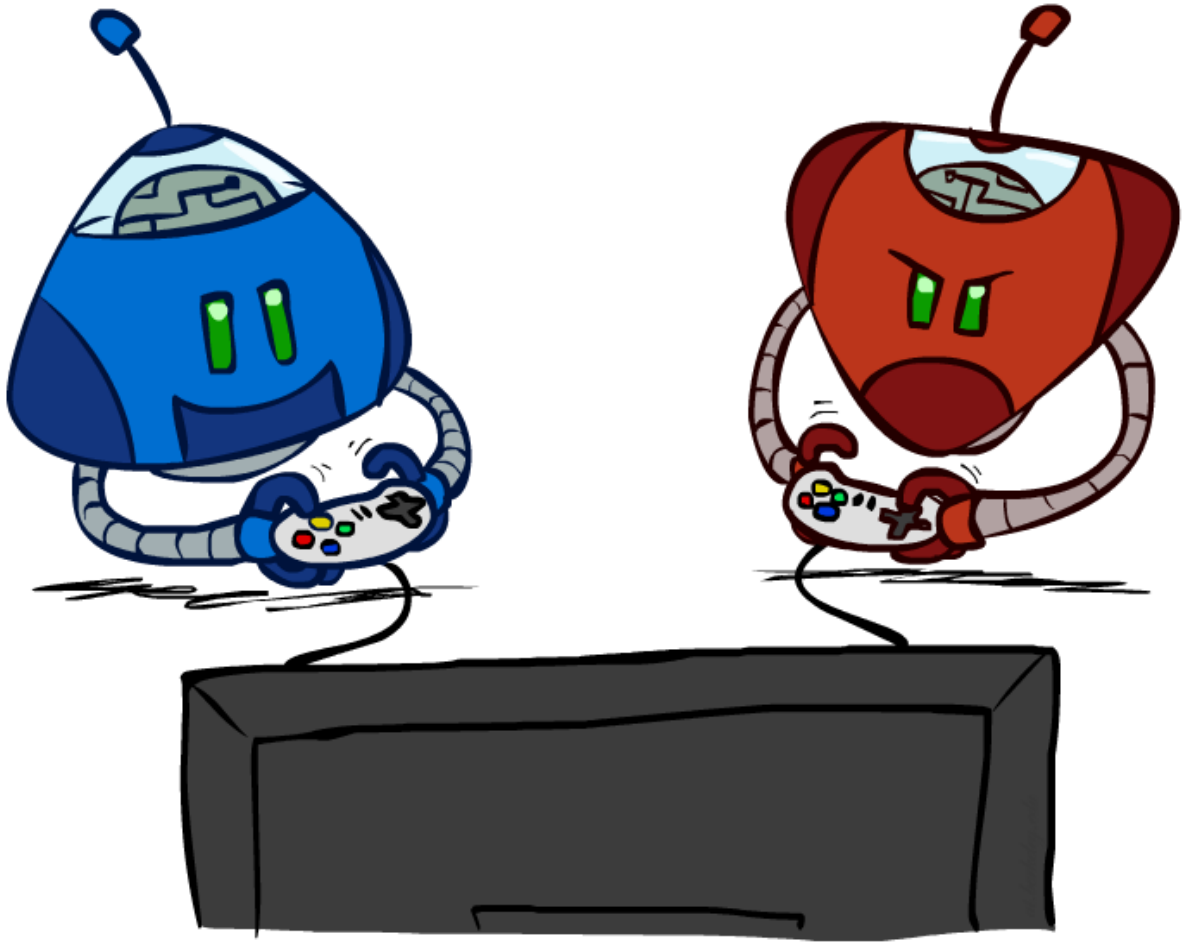
第五章　　对抗搜索



ai.berkeley.edu

主要内容

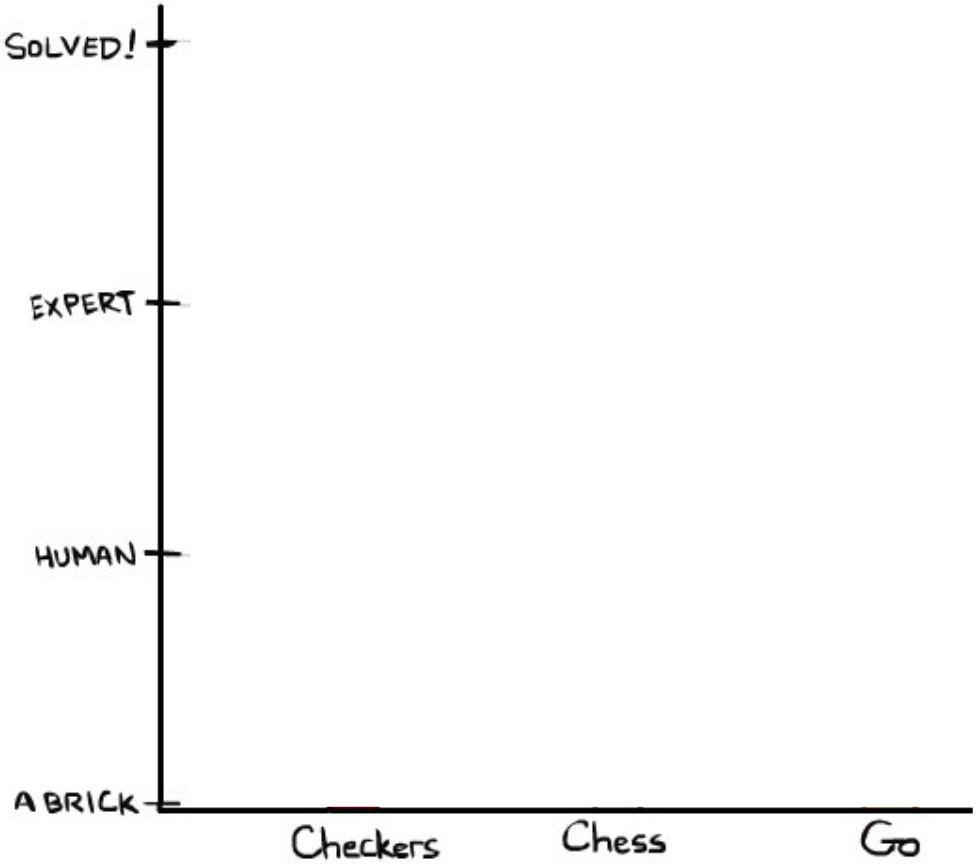
- 5.1 Games theory (博弈论)
- 5.2 极小极大原理
- 5.3 α-β 剪枝
- 5.4 不完美的实时决策

博弈

- 博弈和人类智慧如影随行
- 博弈游戏易于形式化，可以形式化表述为搜索问题
- 博弈也是对真实环境中竞争行为很好的表示模型（军事对抗，谈判，竞买等）

A brief history

- **Checkers:**
  - 1950: First computer player.
  - 1959: Samuel's self-taught program.
  - 1994: First computer world champion: Chinook defeats Tinsley
  - 2007: Checkers solved! Endgame database of 39 trillion states
- **Chess:**
  - 1945-1960: Zuse, Wiener, Shannon, Turing, Newell & Simon, McCarthy.
  - 1960s onward: gradual improvement under "standard model"
  - 1997: Deep Blue defeats human champion Garry Kasparov
  - 2022: Stockfish rating 3541 (vs 2882 for Magnus Carlsen 2015).
- **Go:**
  - 1968: Zobrist's program plays legal Go, barely (b>300!)
  - 1968-2005: various ad hoc approaches tried, novice level
  - 2005-2014: Monte Carlo tree search -> strong amateur
  - 2016-2017: AlphaGo defeats human world champions



博弈论（ Game Theory ）

- 博弈论（ Game Theory ），是现代数学的一个分支， 也是运筹学的一个重要学科。总是以参与者绝对理性为前提，它可能看起来很贴近生活，有很多细节和可能性，但问题却是封闭的，是一门十分严谨的科学。

博弈论中经典问题：囚徒窘境

警察抓了两个嫌疑犯，在他们没有事先串口供的情况下，分开审问。如果两个罪犯都沉默，各判 1 年；互相揭发，各判 8 年；如果一个揭发一个沉默，那么揭发的那个释放，沉默的那个判 10 年。 AB 怎么选择才对自己最有利？

	A沉默	**A揭发B**
B沉默	A、B各1年	A释放，B判10年
B揭发A	A判10年，B释放	A、B各8年

囚徒窘境

	A沉默	A揭发B
B沉默	A、B各1年	A释放，B判10年
B揭发A	A判10年，B释放	A、B各8年

由于 A ， B 事先没有沟通预谋，在不知道对方怎么选择的情况下， 结果会如何呢？

显然最优方案是互相揭发，于是警方判了两个犯人 8 年。

囚徒窘境

	A沉默	A揭发B
B沉默	A、B各1年	A释放，B判10年
B揭发A	A判10年，B释放	A、B各8年

如果审问并不是分开进行，而是二人一起，结果又会如何呢？

稍作思考，A 选择了沉默，B 当然也做出同样的分析。最后两人只被各判 **1** 年，整体的纳什均衡达成。

纳什均衡

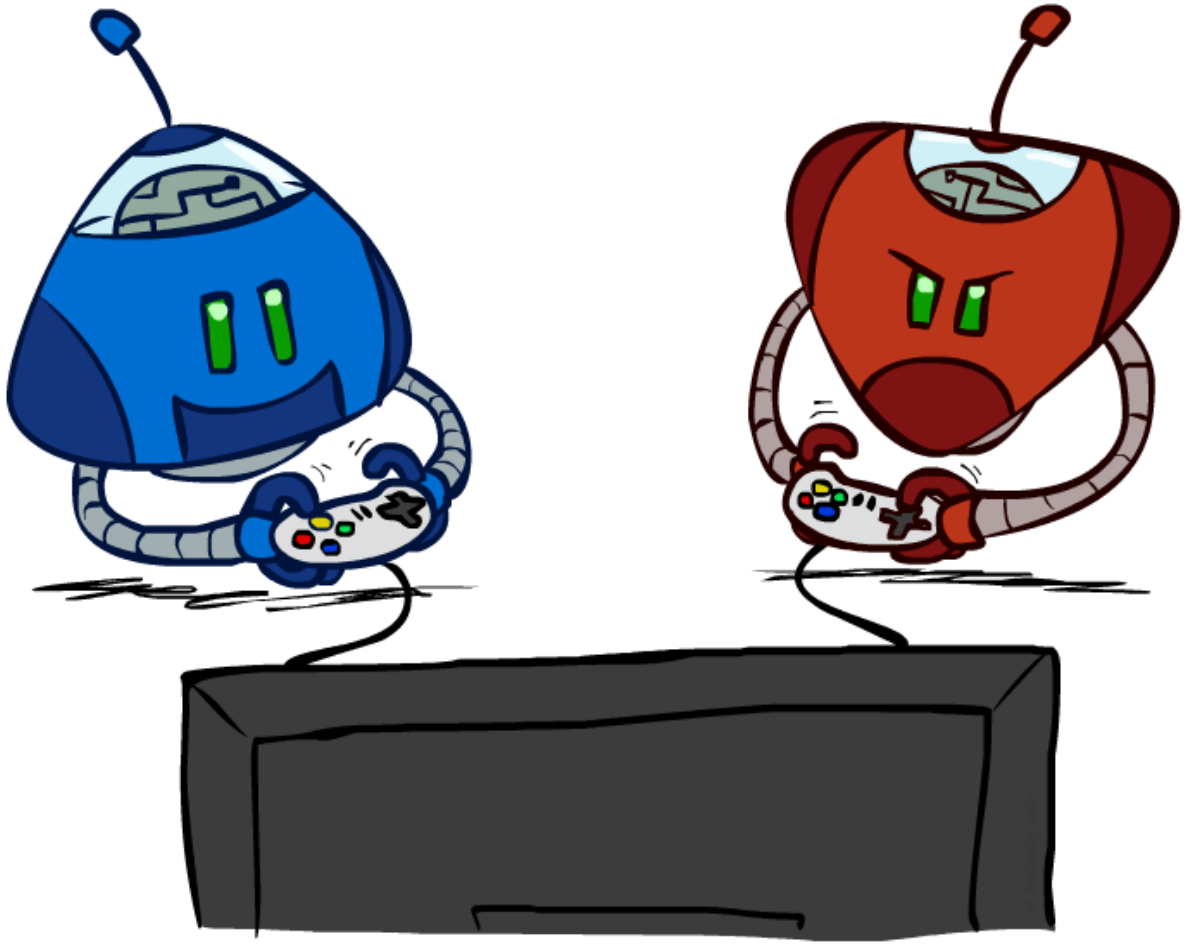
纳什均衡（ Nash equilibrium ）由美国数学家纳什提出，在多人博弈的时候，如果其他人不改变策略，不论我怎么改变也不能增加收益，所有人都是这样，也就达到了纳什均衡。

约翰纳什（ John Nash ），著名经济学家，博弈论创始人。

“ 使双方都不后悔的理性解”

