# Artificial Intelligence

## 5.对抗搜索

# 搜索与对抗搜索

| 搜索 | 对抗搜索 |
| --- | --- |
| 单智能体 | 多智能体 |
| 解是寻找目标的（启发式）方法 | 解是策略（指定对每个可能对手回应的行动策略） |
| 启发式法可以找到最优解 | 时间受限被迫执行一个近似解 |
| 评价函数：穿过给定节点从起始到目标的代价估计 | 评价函数：评估博弈局势的“好坏” |

# 5.1 博弈

数学中的**博弈论**，是经济学的一个分支，把多 Agent 环境看成是博弈，其中每个 Agent 都会受到其他 Agent 的“显著”影响，不论这些 Agent 间是合作的还是竞争的[1]。人工智能中“博弈”通常专指博弈论专家们称为**有完整信息的**、确定性的、轮流行动的、两个游戏者的**零和游戏**（如国际象棋）。术语中，这是指在确定的、完全可观察的环境中两个 Agent 必须轮流行动，在游戏结束时效用值总是相等并且符号相反。例如下国际象棋，一个棋手赢了，则对手一定是输了。正是 Agent 之间效用函数的对立导致了环境是对抗的。

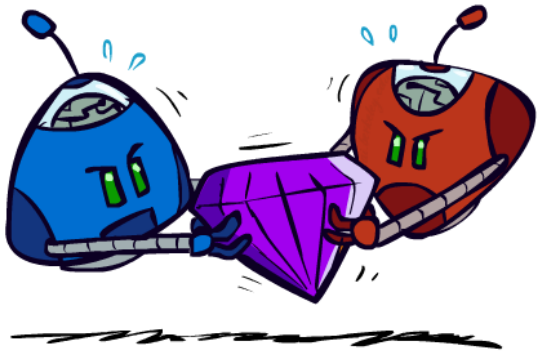
从人类文明产生以来，博弈就和人类智慧如影随形——有时甚至到了令人担忧的程度。对于人工智能研究人员来说，博弈的抽象特性使得博弈成为感兴趣的研究对象。博弈游戏中的状态很容易表示，Agent 的行动数目通常受限，而行动的输出都有严谨的规则来定义。体育游戏如台球和冰球，则有复杂得多的描述，有更大范围的可能行动，也有不够严谨的规则来定义行动的合法性。所以除了足球机器人，体育游戏目前并没有吸引人工智能领域的很大兴趣。

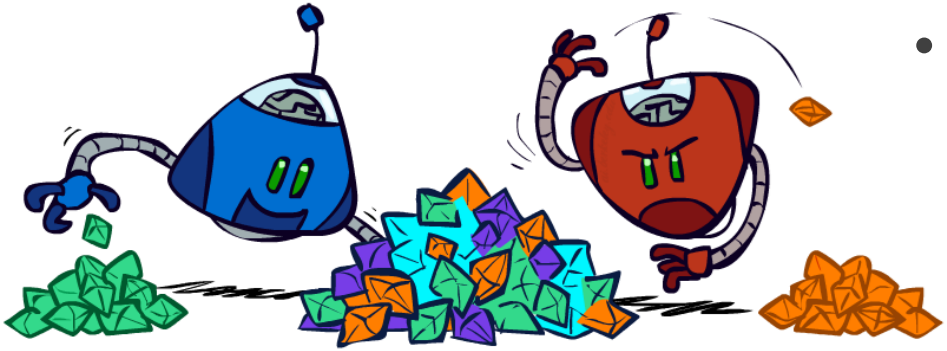
# 博弈的特征

- 两个、或多个玩家（智能体）
- 轮流、与同步行动
- 完全信息、与不完全信息
- 确定性、与随机
- 合作式、与对抗式
- 零和、与非零和

# 零和与非零和博弈

- 零和博弈
  - 智能体之间是对立的方式。
  - 纯竞争：输赢、其和为零。





- 非零和博弈
  - 智能体之间是自主的方式。
  - 合作、中立、竞争、…
  - 双赢、输赢、或双输，其和不为零。

# 囚徒困境

- 有两个犯罪集团的成员被逮捕和监禁。每个囚徒只有二选一的机会：揭发对方并证明其犯罪，或者与对方合作保持沉默。
- 惩罚方式如下：
  - 若A和B彼此揭发对方，则每个囚徒监禁2年。
  - 若A揭发B而B保持沉默，则A被释放而B监禁3年（反之亦然）。
  - 若A和B都保持沉默，则他们仅被监禁1年。

| | 甲沉默（合作） | 甲揭发乙（背叛） |
|---|---|---|
| 乙沉默（合作） | 二人同服刑半年 | 甲即时获释；乙服刑3年 |
| 乙揭发甲（背叛） | 甲服刑3年；乙即时获释 | 二人同服刑2年 |

# 囚徒困境

- 囚徒困境假定每个参与者（即“囚徒”）都是利己的，即都寻求最大自身利益，而不关心另一参与者的利益。另外，没有任何其他力量干预个人决策，参与者可完全按照自己意愿选择策略。
- 囚徒到底应该选择哪一项策略，才能将自己个人的刑期缩至最短？
  - 两名囚徒由于隔绝监禁，并不知道对方选择。
  - 就个人的理性选择而言，检举背叛对方所得刑期，总比沉默要来得低。
- 试设想困境中两名理性囚徒会如何作出选择：
  - 若对方沉默、背叛会让我获释，所以会选择背叛。
  - 若对方背叛指控我，我也要指控对方才能得到较低的刑期，所以也是会选择背叛。
- 二人面对的情况一样，所以二人的理性思考都会得出相同的结论——选择背叛。背叛是两种策略之中的支配性策略。
- 因此，这场博弈中唯一可能达到的纳什均衡，就是双方参与者都背叛对方，结果二人同样服刑2年。

# 纳什均衡

- 在一个博弈过程中，无论对方的策略选择如何，当事人一方都会选择某个确定的策略，则该策略被称作支配性策略。
- 如果两个博弈的当事人的策略组合分别构成各自的支配性策略，那么这个组合就被定义为纳什平衡。

# 智猪博弈

- 猪圈里面有两只猪，一只大，一只小。猪圈很长，一头有一个踏板，另一头是饲料的出口一下踏板，在远离踏板的猪圈的另一边的投食口就会落和食槽。每踩下少量的食物。
  - 如果有一只猪去踩踏板，另一只猪就有机会抢先吃到另一边落下的食物。
  - 当小猪踩动踏板时，大猪会在小猪跑到食槽之前刚好吃光所有的食物；
  - 若是大猪踩动了踏板，则还有机会在小猪吃完落下的食物之前跑到食槽，争吃到另一半残羹。
- 那么，两只猪各会采取什么策略？
- 答案是：小猪将选择“搭便车”策略，也就是舒舒服服地等在食槽边；而大猪则为一点残羹不知疲倦地奔忙于踏板和食槽之间。
- 因为，小猪踩踏板将一无所获，不踩踏板反而能吃上食物。对小猪而言，无论大猪是否踩动踏板，不踩踏板总是好的选择。反观大猪，已明知小猪是不会去踩动踏板的，自己亲自去踩踏板总比不踩强吧，所以只好亲力亲为了。
- “智猪博弈”的结论似乎是，在一个双方公平、公正、合理和共享竞争环境中，有时占优势的一方最终得到的结果却有悖于他的初始理性。

