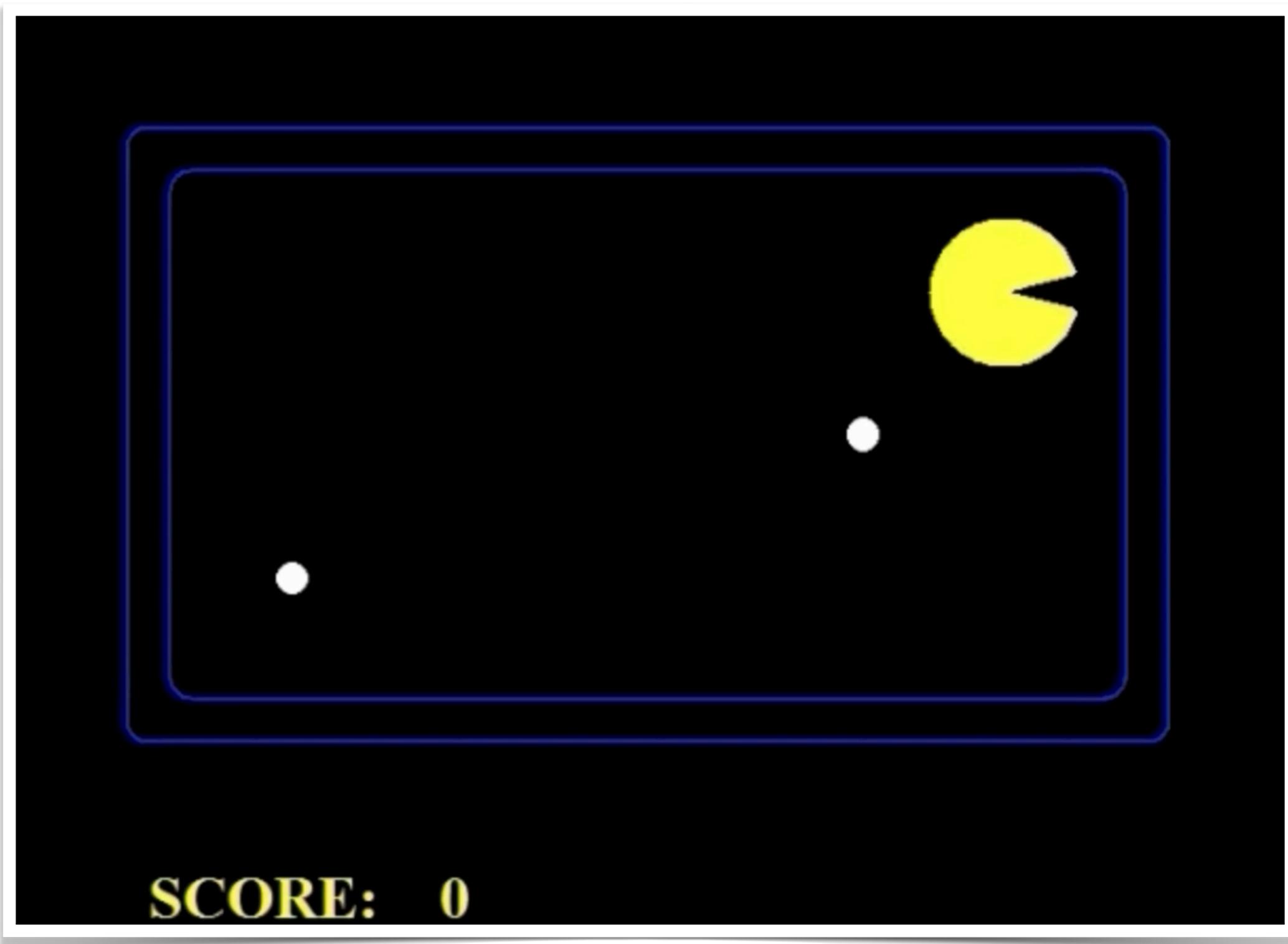


Pacman 为什么会犹豫?

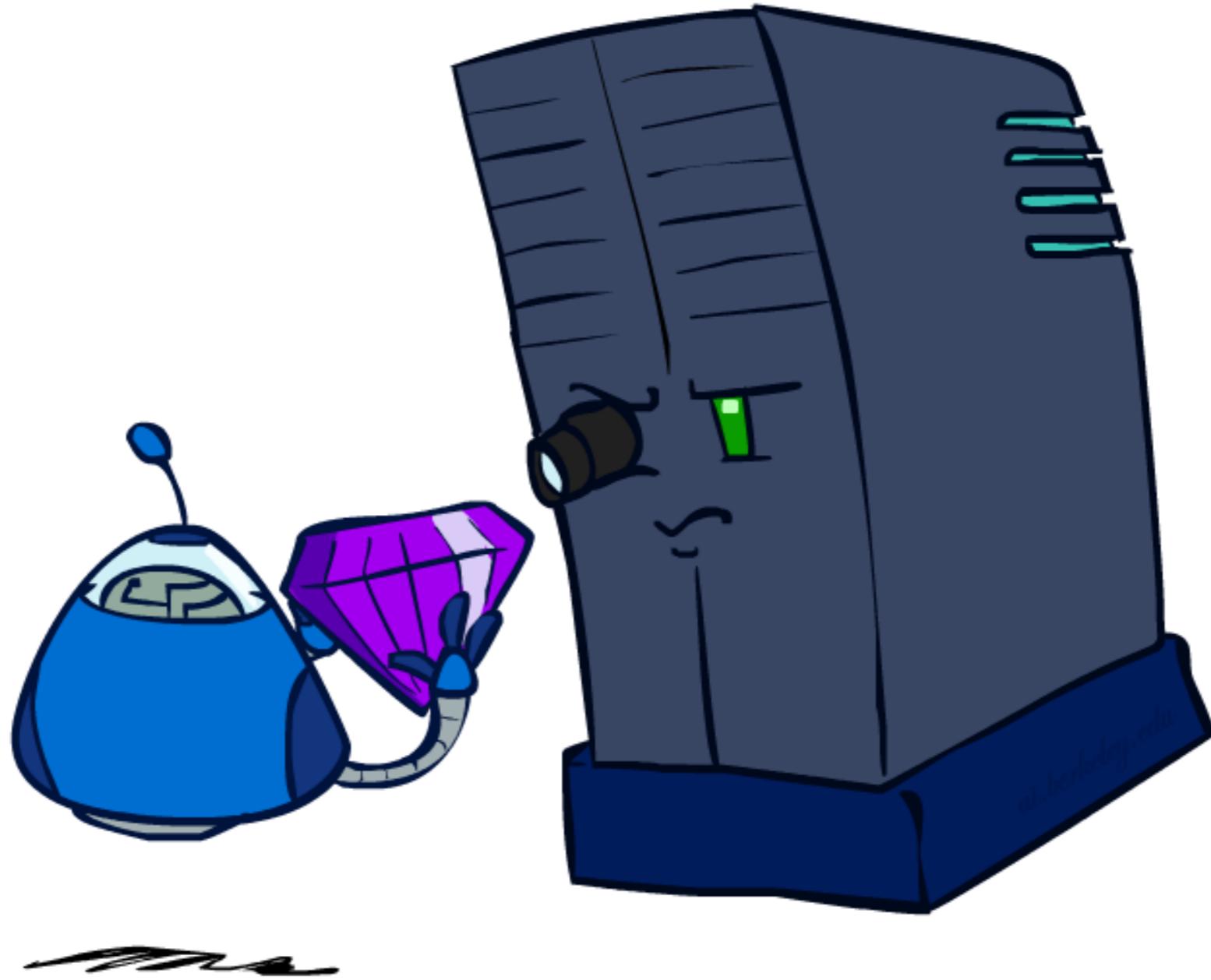


- 智能体重新规划可能面临的处境
 - 在当前两步搜索的情况下，向左吃掉豆子的得分和向右移动的是一样的(后面可能会吃掉豆子)
 - 所以这个时候等待和吃掉豆子的选择结果似乎一样;它有可能向右移动，然后下一轮重新规划时，可能又走回来了!

改进评价函数的有限搜索深度(深度=2)