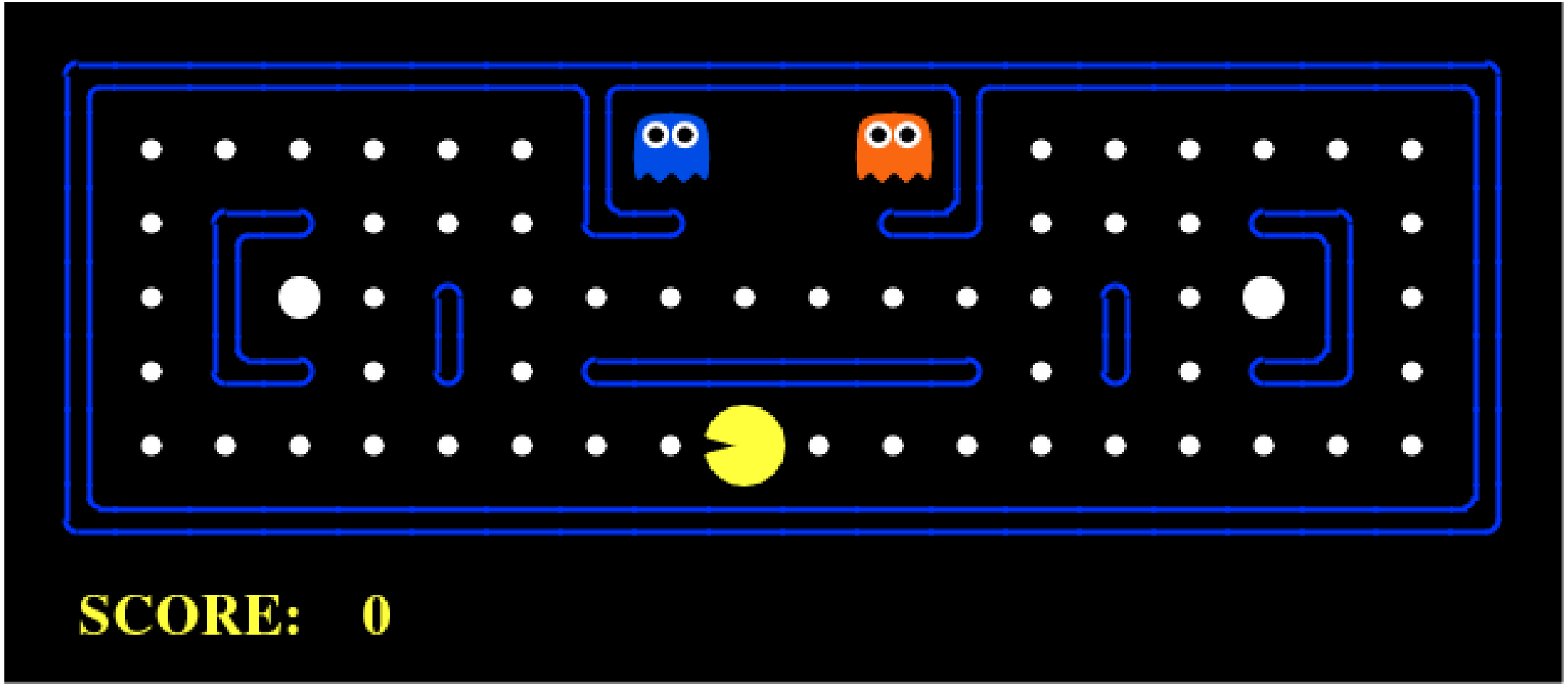
纳什均衡可以认为是博弈论实现人工智能的一个基本基石。

第五章　　对抗搜索



ai.berkeley.edu

Multi-Agent Pacman



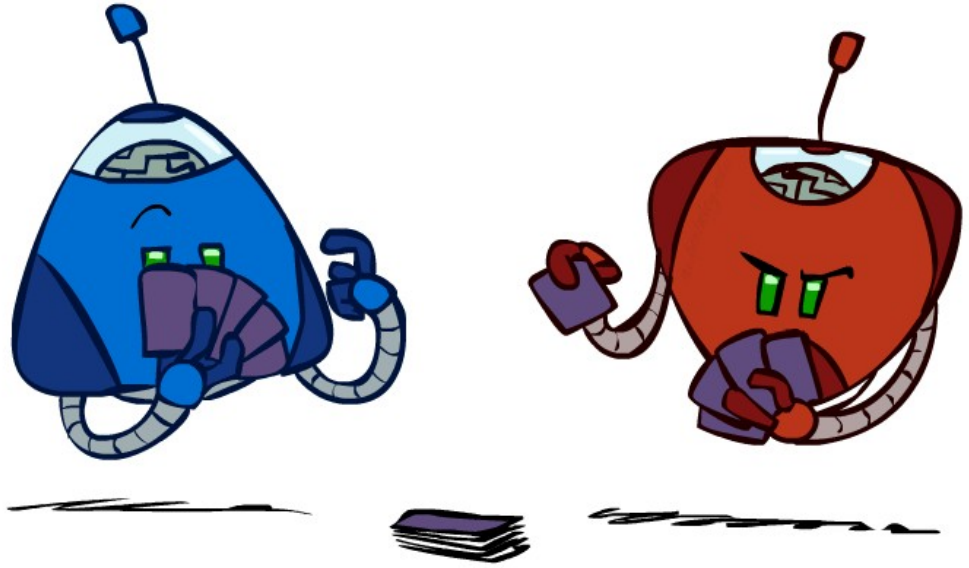
主要内容

- 5.1 Games theory (博弈论)
- 5.2 极小极大搜索算法
- 5.3 α-β 剪枝
- 5.4 不完美的实时决策

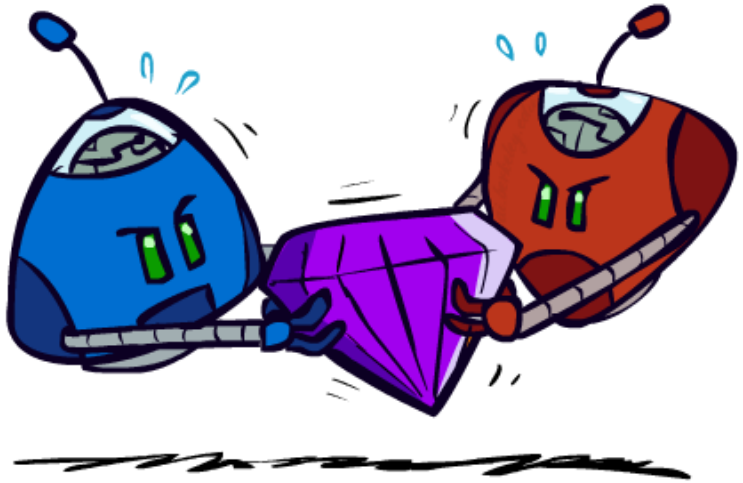
Types of Games

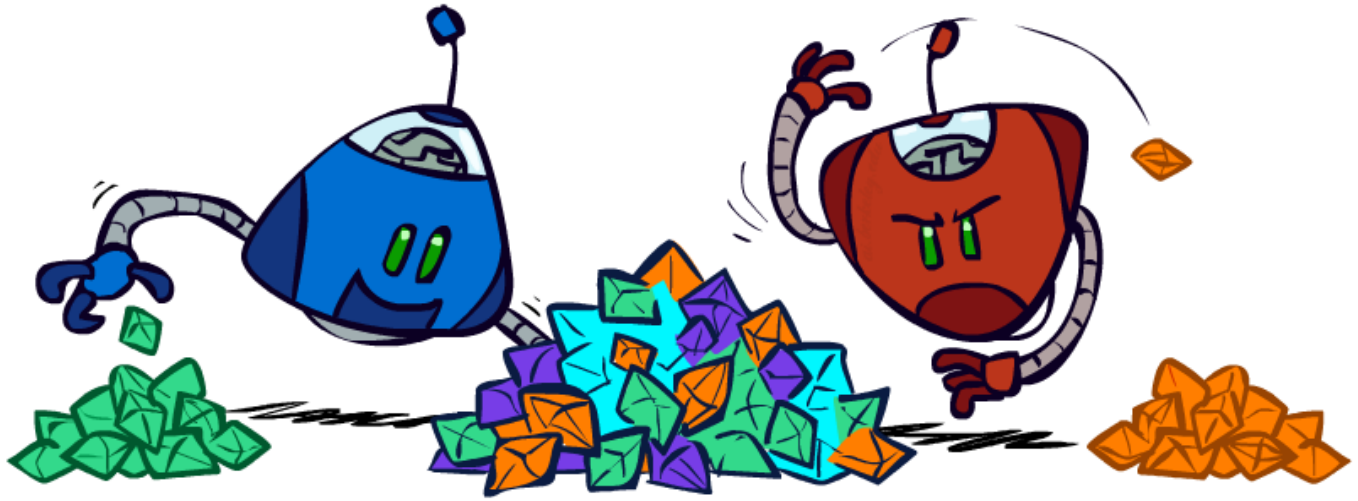
- Game = task environment with > 1 agent

- 任务环境类型：
  - Deterministic or stochastic?
  - Perfect information (fully observable)?
  - Two, three, or more players?
  - Individuals or teams?
  - Turn-taking or simultaneous?
  - Zero sum?



零和博弈



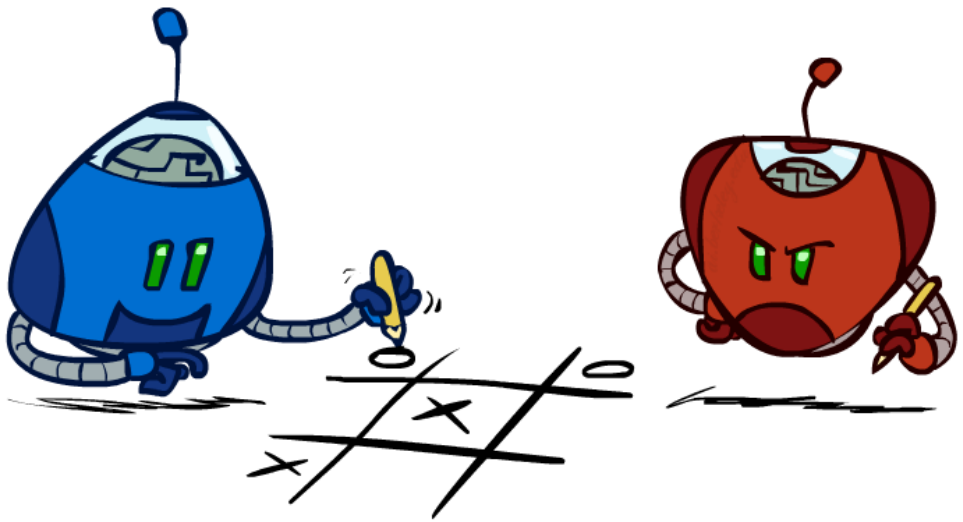


- 零和博弈
  - Agent 具有相反的效用
  - 一个收益：一个最大，另一个最小
  - 对抗性的，纯粹的竞争
- 一般博弈
  - Agent 具有独立的效用
  - 合作、冷漠、竞争等等都是可能的

Deterministic Games

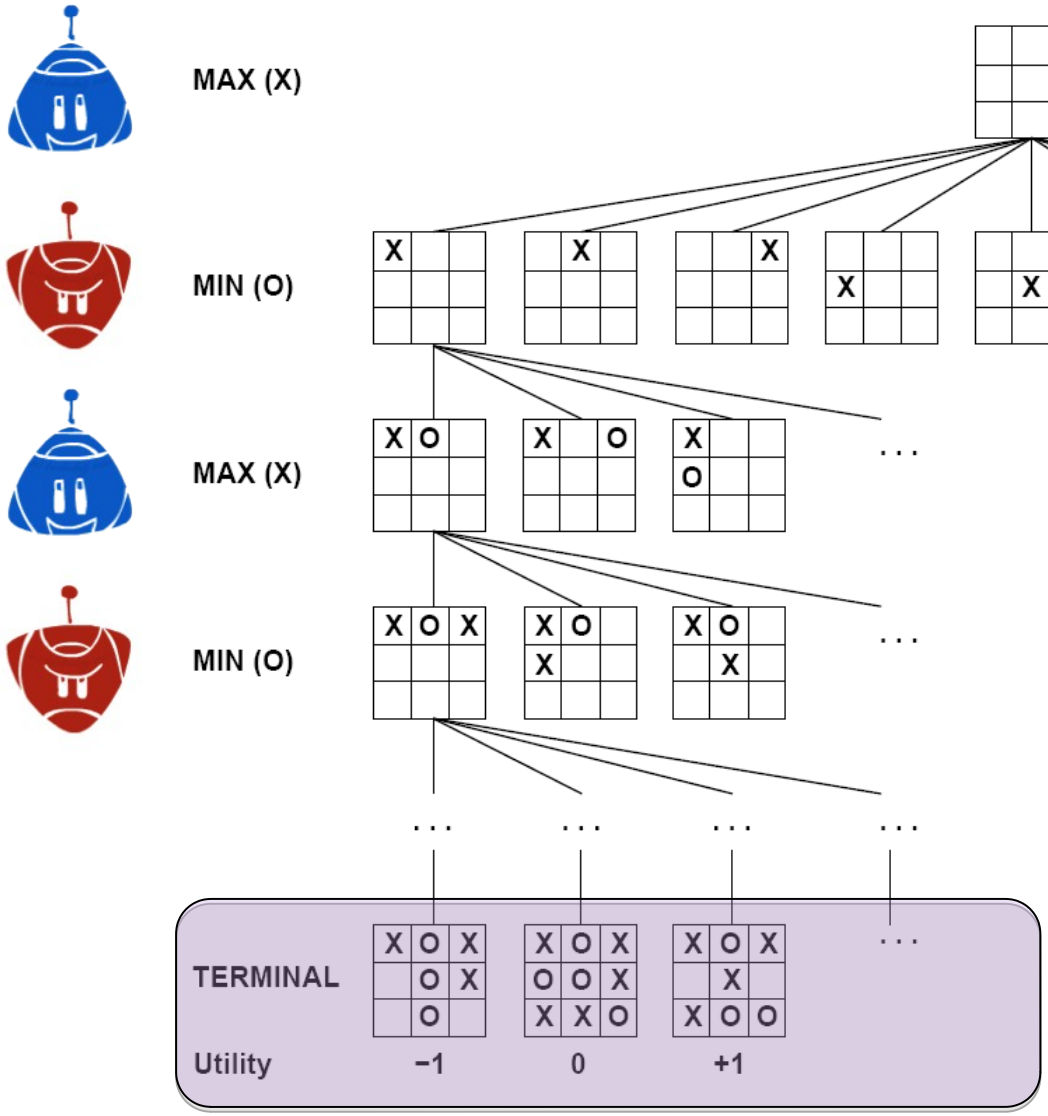
- 问题形式化描述：

  - States: S (start at s_0)
  - Players: P={1…N} (usually take turns)
  - Actions: A (may depend on player / state)
  - Transition Model: RESULT(S,A) → S'
  - Terminal Test: S → {t, f}
  - Utility Function: Utility(S,P) → R

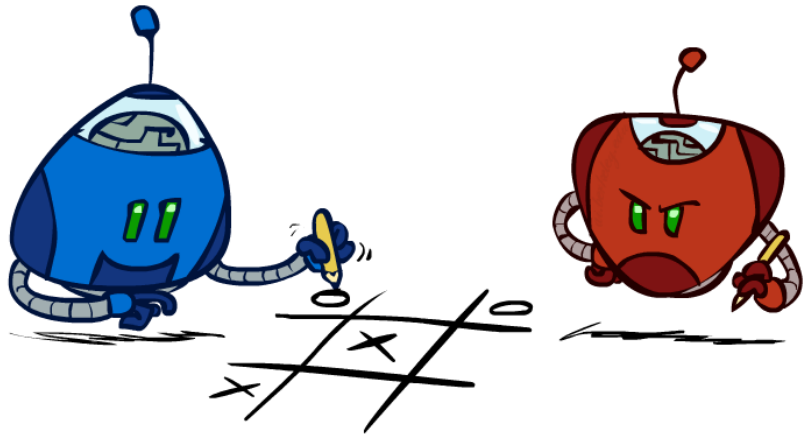


- Solution for *a player* is a policy: S → A(为每个可能的状态指定一个动作)