# 博弈的类型

| | 确定性 | 随机性 |
| --- | --- | --- |
| 完全信息<br>(可完全观测) | 国际象棋、西洋跳棋、围棋、黑白棋 | 西洋双陆棋<br>大富翁 |
| 不完全信息<br>(不可完全观测) | 西洋陆军棋、海战棋 | 桥牌、扑克、拼字游戏 |

# 井字游戏Tic-Tac-Toe

MAX (X)

MIN (O)

MAX (X)

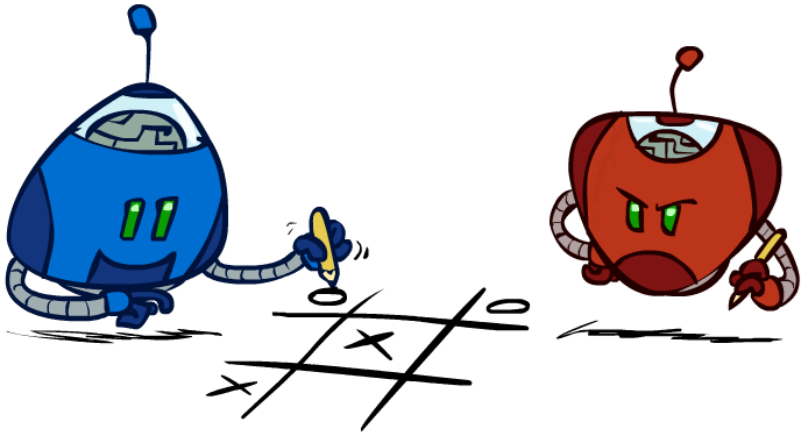
MIN (O)

TERMINAL

Utility

−1 0 +1

MIN(O) 和MAX(X)之间交替走子

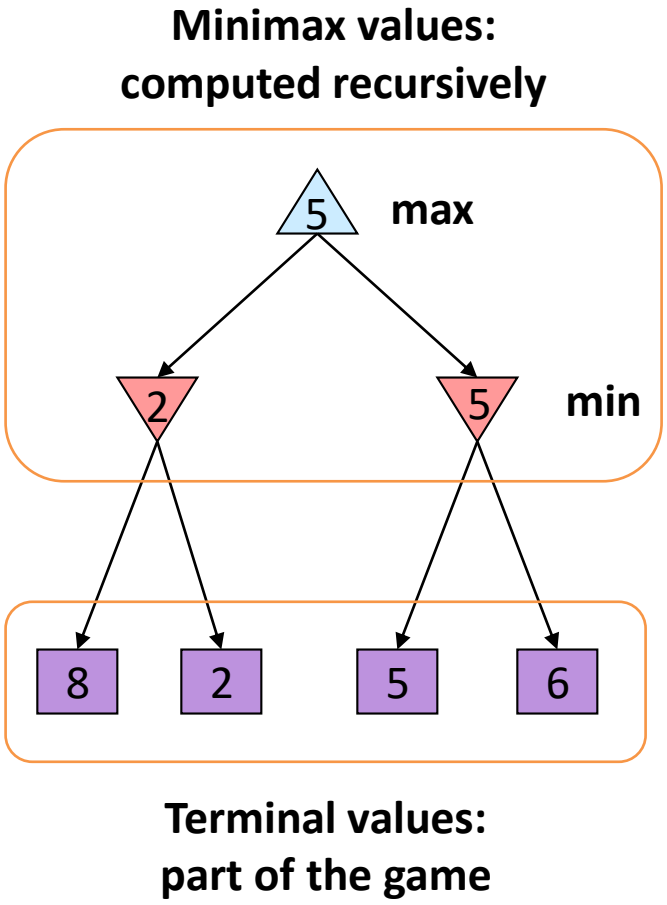


# 5.2 博弈中的优化决策

- 最优解
  - 普通搜索：最优解将是导致获胜的目标状态（终端状态）的一系列动作。
- 对抗搜索
  - MAX和MIN都会有一个最优策略。
  - 在初始状态，MAX必须找到一个策略来确定MAX的动作。
  - 然后MAX针对MIN的每个合理的对应采取相应的动作，以此类推。

# 5.2.1 极大极小算法Minimax

- 确定性，零和游戏：
  - 一字棋、国际象棋、西洋跳棋
  - 一个玩家使结果最大化
  - 另一个最小化结果
- 极大极小算法：
  - 状态空间搜索树
  - 玩家交替操作
  - 计算每个节点的极大极小值



# Minimax算法实现

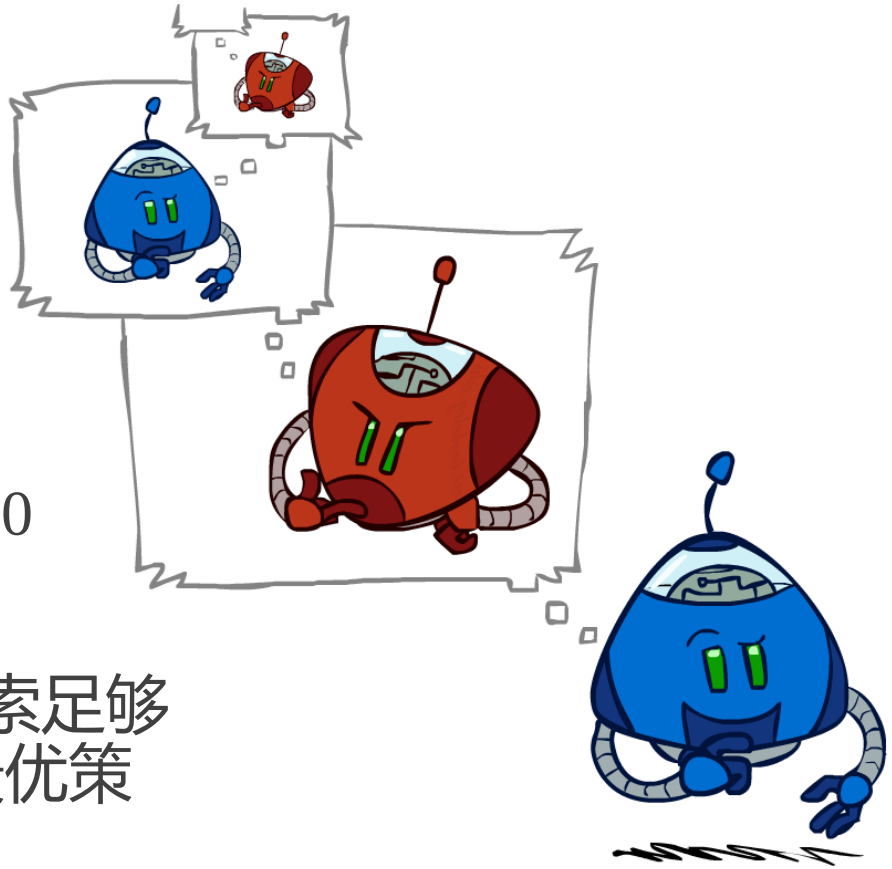
```
def value(state):
    if the state is a terminal state: return the state's utility
    if the next agent is MAX: return max-value(state)
    if the next agent is MIN: return min-value(state)
```

```
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
```