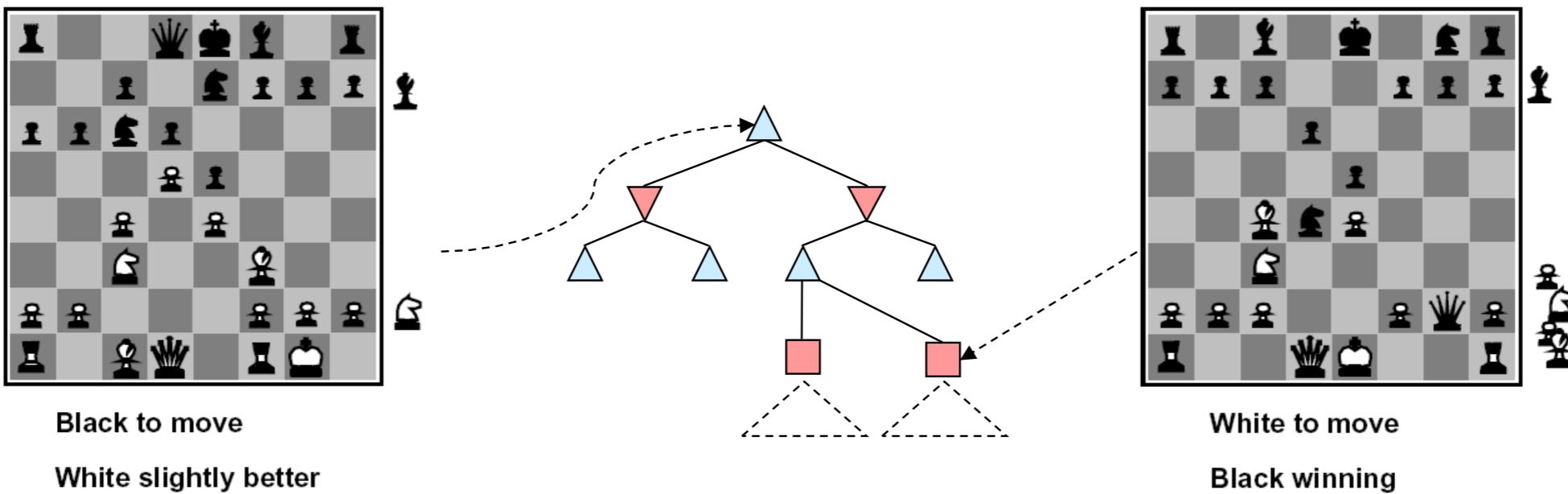


评价函数



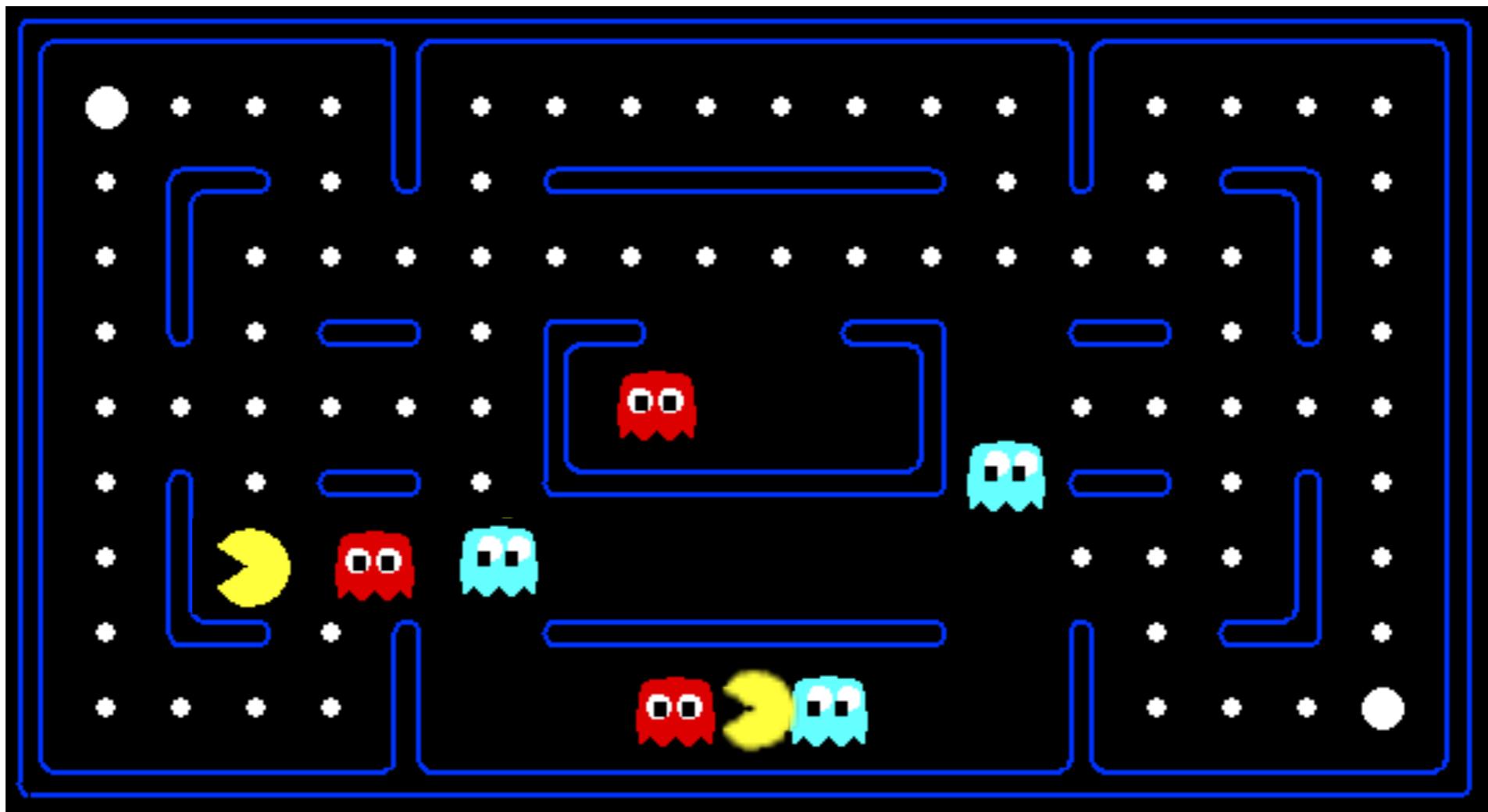
评价函数

- 在一个限定深度搜索中, 用来给非终局节点状态估值。



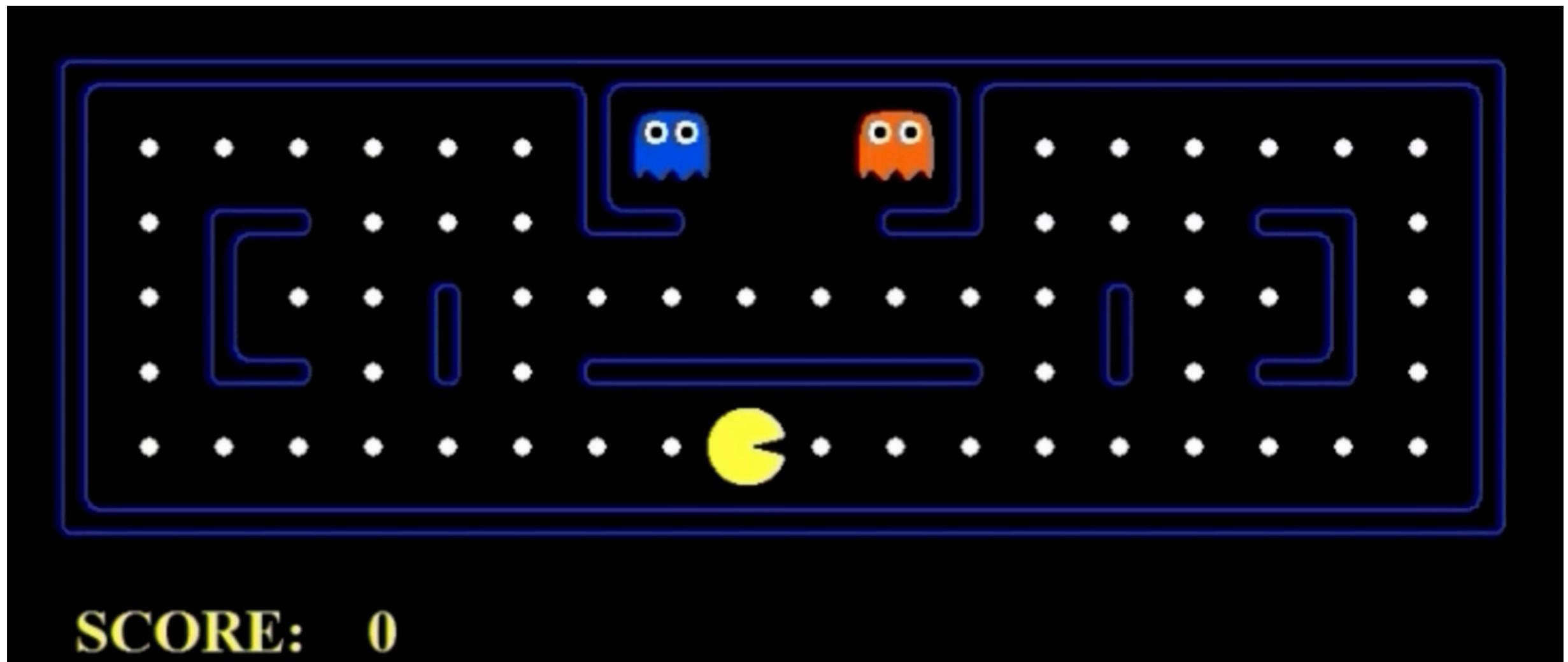
- 理想函数: 返回这个节点状态的实际最小最大值
- 实践中: 特征函数值的加权线性和:
 - $\text{EVAL}(s) = w_1 f_1(s) + w_2 f_2(s) + \dots + w_n f_n(s)$
 - 例如 $w_1 = 9$, $f_1(s) = (\text{白皇后数量} - \text{黑皇后数量})$, 等。

Pacman游戏状态评估



- 评估函数值应反映所处的局面。

聪明的妖怪



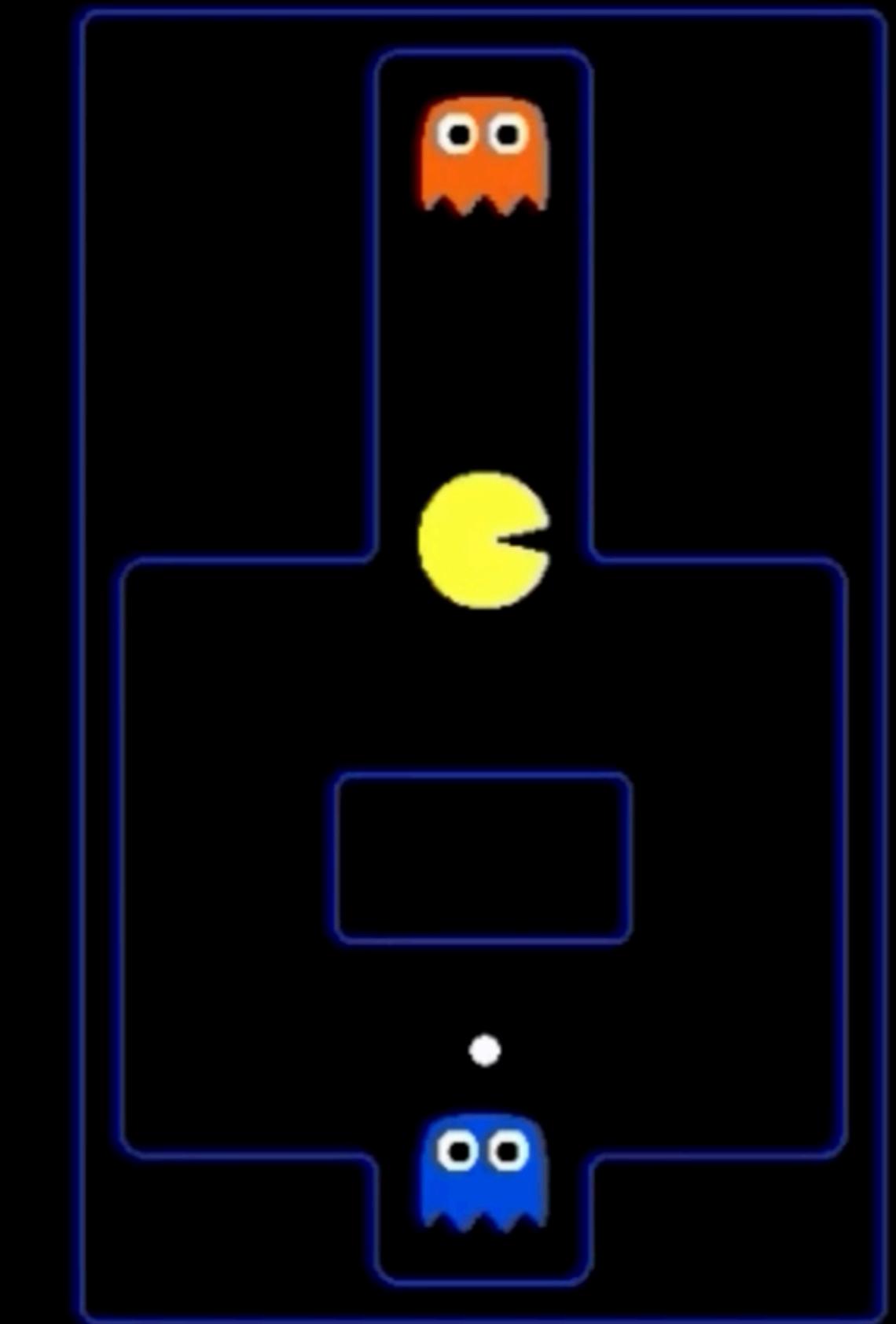
- 共同的评估函数可自动形成合作

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=>更好的表现
- 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

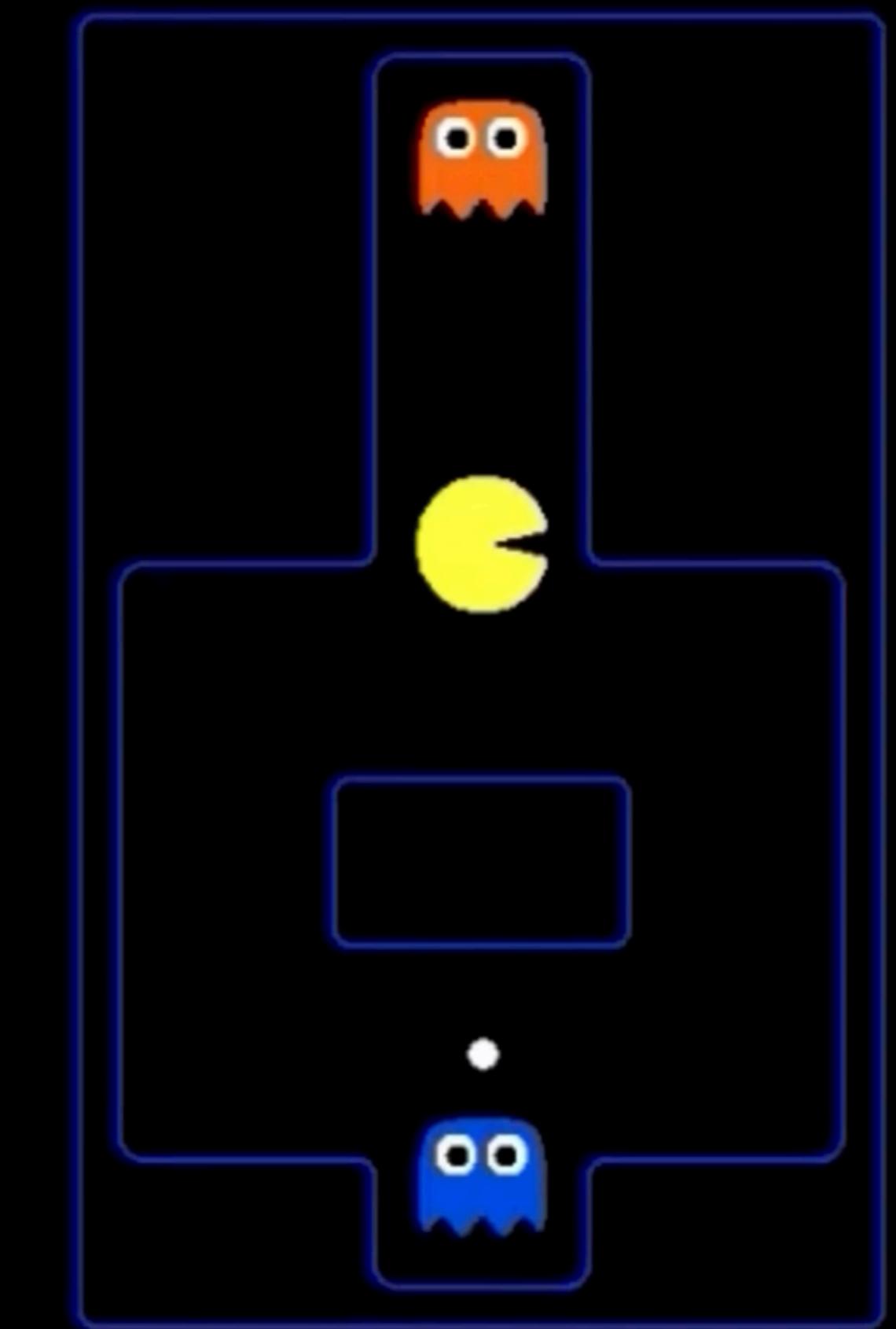


有限搜索深度(深度=2)



SCORE: 0

有限搜索深度(深度=10)

