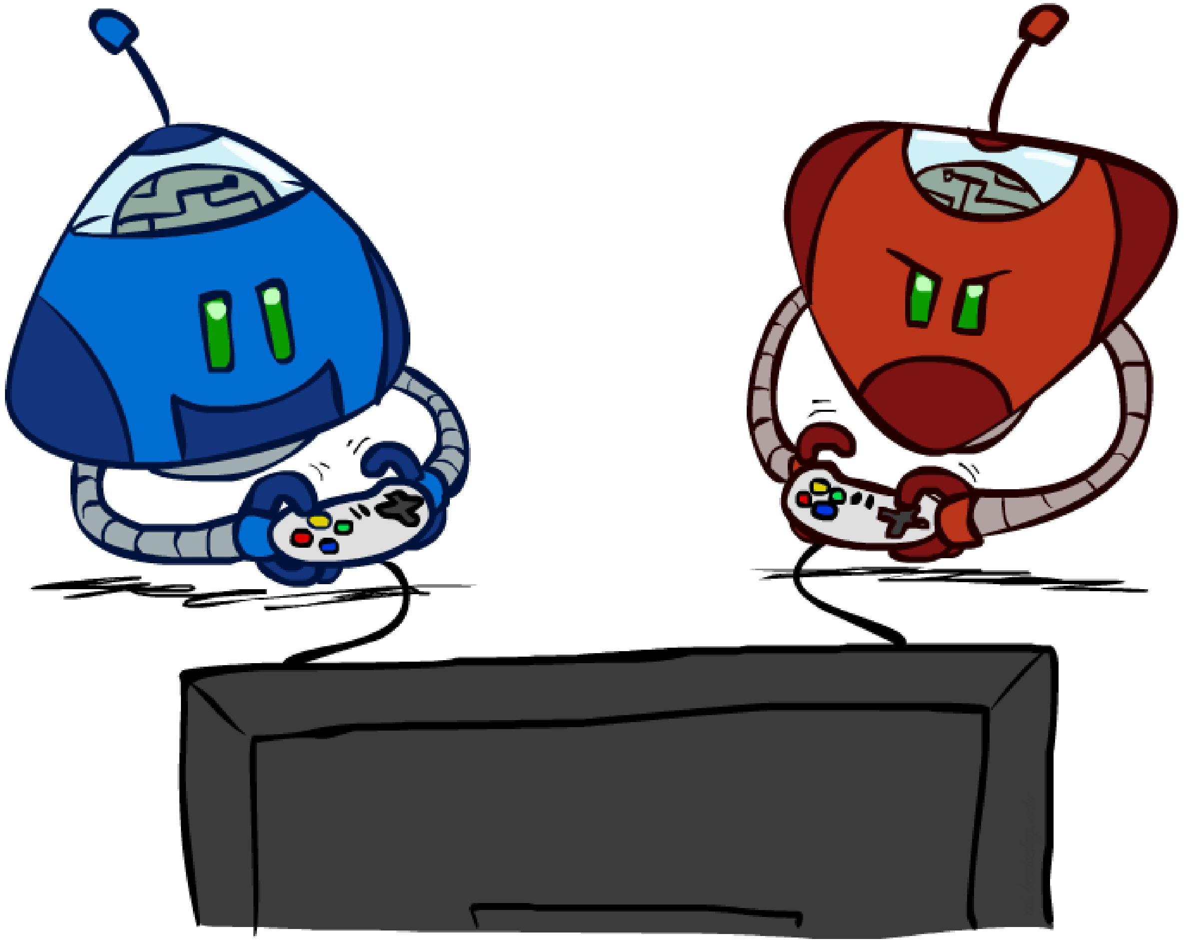
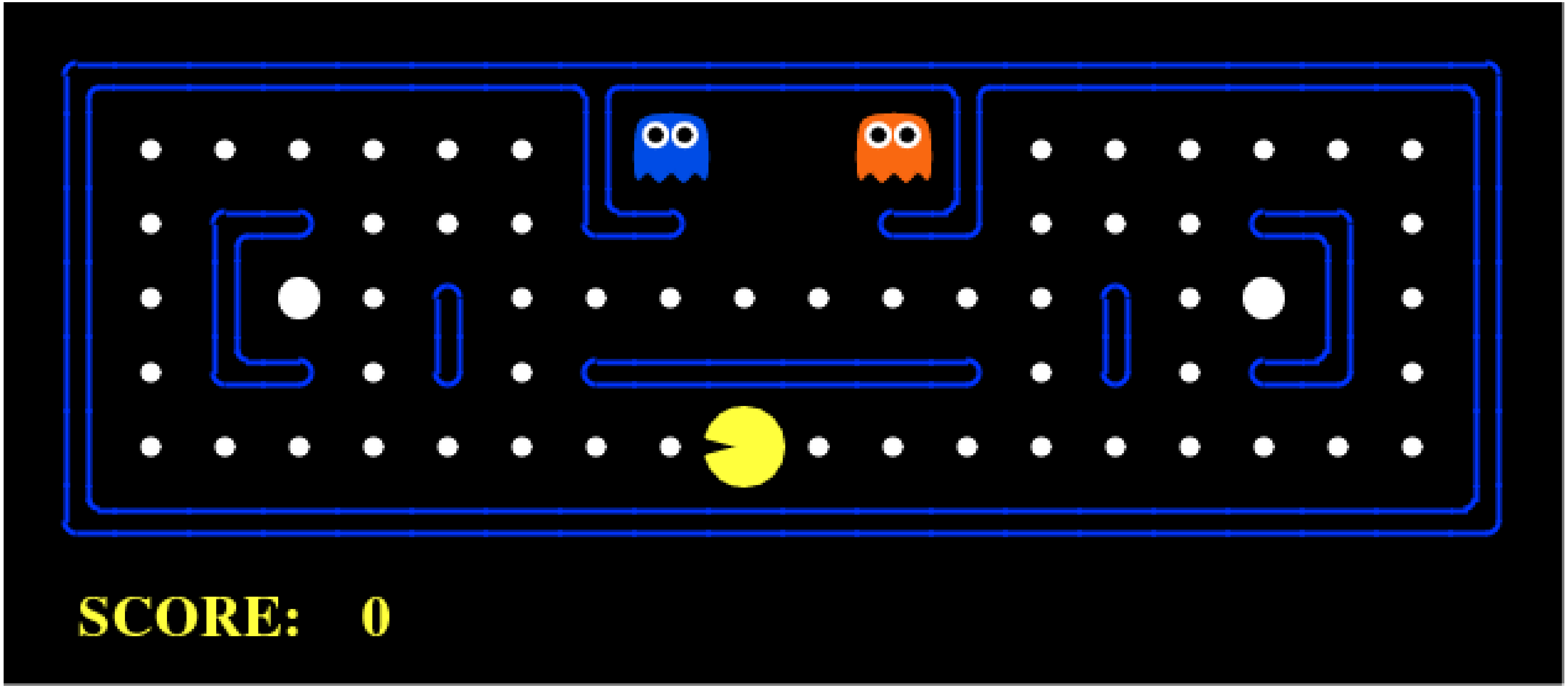
第五章　对抗搜索



ai.berkeley.edu

Multi-Agent Pacman



主要内容

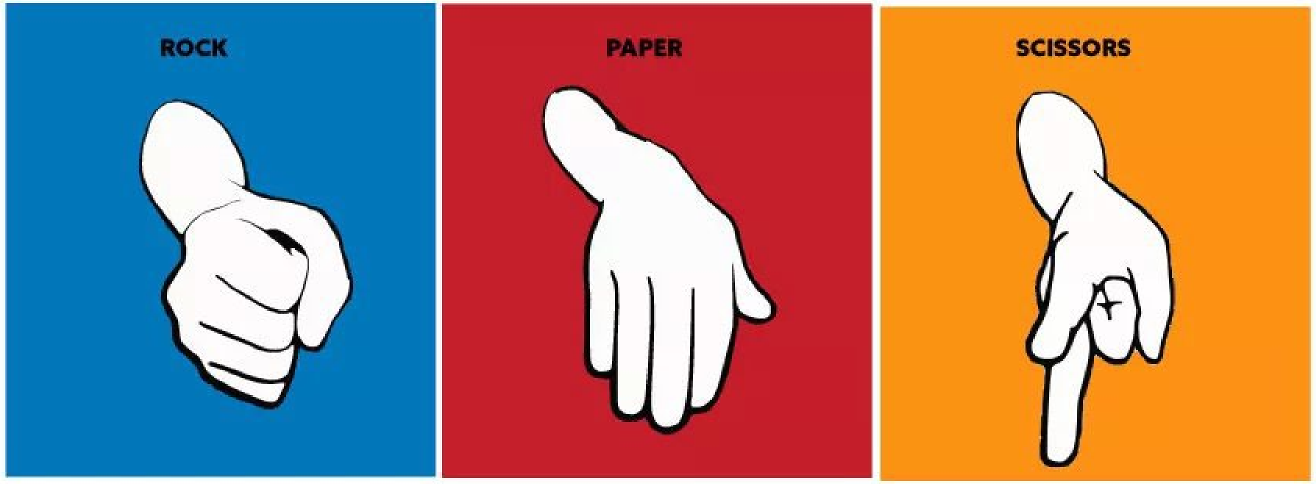
- 5.1 Game theory (博弈论)
- 5.2 极小极大原理
- 5.3 α-β 剪枝
- 5.4 不完美的实时决策

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。

猜拳

- 两人猜拳，规定**连续两局不能出的一样**，在**十分理智**的情况下，**第一把是剪刀平局**，最后会如何?