Tic-Tac-Toe Game Tree

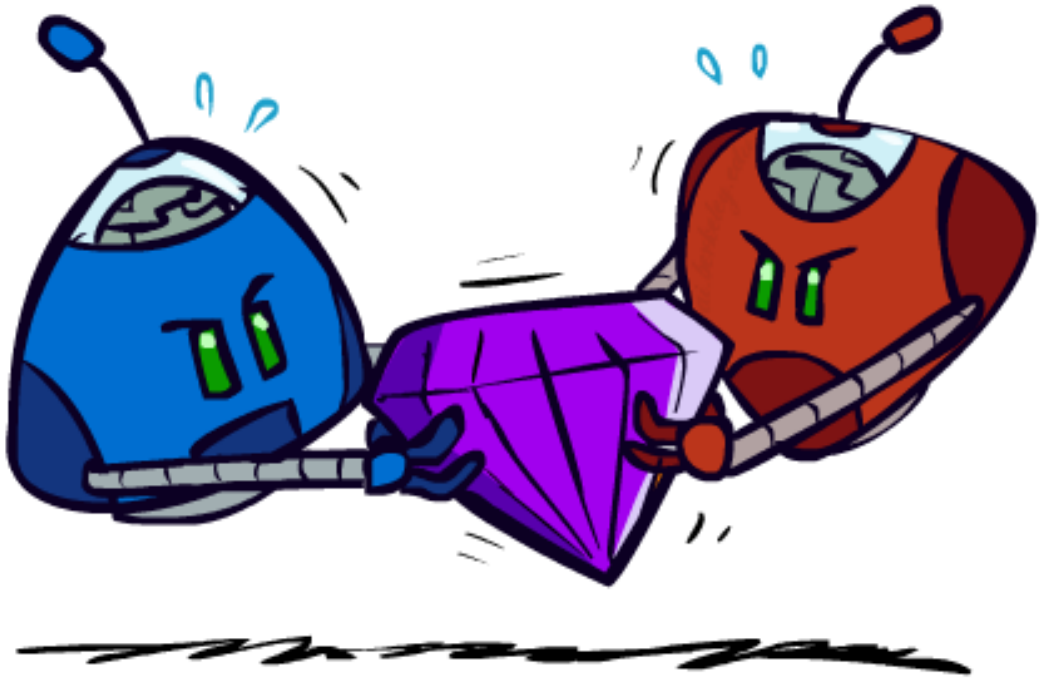


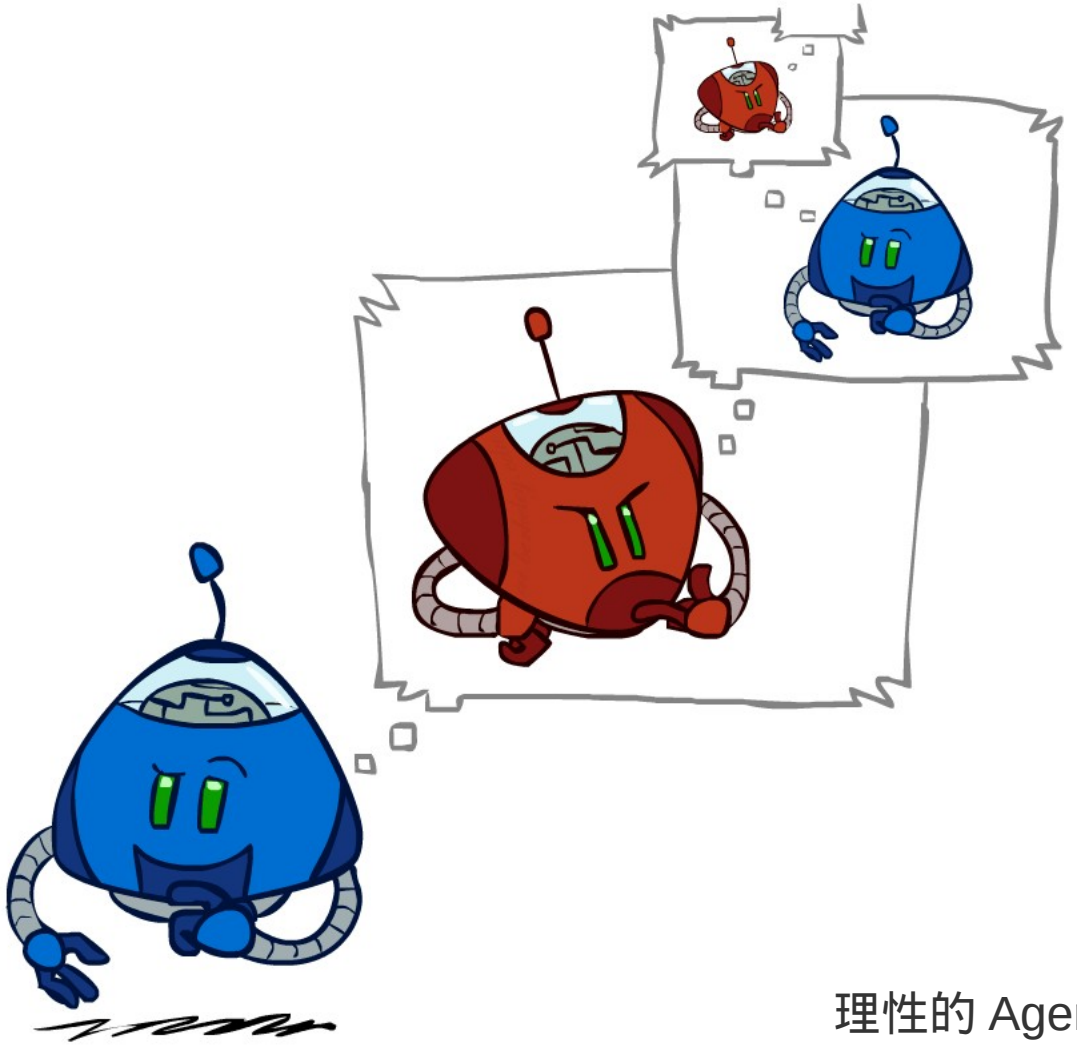
初始状态



叶节点：终止状态对于 MAX 的效用值

Adversarial Games





理性的 Agent

极小极大原理

冯·诺依曼，20 世纪最重要的数学家之一，在现代计算机、博弈论、核武器和生化武器等领域内的科学全才之一，被后人称为“计算机之父”和“博弈论之父”。

1944 年，与奥斯卡•摩根斯特恩合著《博弈论与经济行为理论》被认为是博弈论领域的第一本重要著作。

1926 年，冯·诺依曼理论上证明了所有零和博弈都有一个极小极大值解。

博弈论中经典问题：分蛋糕

- 一块蛋糕，该怎么分才能让两个孩子都满意？

首先，我们要把分蛋糕问题需要转化为两个孩子博弈问题

博弈的规则是：两个孩子分蛋糕，一个切蛋糕 A ，另一个先选蛋糕 B 。

博弈论的目标就是寻找问题的理性解——从理性角度分析所得的答案。

	B 选蛋糕	A 拿到的蛋糕
A 切成两块一样大	一半	一半
A 切成两块不一样大	大块	小块
	小块	大块

博弈论中经典问题：分蛋糕

- 一块蛋糕，该怎么分才能让两个孩子都满意？

	B 选蛋糕	A 拿到的蛋糕
A 切成两块一样大	一半	一半
A 切成两块不一样大	大块	小块
	小块	大块

A 切蛋糕，运用“极小极大原理”：

“ 极小”指的是 B 一定会挑选大块，所以留给自己的肯定是小块；

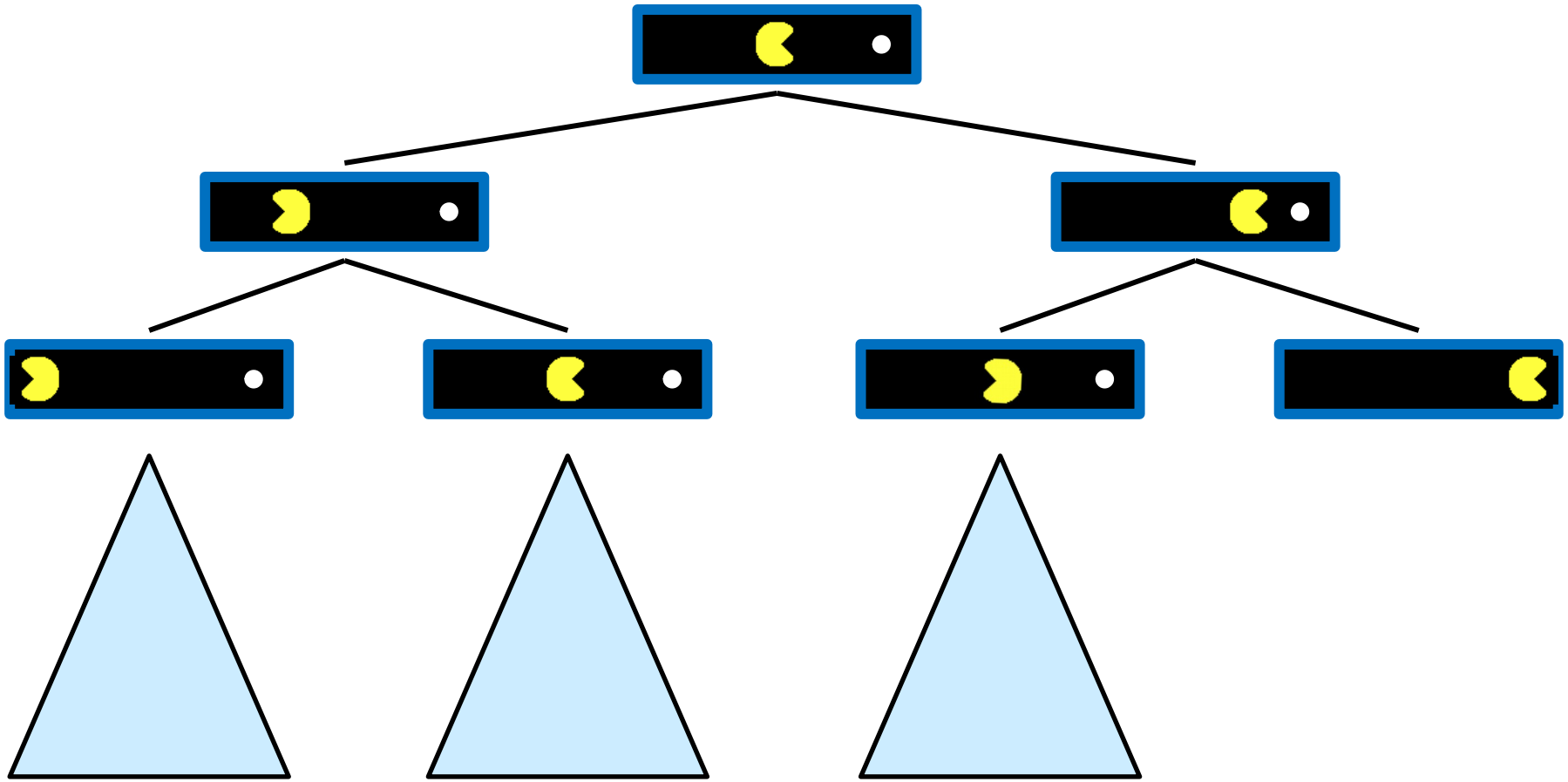
“极大”指的是 A 要使自己的蛋糕尽量大；

“极小极大”组合起来的意思是， A 已知 B 会选大块，所以会把较小的一块切得大一些。对 A 来说，最好的结果就是“一半、一半”，即两人各分得半块蛋糕，这就是这个问题的理性解。

主要内容

- 5.1 Games theory (博弈论)
- 5.2 极小极大搜索算法
- 5.3 α-β 剪枝
- 5.4 不完美的实时决策

单 Agent 的搜索树



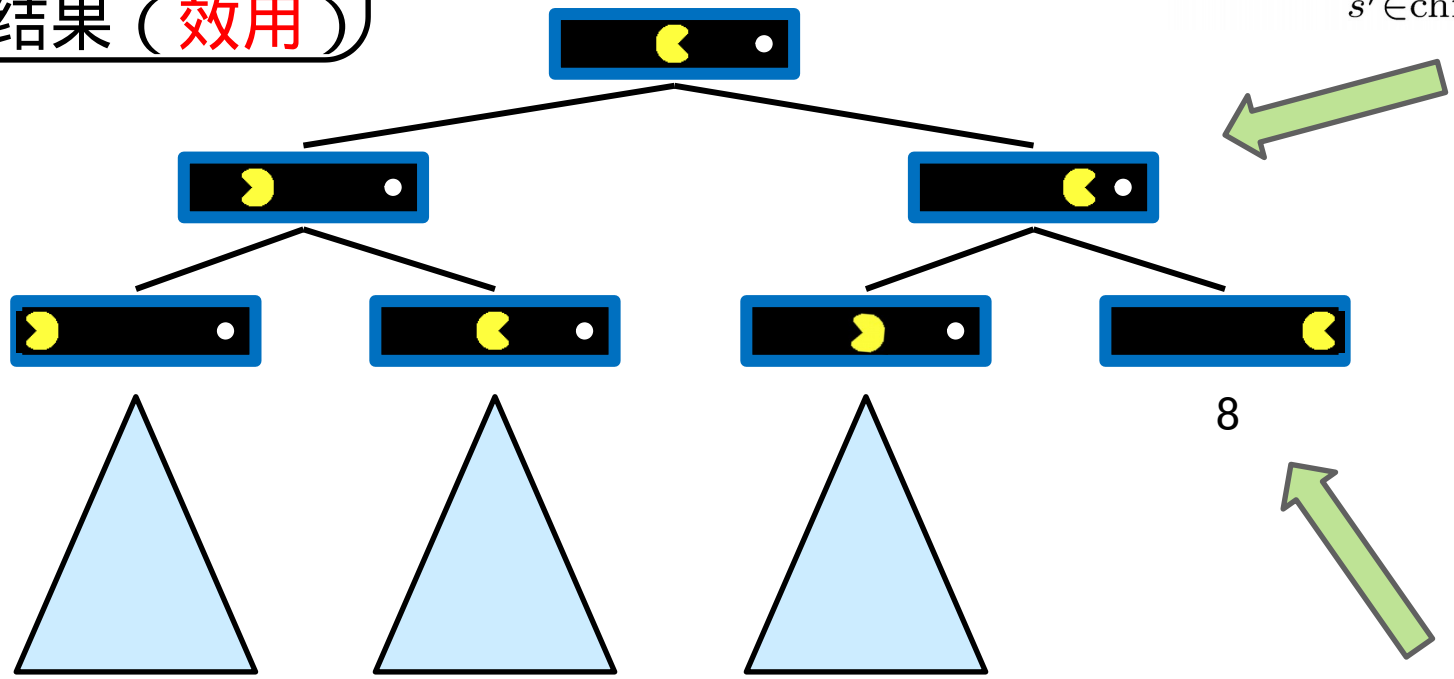
单 Agent 的搜索树

状态值 V(s):