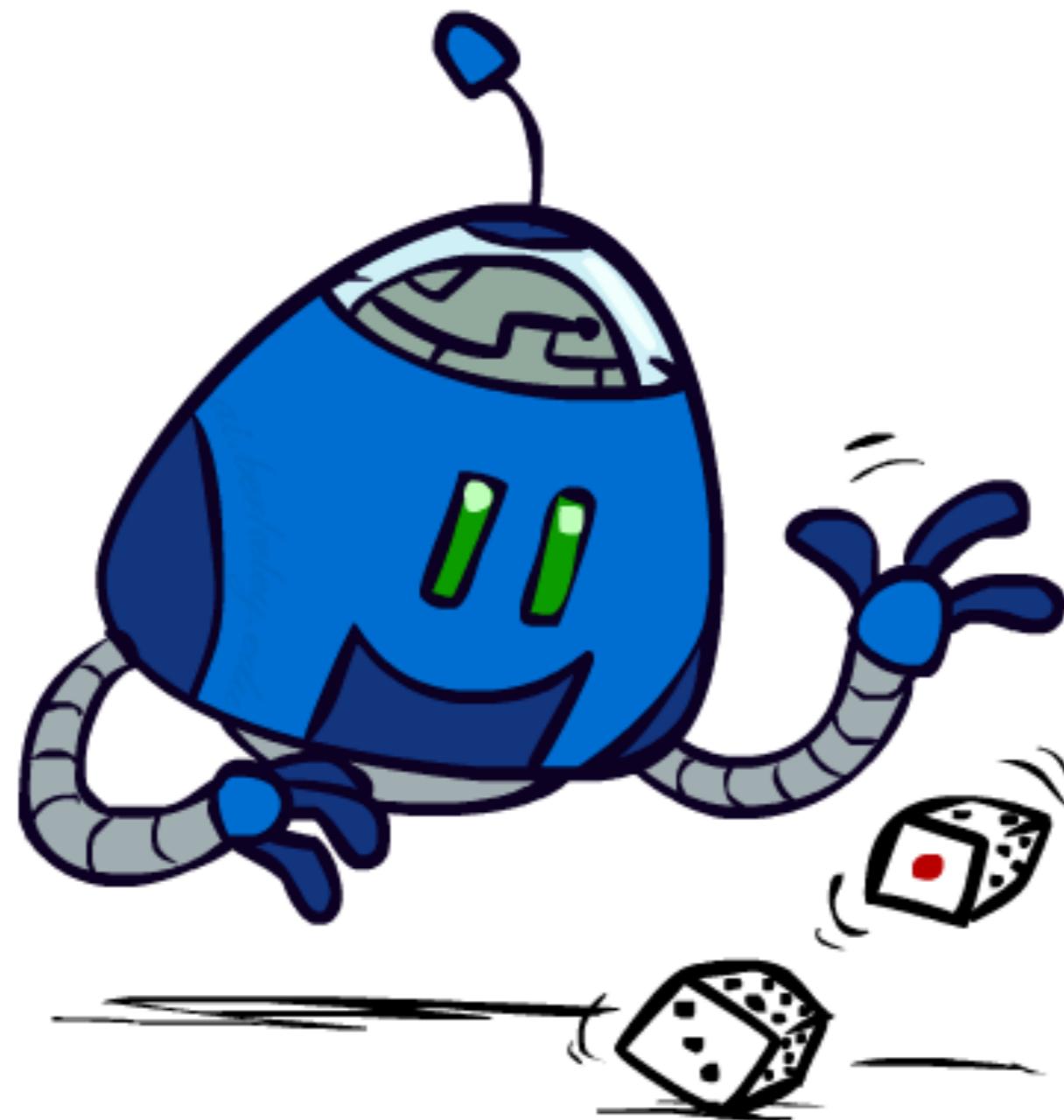


SCORE: 0

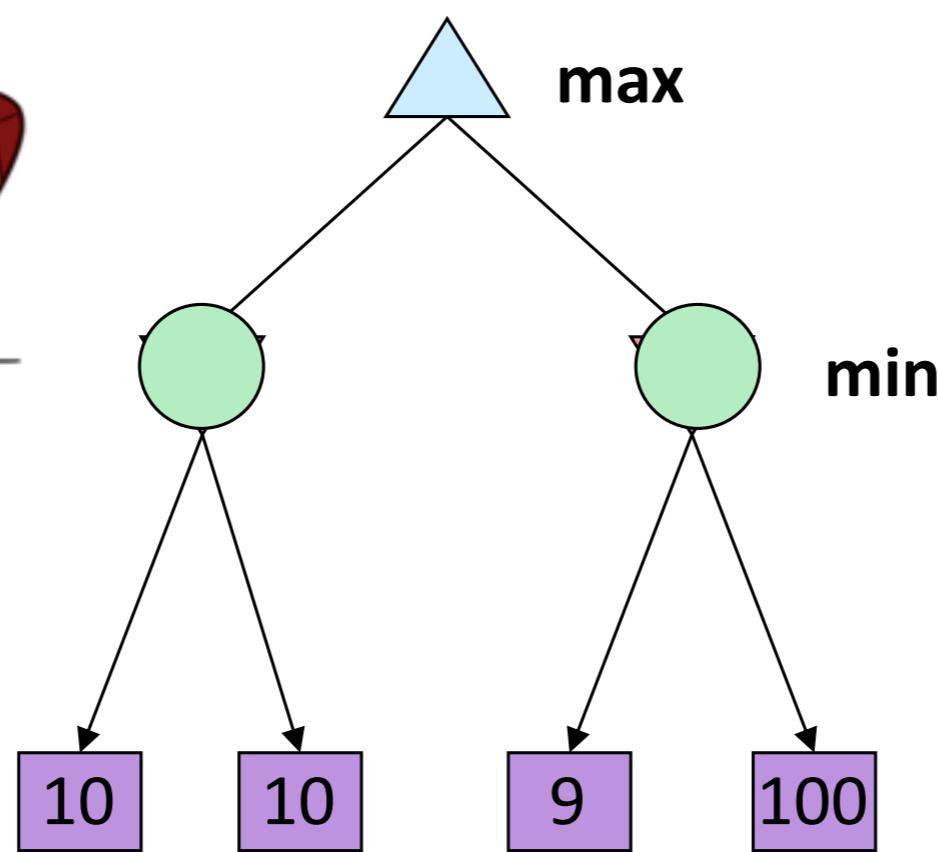
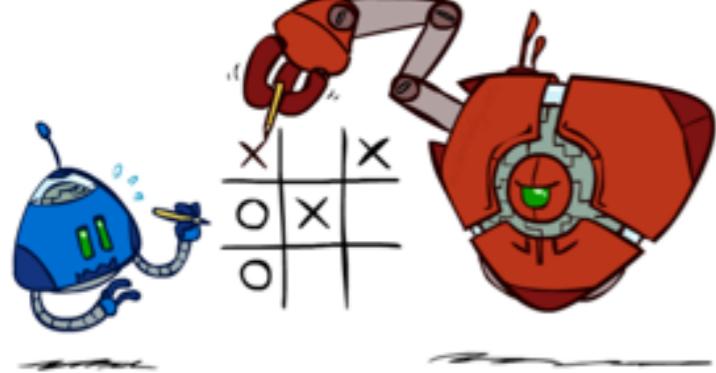
评估函数与Alpha-Beta剪枝的协同作用？

- Alpha-Beta剪枝：修剪的数量取决于扩展顺序
 - 评估函数可以提供指导，以首先扩展最有希望的节点(这使得更有可能在通往根的路径上已经有了一个好的替代方案)
 - (类似于A*的启发式函数、CSP的过滤)
- Alpha-Beta剪枝：
 - 最小节点的值只会继续下降
 - 一旦最小节点的值低于最佳选择的最大值，沿着路径到根，可以进行修剪
 - 因此:如果求值函数在最小节点上提供了值的上界，并且上界已经低于到根路径上的最佳选择max，则可以进行修剪

不确定的结果



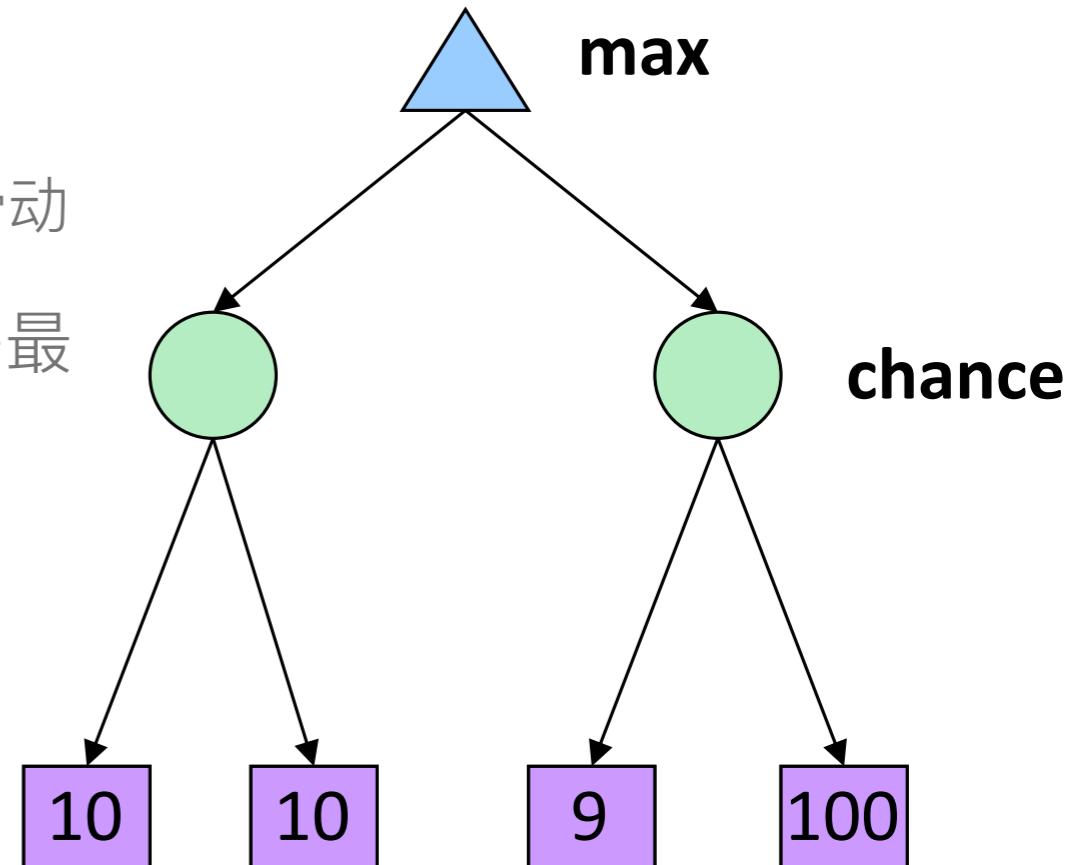
最坏情况vs.平均情况



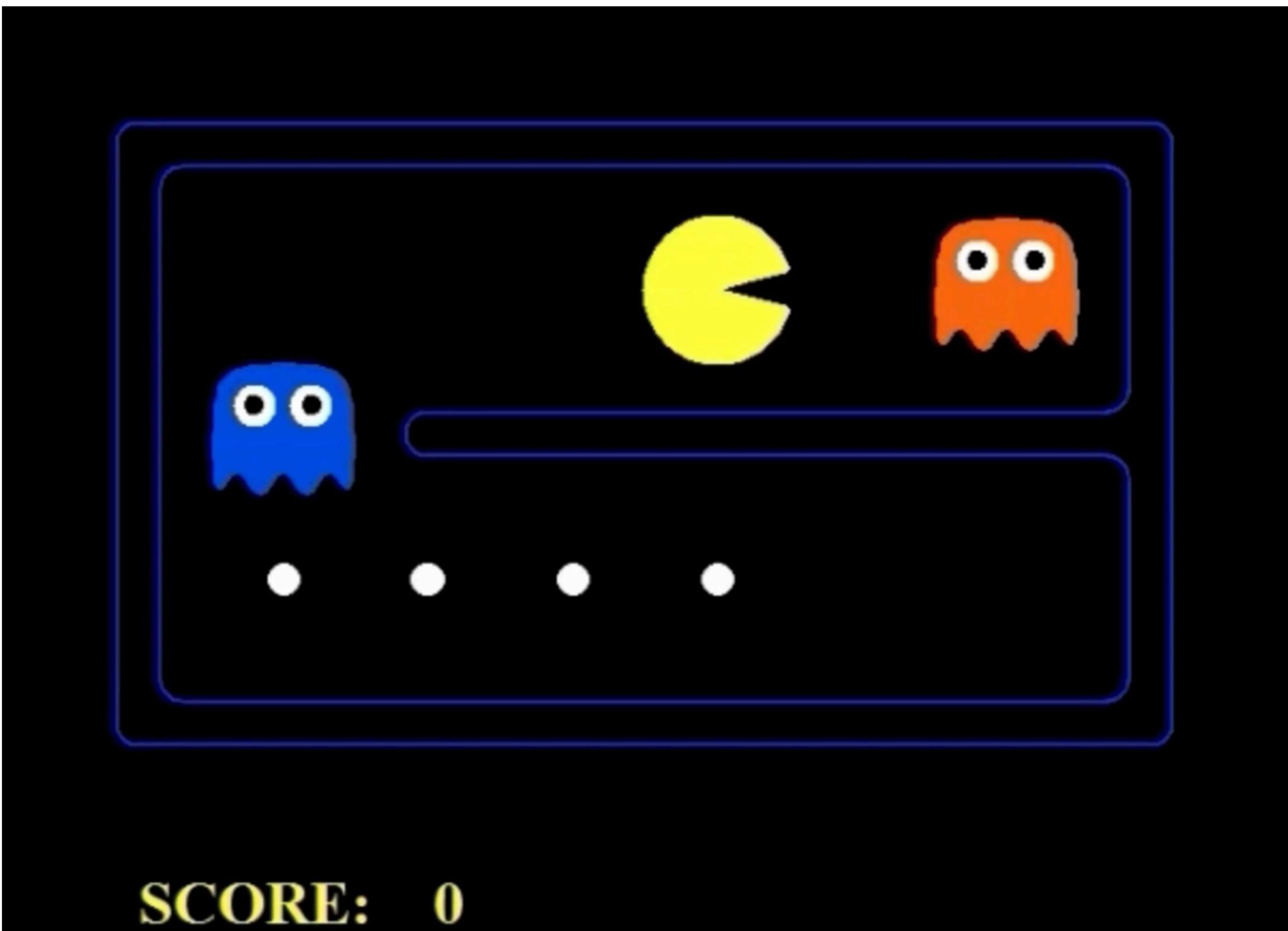
不确定的结果是基于概率的，而不是对手！

最大期望搜索 (Expectimax Search)

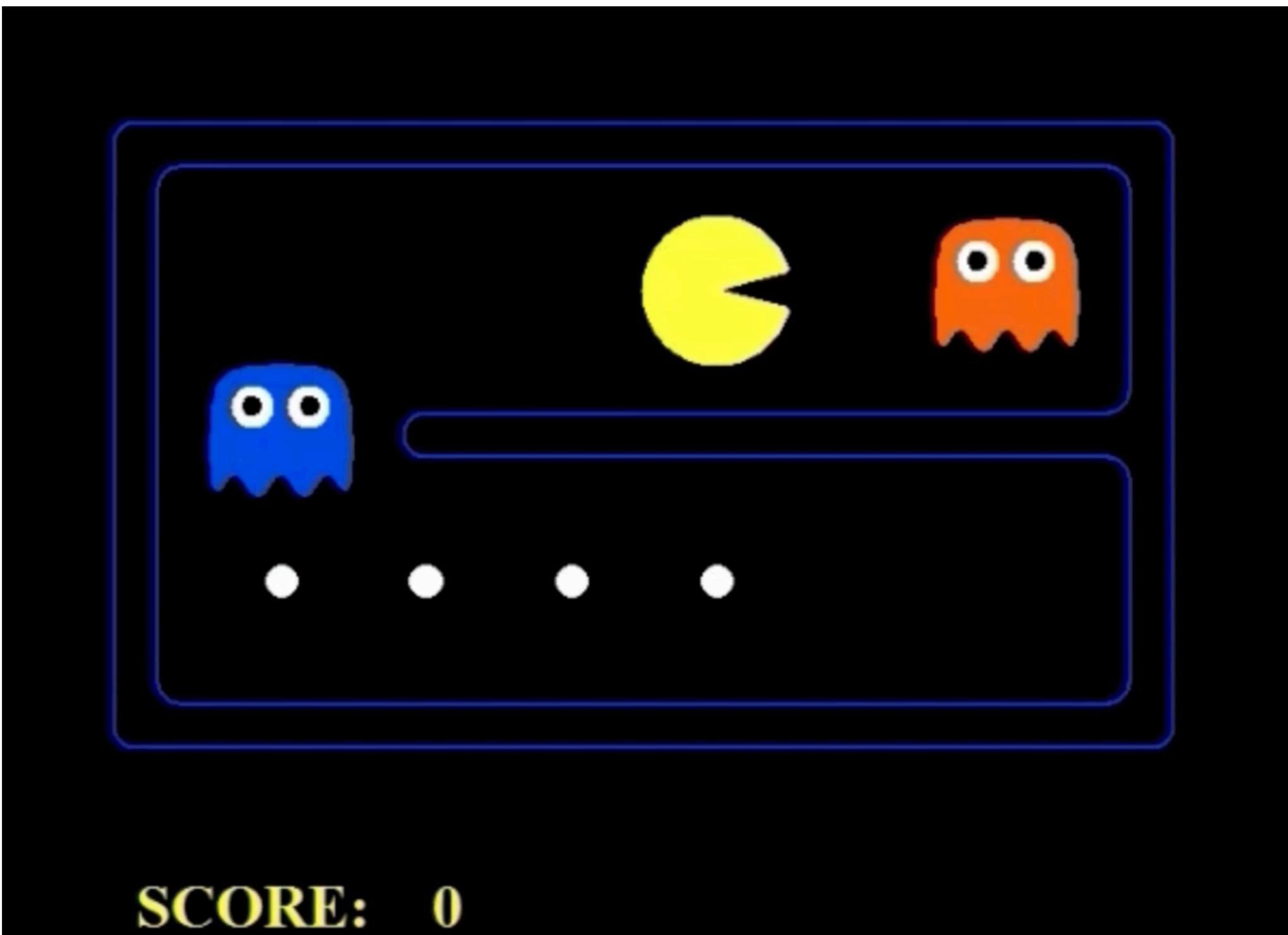
- 什么情况下会有可能不知道行动的结果呢?
 - 随机性的游戏设定, 比如掷骰子
 - 不可预测的对手: 比如乱下棋新手
 - 行动可能失败: 当移动机器人时, 轮子可能会滑动
- 这种情况下应该考虑平均情况的结果, 而不是最坏情况的结果
- **最大期望搜索**: 计算最佳玩法下的平均得分
 - 与极大极小搜索相同的最大节点
 - 机会节点与最小节点相似, 但结果是不确定的
 - 计算期望效用(expected utilities), 即对子节点的期望取加权平均数