# Minimax算法效率

- $b$：搜索分支因子
- $m$：最大深度
- 时间复杂性：$O(b^m)$
- 空间复杂性：$O(bm)$



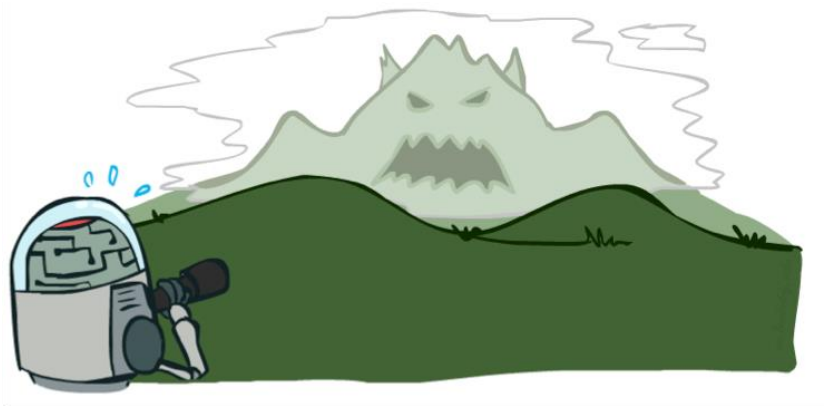
- 以国际象棋为例，b≈35, m ≈ 100
- 求得所有可能解是不现实的
- 实际的博弈中，往往是限时搜索足够的深度后，选择在该深度下的最优策略

# 深度限制

- 解决方案：Depth-limited搜索
  - 只搜索树中的有限深度
  - 用非终端位置的评估函数替换终端实用程序
- 例子:
  - 国际象棋游戏，时间复杂度 $35^m$
  - 计算速度：每秒探索10K个节点
  - 时间限制：每步100s
  - 那么每次移动可以检查1M个节点
  - 从任意一个状态开始，每个Agent的搜索深度为 3.88 步
- 如何提高搜索深度?

# 深度限制

- 一般来说，评价函数需要的参数埋得越深，其预测准确度往往越好，实际中对深度和准确度往往需要做权衡

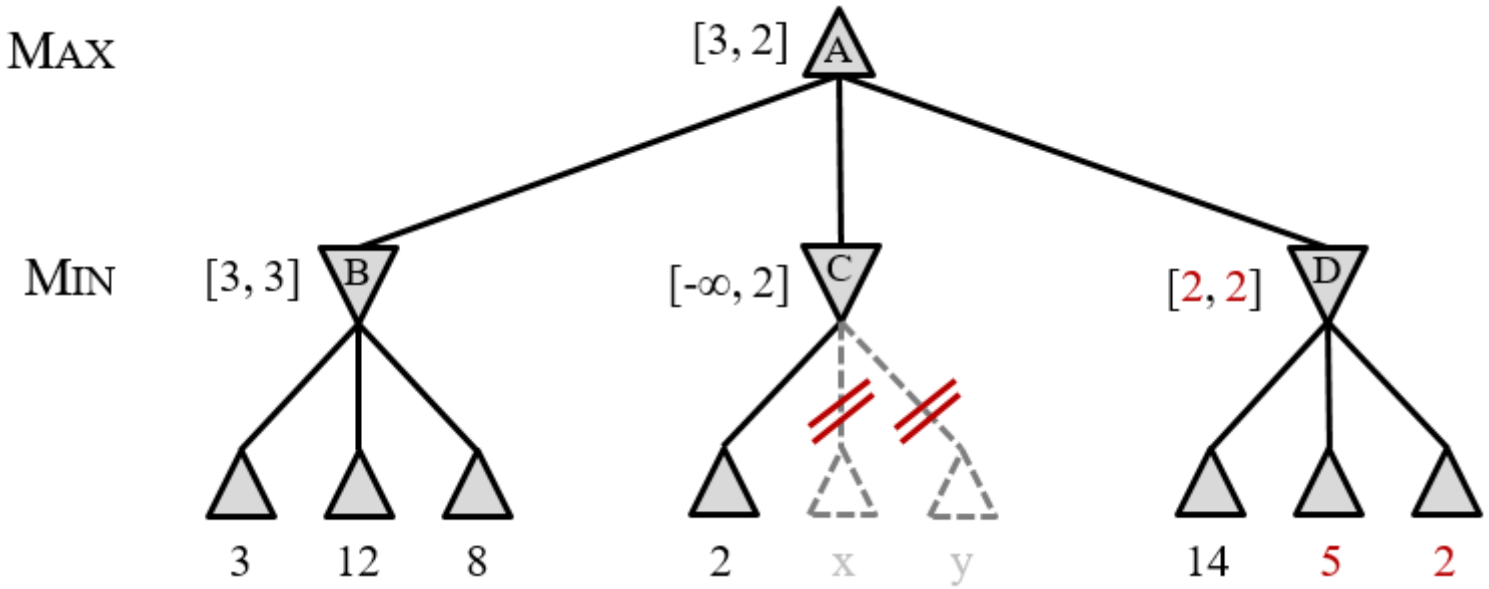


# 为何称其为α-β

- 从如下两个参数得到其名称：
  - α：沿着MAX路径上的任意选择点，迄今为止我们已经发现的最高值。
  - β：沿着MIN路径上的任意选择点，迄今为止我们已经发现的最低值。

- α-β搜索依次完成如下动作：
  - 边搜索边更新α和β的值
  - 一旦得知当前节点的值比当前MAX或MIN的α或β值更差，则在该节点剪去其余的分枝。

# α-β剪枝过程



# α-β剪枝实现

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
```

# 例：石子游戏

- Alice 和 Bob 用几堆石子在做游戏。几堆石子排成一行，每堆石子都对应一个得分，由数组 stoneValue 给出。
- Alice 和 Bob 轮流取石子，Alice 总是先开始。在每个玩家的回合中，该玩家可以拿走剩下石子中的的前 1、2 或 3 堆石子 。比赛一直持续到所有石头都被拿走。
- 每个玩家的最终得分为他所拿到的每堆石子的对应得分之和。每个玩家的初始分数都是 0 。比赛的目标是决出最高分，得分最高的选手将会赢得比赛，比赛也可能会出现平局。
- 假设 Alice 和 Bob 都采取最优策略 。如果 Alice 赢了就返回 "Alice" ，Bob 赢了就返回 "Bob"，平局（分数相同）返回 "Tie" 。

# 石子游戏

- 示例 1：
  - values = [1,2,3,7]，赢家："Bob"
  - 解释：Alice 总是会输，她的最佳选择是拿走前三堆，得分变成 6 。但是 Bob 的得分为 7，Bob 获胜。
- 示例 2：
  - values = [1,2,3,-9]，赢家："Alice"
  - 解释：Alice 要想获胜就必须在第一个回合拿走前三堆石子，给 Bob 留下负分。如果 Alice 只拿走第一堆，那么她的得分为 1，接下来 Bob 拿走第二、三堆，得分为 5 。之后 Alice 只能拿到分数 -9 的石子堆，输掉比赛。如果 Alice 拿走前两堆，那么她的得分为 3，接下来 Bob 拿走第三堆，得分为 3 。之后 Alice 只能拿到分数 -9 的石子堆，同样会输掉比赛。
- 示例 3：
  - values = [1,2,3,6]，赢家："Tie"
  - 解释：Alice 无法赢得比赛。如果她决定选择前三堆，她可以以平局结束比赛，否则她就会输。

# 代码