可以得到的最好的结果（效用）

非终止状态：

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

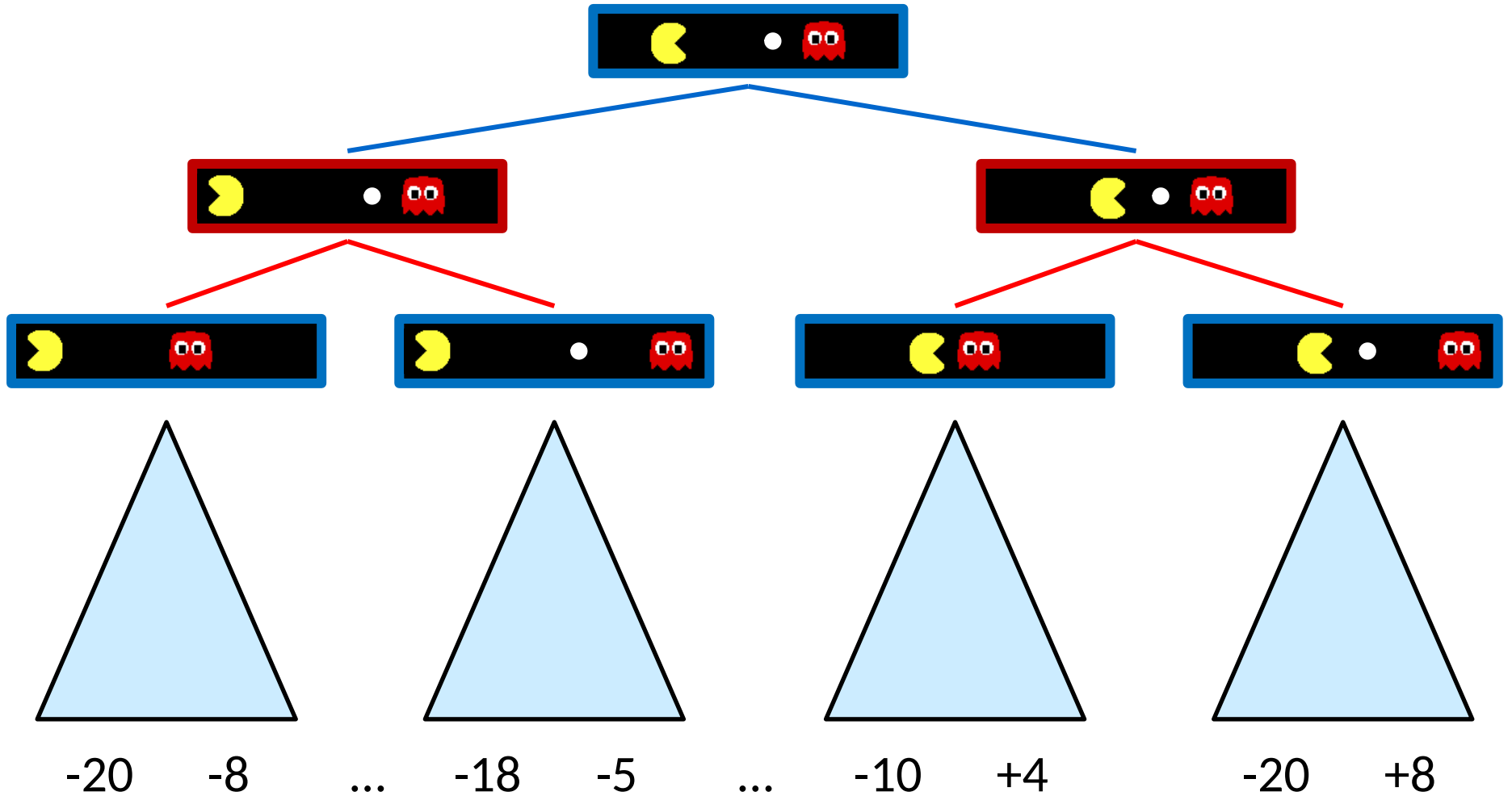


终止状态：

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

搜索目标：找到初始状态到终止状态的最优解路径

对抗博弈树



双方都是理性的 Agent

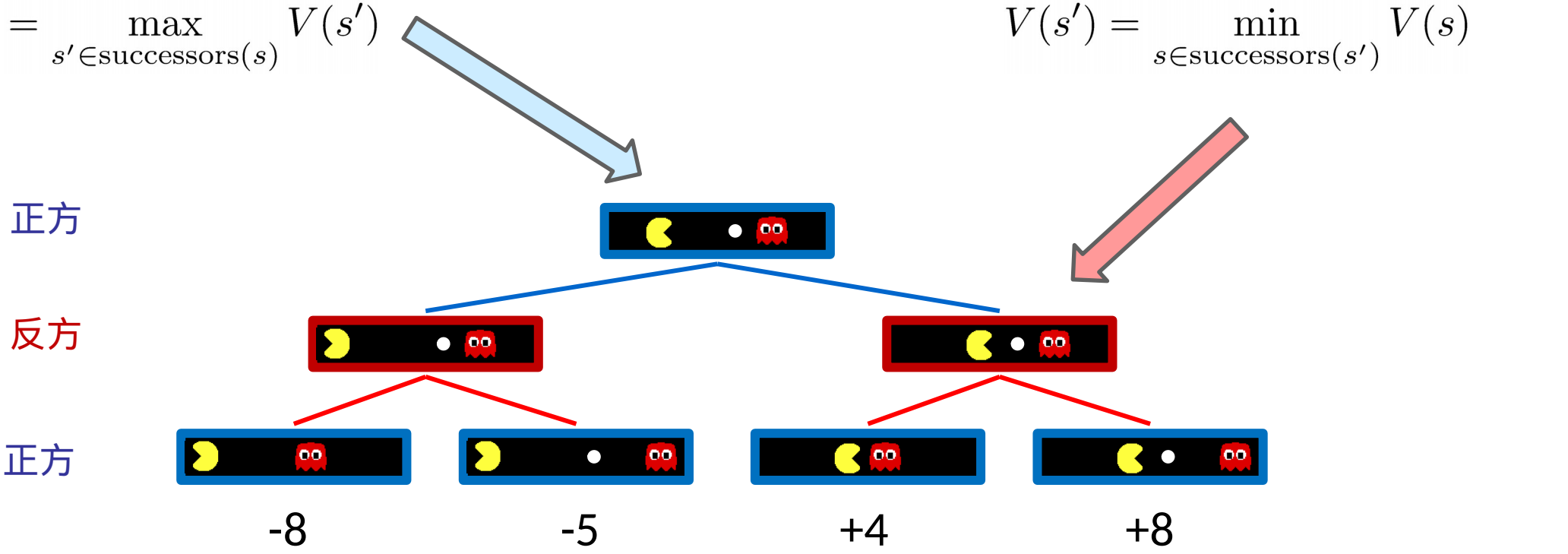
Minimax Values

正方的状态值：

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

反方的状态值：

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



终止状态： 正方的效用值 $V(s) = \text{known}$

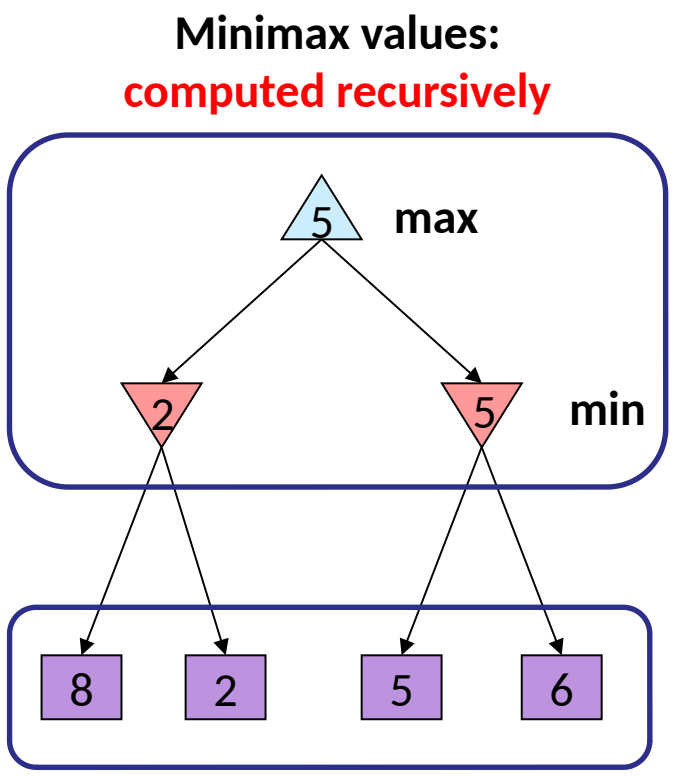
搜索目标：从初始状态出发，找到到达终止状态的最优解路径

对抗搜索： 对抗理性（最优）对手可实现最佳效用

对抗搜索 (Minimax)

- 确定性的零和博弈：
  - Tic-tac-toe, chess, checkers
  - One player maximizes result
  - The other minimizes result

- Minimax 搜索：
  - 状态空间搜索树
  - Players 交替轮流
  - 计算每个结点的 minimax value: 对抗理性（最优）对手可实现的最佳效用

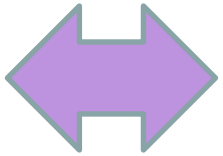


Minimax Implementation

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

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

正方的状态值 :



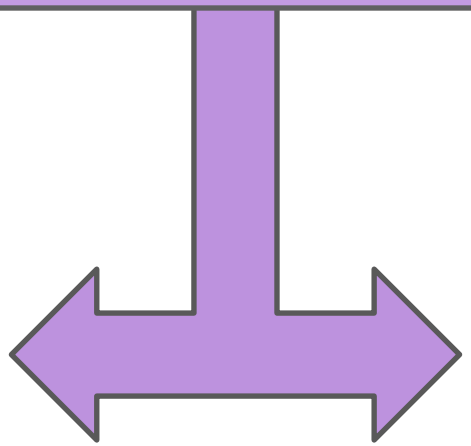
```
def min-value(state):
    initialize v = +∞
for each successor of
    state:
        v = min(v, max-value(successor))
    return v
```

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

反方的状态值 :

Minimax Implementation (Dispatch)

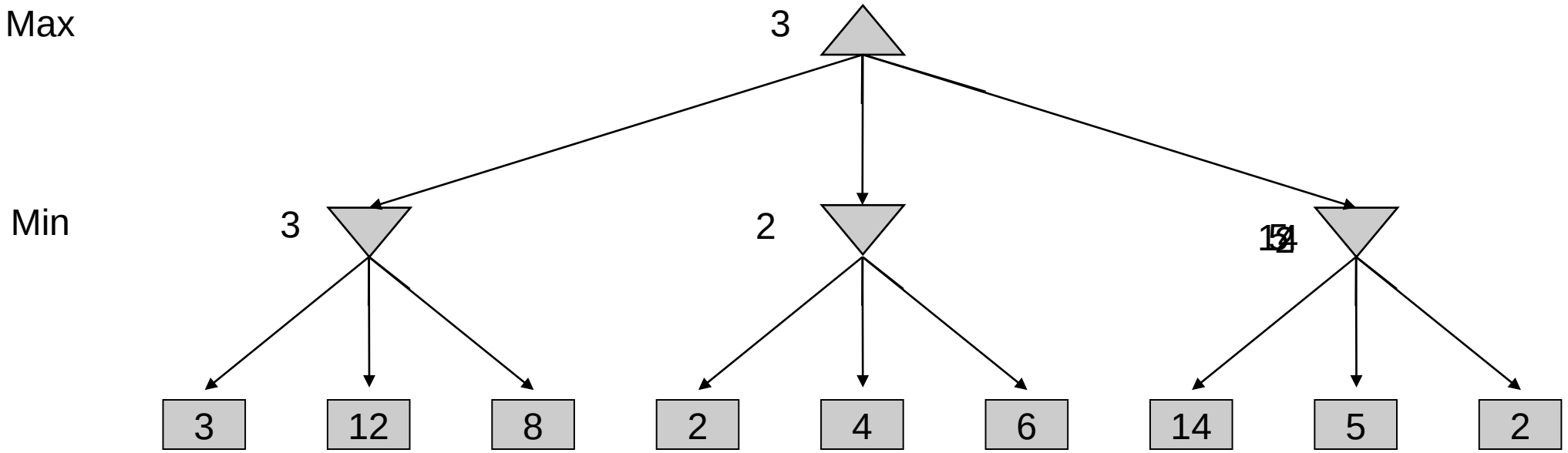
```
def value(state):
    if the state is a terminal state: return the state's utility
    if the next agent is MAX: return max-value(state)
    if the next agent is MIN: return min-value(state)
```



```
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
```