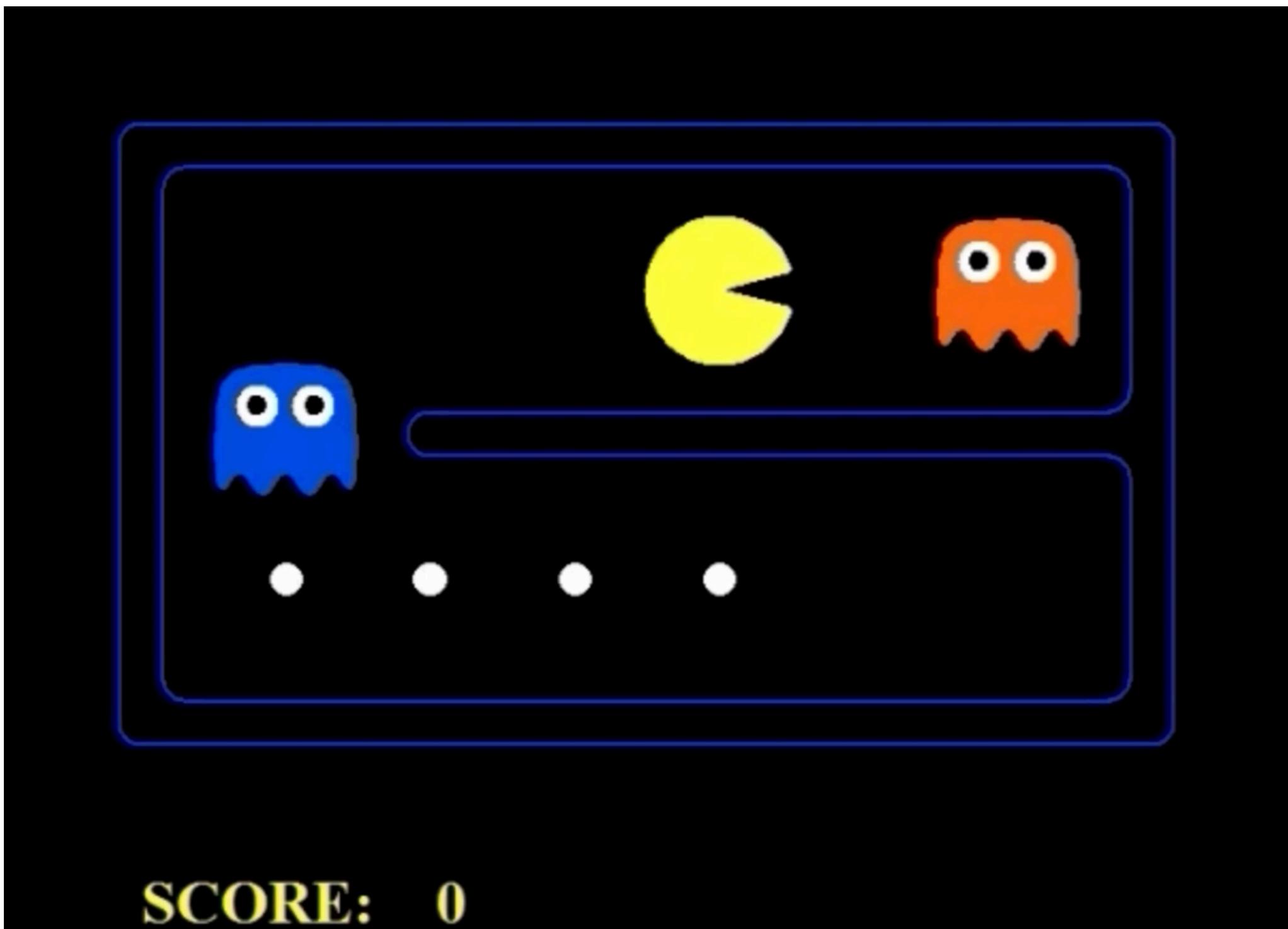
极大极小搜索vs最大期望搜索 (Minimax)



极大极小搜索vs最大期望搜索 (Expectimax)



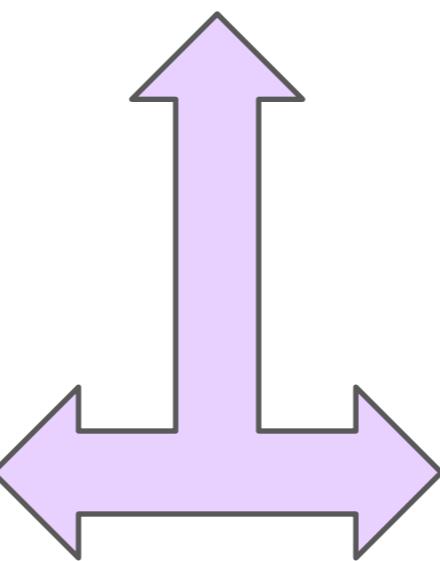
极大极小搜索vs最大期望搜索 (Expectimax)



最大期望搜索

```
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)
```

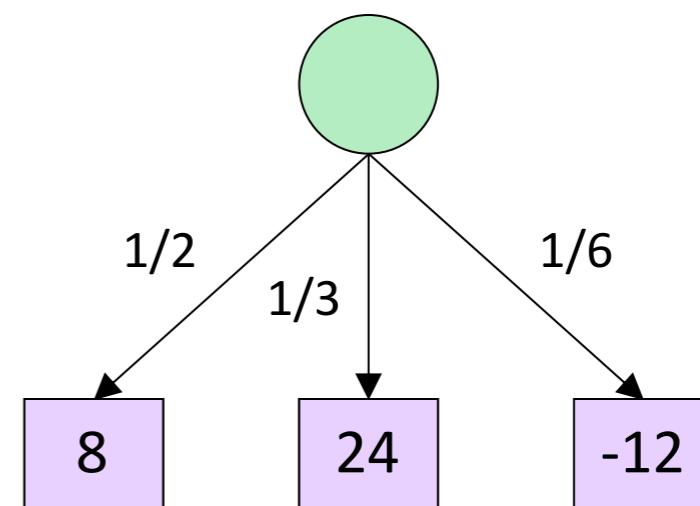
```
def max-value(state):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor))
    return v
```