```
stoneGame(int[] stoneValue){
        int total=0;
        for(int i=0;i<stoneValue.length;i++){
            total+=stoneValue[i];
        }
        int alice=value(stoneValue, 0, 0, 0, true,
                -Integer.MAX_VALUE, Integer.MAX_VALUE);
        if(alice>total-alice){
            return "Alice";
        }else if(alice==total-alice){
            return "Tie";
        }else{
            return "Bob";
        }
    }
```

# 代码

```
value(){
        if(p==stoneValue.length){
            return alice;
        }
        if(turn){
            int maxV=maxValue(stoneValue, p, alice, bob, a, b);
            return maxV;
        }else{
            int minV=minValue(stoneValue, p, alice, bob, a, b);
            return minV;
        }
    }
```

```
maxValue(){
        int val=-Integer.MAX_VALUE;
        for(int i=1;i<=3;i++){
            if(p+i-1<stoneValue.length){
                alice+=stoneValue[p+i-1];
                int valI=value(stoneValue, p+i, alice, bob, false, a, b);
                val=val>valI ? val : valI;
                if(val>=b){
                    return val;
                }
                a=a>val ? a : val;
            }
        }
        return val;
    }
minValue(){
        int val=Integer.MAX_VALUE;
        for(int i=1;i<=3;i++){
            if(p+i-1<stoneValue.length){
                bob+=stoneValue[p+i-1];
                int valI=value(stoneValue, p+i, alice, bob, true, a, b);
                val=val<valI ? val : valI;
                if(val<=a){
                    return val;
                }
                b=b<val ? b : val;
            }
        }
        return val;
    }
```

# 5.4 不完美的实时决策

- minimax算法生成整个博弈搜索空间。
- α-β剪枝算法允许我们将其剪去大部分。
- 然而，α-β仍然需要搜索抵达终端状态的所有途径、至少是搜索空间的一部分。
- 这个深度通常是不实际的，因为移动必须在合理的时间内完成。
- 克劳德·香农提出：程序应该早一些剪断搜索，并在搜索中对状态应用启发式评估函数，有效地将非终端节点转换为终端叶节点。该建议是用如下两种方法：
  - 用EVAL来代替UTILITY，EVAL：一个启发式评价函数，用于估计位置的效用。
  - 用CUTOFF-TEST来代替TERMINAL-TEST，CUTOFF-TEST：确定何时应用EVAL函数。

# 5.4.1 评估函数

- 一个评价函数返回从一个给定位置该博弈的期望效用估计。如何设计好的评价函数?
- 它应该与真实效用函数相同的方式对终端状态进行整理:
  a) 获胜状态必须评价为优于平局,
  b) 平局状态必须评价为优于失败。