Minimax Example

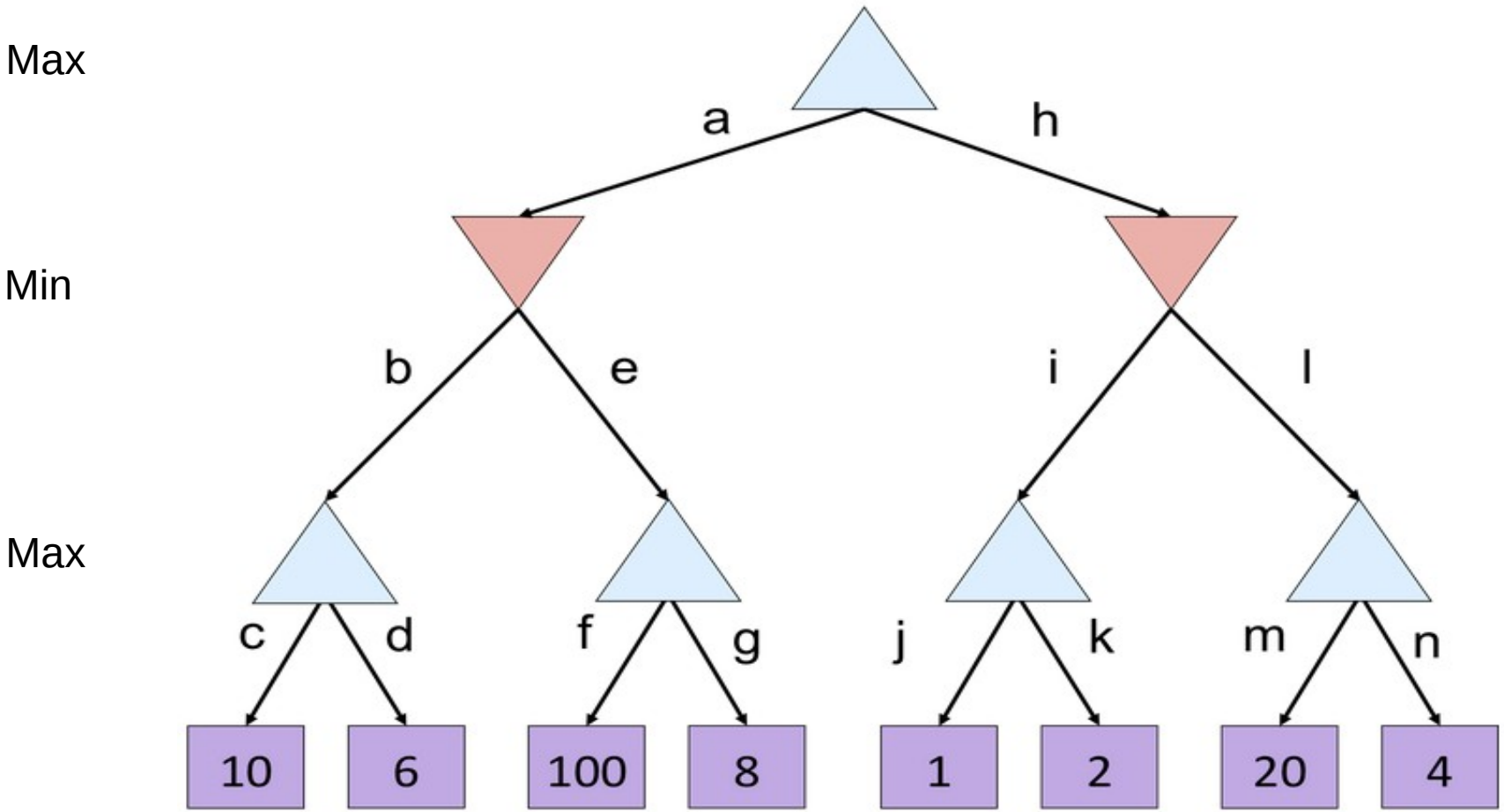


思考题：

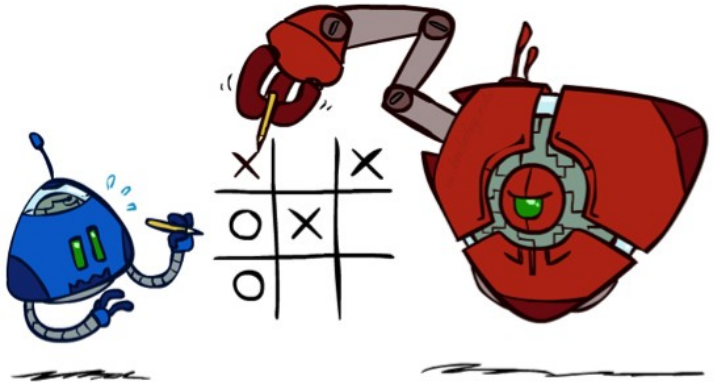
Minimax 搜索过程，类似于树的？搜索策略

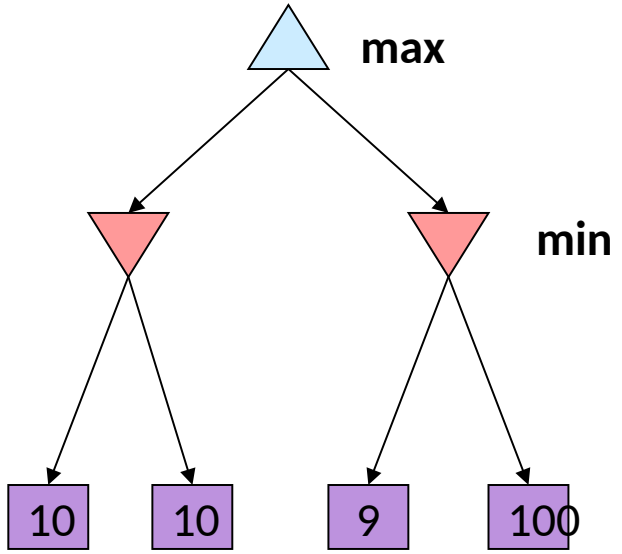
Minimax 课堂练习

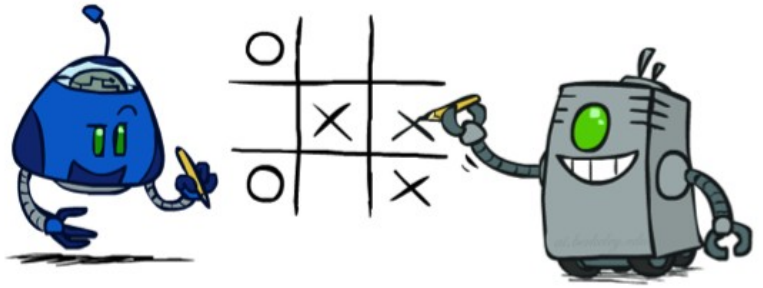
按从左到右的顺序搜索，使用极小极大算法，标明各状态的 Utility 值



Minimax Properties





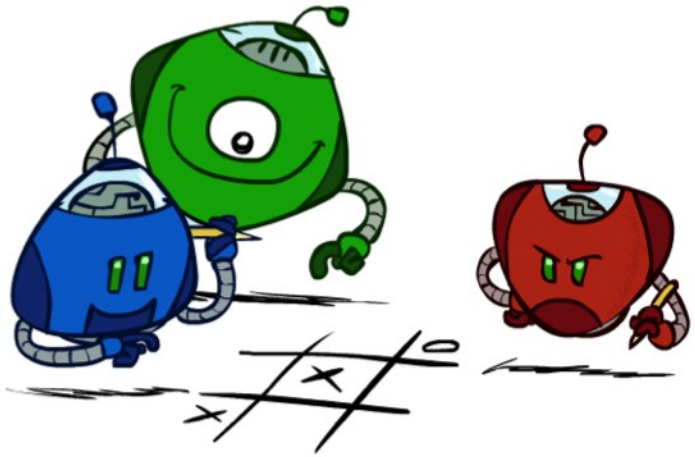


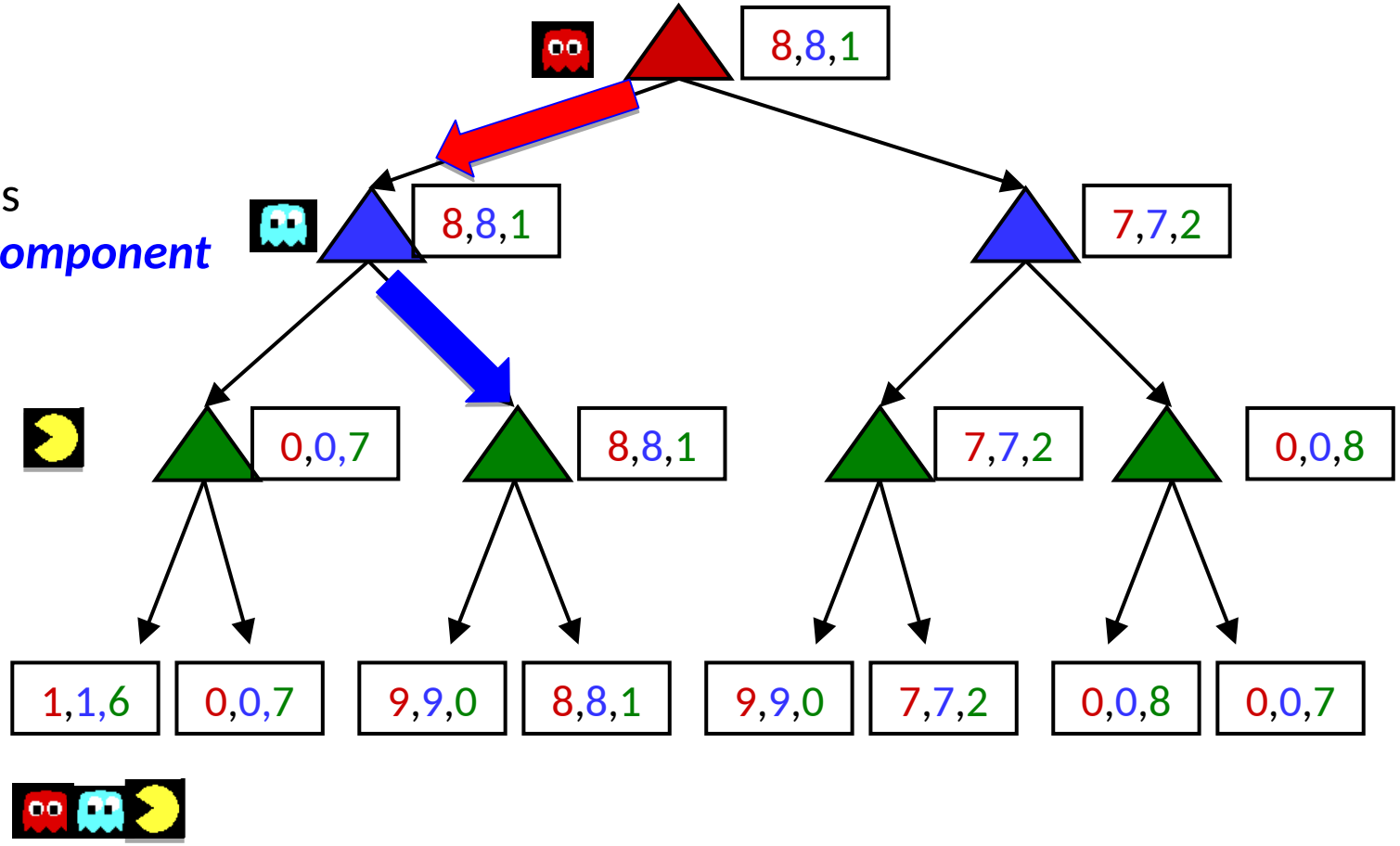
Optimal against a perfect player. Otherwise?

Generalized minimax

- What if the game is not zero-sum, or has multiple players?

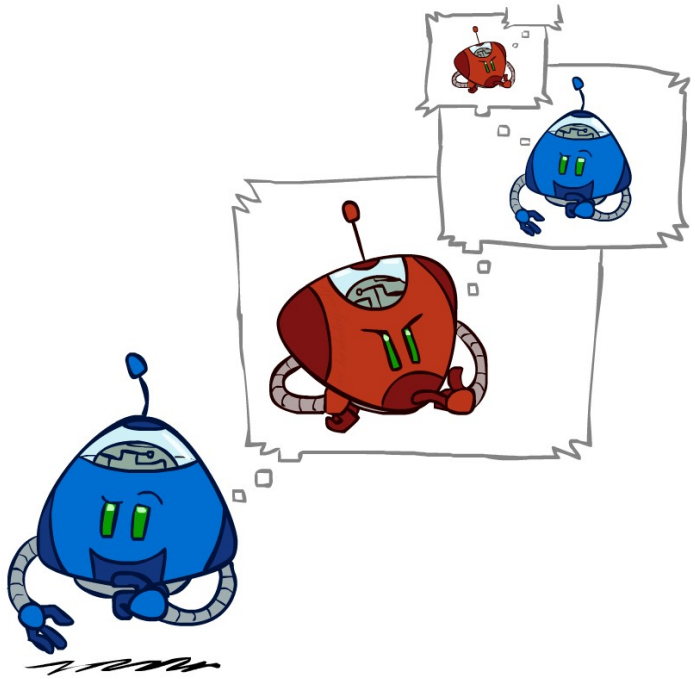
- Generalization of minimax:
  - Terminals have ***utility tuples***
  - Node values are also utility tuples
  - ***Each player maximizes its own component***
  - Can give rise to cooperation and competition dynamically…





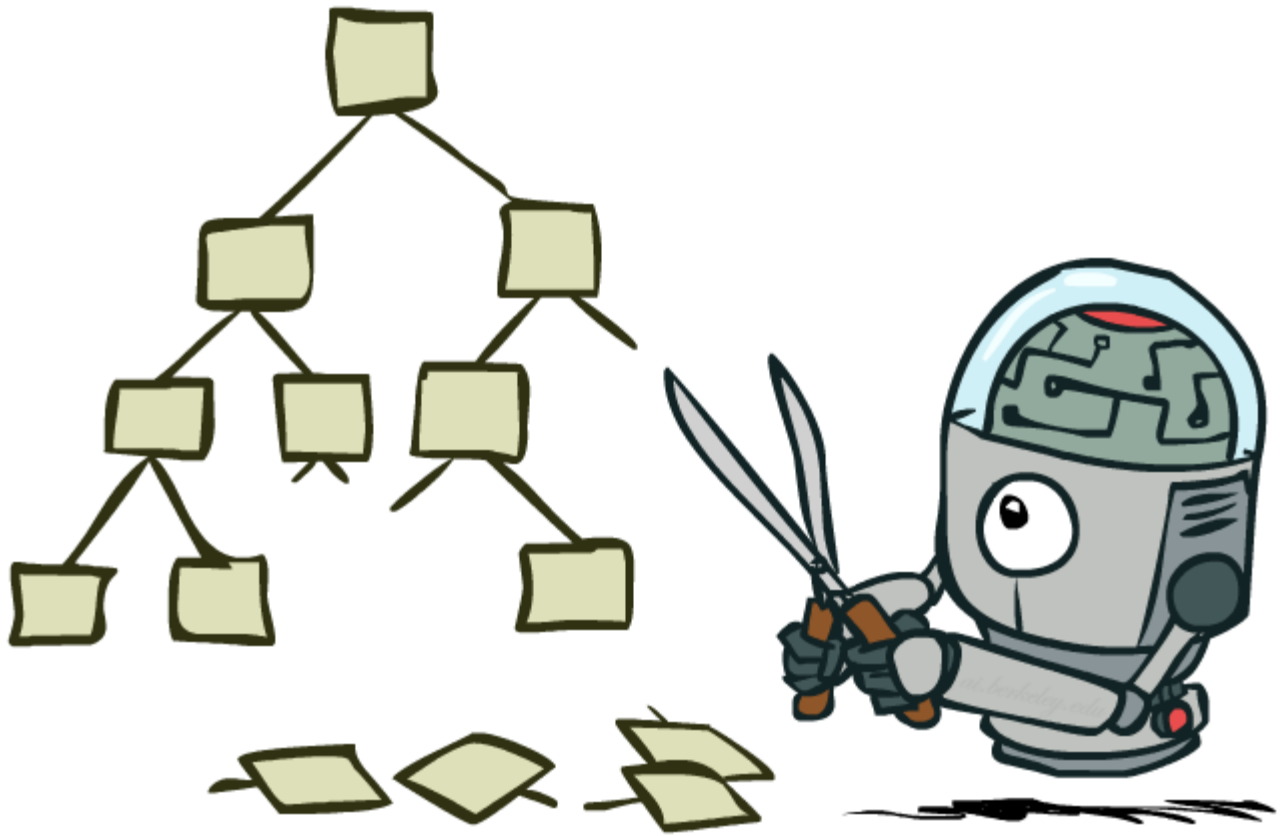
极小极大算法性能

- 完备性？Yes (if tree is finite)
- 最优性？Yes (against an optimal opponent)
- 时间复杂度？$O(b^m)$ (b: 分支因子 ; m: 最大树的深度)
- 空间复杂度？O(bm) (depth-first exploration)



- Example: 象棋 $b \approx 35, m \approx 100$
  - 不可能精确求解
  - 问题：是否有必要探索整棵树？

博弈树剪枝 Pruning



第六周作业： 5.9abcd 井字棋

5.9 本题以井字棋（圈与十字游戏）为例练习博弈中的基本概念。定义 X_n 为恰好有 n 个 X 而没有 O 的行、列或者对角线的数目。同样 O_n 为正好有 n 个 O 的行、列或者对角线的数目。效用函数给 $X_3 = 1$ 的棋局+1，给 $O_3 = 1$ 的棋局–1。所有其他终止状态效用值为 0。对于非终止状态，使用线性的评估函数定义为 $Eval(s) = 3X_2(s) + X_1(s) - (3O_2(s) + O_1(s))$。