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

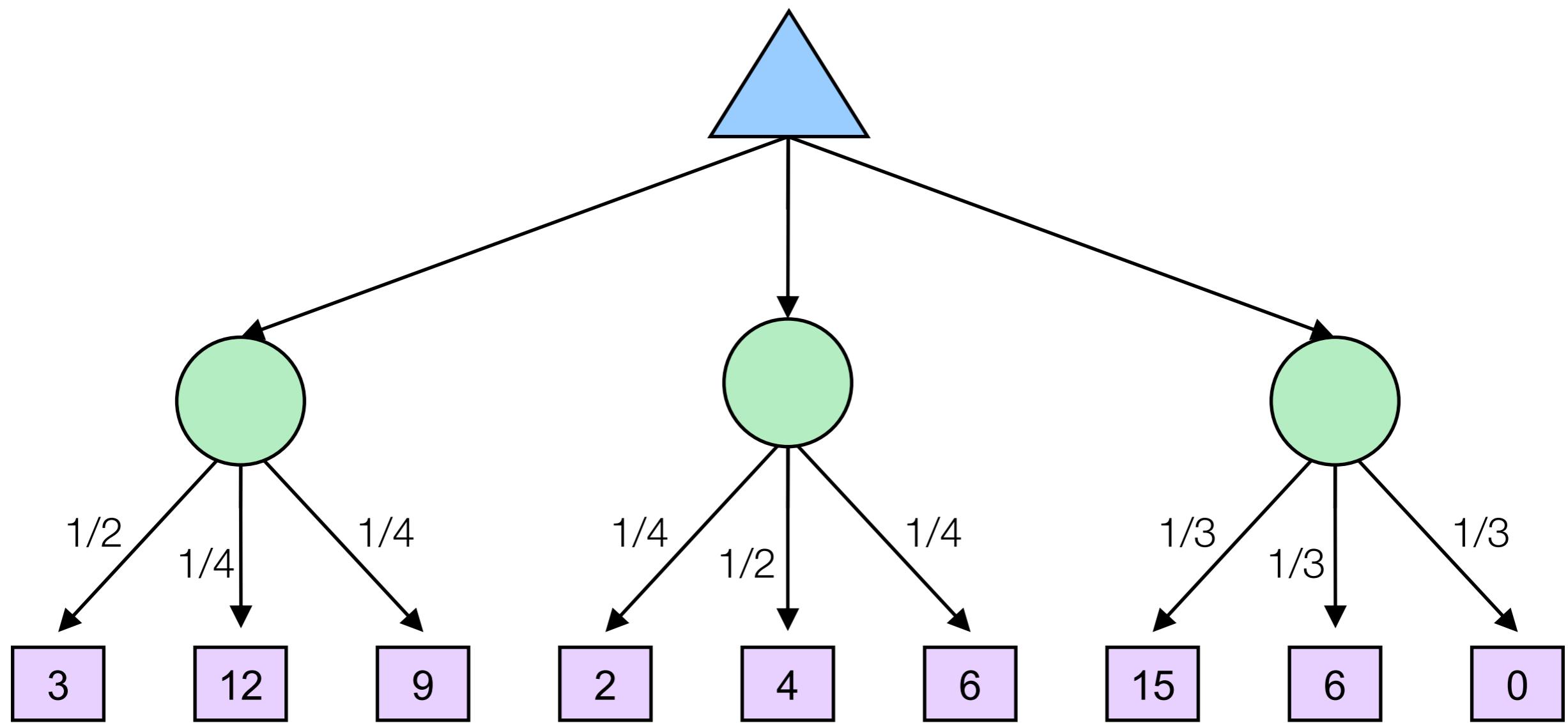
最大期望搜索

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

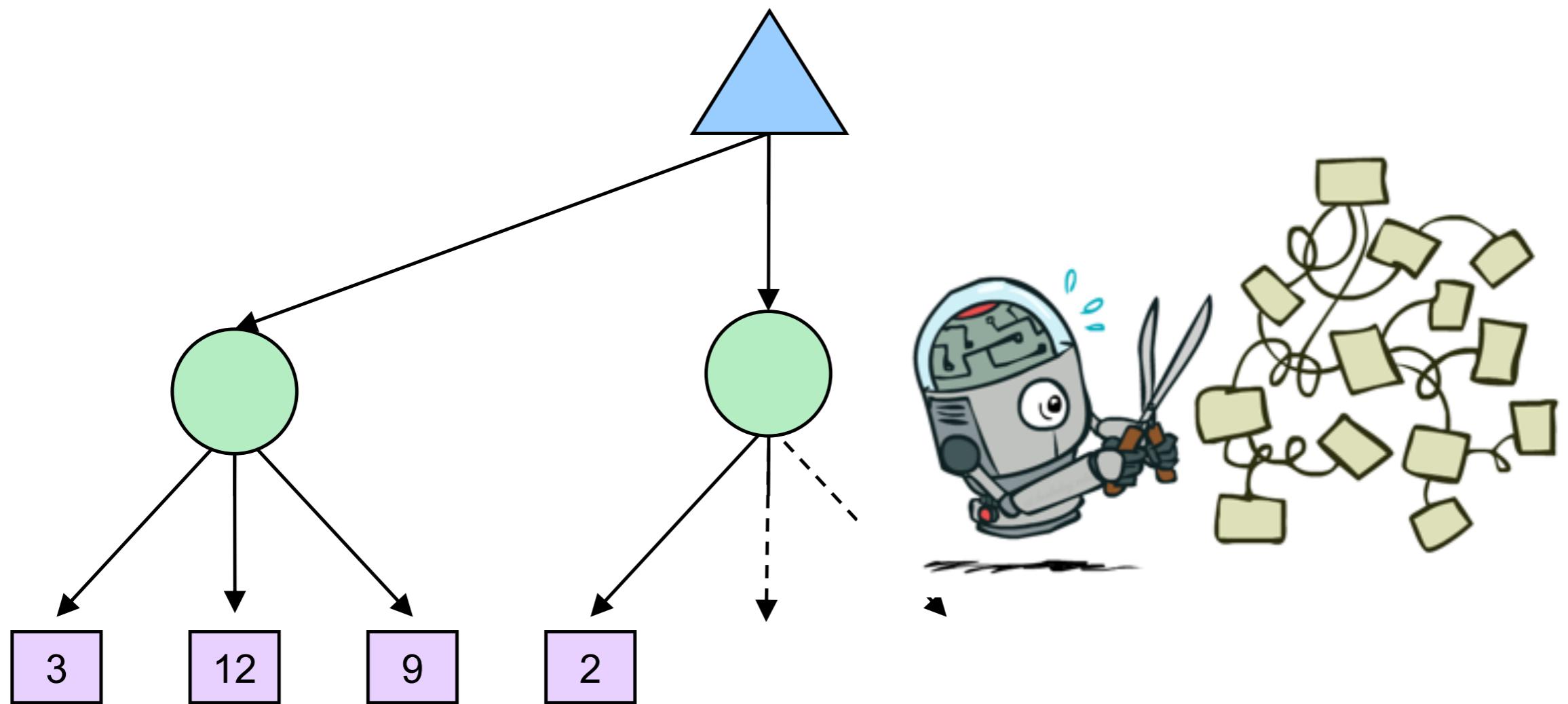


$$v = (1/2)(8) + (1/3)(24) + (1/6)(-12) = 10$$

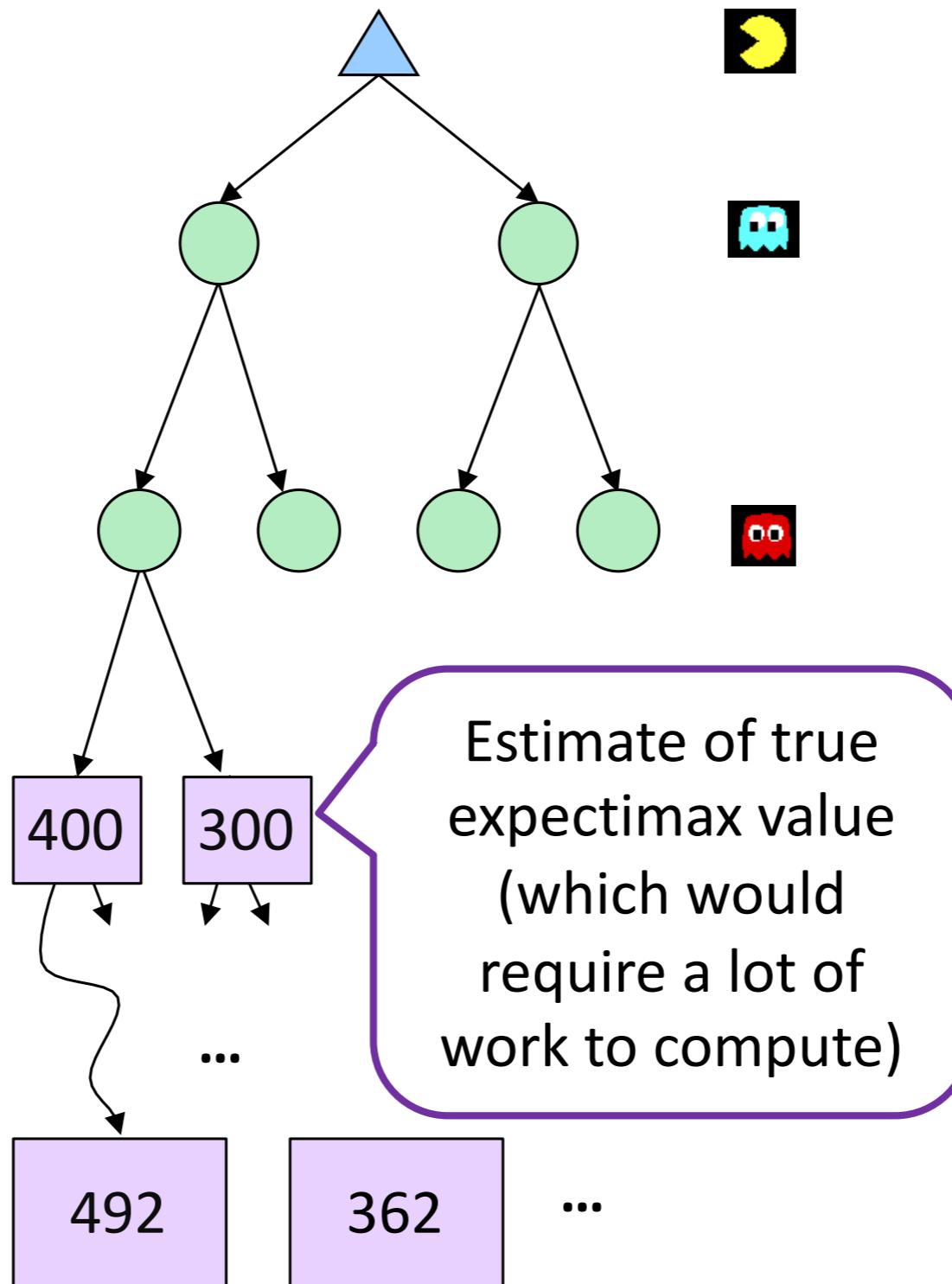
最大期望搜索示例



最大期望搜索剪枝?

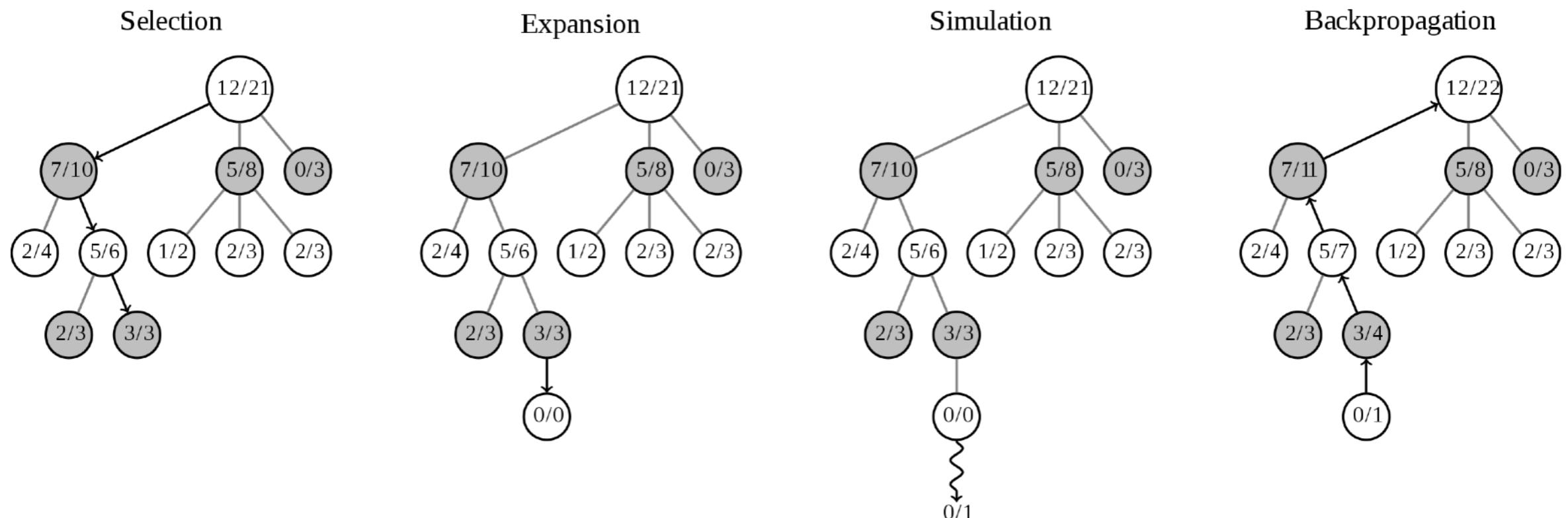


深度受限的最大期望搜索

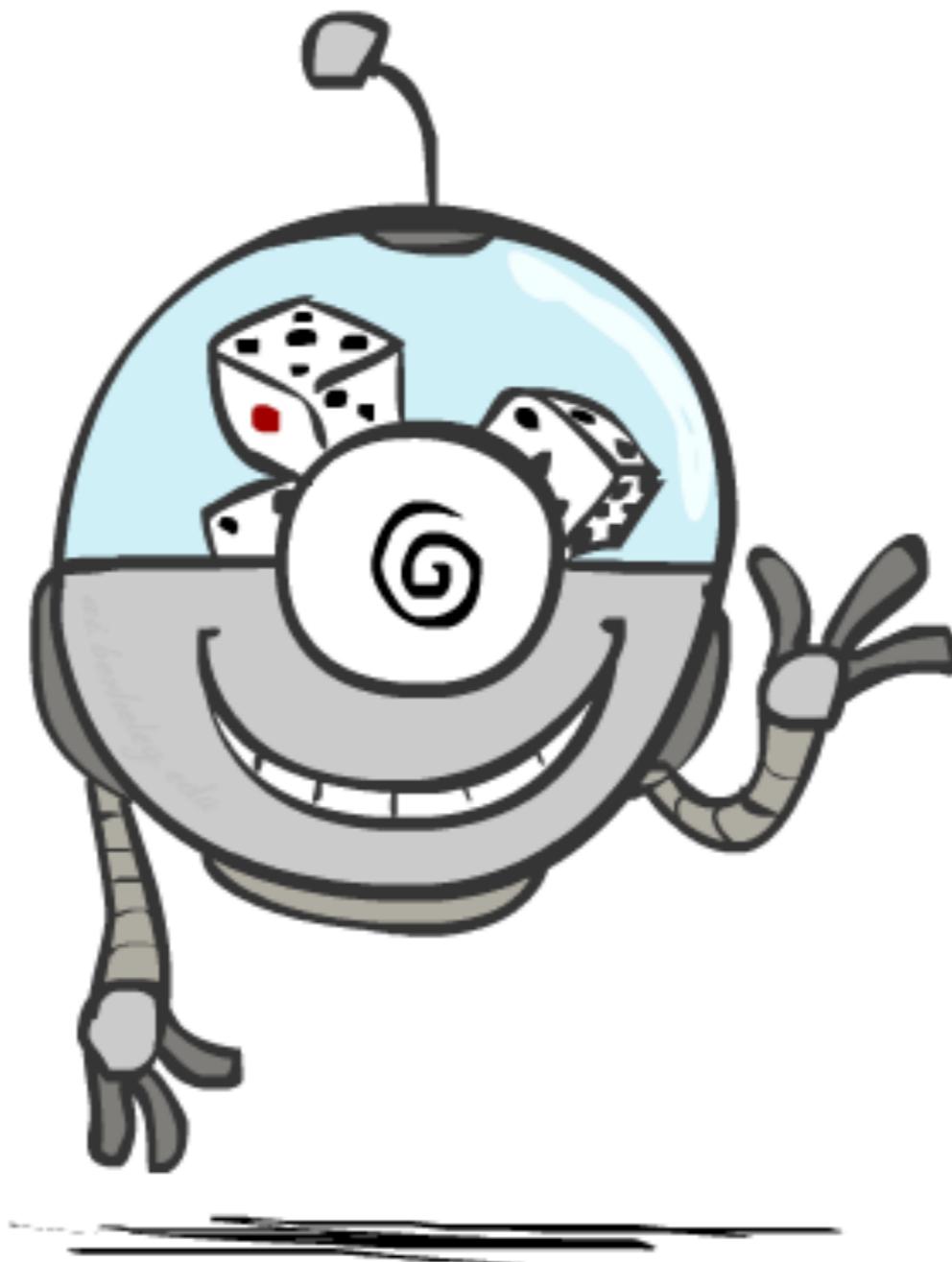


蒙特卡洛树搜索

- 通过某种“试验”的方法，得到事件出现的频率，或者随机变数的平均值，并用它们作为问题的解。



概率



什么是概率

- 一个用来表示未知事件是否发生的随机变量

- 一个结果的权重分配的概率分布

- 示例：高速路上的交通情况

- 随机变量： $T = \text{交通是否拥堵}$

- 结果： $T \in \{\text{none}, \text{light}, \text{heavy}\}$

- 分布： $P(T=\text{none}) = 0.25, P(T=\text{light}) = 0.50, P(T=\text{heavy}) = 0.25$

- 随着观察到更多信息，概率可能会发生变化：

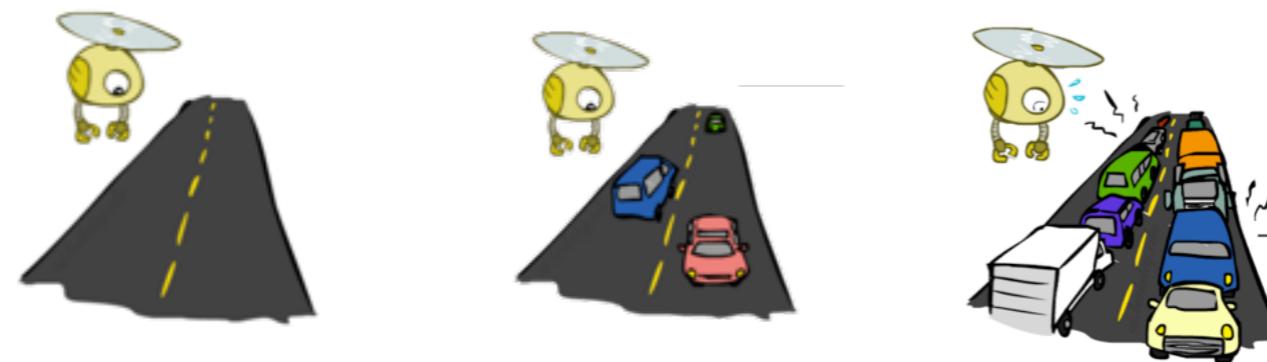
- $P(T=\text{heavy}) = 0.25, P(T=\text{heavy} | \text{Hour}=8\text{am}) = 0.60$



什么是期望

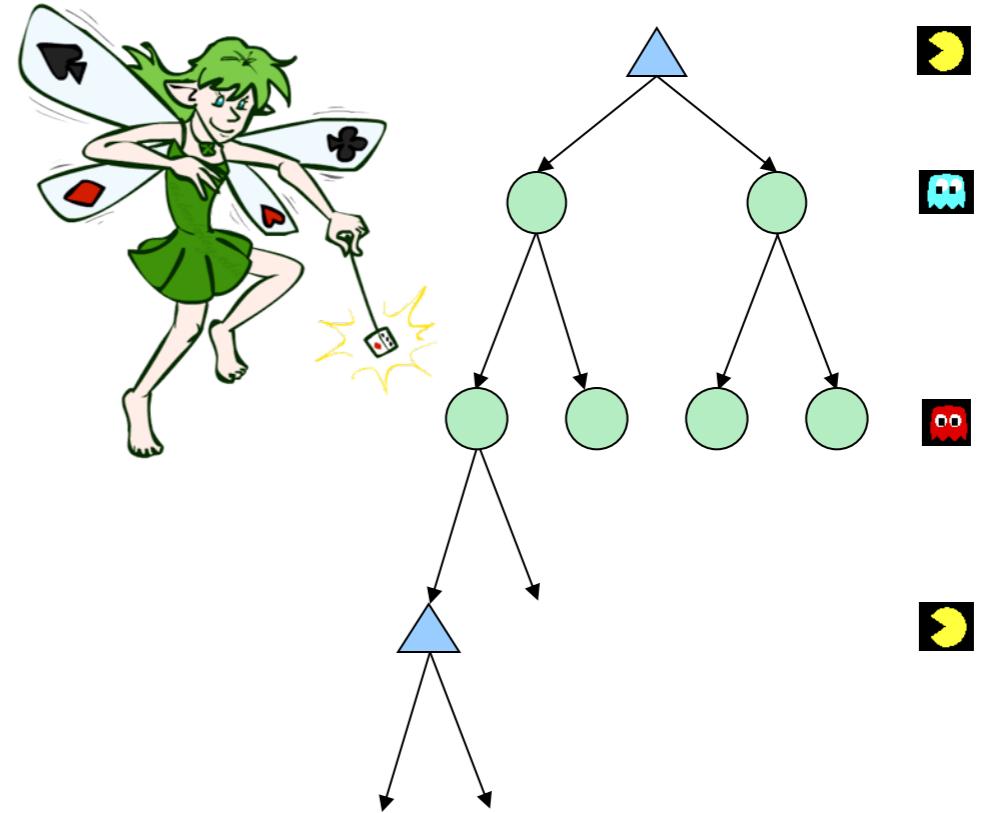
- 随机变量函数的期望值是按结果的概率分布加权的平均值。
- 示例：到机场要多长时间？