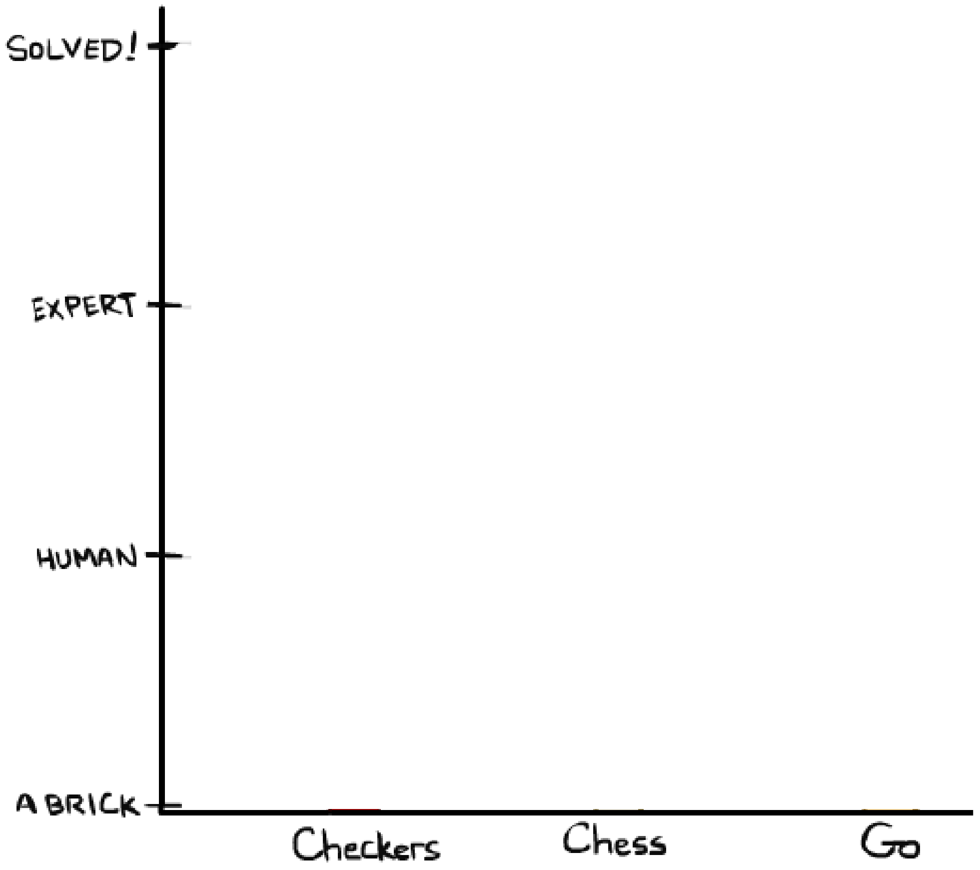


博弈

- 博弈和人类智慧如影随行
- 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**）
- 博弈游戏易于**形式化**，可以形式化表述为**搜索问题**

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



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

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。

博弈论中经典问题：囚徒窘境

警察抓了两个嫌疑犯，<u>在他们没有事先串口供的情况下，**分开审问**</u>。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

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?

由于每位参与者都是从自我利益最大化角度出发的，此时**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

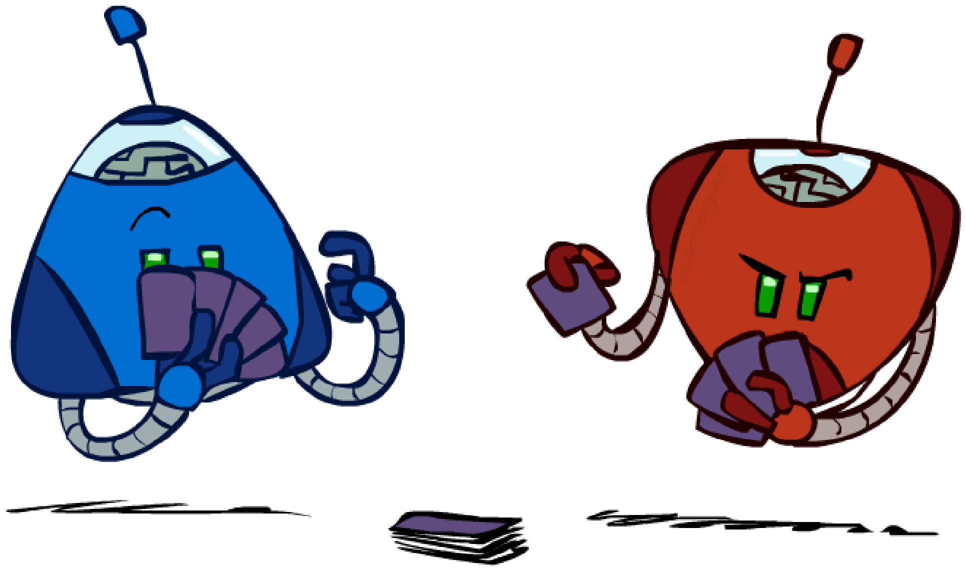
如果审问并不是分开进行，结果又会如何呢?

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。**

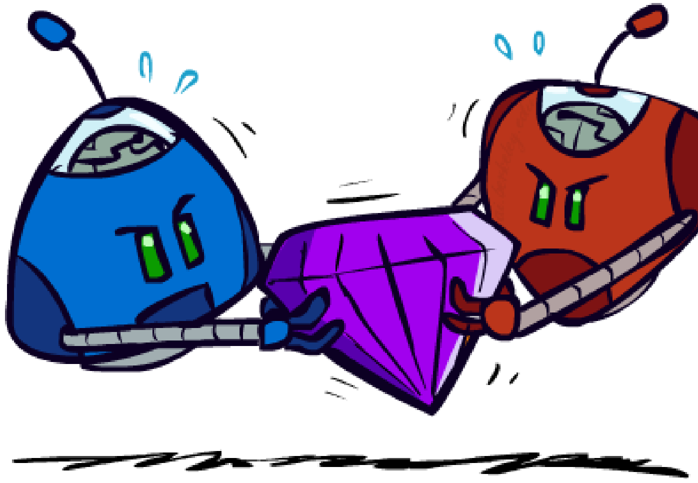
Types of Games

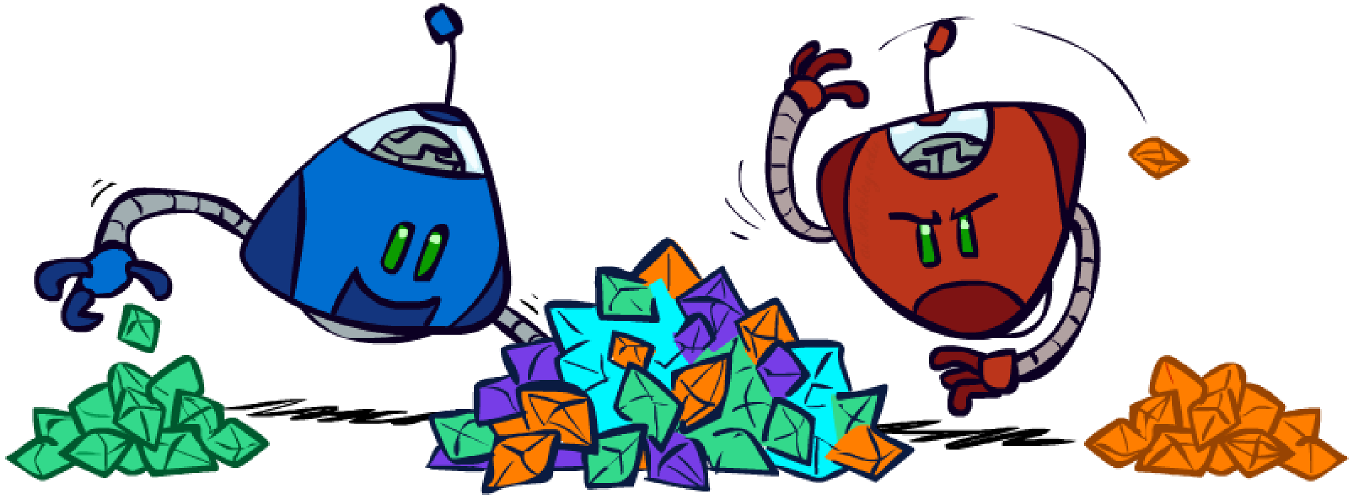
- Game = task environment with > 1 agent

- 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

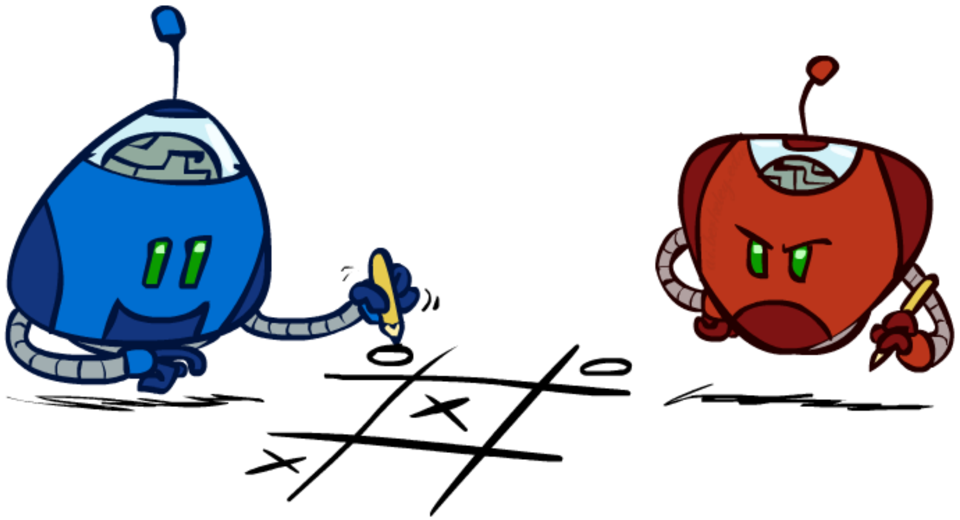




- 零和博弈
  - Agent具有相反的效用
  - 一个收益：一个最大，另一个最小
  - 对抗性的，纯粹的竞争
- 一般博弈
  - Agent具有独立的效用
  - 合作、冷漠、竞争等都是可能的