a. 估算可能的井字棋局数。

b. 考虑对称性，给出从空棋盘开始的深度为 2 的完整博弈树（即，在棋盘上一个 X 一个 O 的棋局）。

c. 标出深度为 2 的棋局的评估函数值。

d. 使用极小极大算法标出深度为 1 和 0 的棋局的倒推值，并根据这些值选出最佳的起始行棋。

无论胜负，均下完 9 个棋子，算一局

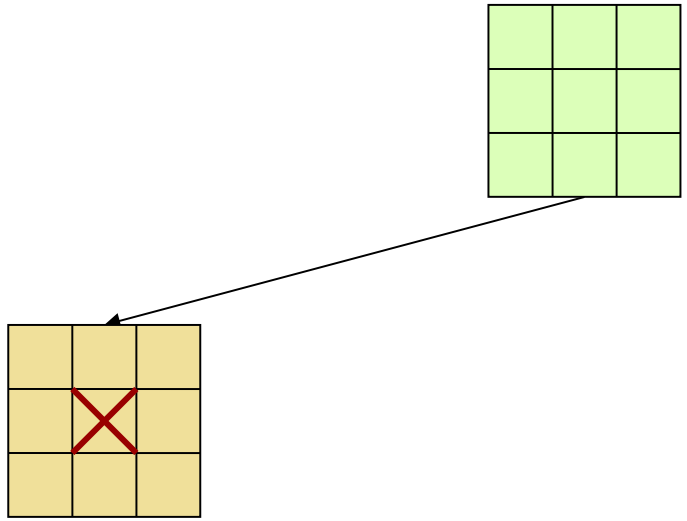
主要内容

- 5.1 Games theory (博弈论)
- 5.2 极大极小原理
- 5.3 α-β 剪枝
- 5.4 不完美的实时决策

α-β 剪枝

Max

Min

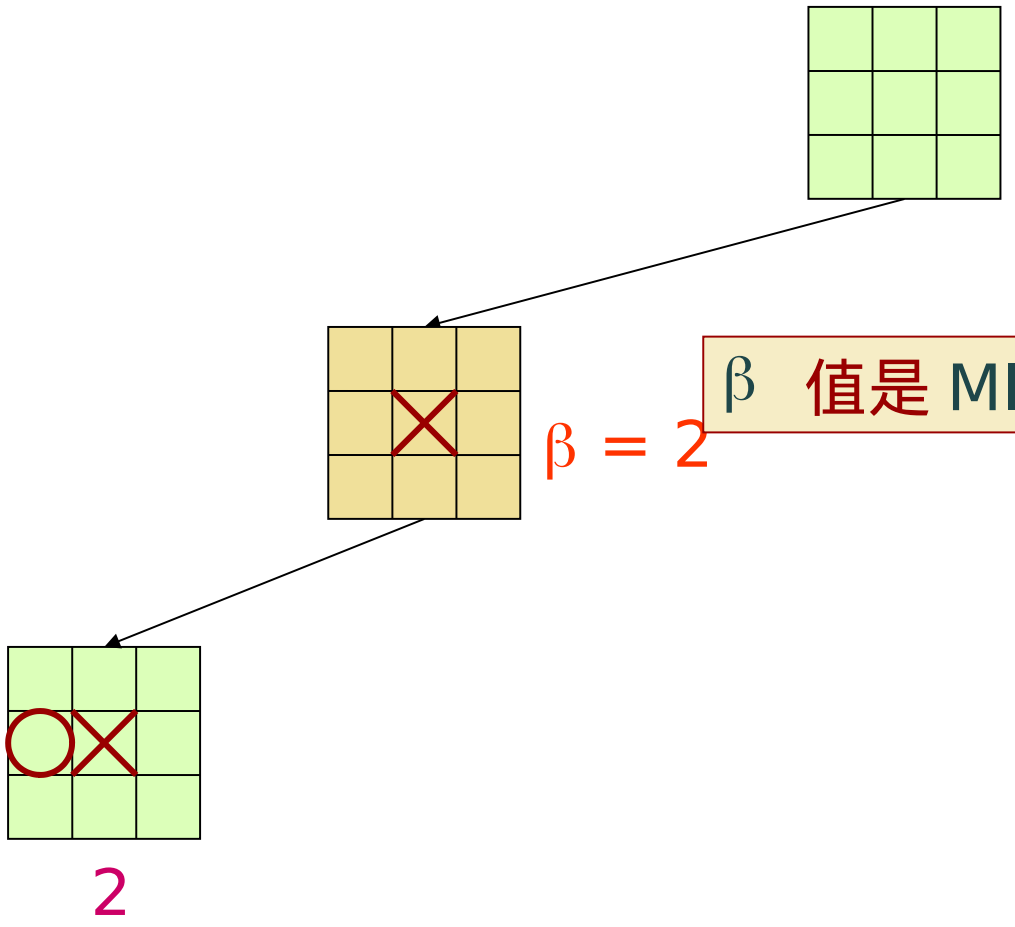


α-β 剪枝

Max

Min

Max



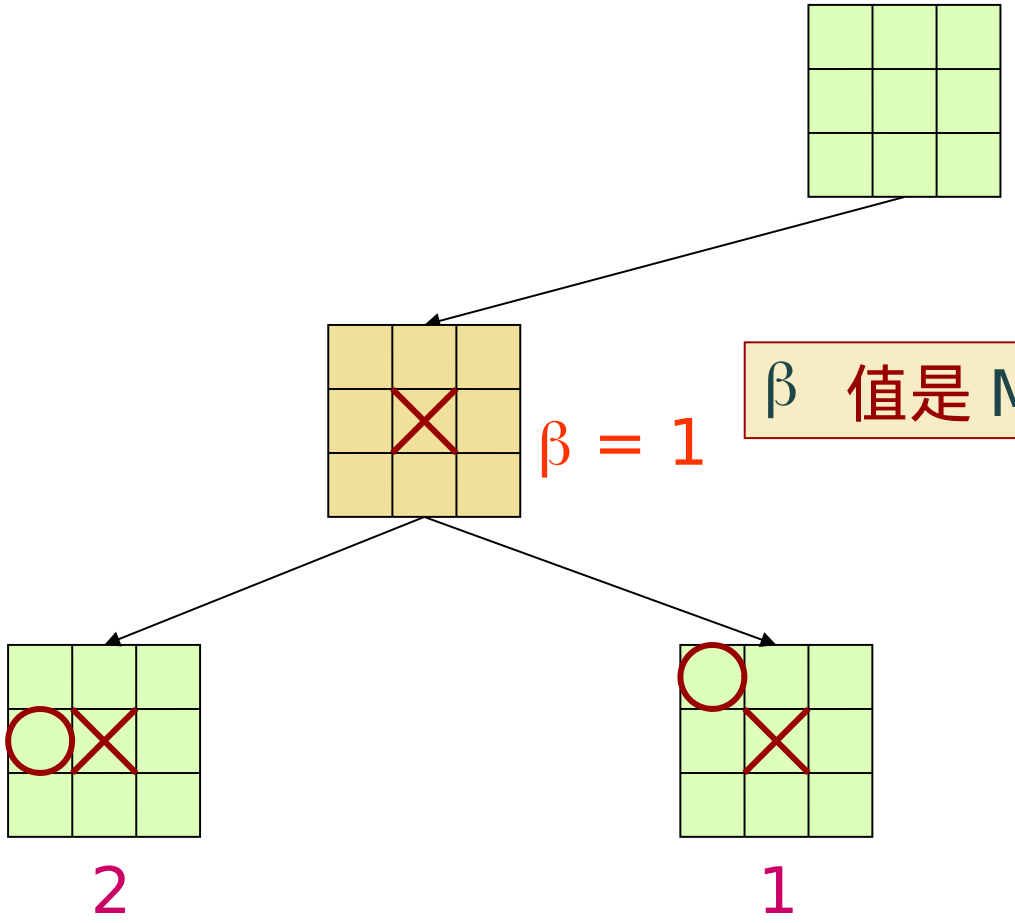
β 值是 MIN 节点的最佳（极小值）选择

α-β 剪枝

Max

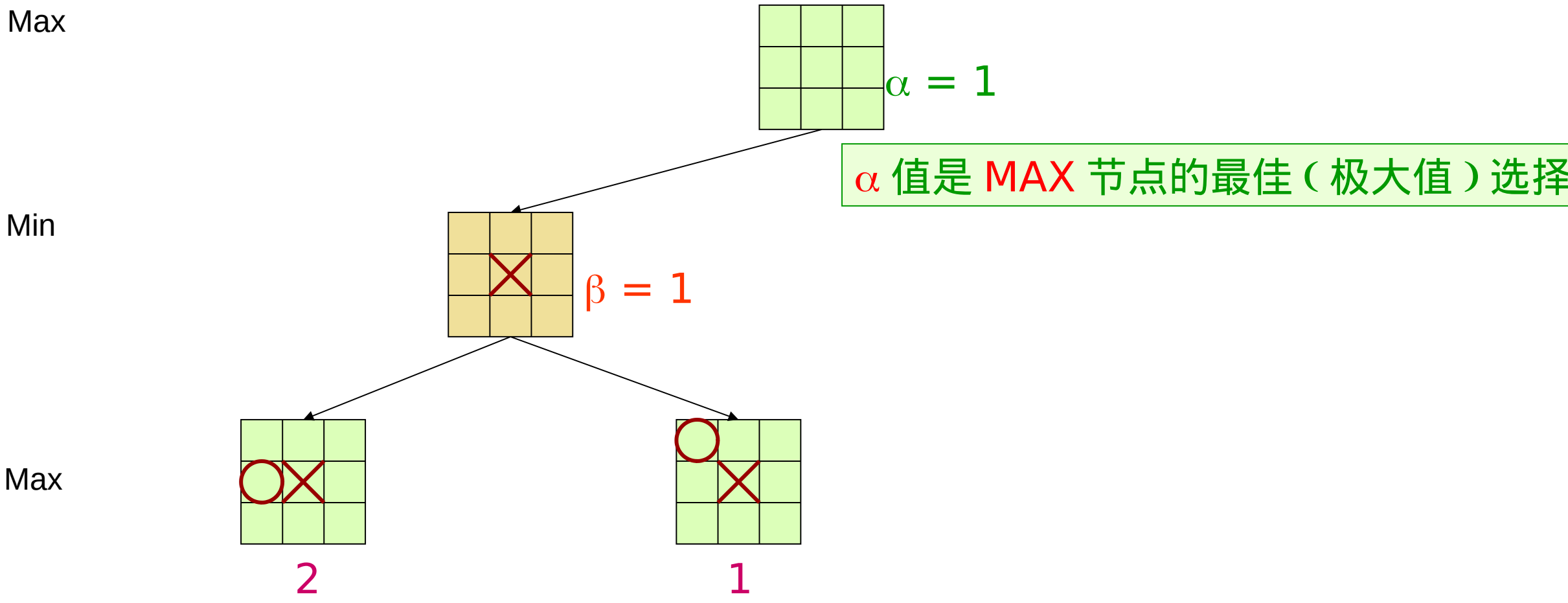
Min

Max

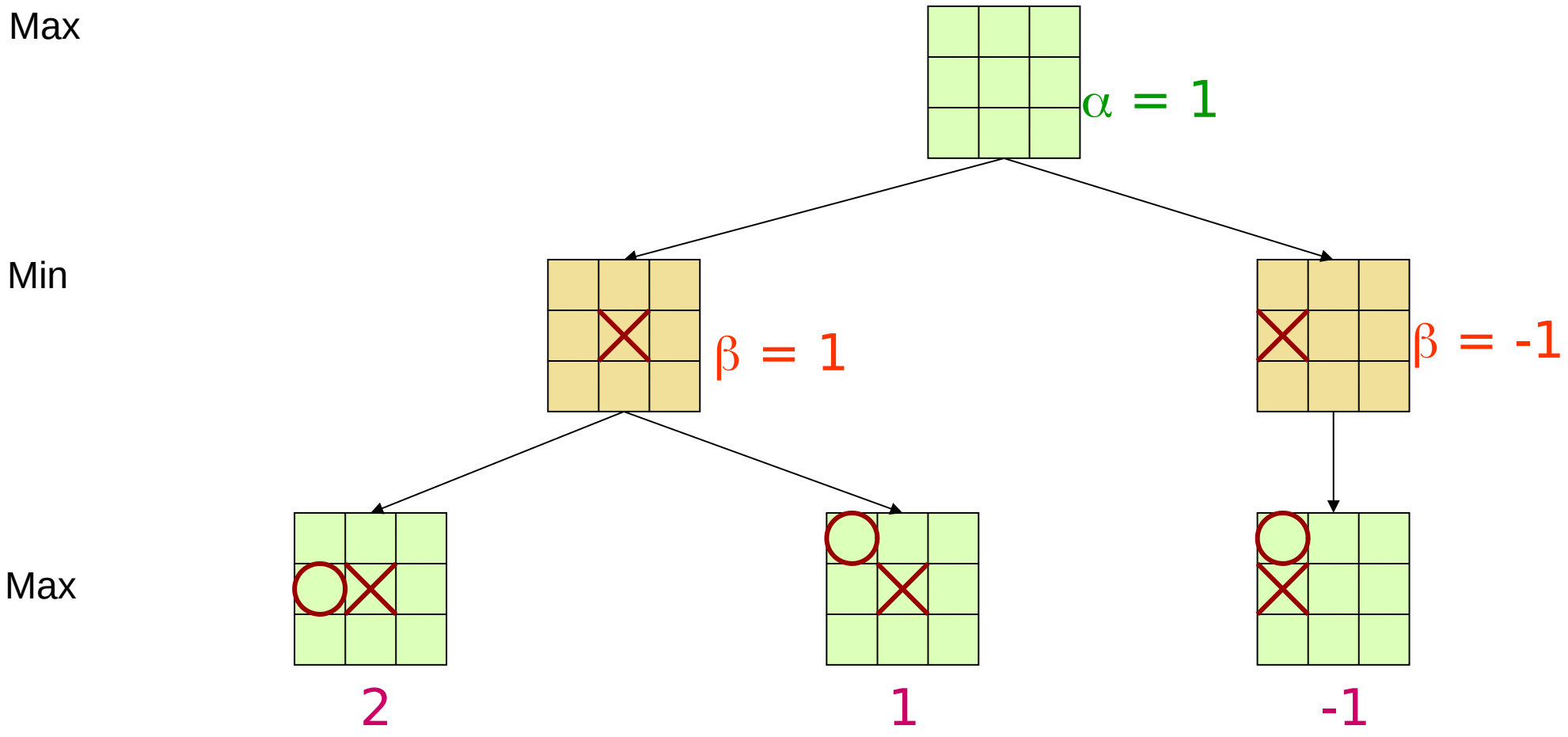


β 值是 MIN 节点的最佳（极小值）选择

α-β 剪枝



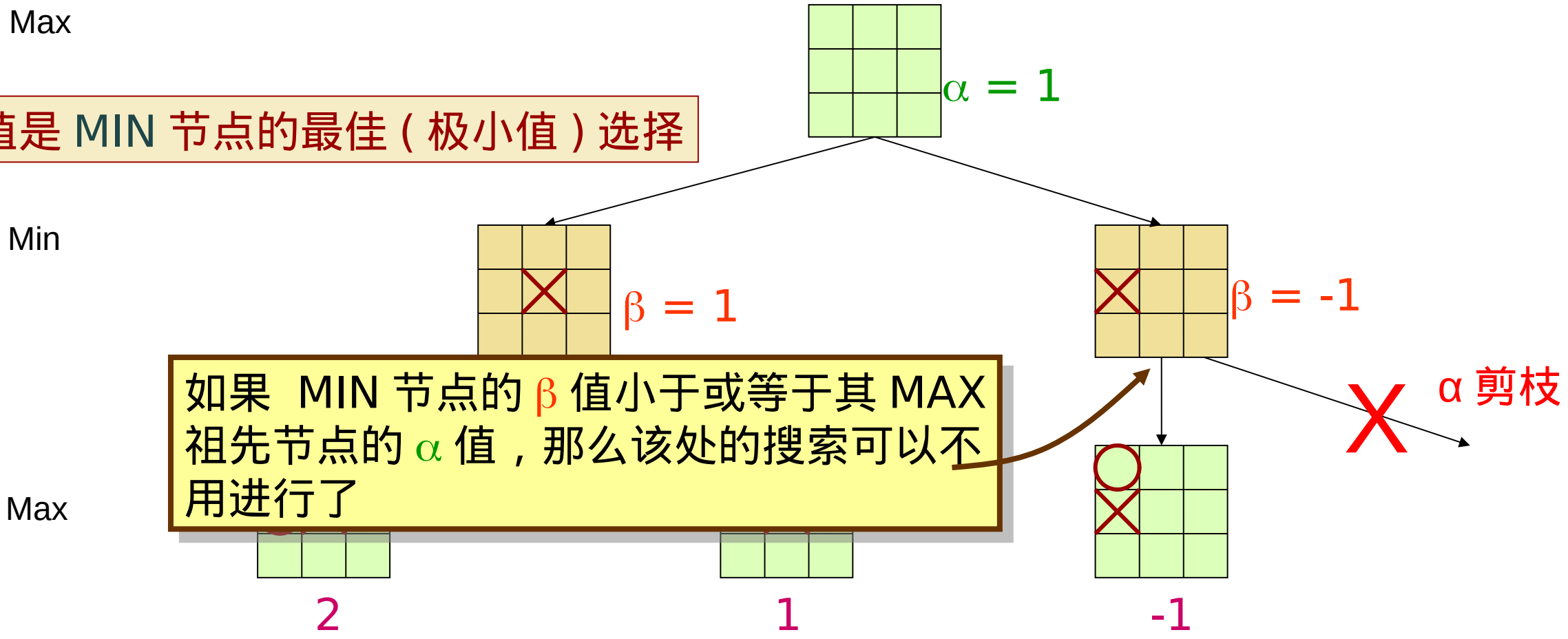
α-β 剪枝



α-β 剪枝

α 值是 MAX 节点的最佳（极大值）选择

β 值是 MIN 节点的最佳（极小值）选择



极小极大搜索是深度优先的，所以任何时候只需要考虑树中某条单一路径上的结点

主要内容

- 5.1 Games theory (博弈论)
- 5.2 极大极小原理
- 5.3 α-β 剪枝
- 5.4 不完美的实时决策

MAX (X)

MIN (O)

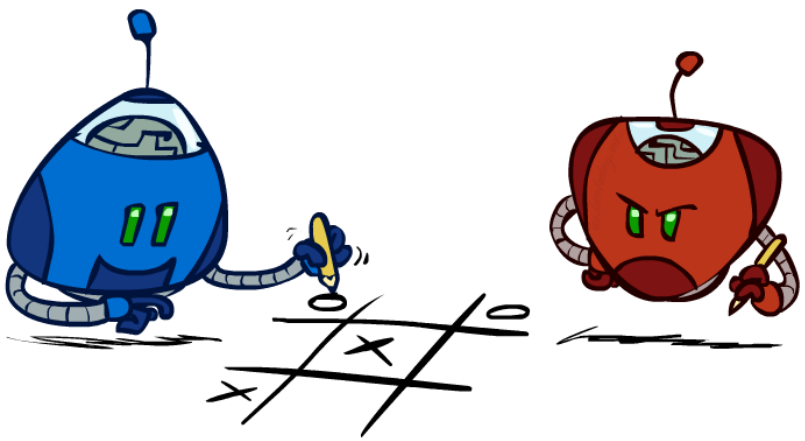
MAX (X)

MIN (O)