Time:	20 min	x	+	30 min	x	+	60 min	x	35 min
Probability:	0.25			0.50			0.25		



使用概率建模

- 在expectimax搜索中，我们有一个关于对手（或环境）在任何状态下的行为的概率模型
 - 模型可以是一个简单的均匀分布（掷骰子）
 - 模型可能很复杂，需要大量的计算
 - 任何一个我们的对手都有可能失去控制
 - 这个模型可能会说敌对行动很可能发生！
- 现在，假设每个机会节点都有一个指定其结果分布的概率

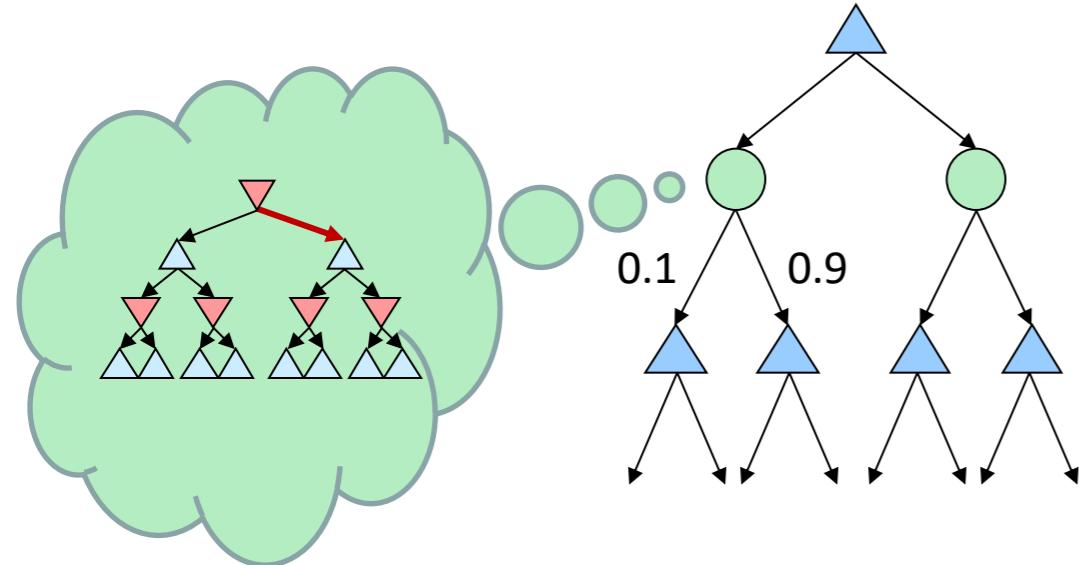


对另一个Agent的行动有一个概率性的信念，并不意味着该Agent的行为是随机的！

例子： 知情概率

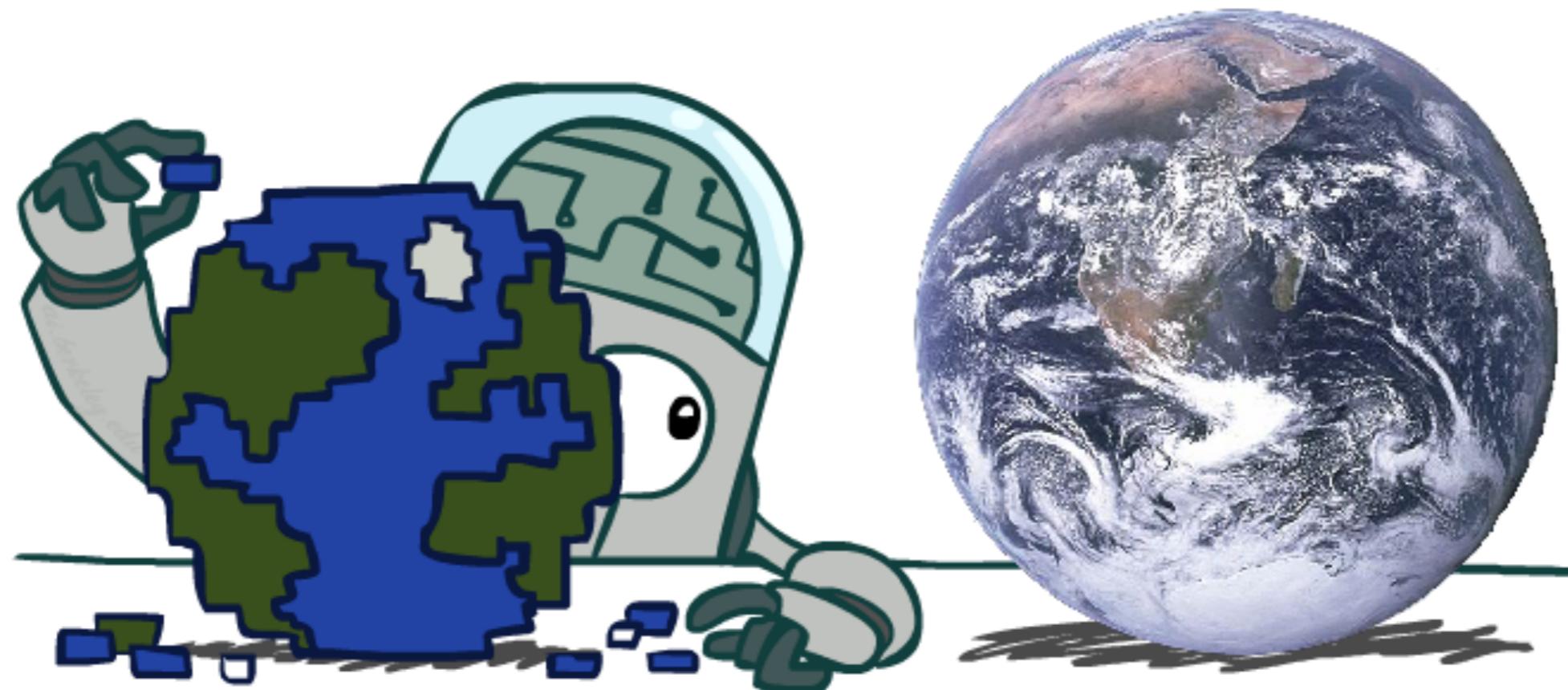
- 假设你知道对手在进行深度为2的Minimax搜索，并且在80%的时候使用此搜索的结果作为移动策略，余下的20%则随机移动。
- 问题：你应该使用何种搜索策略？

- 答案：最大期望搜索



- 要弄清楚每个机会节点的概率，必须对对手进行模拟
- 这种模拟将会变得很慢
- 更糟糕的是，如果需要模拟对手来模拟你.....

建模假设



乐观主义和悲观主义的危险

乐观主义的危险

当世界很危险时，仍然假设存在着“机会”

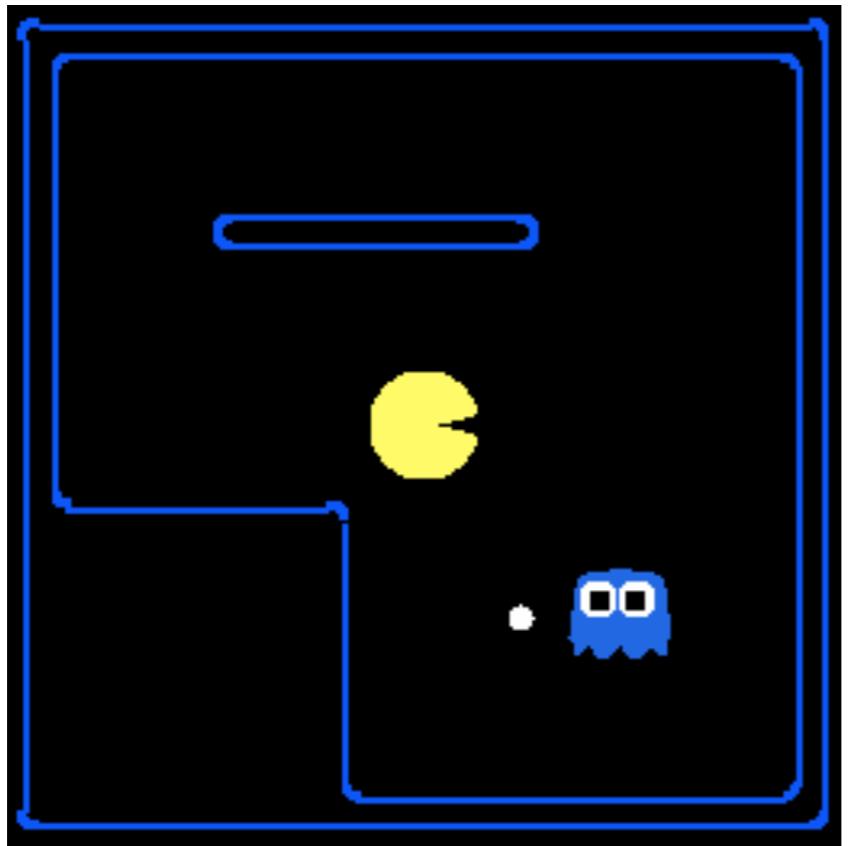


悲观主义的危险

假设不太会发生的“坏”情况



假设与现实

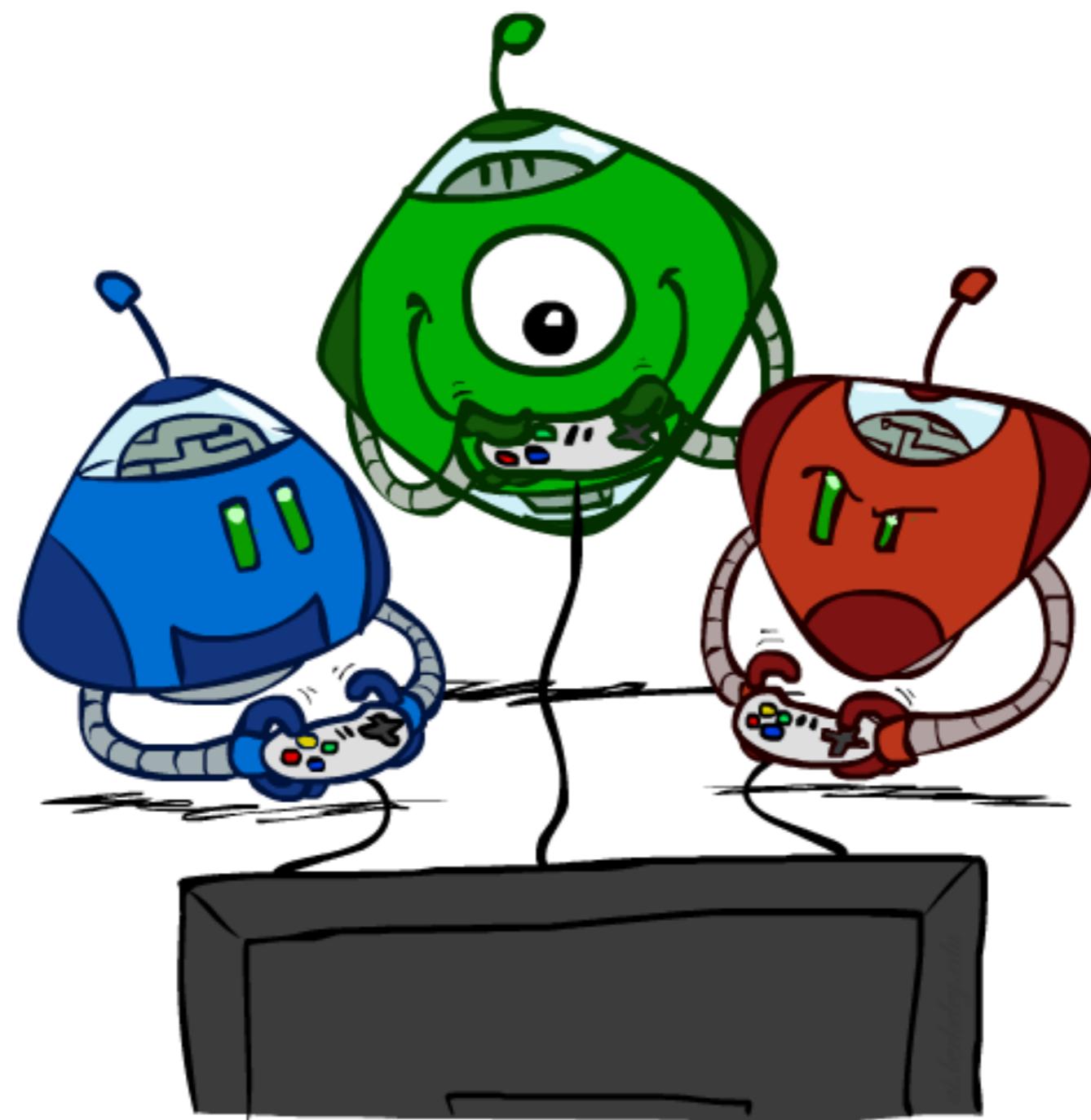


	Adversarial Ghost	Random Ghost
Minimax Pacman	Won 5/5 Avg. Score: 483	Won 5/5 Avg. Score: 493
Expectimax Pacman	Won 1/5 Avg. Score: -303	Won 5/5 Avg. Score: 503