Deterministic Games

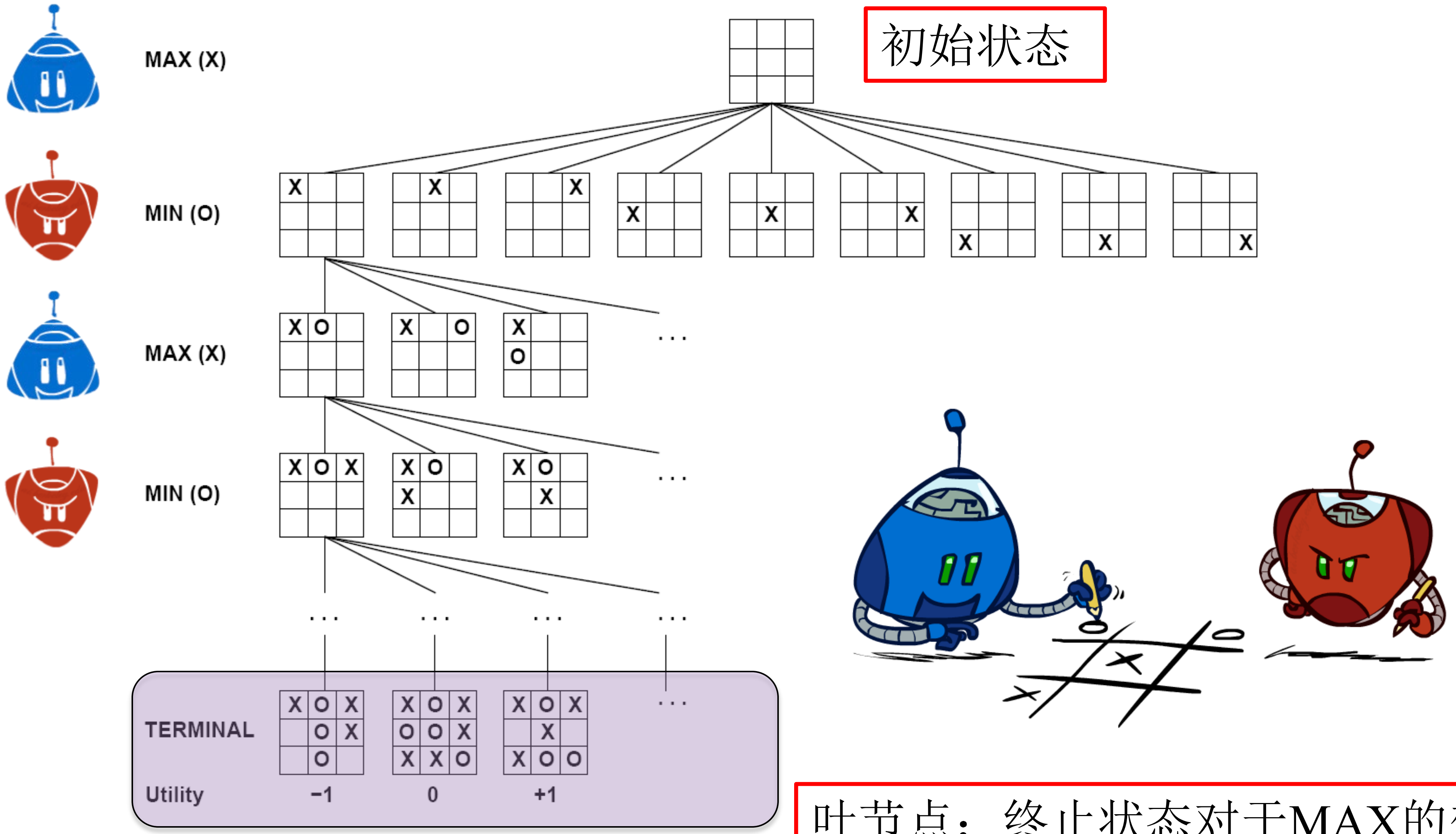
- 问题形式化描述:
  - States: S (start at s_0)
  - Players: P={1...N} (usually take turns)
  - Actions: A (may depend on player / state)
  - Transition Model: RESULT(S,A) $\rightarrow$ S’
  - Terminal Test: S $\rightarrow$ {t, f}
  - Utility Function: Utility(S,P) $\rightarrow$ R



- Solution for *a player* is a policy: S $\rightarrow$ 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

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

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，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，另一个先选蛋糕。

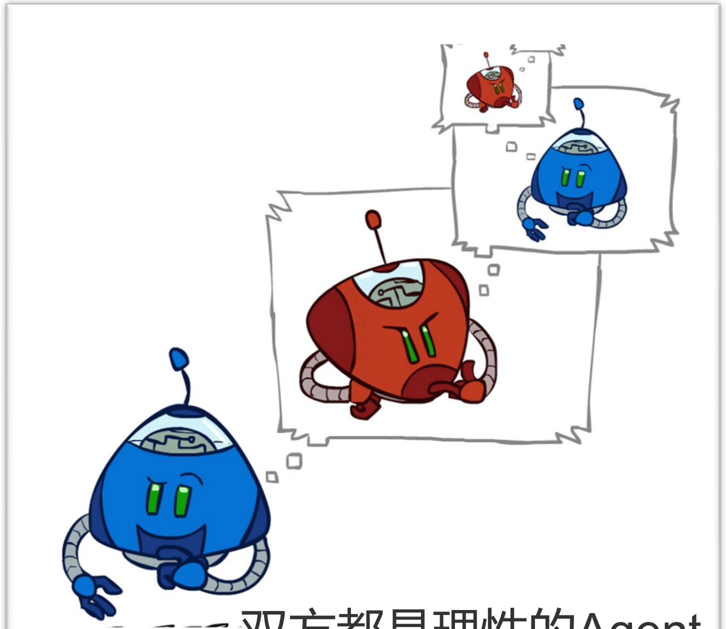
博弈论的目标就是寻找问题的理性解——从理性角度分析所得的答案。

	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



双方都是理性的Agent

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；

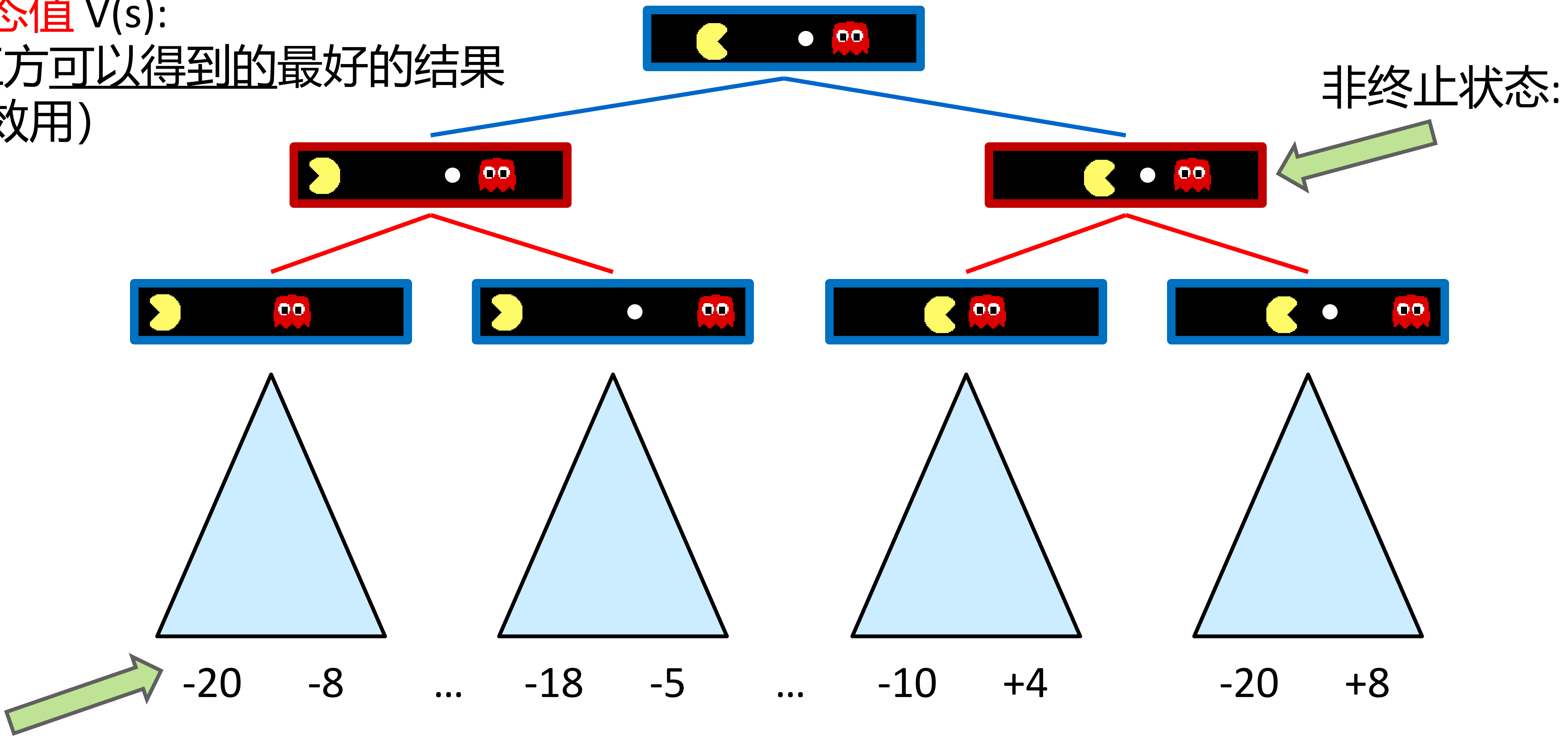
“极小极大”组合起来的意思是，A已知B会选大块，所以会把较小的一块切得大一些。对A来说，最好的结果就是 “一半、一半”，即两人各分得半块蛋糕，这就是这个问题的理性解。

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

对抗博弈树

状态值 V(s):
正方可以得到的最好的结果
(效用)



终止状态: $V(s) = \text{known}$ 正方的效用值

Minimax Values

正方的状态值:

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

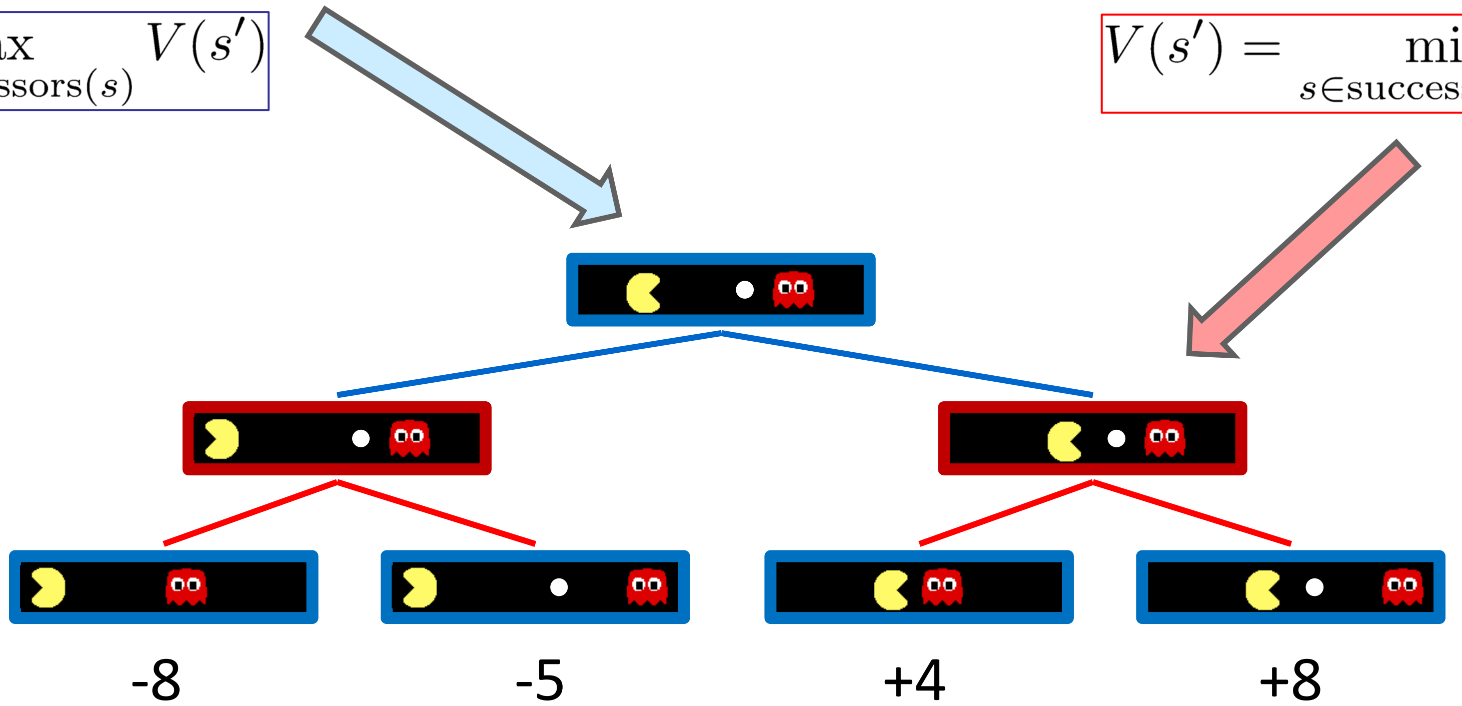
反方的状态值:

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

正方

反方

正方



终止状态:

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

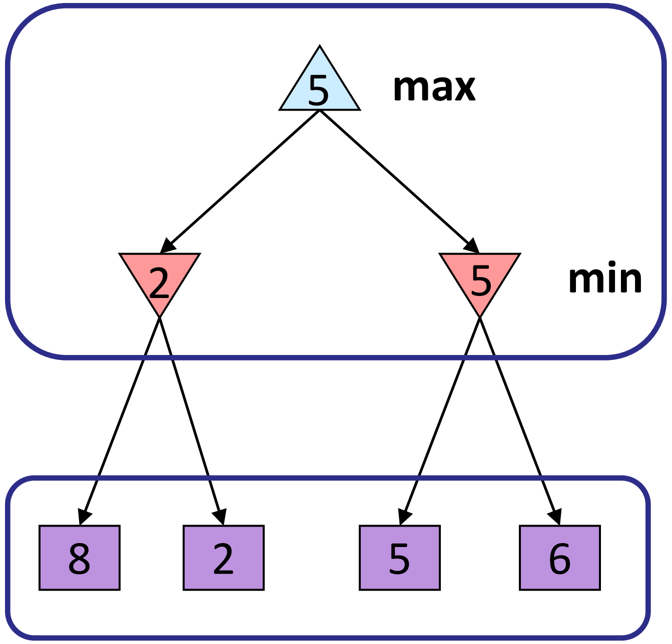
搜索目标：从初始状态出发，找到到达终止状态的最优解路径

对抗搜索：对抗理性（最优）对手，获取可实现的最佳效用

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：对抗理性对手可实现的最佳效用

Minimax values:
computed recursively



Terminal values:
part of the game

Minimax Implementation

```
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
```

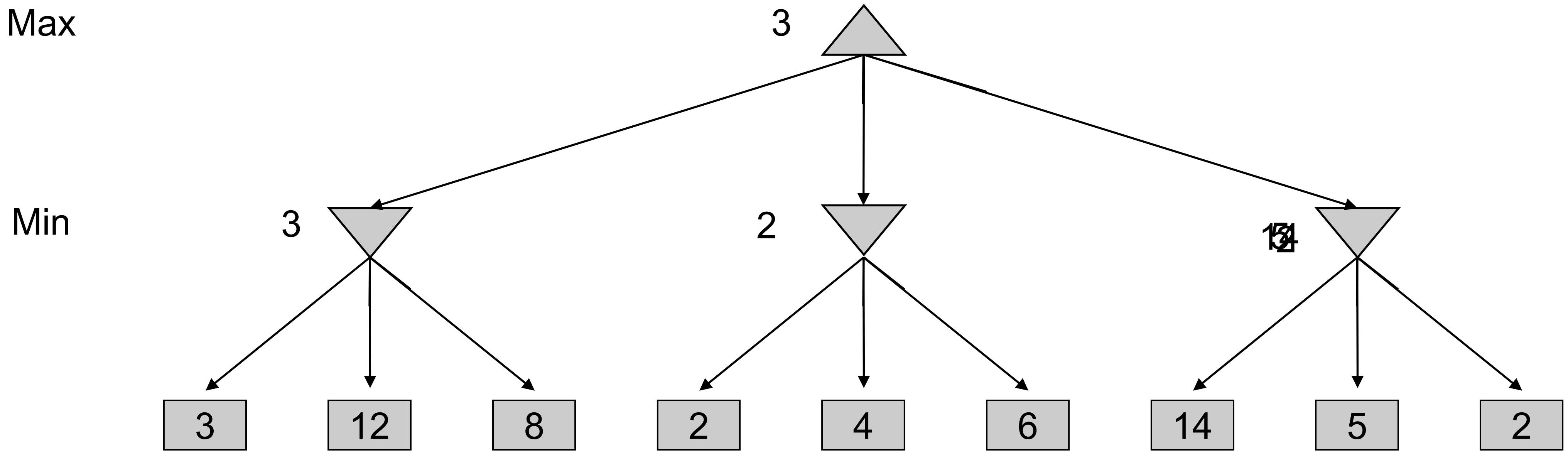
$$V(s) = \max_{s' \in \text{successors}(s)} V(s')$$

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

正方的状态值:

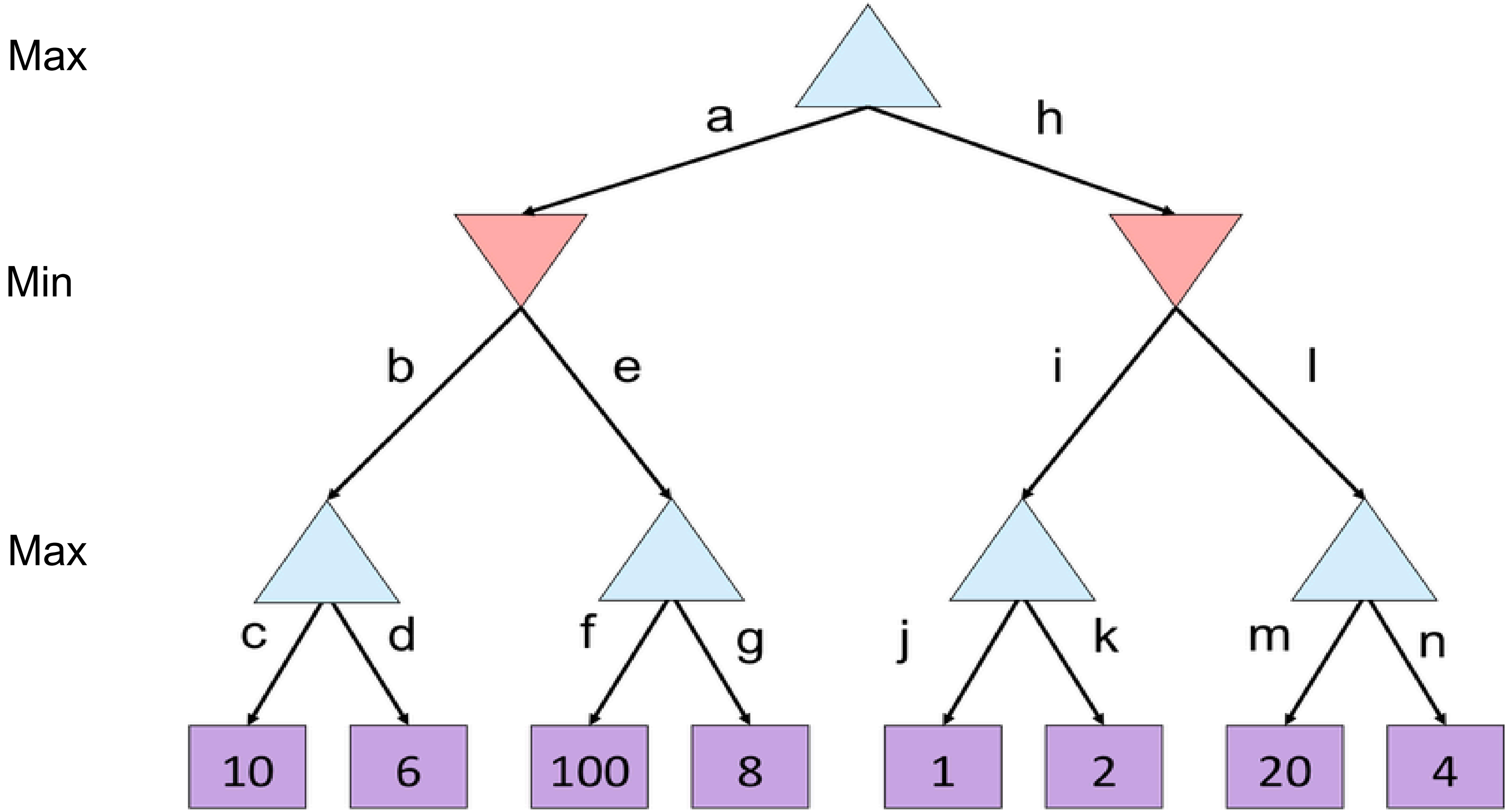
反方的状态值:

Minimax Example

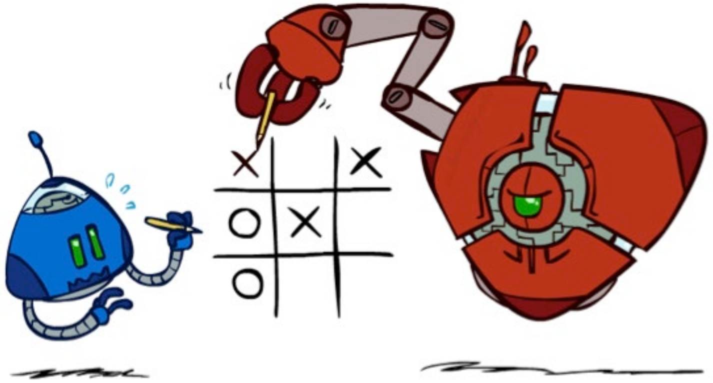


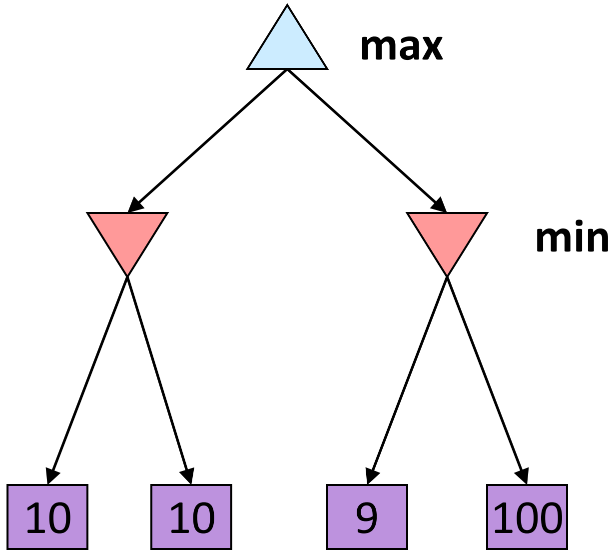
Minimax 课堂练习

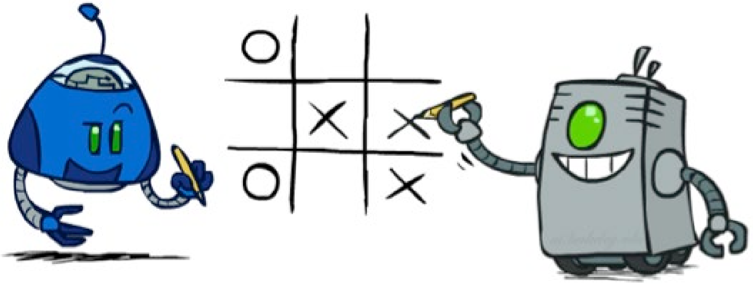
按从左到右的顺序搜索，使用极小极大算法，标明各状态的Utility值



Minimax Properties





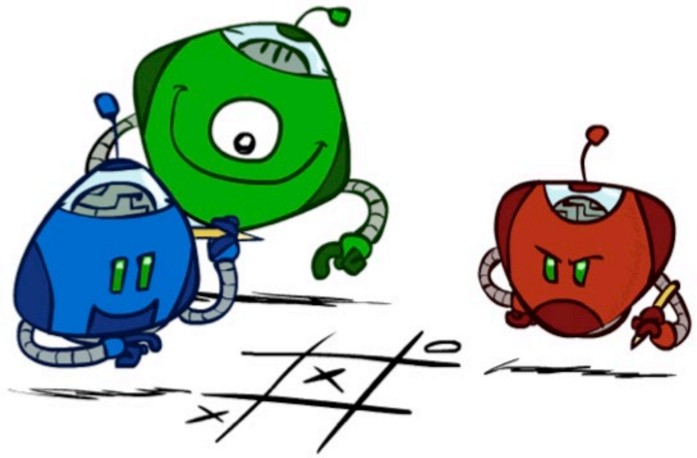


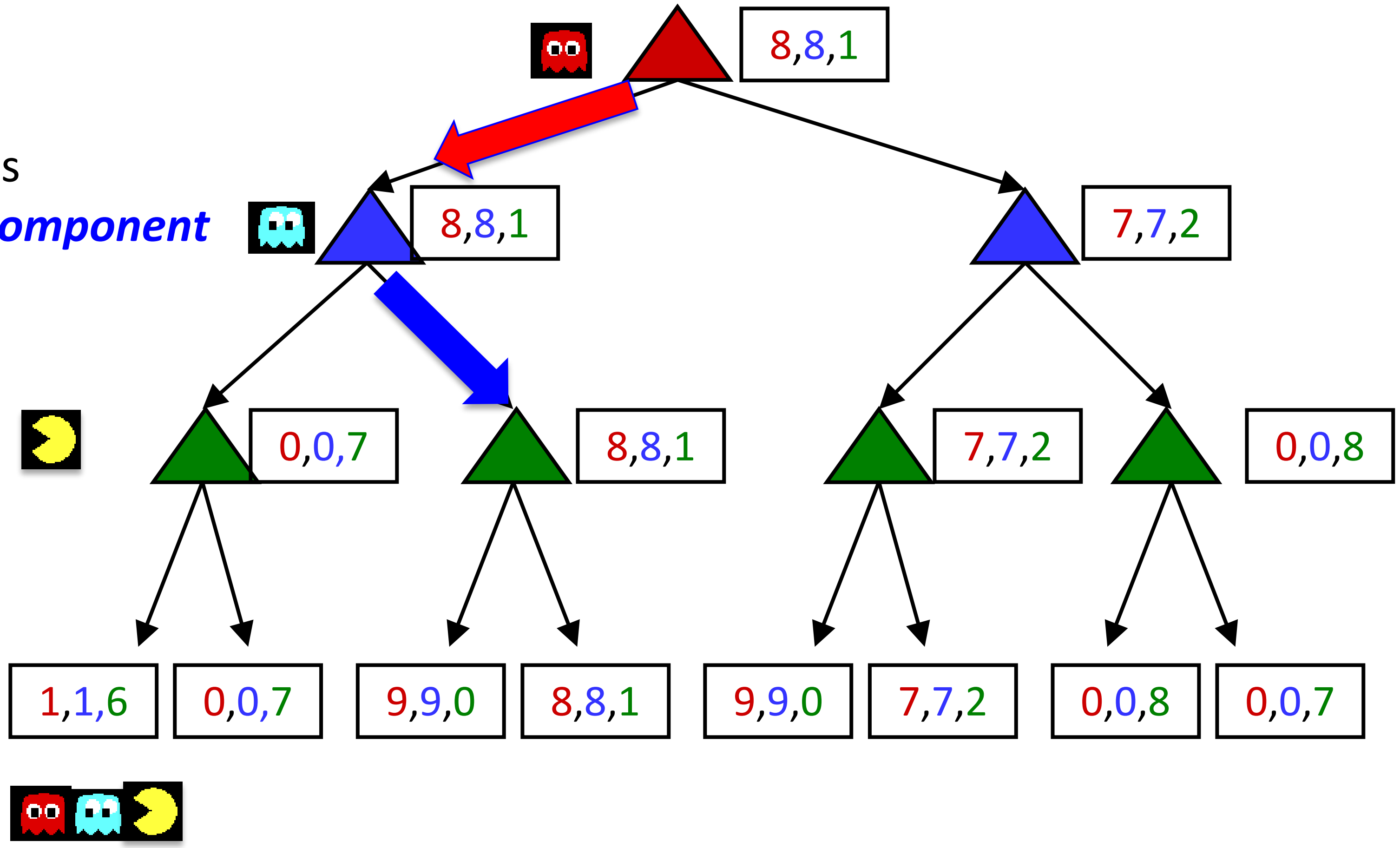
Optimal against a perfect player. Otherwise?

Generalized minimax

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

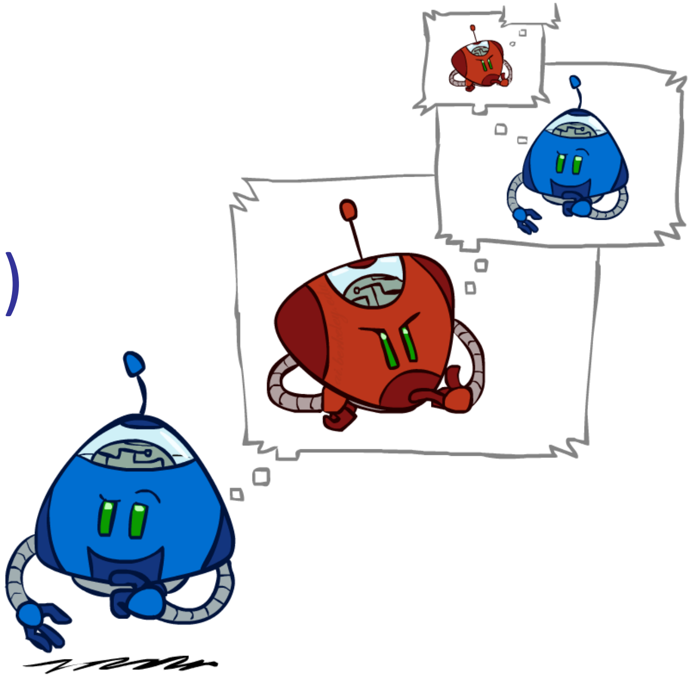
- Generalization of minimax:
  - Terminals have ***utility tuples***
  - Node values are also utility tuples
  - ***Each player maximizes its own component***
  - Can give rise to cooperation and competition dynamically…