TERMINAL

Utility

−1 0 +1



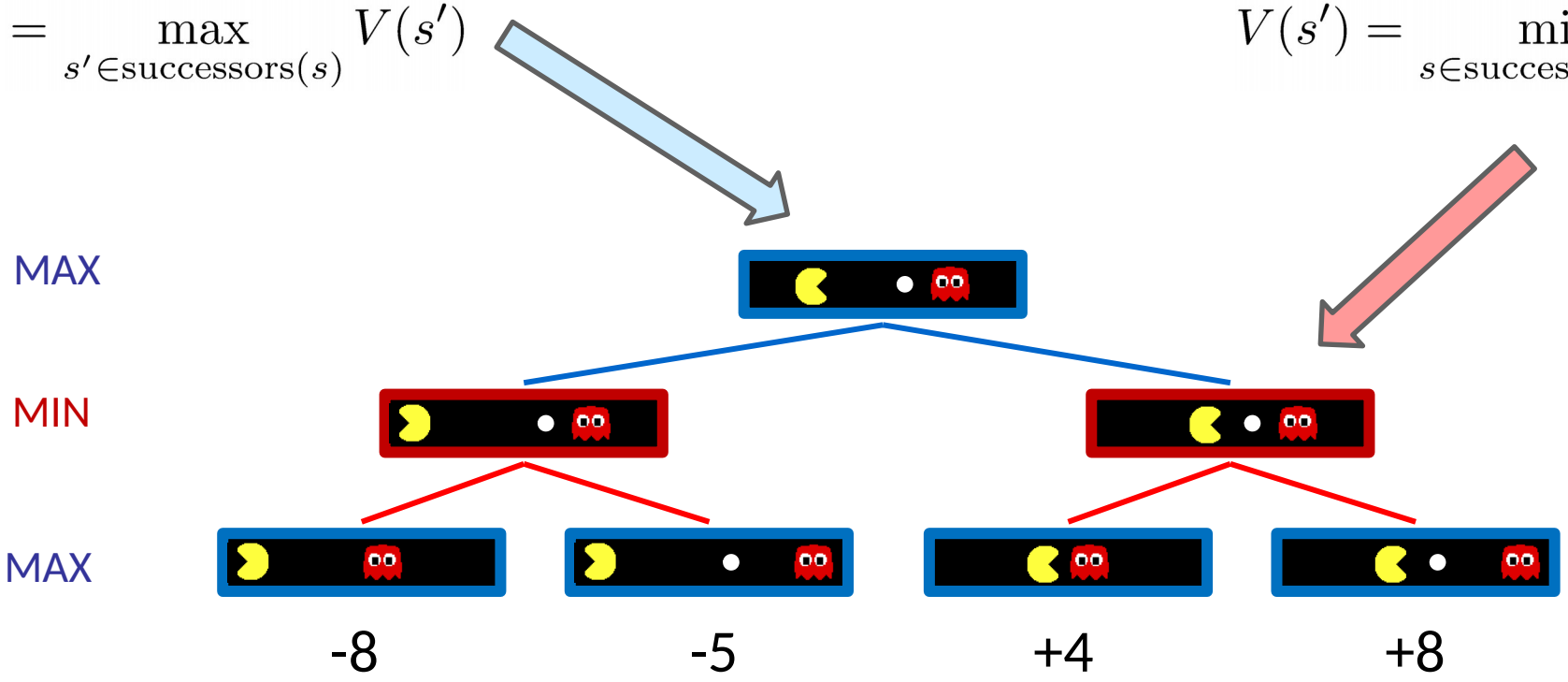
Minimax Values

MAX 的状态值：

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

MIN 的状态值：

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



终止状态： MAX 的效用值 $V(s) = \text{known}$

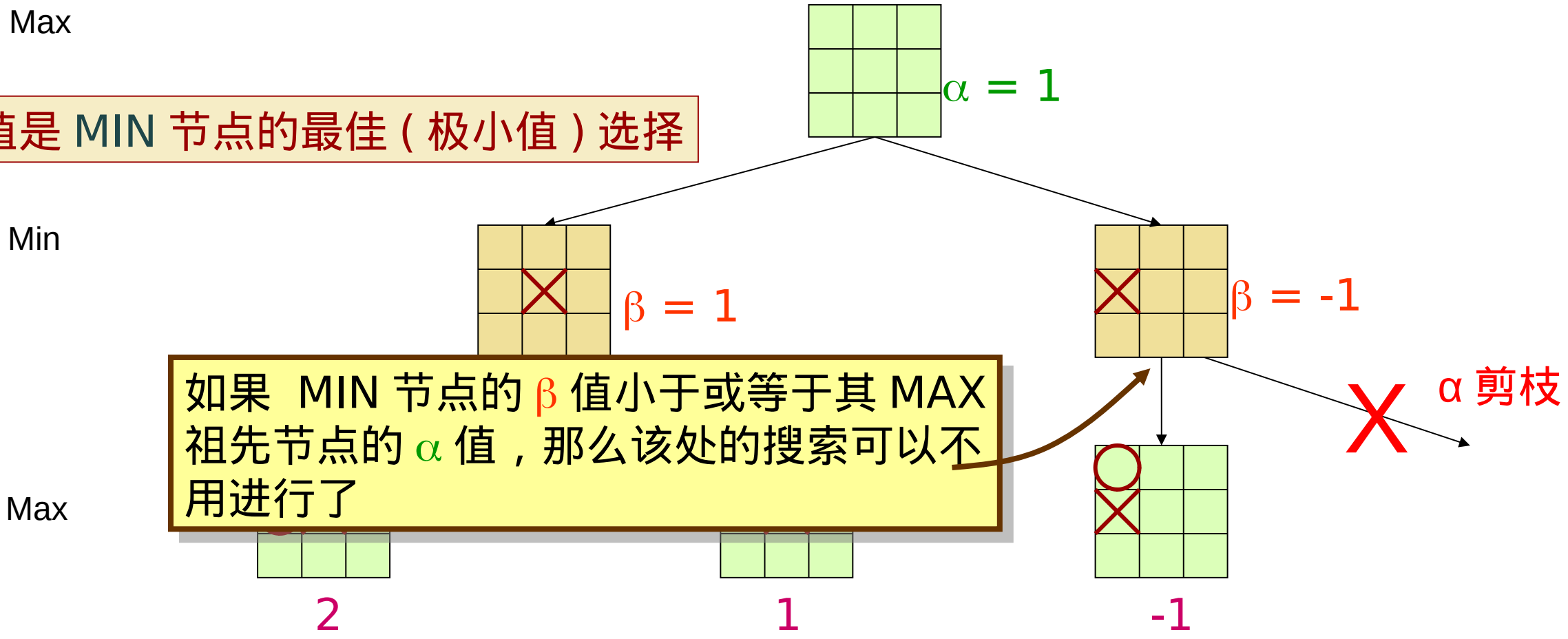
搜索目标：从初始状态出发，找到到达终止状态的最优解路径

对抗搜索： 对抗理性（最优）对手可实现最佳效用

α-β 剪枝

α 值是 MAX 节点的最佳（极大值）选择

β 值是 MIN 节点的最佳（极小值）选择



极小极大搜索是深度优先的，所以任何时候只需要考虑树中某条单一路径上的结点

Alpha-Beta Implementation

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

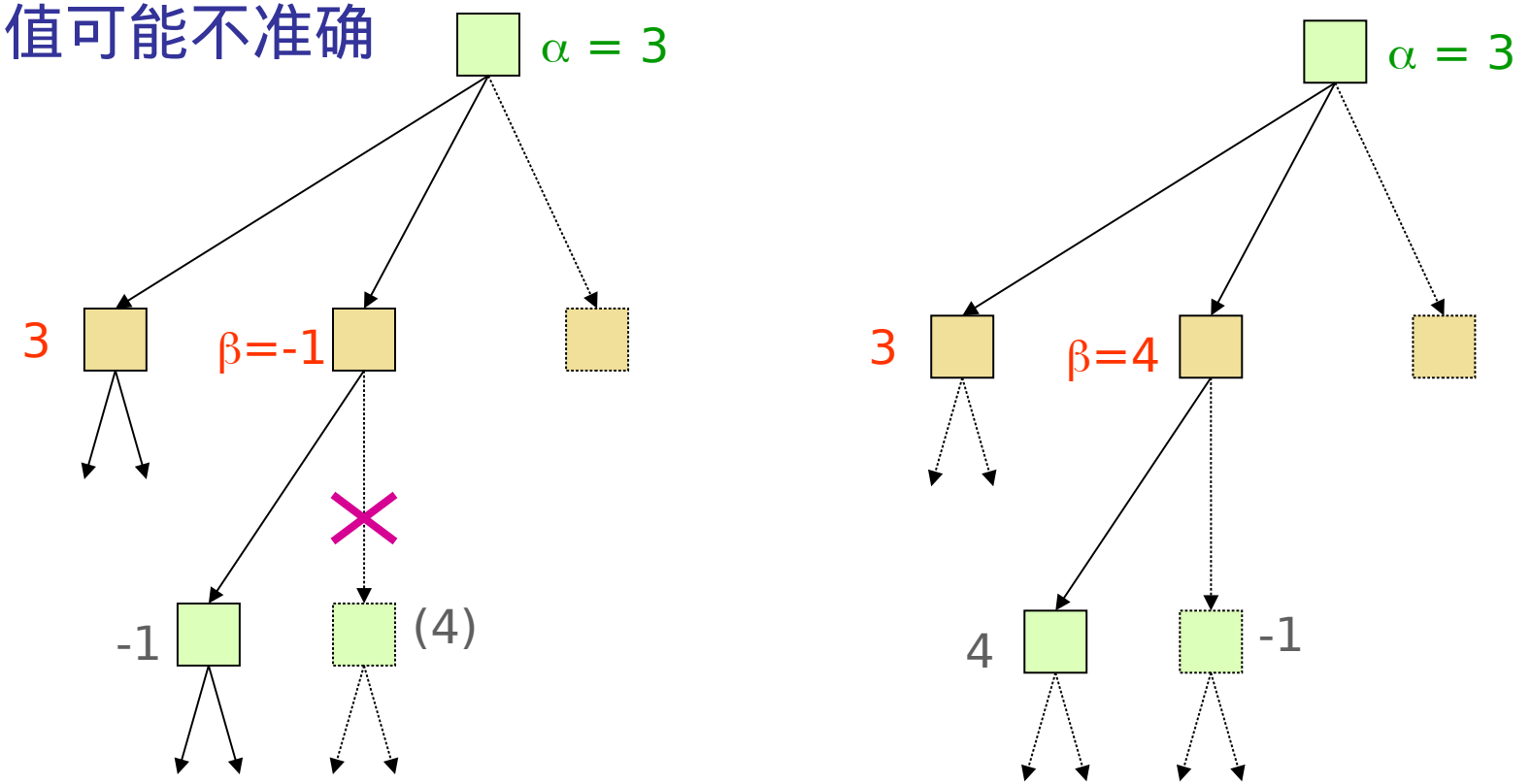
```
def max-value(state, α, β):
    initialize v = -∞
for each successor of
    state:
    v = max(v, value(successor, α, β)) if v
        ≥ β return v
        α = max(α, v)
    return v
```

```
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
```