- 计算的时间一定不能太长。
- 非终端状态应该与实际获胜的机会密切相关。
- 加权线性函数:

$$\mathrm{EVAL}(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s) = \sum_{i=1}^{n} w_i f_i(s)$$

$w_i$ – 每个特征权重（一般0-1之间，所有特征之和为1）

$f_i$ -- 棋局某个特征（如: “白棋棋子位置平均值-黑棋棋子位置平均值”、“白棋所有棋子数量-黑棋所有棋子数量”）

# 非线性特征

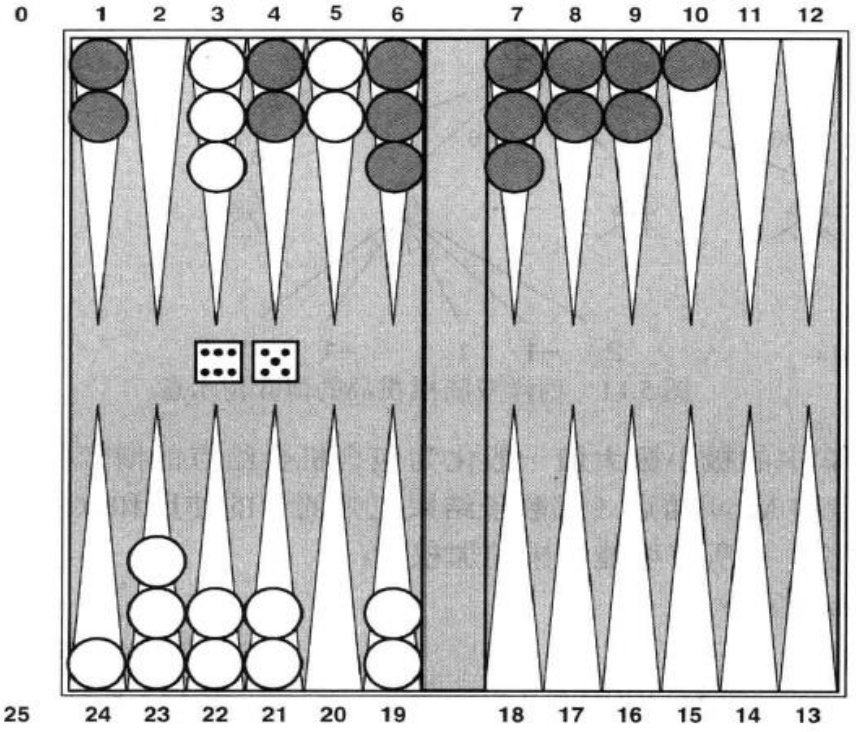
（a）白棋要走子

(b) 白棋走子后

- 两盘国际象棋布局仅在右下角车的位置不同：
  - (a) 黑棋多一个马和两个兵，应该赢得这盘棋。
  - (b) 白棋将捕获皇后，使得它处于足以获胜的态势。
- 线性特征：每个特征的贡献独立于其它的值
- 非线性特征：两个象合起来的作用超过单个象x2

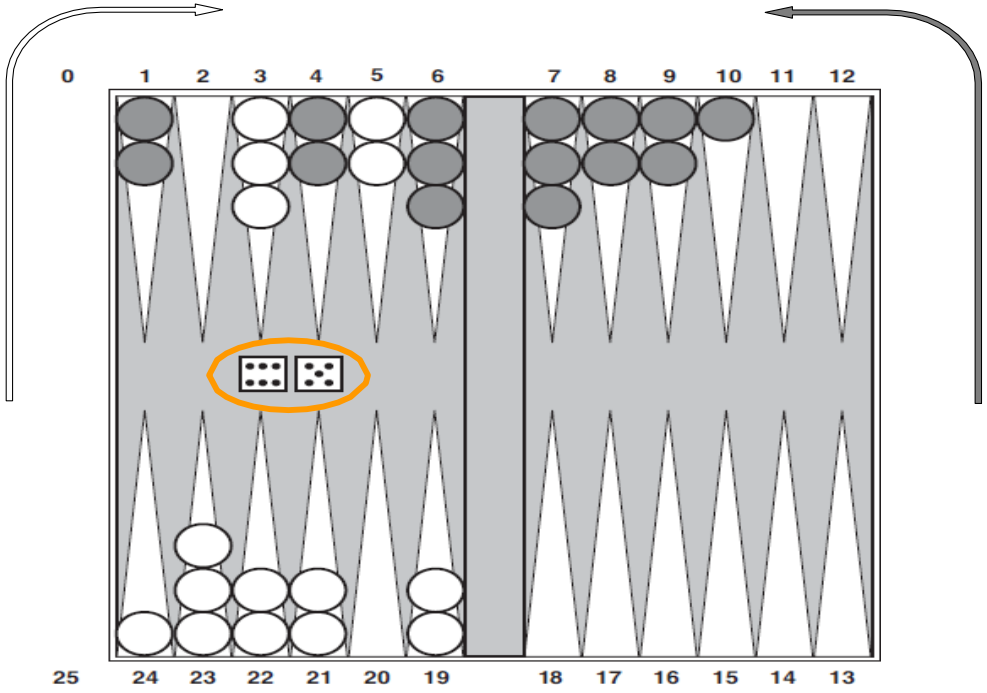
# 5.5 随机博弈

- 一种具有概率转换的动态博弈，有一个或多个玩家，于1950年代初提出的。
- 在现实生活中，许多无法预测的事件可以使我们陷入始料不及的处境。
- 许多博弈通过引入一种随机元素来仿照这种不可预测性，例如掷骰子。
- 西洋双陆棋
  - 走棋是根据掷骰子来决定的，在对手之前将所有的棋子移到棋盘外的玩家则获胜。
  - 随着每次掷骰子，玩家们必须从许多选项中选择如何移动棋子，并且要预见对手可能的对攻棋。



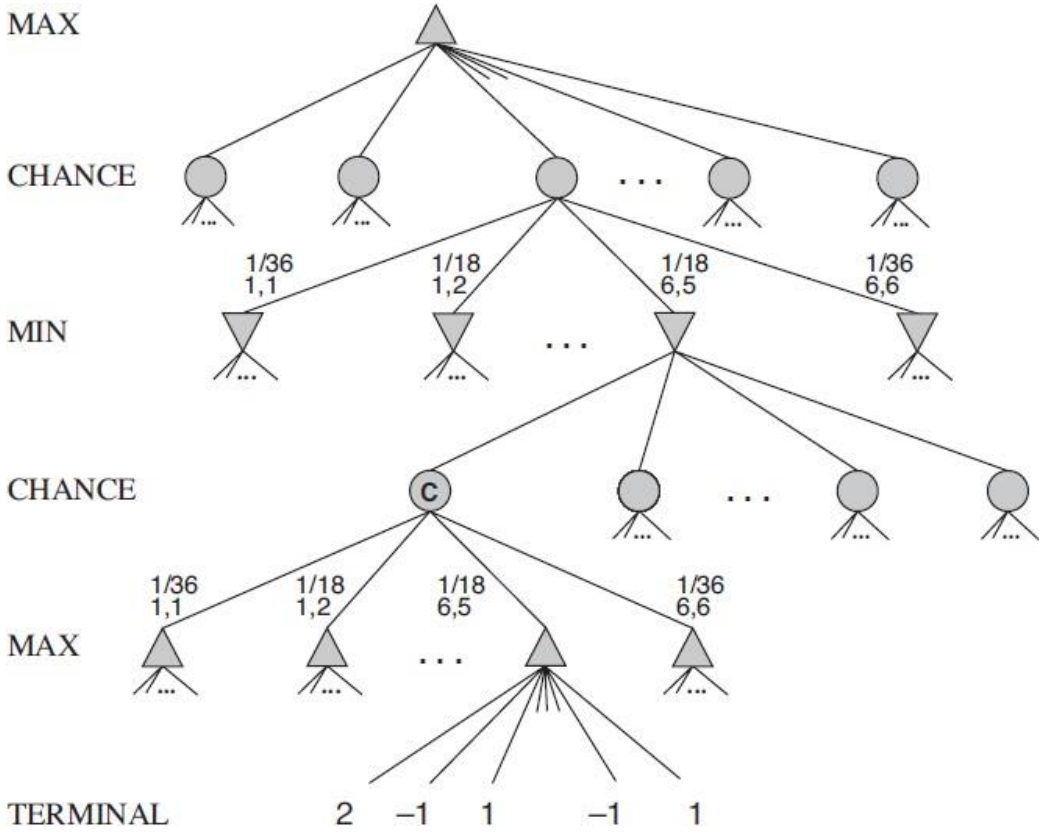
# 西洋双陆棋

- 白棋顺时针移到25，然后黑棋逆时针移到0。目标是将所有的棋子移到棋盘外。
- 一个棋子可以移到任意位置，除非在那里有多个对手的棋子；如果有一个对手的棋子，它就被抓住、然后必须重新开始。
- 如该棋局所示，白棋已经掷了6-5，因而必须从四种合法的走棋 中选择：
  - (5-10, 5-11),
  - (5-11, 19-24),
  - (5-10, 10-16),
  - (5-11, 11-16).
- (5-11, 11-16)指的是把棋子从5移到11，接着再移动到16



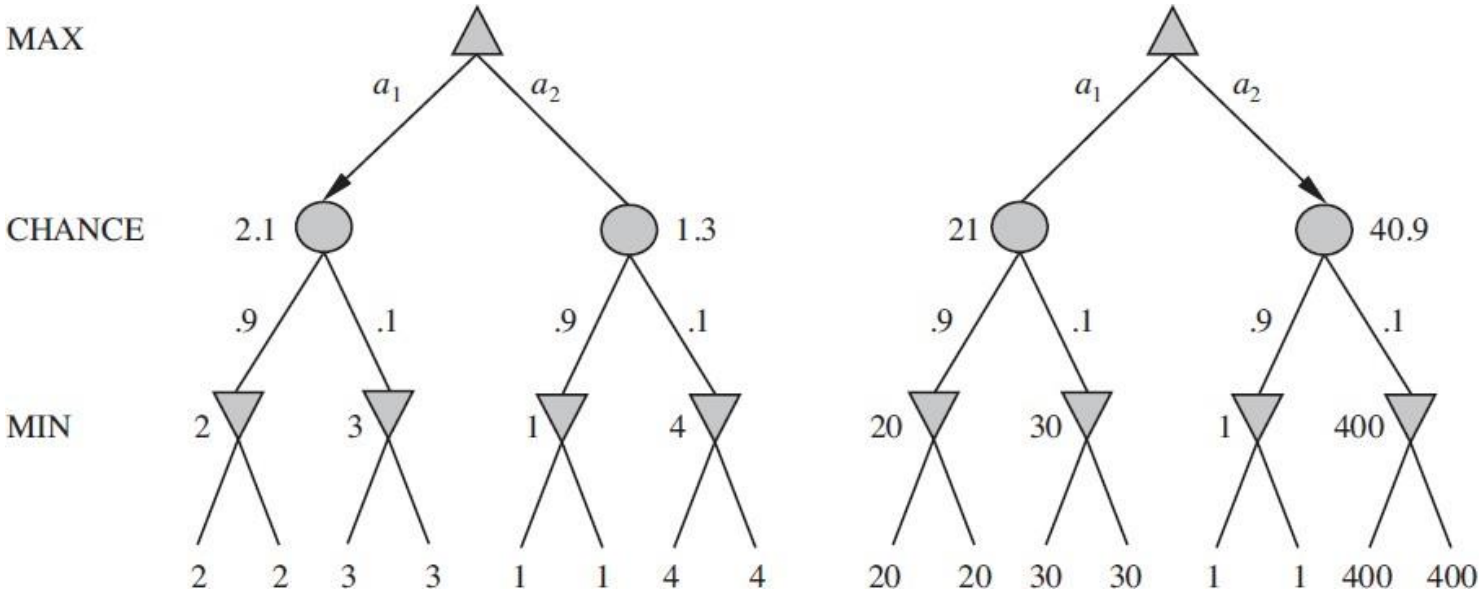