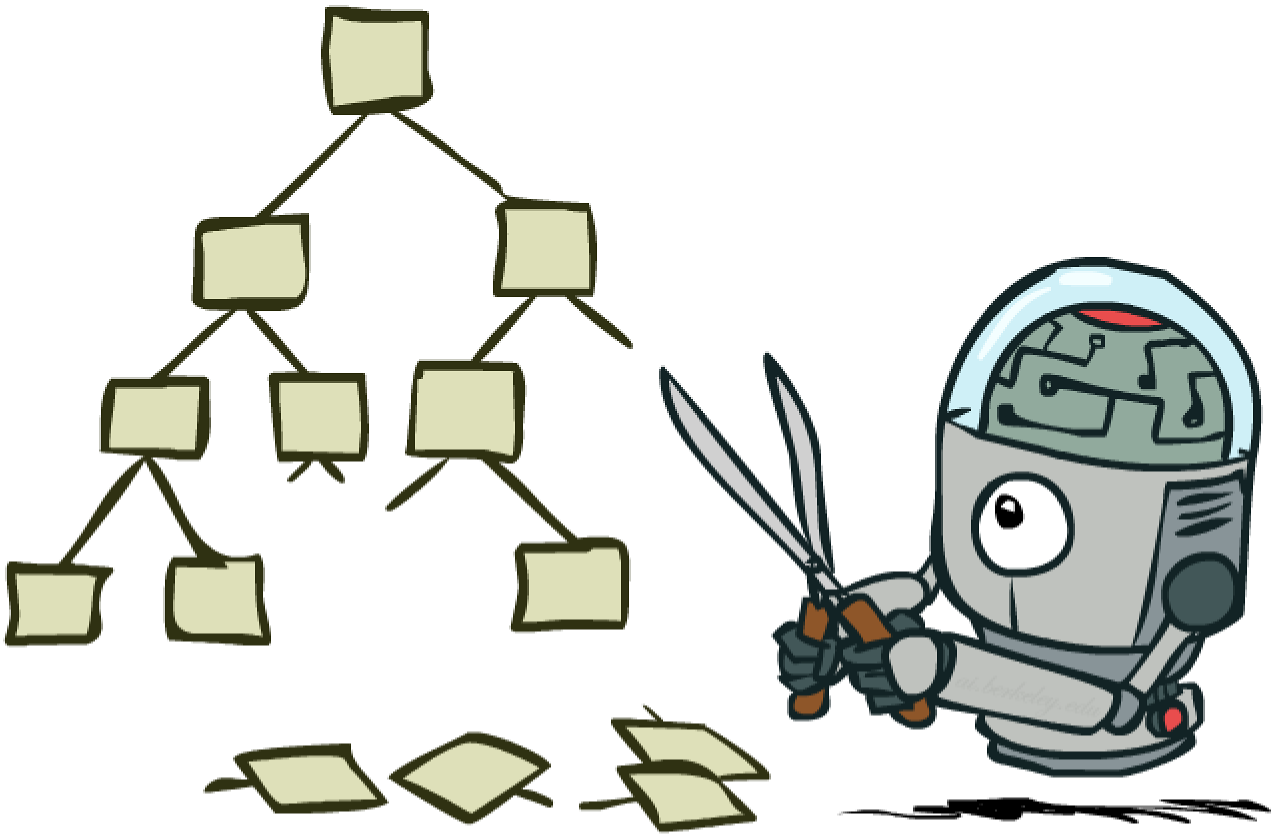
极小极大算法性能

- 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。

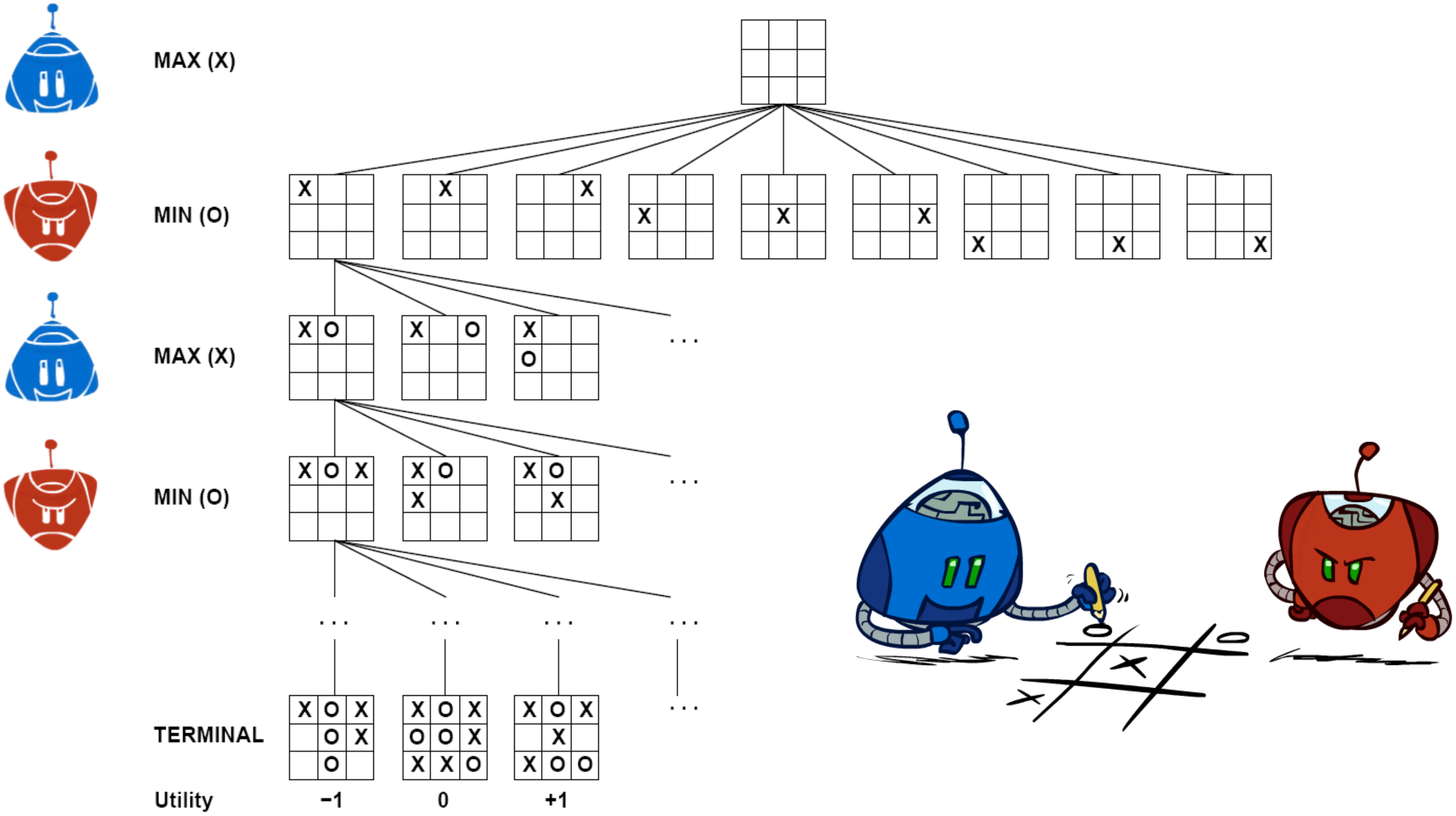
b. 考虑对称性，给出从空棋盘开始的深度为 2 的完整博弈树（即，在棋盘上一个 X 一个 O 的棋局）。