Pacman使用depth 4 search和eval函数来避免被吃

Ghost使用depth 2 search和eval函数来查找Pacman

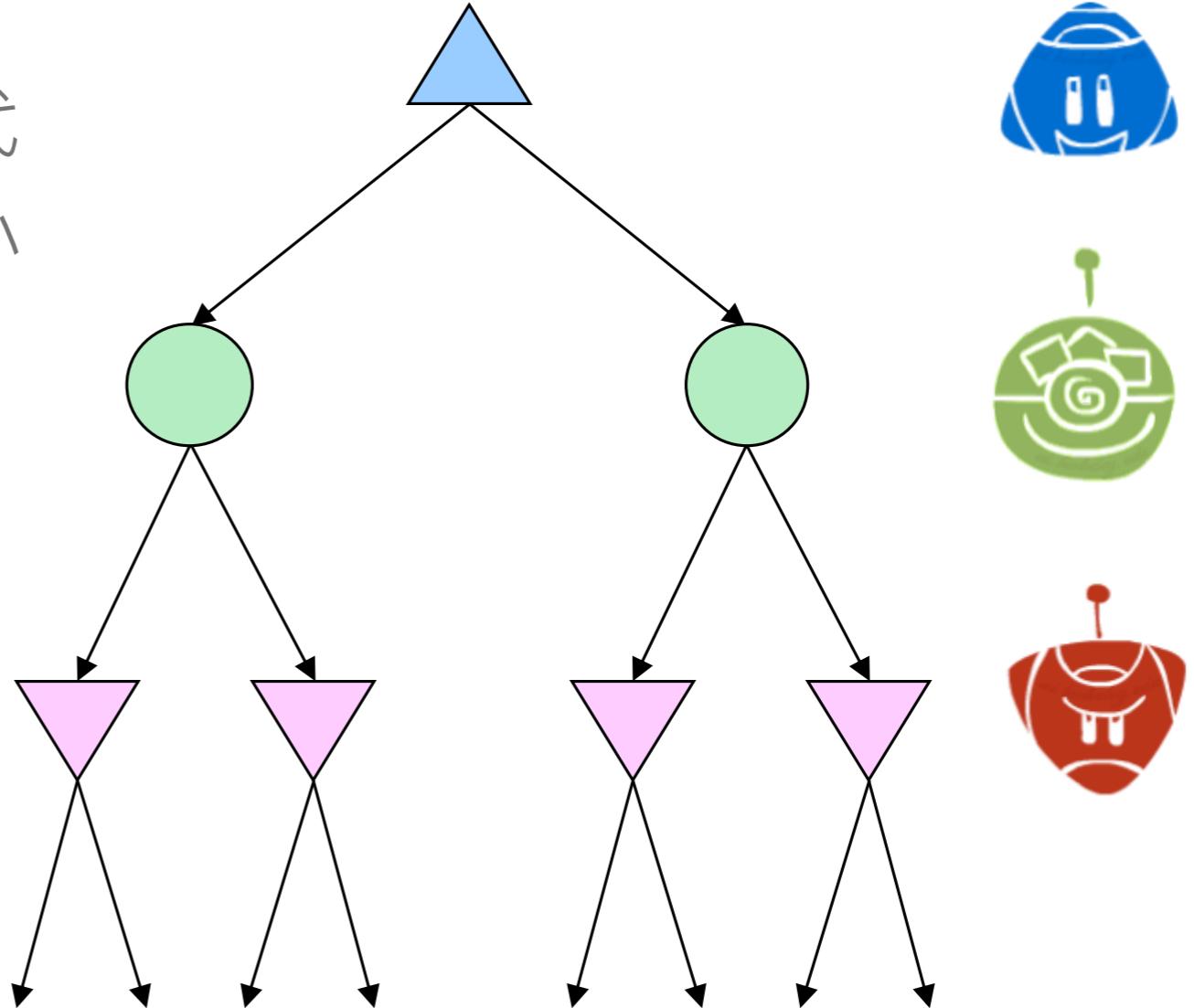
其他类型的博弈



多种层类型的混合

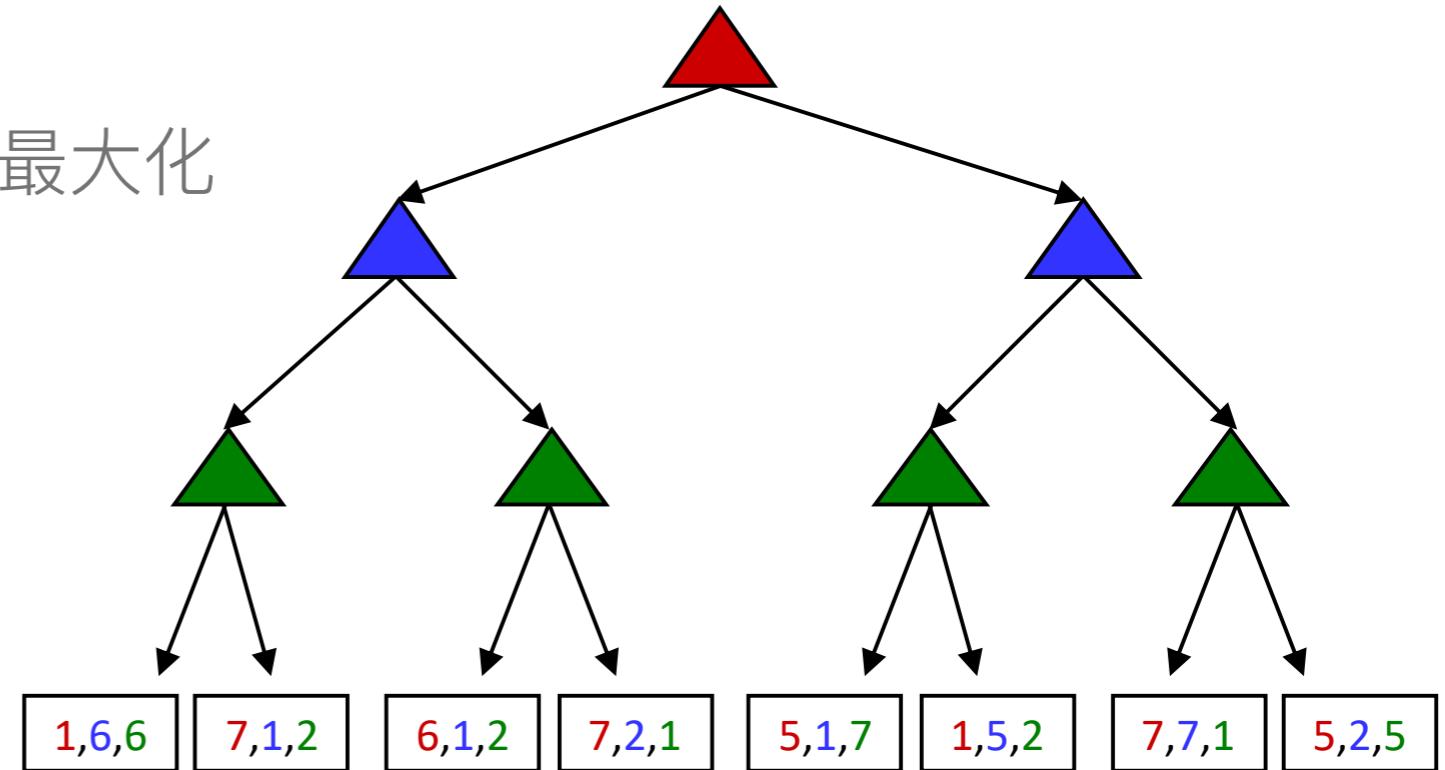
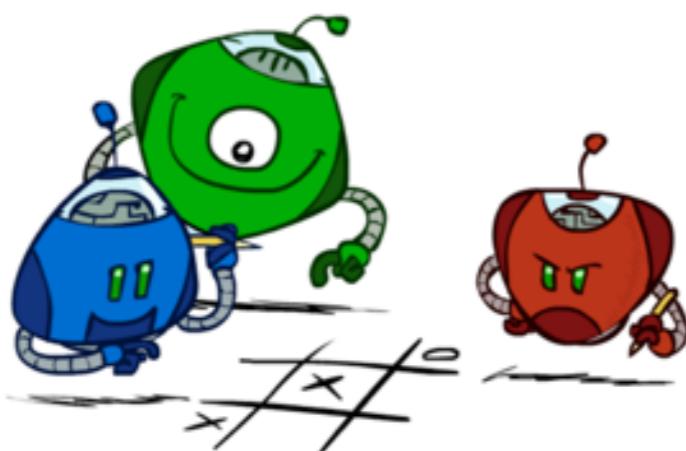
- 例如双陆棋

- 环境是一个额外的“随机代理”播放器，它在每个最小/最大代理之后移动

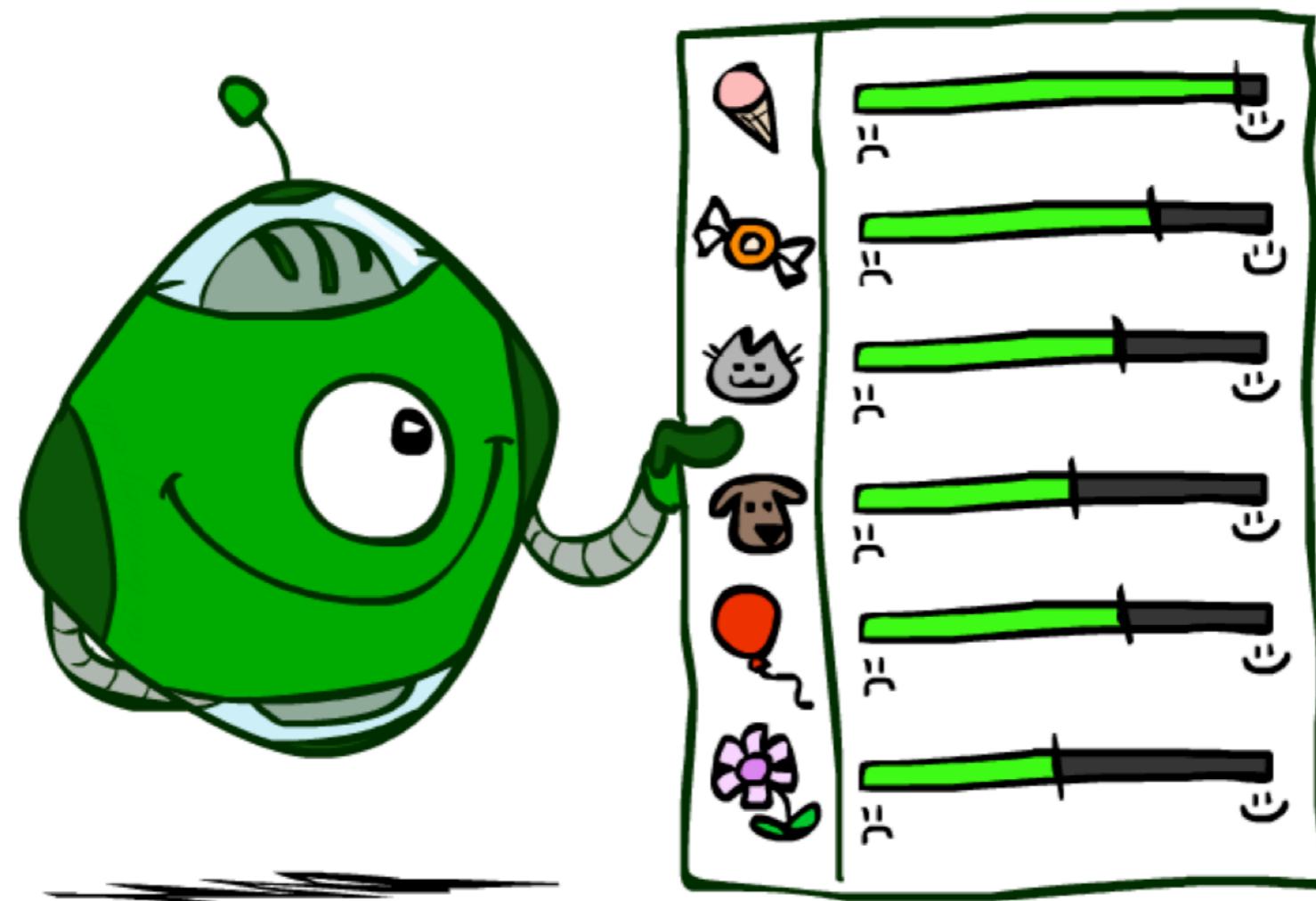


多Agent博弈

- 如果游戏不是零和游戏，或者有多个玩家怎么办？
- 一般化的minimax：
 - 终端有效用三元组
 - 节点值也是三元组
 - 每个参与者都将自己的效用最大化
 - 可以引起合作和动态竞争...



效用

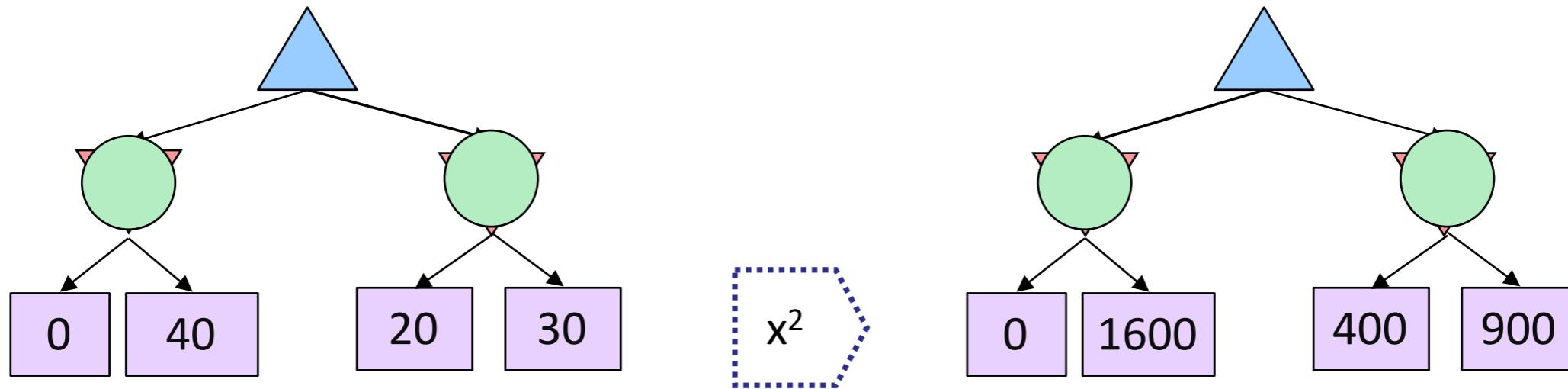


最大预期效用

- 为什么要平均效用？为什么不是minimax？
- 最大预期效用原则：
 - 鉴于其知识，理性Agent应选择能最大化其预期效用的行动
- 问题：
 - 效用来自哪里？
 - 我们怎么知道这样的效用存在？
 - 我们怎么知道平均效用有意义？
 - 如果效用无法描述我们的行为（偏好）怎么办？



使用什么效用?