# 西洋双陆棋棋局的博弈树

- 投掷两个骰子有36种方式，每种都有同样可能；但是由于(6, 5)与(5, 6)相同，故仅有21种不同 的投掷。
- 六对儿双数，即 (1, 1) 到 (6, 6)，每对儿的概率为1/36，故：P(1, 1) = 1/36。
- 其他15种不同的掷骰子，每种有1/18的概率。



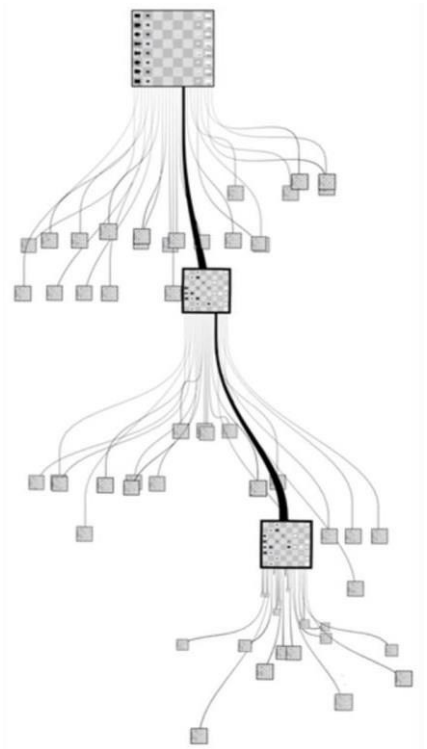
# 5.5.1 机会博弈

和极小极大值一样，期望极小极大值的近似估计可以通过在某结点截断搜索并对每个叶结点计算其评估函数来进行。你也许会认为像西洋双陆棋的评估函数应该和国际象棋的评估函数类似——对好棋局给予高分。但实际上，机会结点的存在意味着人们需要更加仔细地考虑评估值的含义。图 5.12 指出：叶结点的评估函数值为[1, 2, 3, 4]，$a_1$ 是最佳棋招；但如果评估值为[1, 20, 30, 400]，$a_2$ 是最佳棋招。可以看出，评估值取值范围不同，程序行棋会表现得完全不一样！为了避免这种敏感性，评估函数应该与棋局获胜概率（或者更一般的说，是棋局的期望效用值）成正线性变换。这在涉及不确定性的情况中是非常重要和普遍的特性，将在第 16 章中进行进一步讨论。



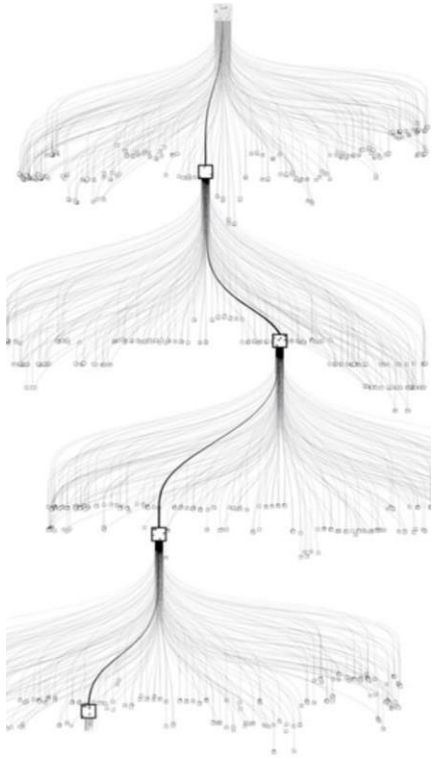
# 5.6 蒙特卡洛方法

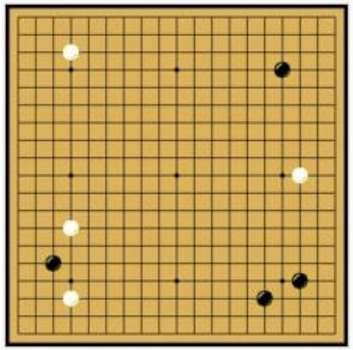
• 围棋一直被视为最复杂的博弈之一、而且是最具挑战性的AI经典博弈。



Chess (b ≈ 35, d ≈ 80)

8 x 8 = 64, possible games ≈ $10^{120}$





Go (b ≈ 250, d ≈ 150)

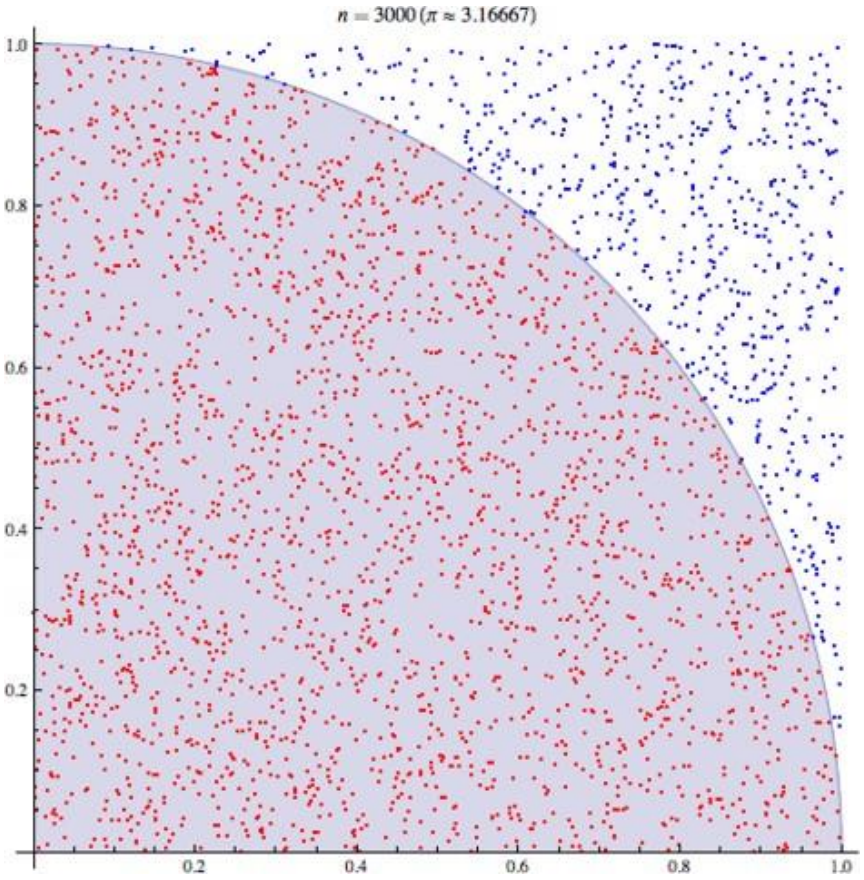
19 x 19 = 361, possible games ≈ $10^{170}$

# 蒙特卡罗方法

- 蒙特卡罗方法是一大类计算算法，它凭借重复随机采样来获得数值结果。
- 它们往往遵循如下特定模式：
  - 定义一个可能的输入域；
  - 从该域的一个概率分布随机地生成输入；
  - 对该输入进行确定性计算；
  - 将结果聚合。

# 用蒙特卡罗方法估计π

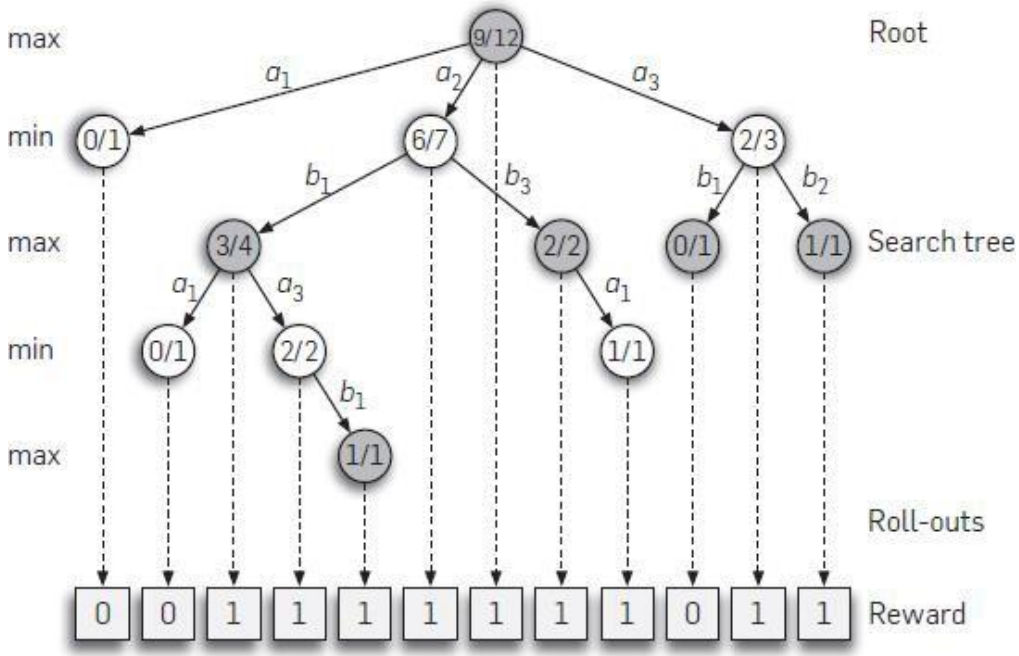