α 剪枝

Alpha-Beta Pruning Properties

- 剪枝对为根结点 minimax 值的计算没有影响！
- 中间节点的值可能不准确

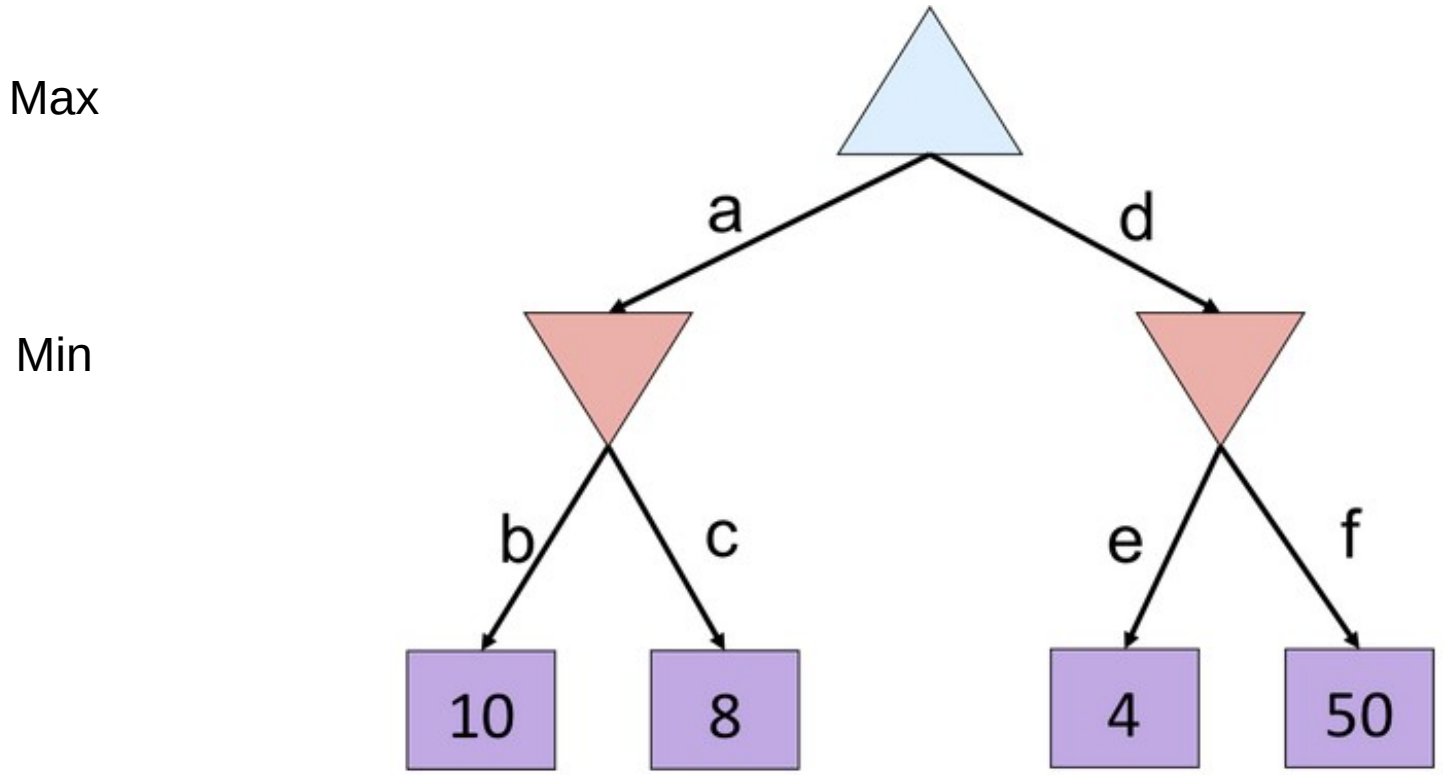


Alpha-Beta Quiz

α 值是 MAX 节点的最佳（极大值）选择

β 值是 MIN 节点的最佳（极小值）选择

按从左到右的顺序进行 α-β 剪枝搜索，标出树上状态的 utility 值，标明何处发生剪枝，属于 α 剪枝还是 β 剪枝。

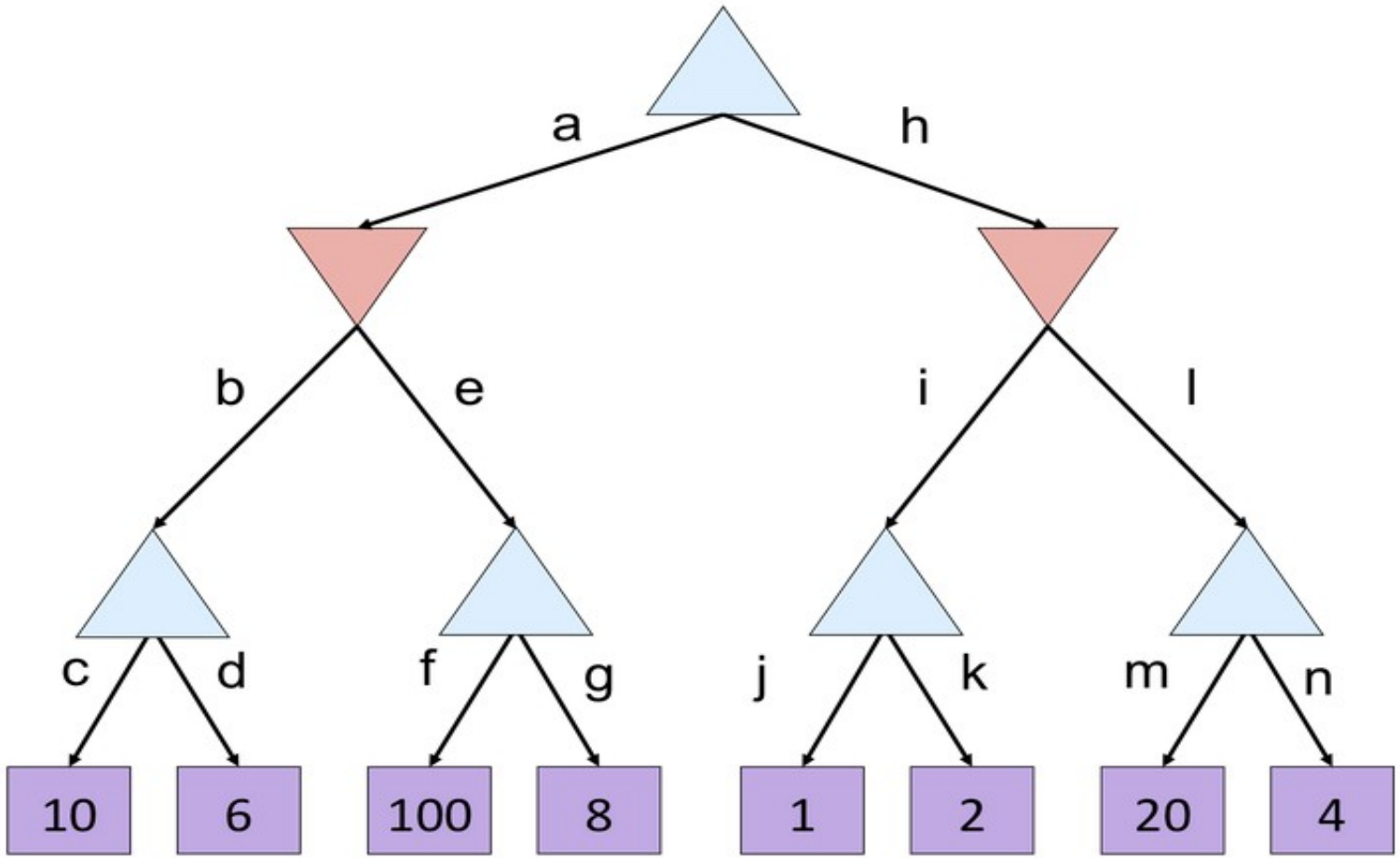


Alpha-Beta Quiz 2

α 值是 MAX 节点的最佳（极大值）选择

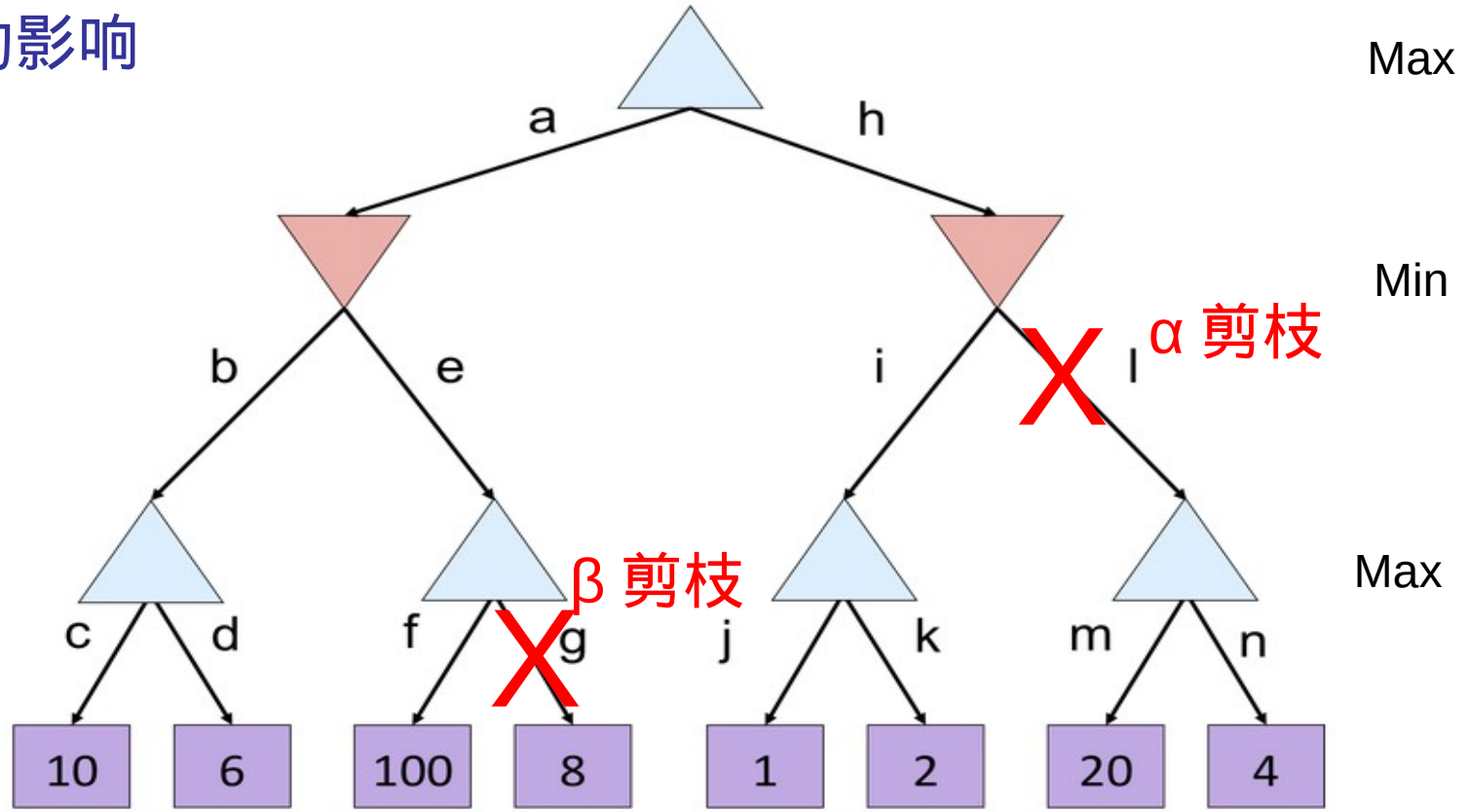
β 值是 MIN 节点的最佳（极小值）选择

按从左到右的顺序进行 α-β 剪枝搜索，标出树上状态的 utility 值，标明何处发生剪枝，属于 α 剪枝还是 β 剪枝。



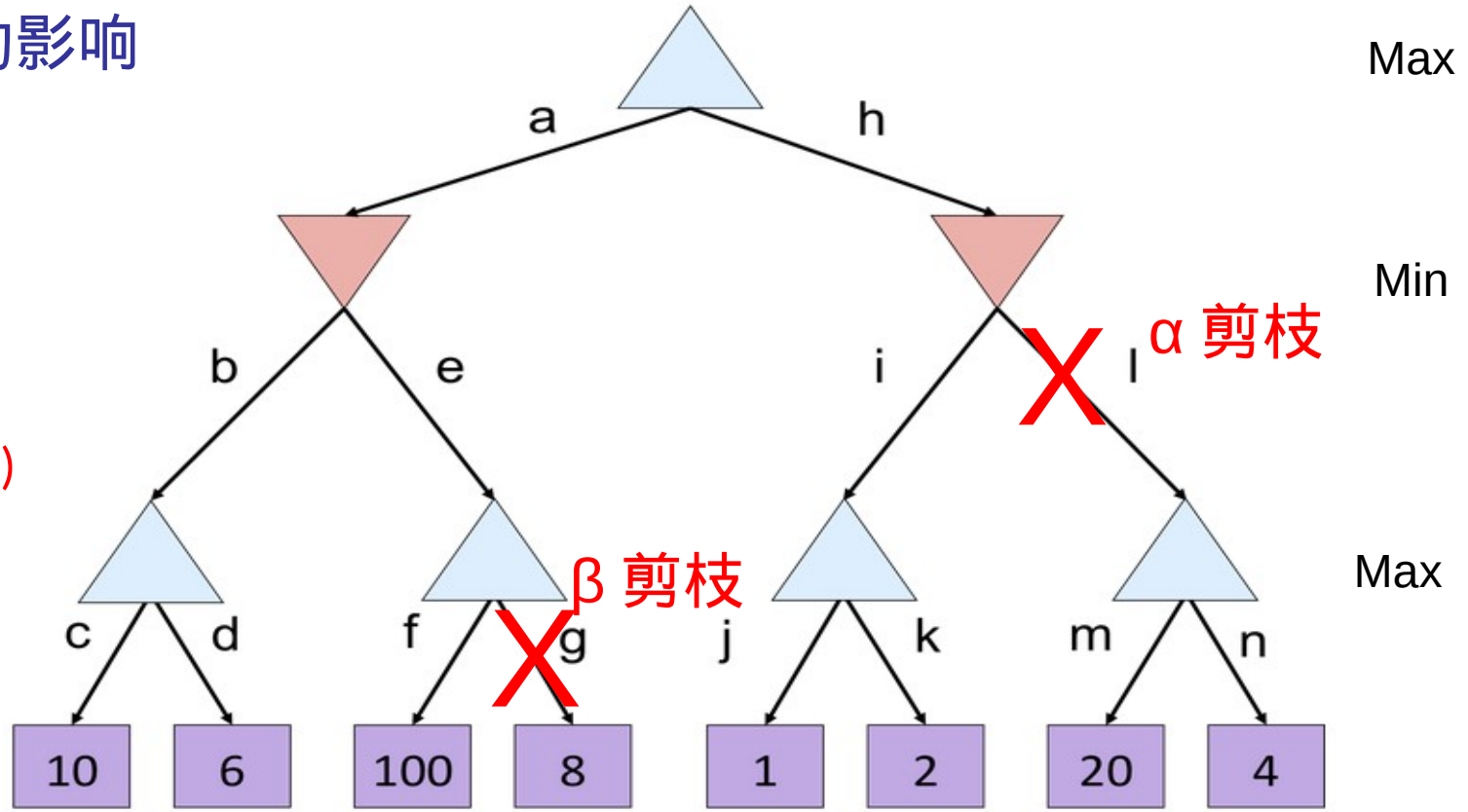
Alpha-Beta Pruning Properties

- Good children order 对剪枝的影响



Alpha-Beta Pruning Properties

- Good children order 对剪枝的影响
- With “perfect ordering”:
  - 时间复杂度可以降低至 $O(b^{m/2})$
  - 但是仍然不能用于象棋等

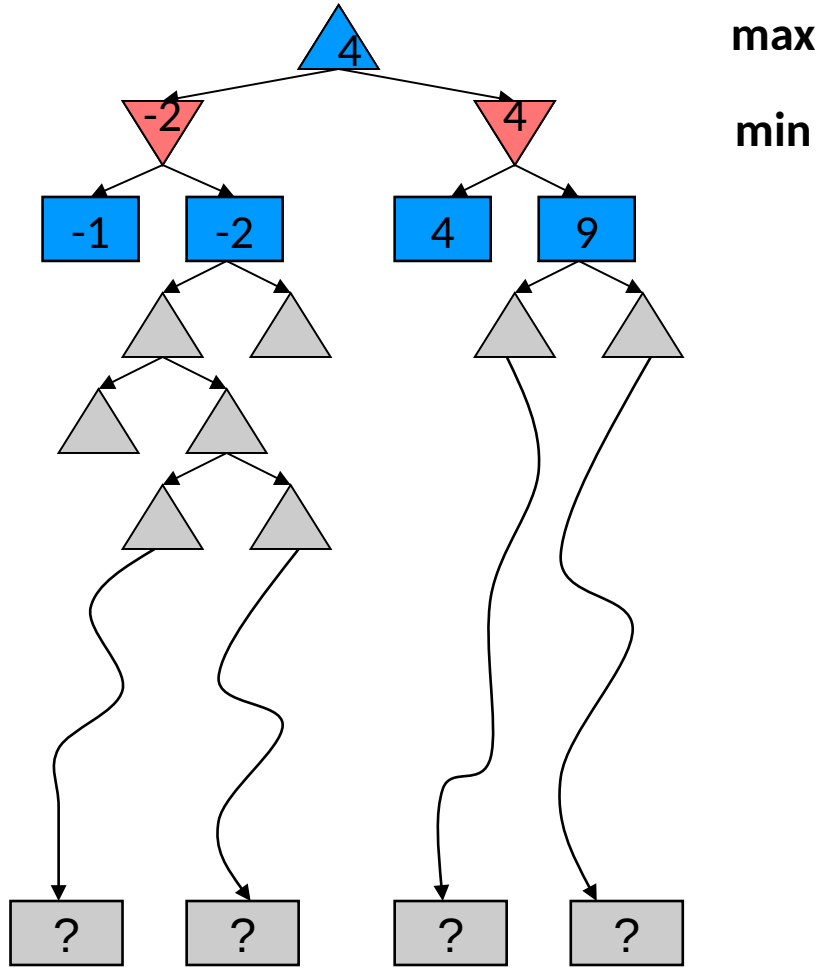


主要内容

- 5.1 Games theory (博弈论)
- 5.2 极大极小原理
- 5.3 α-β 剪枝
- 5.4 不完美的实时决策

资源受限

- Problem: 在真实游戏中，难以搜索到叶子结点！
- Solution: 深度受限搜索
  - 将搜索限制在给定深度内
  - 终止结点用效用值
  - 非终止结点用评估函数估计效用
- Example:
  - Suppose we have 100 seconds, can explore 10K nodes / sec
  - So can check 1M nodes per move
  - α-β reaches about depth 8 – decent chess program
- 无法保证最优决策
- 时间受限，可以结合迭代加深搜索算法



算法在棋类比赛中的应用

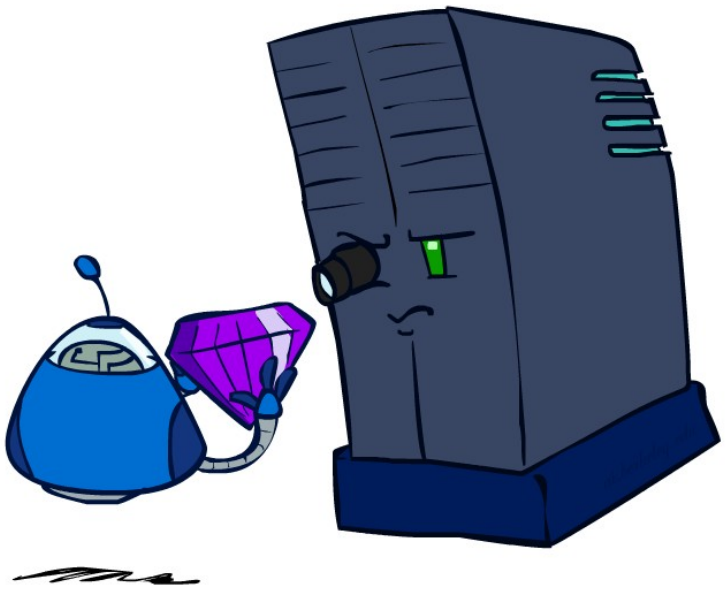
深蓝：下棋核心“算”，成功要素有三个：

1 ）经验，分析 70 万盘人类大师下过的棋局及全部的 5-6 只残局，总结成下棋的规则（考虑子力、棋子位置、王的安全性、布局节奏等）

2 ）算法，深蓝使用阿尔法 - 贝塔剪枝算法，速度很快。如果不剪枝依靠暴力计算，每步棋需要搜索 17 年

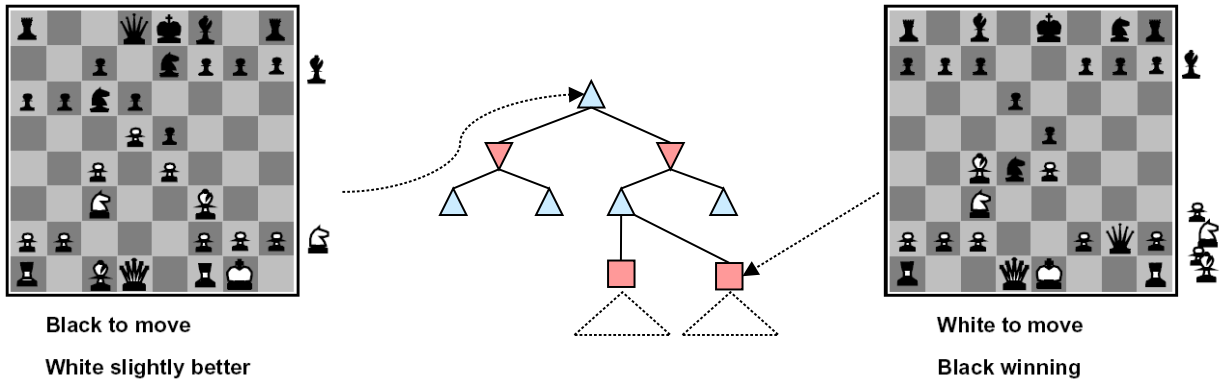
3 ）算力， RS/6000SP2 机器，每秒能够分析 2 亿

评估函数



评估函数

- 在深度受限搜索中，评估函数给非终止结点打分

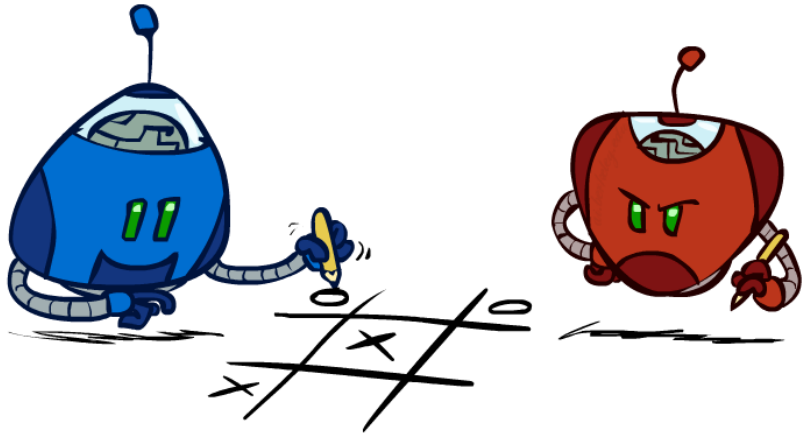


- 理想的函数：返回状态的真实的极小极大值
- 实践中：加权线性函数：

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

- e.g. 对于国际象棋，$f_1(s)$ = (num white queens – num black queens), etc.

井字棋的评估函数



- *MAX* 标记 **X** *MIN* 标记 O
- MAX 先移动 .

– 对状态 s 的评估函数 $e(s)$

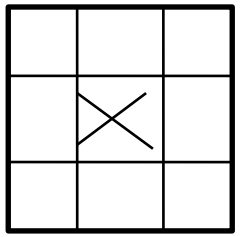
- If s is not a winning for either player, $e(s)$ = (no. of complete rows, columns, or diagonals that are still open for MAX) - (no. of complete rows, columns, or diagonals that are still open for MIN)
- If s is a win of *MAX*, $e(s) = \infty$
- If s is a win of *MIN*, $e(s) = -\infty$

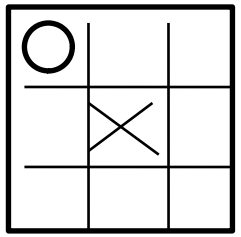
井字棋的评估函数

– 对状态 s 的评估函数 $e(s)$

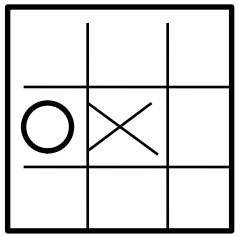
- If s is not a winning for either player,

  $e(s)$ = (no. of complete rows, columns, or diagonals that are still open for *MAX*) - (no. of complete rows, columns, or diagonals that are still open for *MIN*)





e(s)=5-4=1



e(s)=6-4=2

谢谢！