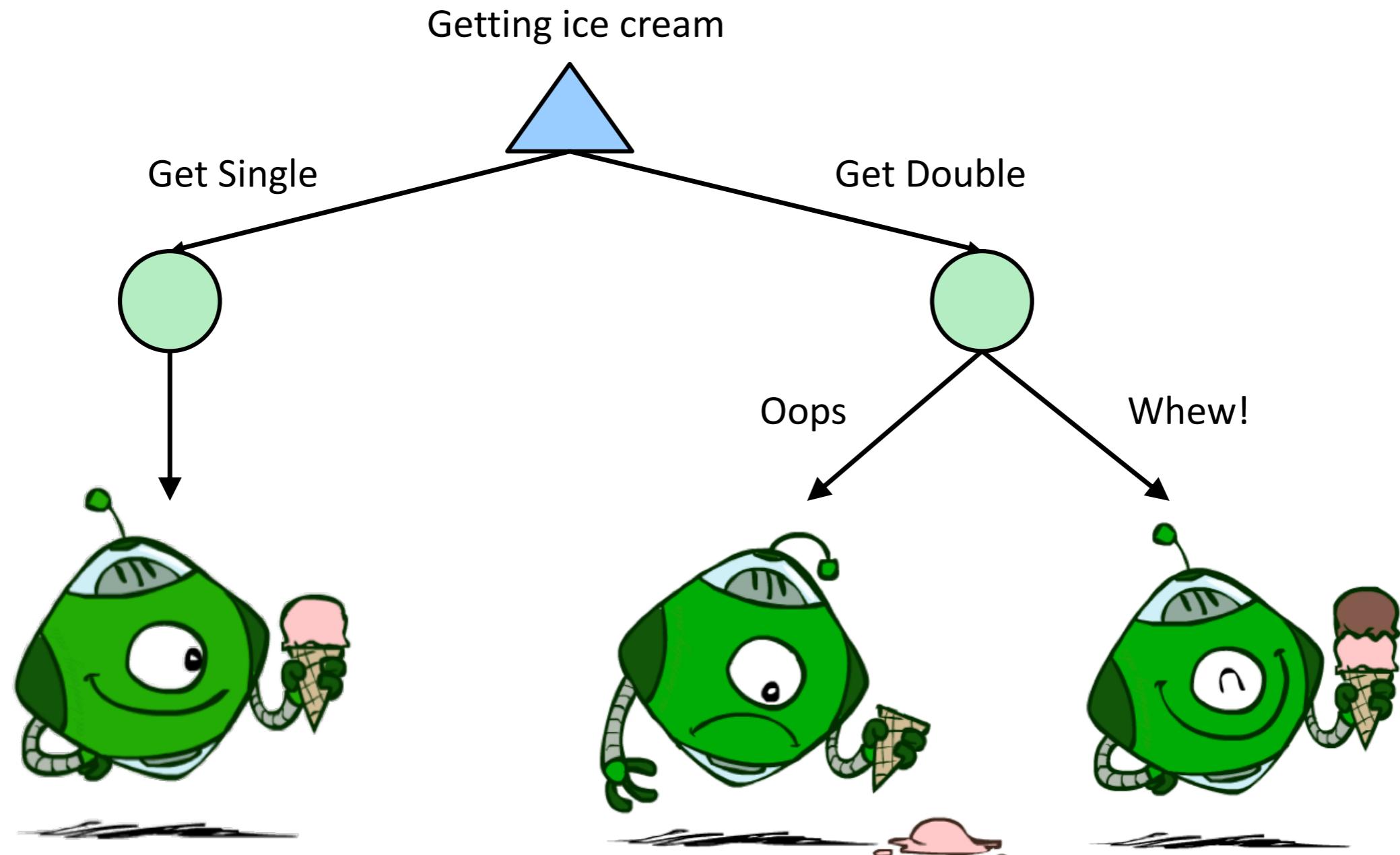
- 对于最坏情况的minimax推理，效用取值范围无关紧要
 - 我们只希望更好的状态有更高的评价（正确排序）
 - 我们称这种现象为：**对单调变换不敏感**
- 对于一般情况下的expectimax推理，我们需要有意义的效用值

效用

- 效用是从结果（世界状态）到描述Agent偏好的函数
- 效用从何而来？
 - 在游戏中，可能很简单 (+1/-1)
 - 效用代表了Agent的目标
 - 定理：任何“理性”的偏好都可以概括为一个效用函数
- 我们编写效用函数，从而实现行为决策
 - 我们为什么不让Agent选择效用函数？
 - 我们为什么不规定行为？



效用：不确定的结果



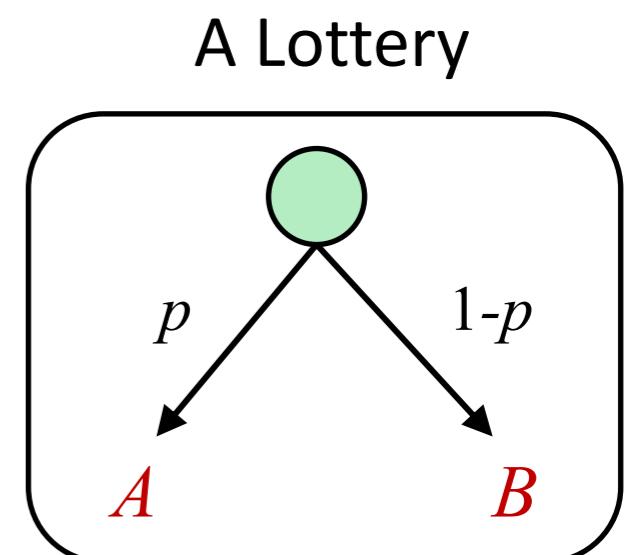
偏好

- 一个Agent必须要在不同的结果中有所偏好：

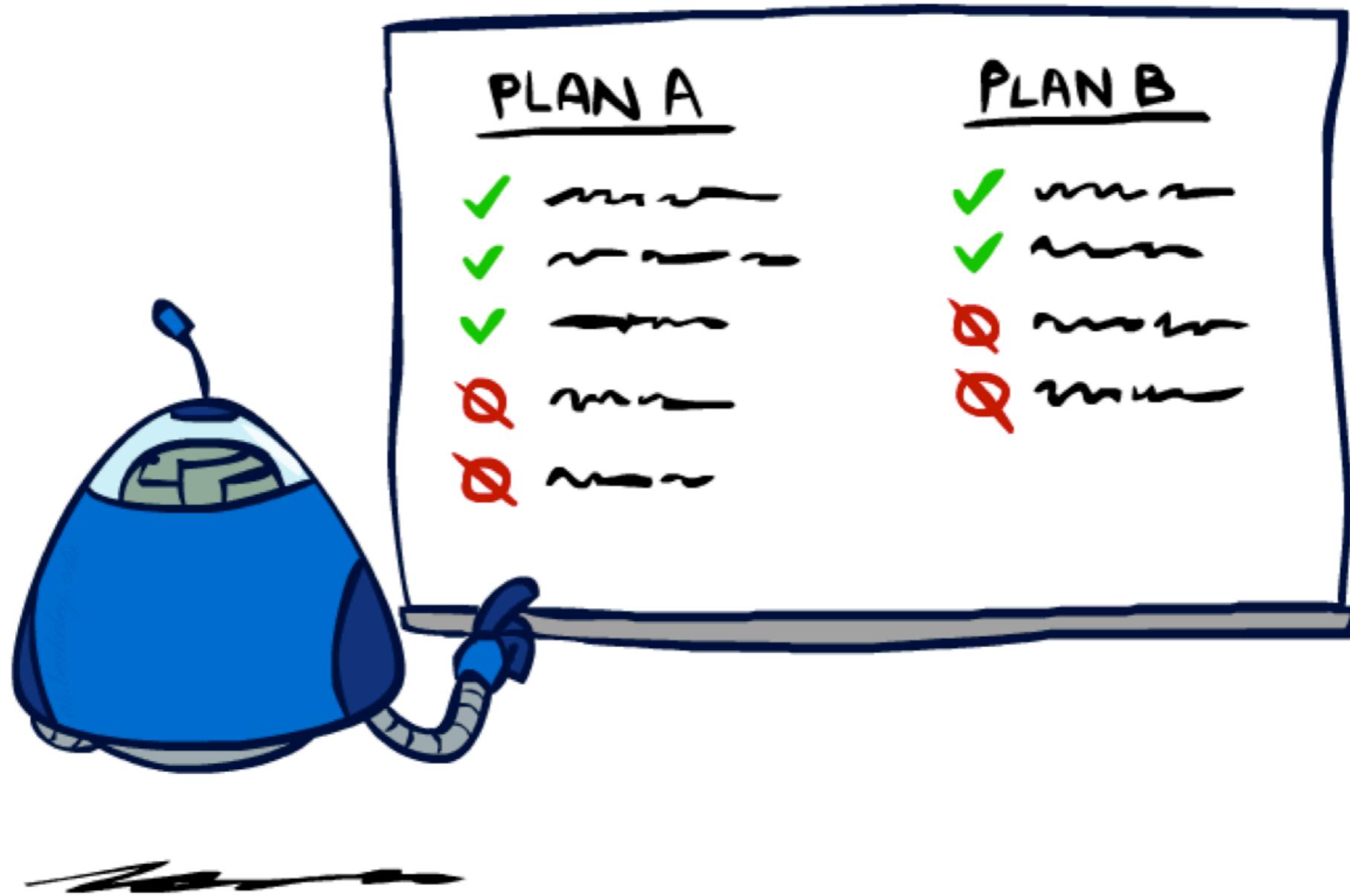
- 奖品：A, B, etc
 - 抽奖：奖品不确定的情况

$$L = [p, A; \ (1 - p), B]$$

- 符号
 - 偏好: $A \succ B$
 - 没有偏好: $A \sim B$



理性

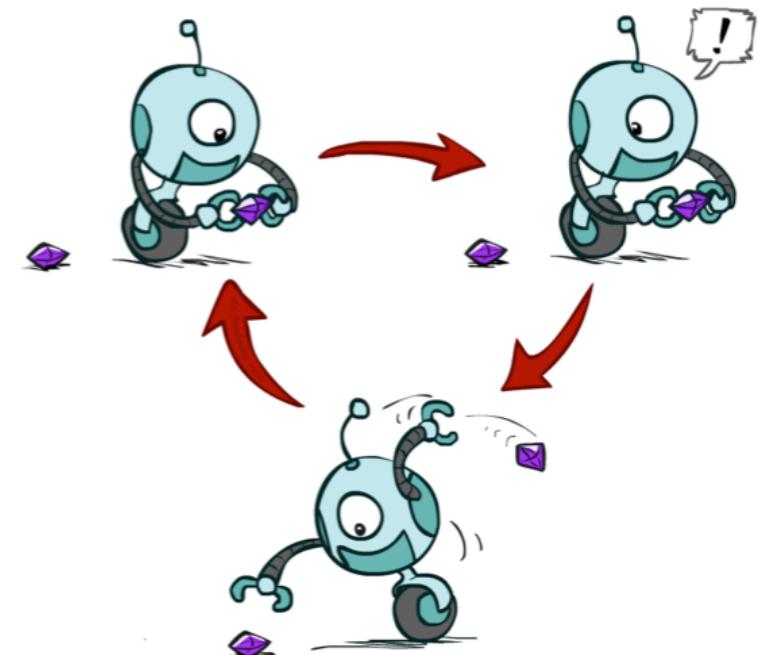


理性偏好

- 在我们称之为理性偏好之前，我们需要一些约束条件，例如：

偏好的传递性: $(A \succ B) \wedge (B \succ C) \Rightarrow (A \succ C)$

- 例如：一个具有不传递偏好的代理
 - 如果 $B > C$, 那么Agent将支付一定代价来获得B
 - 如果 $A > B$, 则Agent将支付一定代价来获得A
 - 如果 $C > A$, 则Agent将支付一定代价来获得C



理性偏好

Orderability

$$(A \succ B) \vee (B \succ A) \vee (A \sim B)$$

Transitivity

$$(A \succ B) \wedge (B \succ C) \Rightarrow (A \succ C)$$

Continuity

$$A \succ B \succ C \Rightarrow \exists p [p, A; 1 - p, C] \sim B$$

Substitutability

$$A \sim B \Rightarrow [p, A; 1 - p, C] \sim [p, B; 1 - p, C]$$

Monotonicity

$$A \succ B \Rightarrow$$

$$(p \geq q \Leftrightarrow [p, A; 1 - p, B] \succeq [q, A; 1 - q, B])$$



- 定理：理性偏好意味着行为可以描述为期望效用的最大化

最大期望效用 (MEU) 原则

- 定理：给定满足这些约束的任何偏好，就存在一个实值函数 U ，使得：

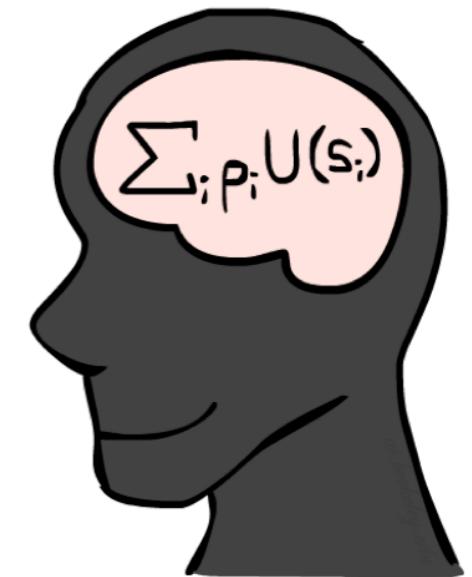
$$U(A) \geq U(B) \Leftrightarrow A \succeq B$$

$$U([p_1, S_1; \dots; p_n, S_n]) = \sum_i p_i U(S_i)$$

- 即 U 的值保留了偏好！

- 最大期望效用 (MEU) 原则：

- 选择最大化期望效用的操作
- 注意：Agent可以是完全理性的（与MEU一致），而无需表示或操纵实用程序和概率
- 例如，用于完美井字棋的查找表，反射式真空吸尘器



作业

- 请编程计算博弈树的根节点的值。（机遇节点下每个可能的行动是相同概率发生的）