- 鉴于圆形与正方形面积之比为π/4，则π的值可采用蒙特卡罗方法近似得出：
- 先画出一个正方形，然后在其中画一个圆弧。
- 将尺寸大小一致的小颗粒散落在正方形上。
- 计算圆形和正方形中小颗粒的数量和总的数量。
- 两个数量之比为两个面积的估算，即π/4。结果乘以4得出π。



# 蒙特卡罗树搜索

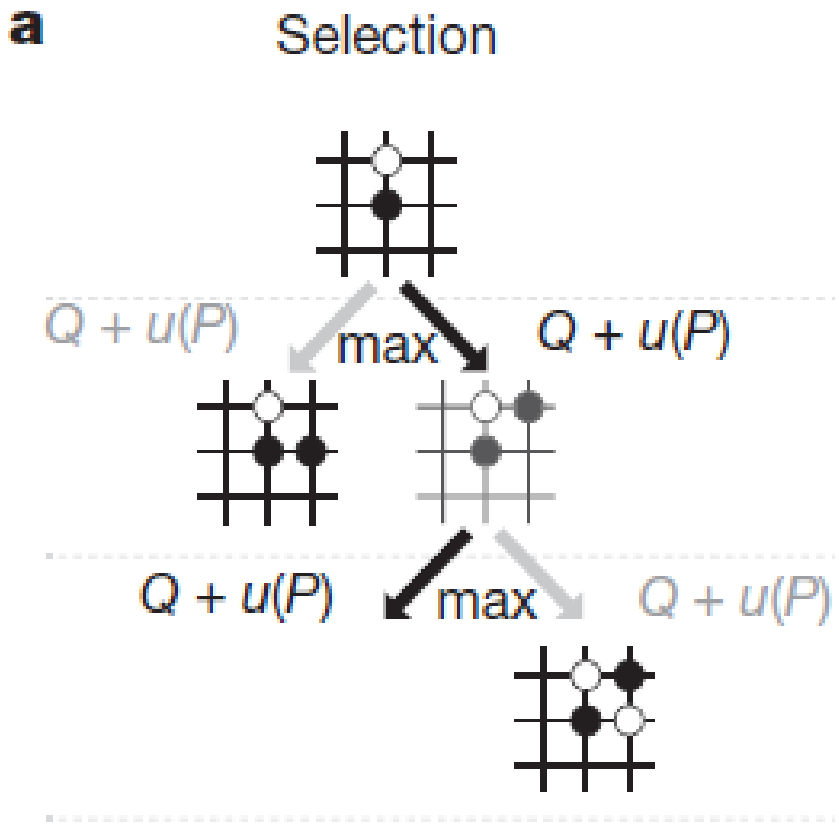
- MCTS将蒙特卡罗仿真与博弈树搜索相结合。
- 和minimax一样，每个节点对应于一个的博弈状态。
- 不同于minimax，节点的值通过蒙特卡罗仿真来估值。

- W(a)/N(a) = 动作a的值
  - W(a) = 总的奖励
  - N(a) = 仿真的数量



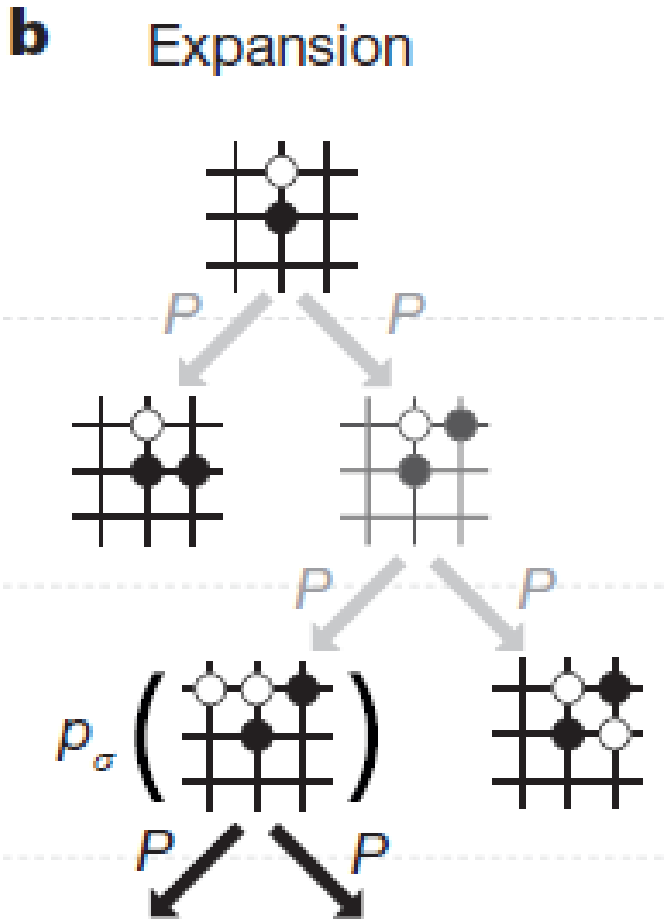
# AlphaGo的蒙特卡罗树搜索

• 选择：每次仿真通过选择边与最大动作值$Q$ + 奖励 $u(P)$ 对搜索树进行遍历，依赖于该条边存储的先验概率$P$。



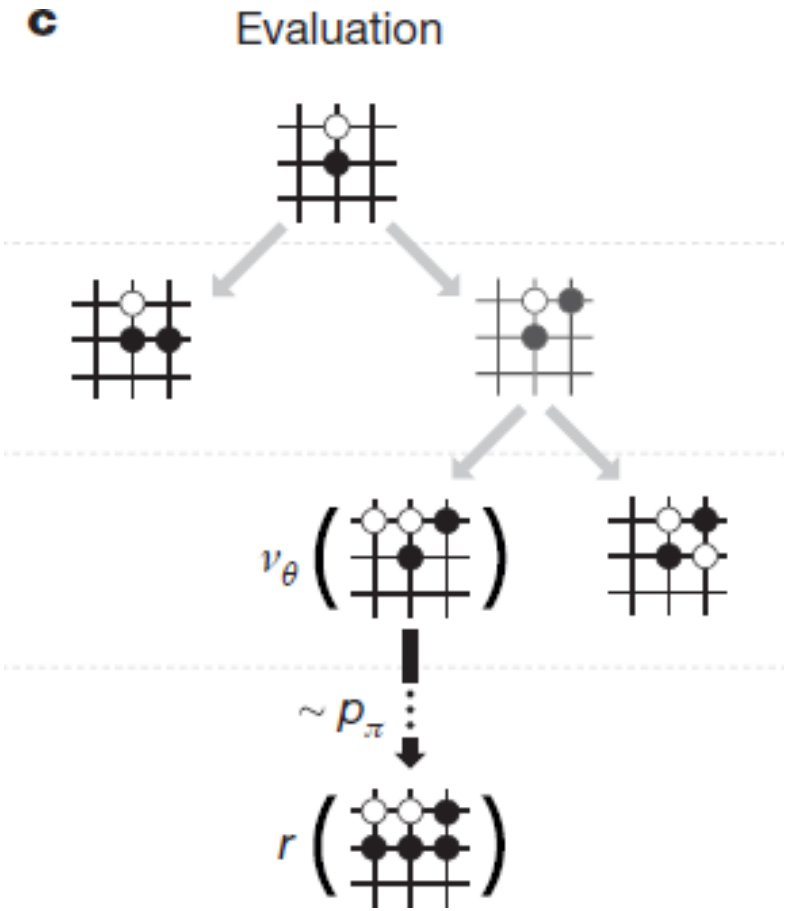
# AlphaGo的蒙特卡罗树搜索

• 扩展：叶节点可以扩展；新的节点先由策略网络 $p_\sigma$ 处理，然后其输出概率存储为每个动作的先验概率P。



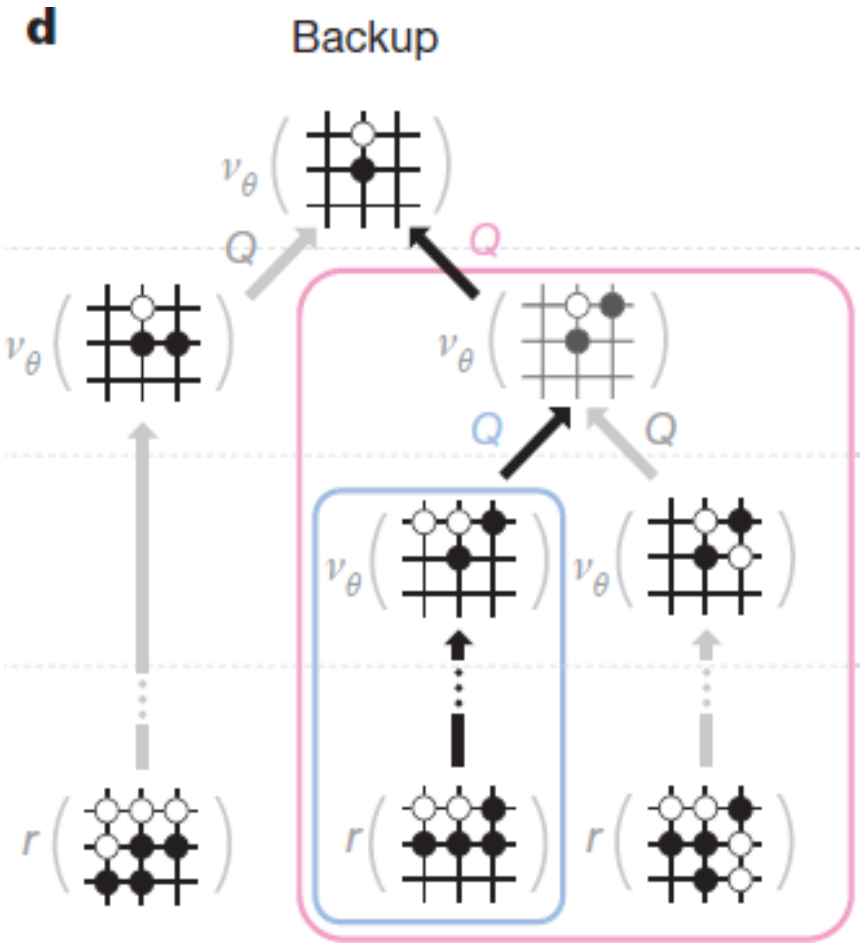
# AlphaGo的蒙特卡罗树搜索

- 评价（仿真）：叶节点用两种方法评价：
  - 1) 使用价值网络$v_\theta$；
  - 2) 使用快速走子策略$p_\pi$运行到博弈结束，然后用函数r计算出胜者。



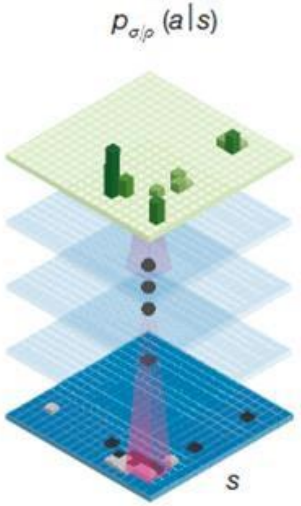
# AlphaGo的蒙特卡罗树搜索

• 后援（反向传播）：更新动作值Q来跟踪在该动作下面子树的所有评价函数 r 和 $v_\theta$ 的平均值。



# AlphaGo的神经网络架构

- 策略网络：用于评估棋局
  - 输入：棋盘位置s
  - 将s穿过具有参数σ或ρ的卷积层
  - 输出：一个合法走子a的概率分布。




- 价值网络：用于选择走棋
  - 输入：棋盘位置s′
  - 同样采用具有参数θ的卷积层
  - 输出：一个预测期望输出的标量值$v_\theta(s')$