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

第六周作业：5.9 井字棋

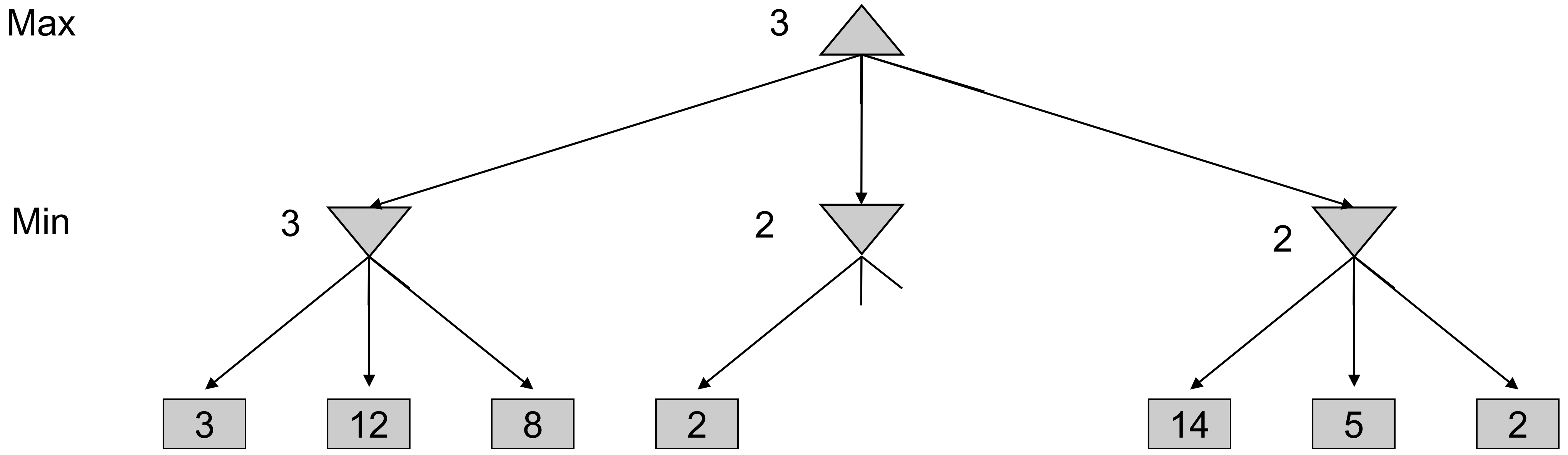


主要内容

- 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

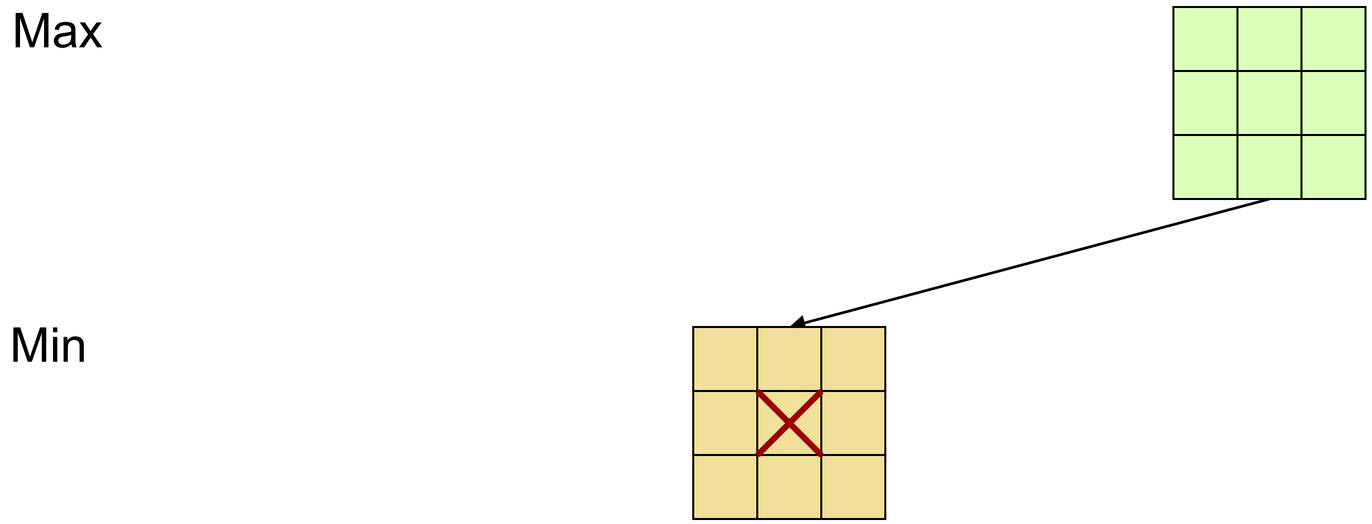
Minimax Pruning

α值是MAX节点的最佳(极大值)选择

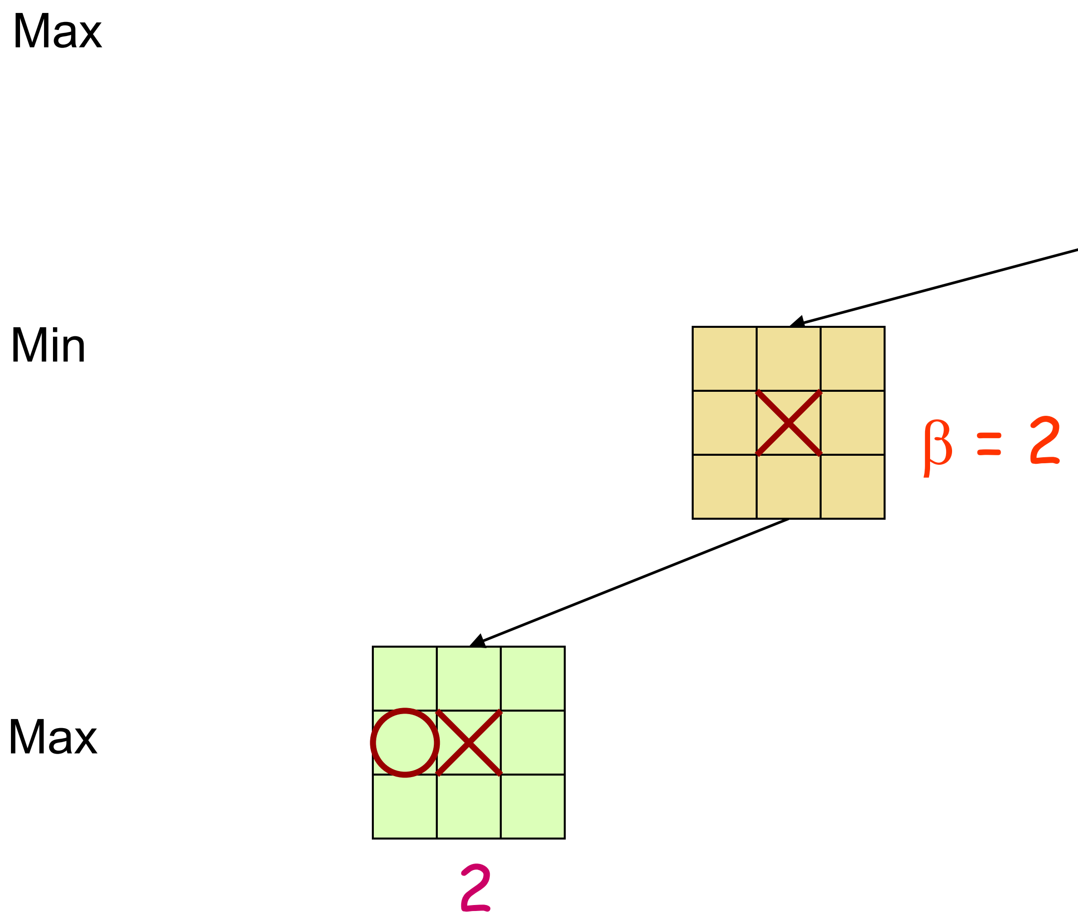


β 值是MIN节点的最佳(极小值)选择

α-β剪枝

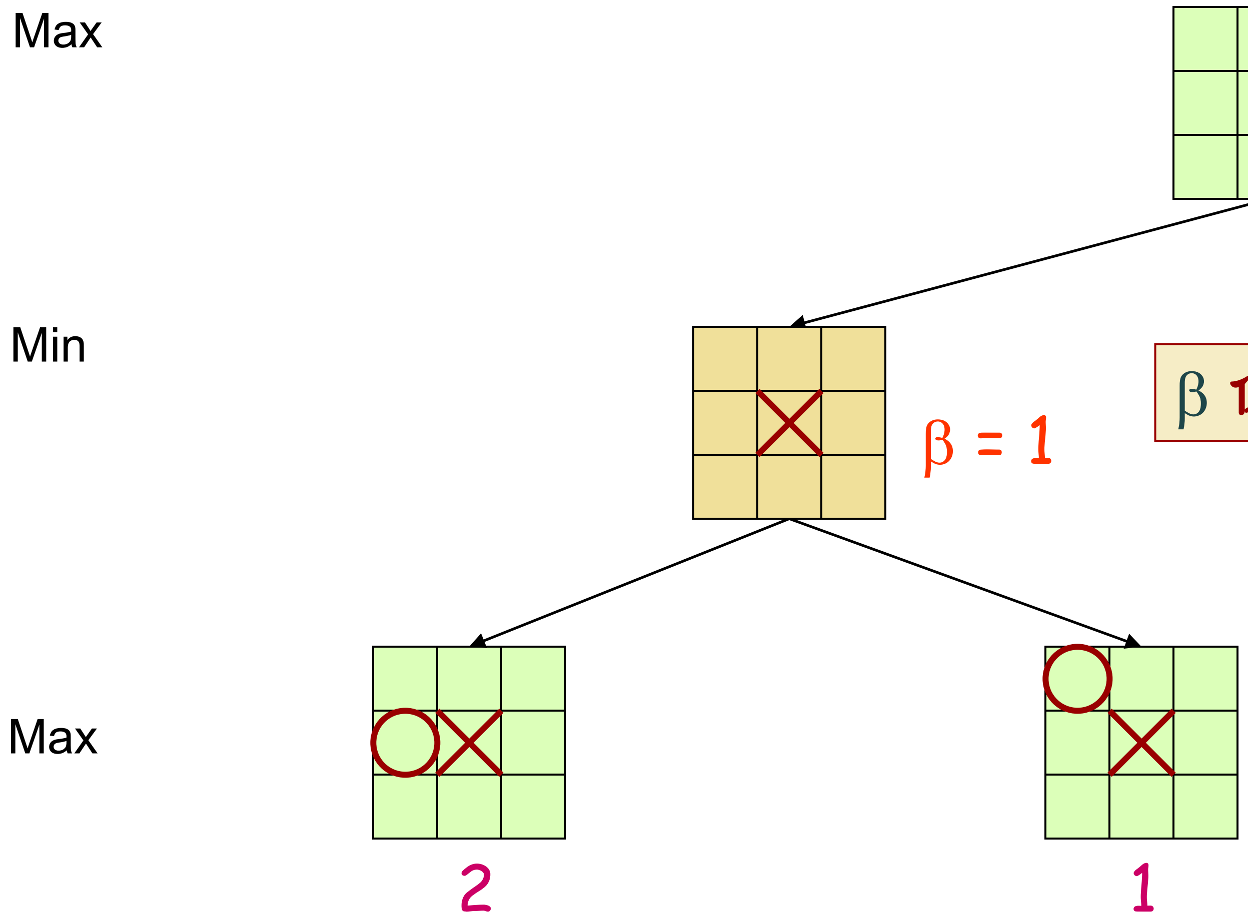


α-β剪枝



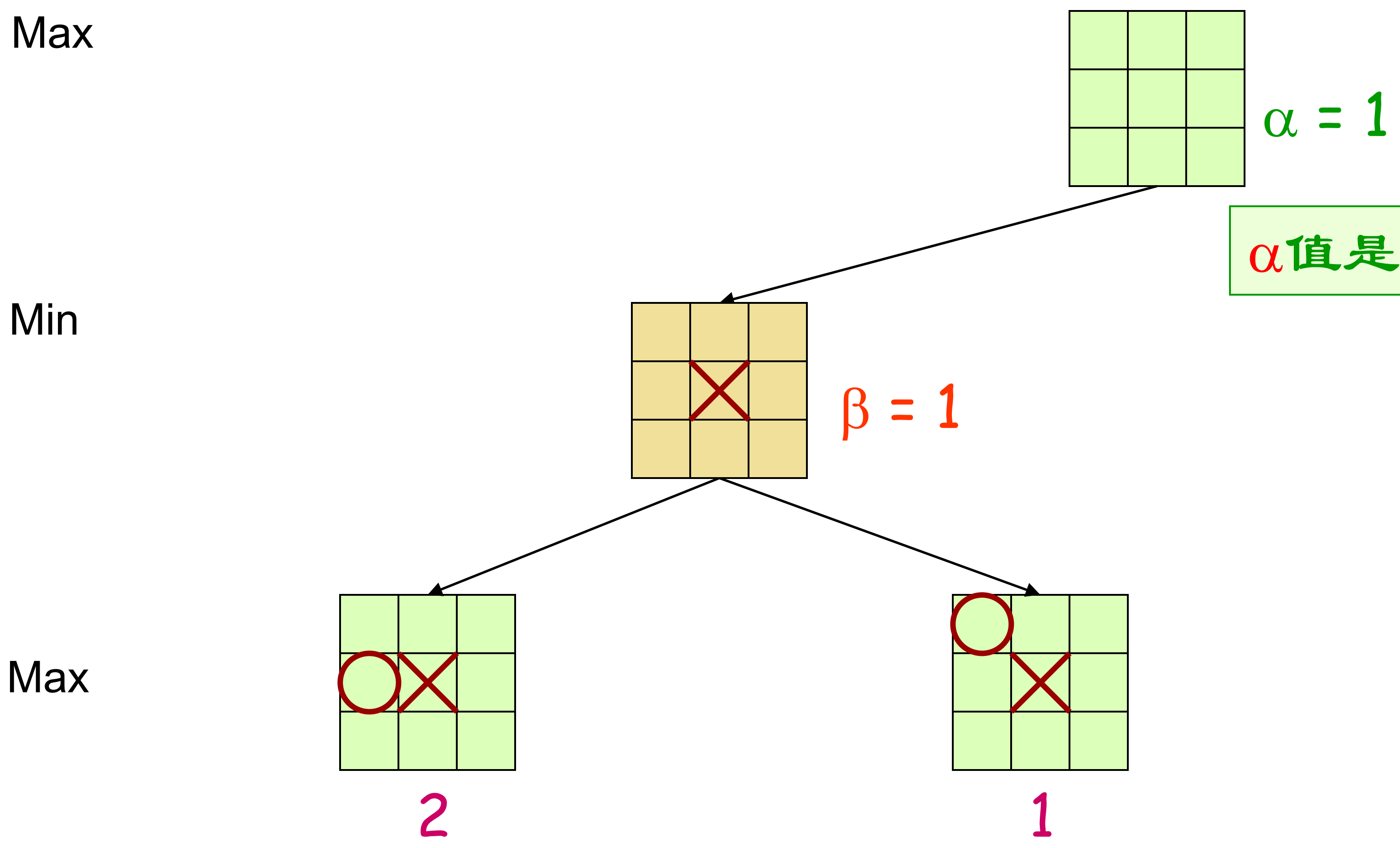
β 值是MIN节点的最佳(极小值)选择

α-β剪枝



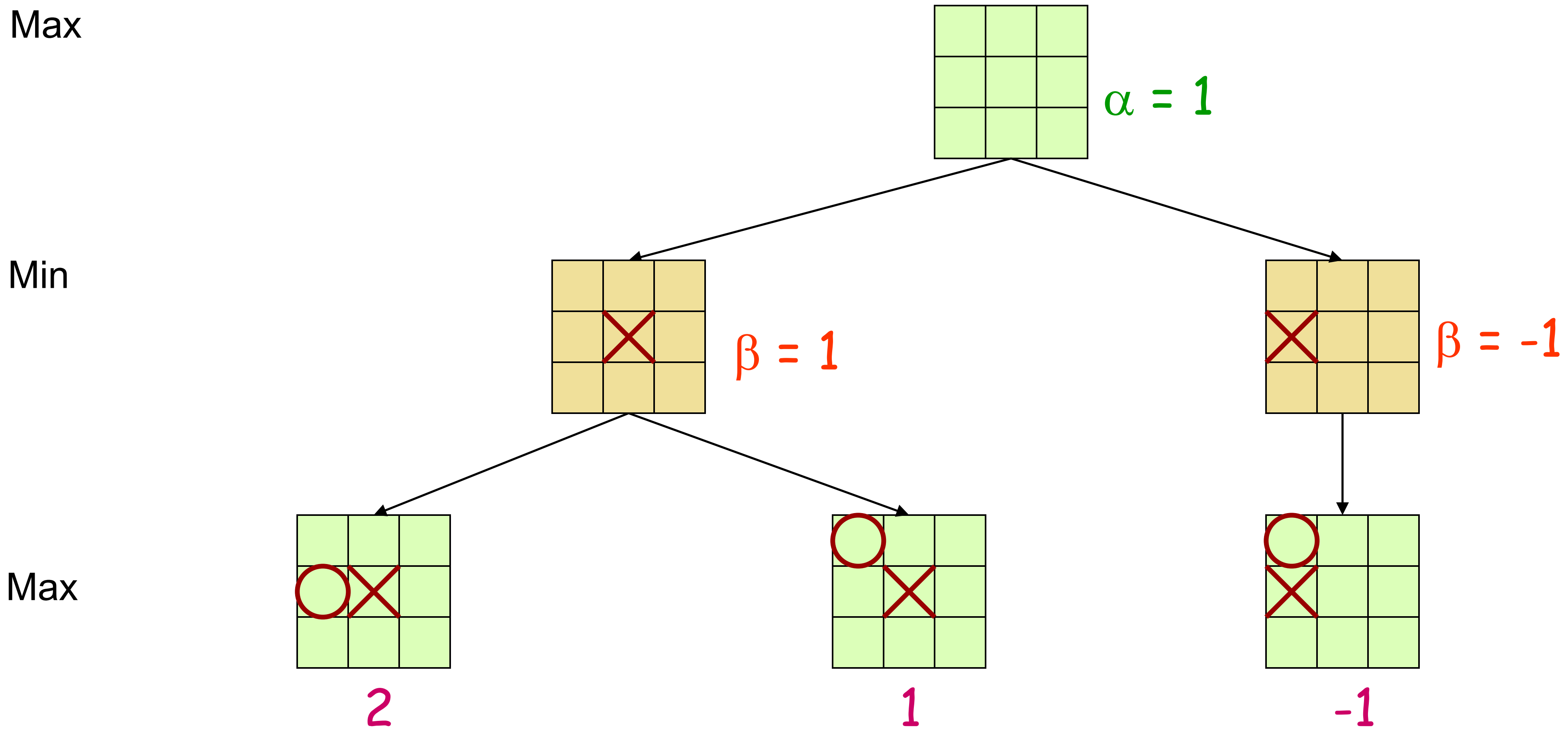
β 值是MIN节点的最佳(极小值)选择

α-β剪枝



α值是MAX节点的最佳(极大值)选择

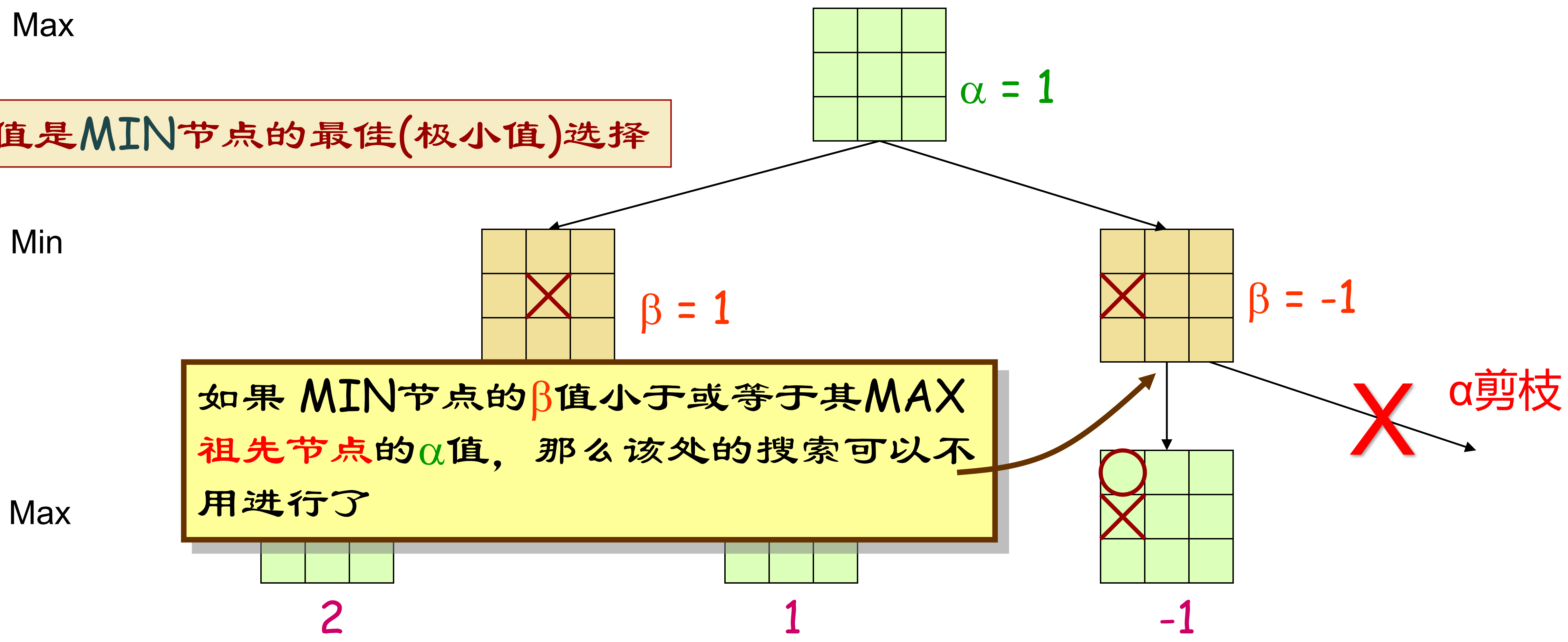
α-β剪枝



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

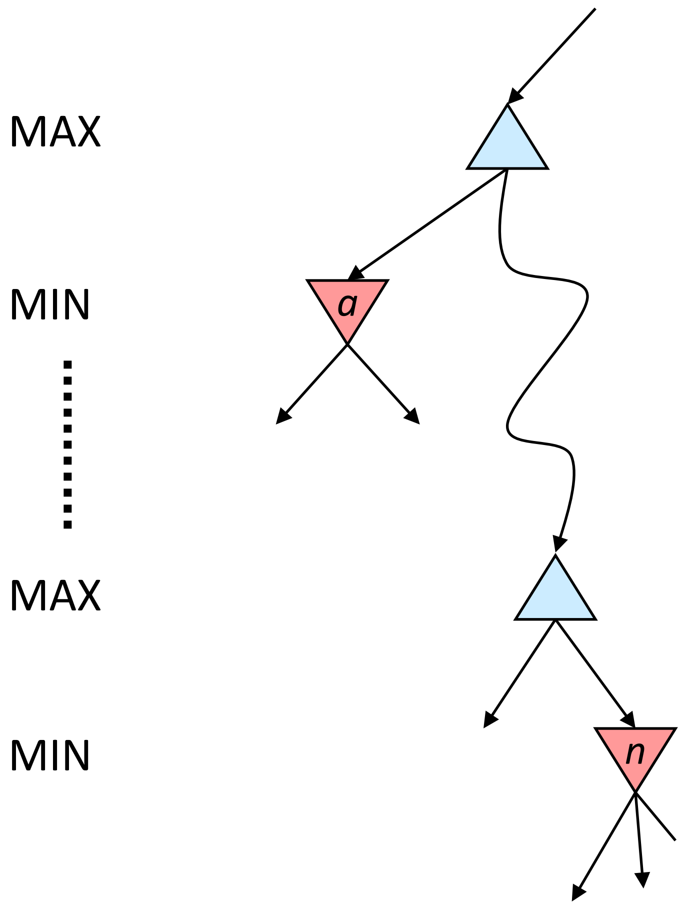


如果 MIN节点的β值小于或等于其MAX祖先节点的α值，那么该处的搜索可以不用进行了

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

α-β 剪枝

- General configuration (MIN version)
  - We're computing the MIN-VALUE at some node *n*
  - We're looping over *n*'s children
  - *n*'s estimate of the childrens' min is dropping
  - Who cares about *n*'s value? MAX
  - Let *a* be the best value that MAX can get at any choice point along the current path from the root
  - If *n* becomes worse than *a*, **MAX will avoid it**, so we can stop considering *n*'s other children (it's already bad enough that it won't be played)



- MAX version is symmetric

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, min-value(successor, α, β))
        if v ≥ β return v
        α = max(α, v)
    return v
```

β剪枝

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

α剪枝