第 8 章 蒙特卡罗博弈方法

计算机博弈理论的研究希望计算机能够像人一样、思维、判断和推理，并能够做出理性的决策。棋类博弈由于规则明确、竞技性高，且人类选手往往胜于计算机等原因，在计算机博弈理论的研究过程中一直受到重要关注和深入的探讨，并促进了计算机博弈理论的发展。传统的基于博弈树搜索和静态评估的博弈方法在国际象棋、中国象棋等棋类项目中获得了明显的成功，该类项目的盘面估计与博弈树搜索过程相对独立，棋子在盘面中的作用相对明确，且棋局中的专家规则相对较为容易概括和总结。

然而传统的博弈理论在计算机围棋博弈中遇到了明显的困难：围棋具有巨大的搜索空间；盘面评估与博弈树搜索紧密相关，只能通过对将来落子的可能性进行分析才能准确地确定棋子之间的关系；与此同时，高层次的围棋知识也很难归纳，归纳之后常有例外，并且在手工构建围棋知识和规则的过程中常会出现矛盾而导致不一致性。这些独特的因素为围棋及拥有类似性质的计算机博弈问题研究带来了新的挑战。

从 2006 年开始，计算机围棋博弈的相关研究有了跨越式的发展，基于蒙特卡罗模拟的博弈树搜索算法获得了重要的成功，并开始逐步引领计算机博弈理论研究的方向。在本章，我们将介绍蒙特卡罗博弈理论及其在围棋等棋类博弈中的应用。

8.1 基本概念

8.1.1 马尔科夫决策过程

马尔科夫决策过程是序贯决策过程的主要研究领域之一，一个序贯决策过程包括以下几点：

1. 所有的决策时刻点集；
2. 系统的所有可能的状态集合；
3. 可以采用的全体行动集合；
4. 与状态和行动相关联的既得回报或费用集合；
5. 与状态和行动相关联的转移概率的集合。

一般来讲，我们总认为决策者在开始做决策的时候这些量是已知的，在此基础上，马尔科夫决策过程是一类特殊的序贯决策问题，其特点是可采用的行动集、既得回报和转移概率只是依赖于当前的状态和选取的行动，而与过去的历史无关。一

个马尔科夫决策过程的主要组成包括：决策周期、状态、行动、转移概率和报酬。作为决策者，所面对的问题根据系统的转移概率抓住特定的机会，适时的做出一系列的行动或选择，以期达到决策者心中的某种最优化目标。由于受控制的系统在持续发展，过去的决策可以通过状态的转移影响到当前的决策，一般来讲，当前一步的最优选择不一定是全局最优的决策，必须要考虑系统在将来状态上的预期机会和费用。

决策时刻与决策周期

选取行动的时间点被称为决策时刻，并用T记录所有决策时刻的点集。T是非负实直线上的子集，它可以是有限点集、可列无限点集或者是连续集合。在T为离散的情况下，决策都是在决策时刻做出的；而在T为连续集的情况下，系统可以连续的做决策，也可以在某些随机点或某些事件发生时做决策，甚至由决策者选择时机做出决策等。

对于离散时间问题，两个相邻的决策时刻被称为决策周期或者阶段，我们把有限阶段的决策时刻集记为$T = \{0,1,2,\dots,N\}$，而把无限阶段的决策时刻记为$T = \{0,1,2,\dots\}$。

状态与行动集

在每一个决策时刻上对系统的唯一描述符就是“状态”，记博弈系统的所有可能状态集合为S，S也被称为“状态空间”。如果在任一个决策时刻，决策者所观察到的状态为$i \in S$，则他可以在状态i的可用行动集$A(i)$中选取行动$a \in A(i)$，其中$A(i)$也称为当前状态的“行动空间”。令：

$$A = \bigcup_{i \in S} A(i)$$

并且假设S和$A(i)$都不依赖于时刻t，状态集合S和行动集合$A(i)$可以是任意的有限集合、可数的无限集合、有限维欧氏空间的紧致子集或者是完备可分度量空间上的博雷尔（Borel）子集，除非特别声明，我们总考虑S和$A(i)$均为离散集的情况。

行动的选取可以是确定性的选取一个，也可以在多个允许的行动中随机性地选取。我们记$\mathcal{P}(A(i))$为$A(i)$的博雷尔子集上的所有概率分布，记$\mathcal{P}(A)$为A的博雷尔子集上的所有概率分布，随机选取行动就是选取一个概率分布$P(\cdot) \in \mathcal{P}(A(i))$，其中，选取行动$a \in A(i)$的概率是$P(a)$，如果这个分布是退化的，则为确定性地选择行动。

转移概率和报酬

对于任意一个决策时刻，在状态i采取行动$a \in A(i)$之后将产生两个结果，一是决策者获得报酬$r(i,a)$；二是下一个决策时刻系统所处的状态将由概率分布$P(\cdot\,|i,a)$

决定。

报酬$r(i,a)$是定义在$i \in S$和$a \in A(i)$上的实值函数，当$r(i,a)$为正值时，表示决策者所获得的收入，当其为负值时，表示决策者所付出的费用。从模型的角度来看，报酬$r(i,a)$是即时的，但是在这个决策周期内它是何时或如何获得的并不重要，在选取行动后，模型只需知道它的值或期望值。实际上，报酬可以包括到下一个决策时刻的一次性收入、持续到下一阶段的累积收入，或转移到下一个状态的随机收入等。一般来讲报酬还依赖下一个决策时刻的状态j，即$r(i,a,j)$。此时，采取行动a的期望报酬值为：

$$r(i,a) = \sum_{j \in S} r(i,a,j)P(j|i,a)$$

上式中非负函数$P(j|i,a)$是下一个决策时刻系统转移到状态j的概率，函数$P(\cdot|i,a)$被称为转移概率函数。需要注意的是，在一般的实际问题中，状态转移是可以发生在两个决策时刻的中间的，但是在不影响决策的情况下，我们的模型依然适用。通常我们假设：

$$\sum_{j \in S} P(j|i,a) = 1$$

我们把五重组$\{T,S,A(i),P(\cdot|i,a),r(i,a)\}$称为一个“马尔科夫决策过程”，其转移概率和报酬仅仅依赖于当前的状态和决策者选取的行动，而不依赖于过去的历史。这里我们把包括了最优准则的马尔科夫决策过程称为“马尔科夫决策问题”。

8.1.2 围棋落子模型

由于围棋是一种策略性二人棋类游戏，棋手在相互对弈的过程中所下的每一步棋，都是经过深思熟虑、精密决策的结果。在对弈时，棋手不仅要考虑当前所下棋子取得的效果，也要照顾到长远的利益。同时，棋手在下棋的过程中只针对当前盘面进行决策，对于同样的盘面，棋手不用去考虑其中经历的不同步骤。可以说，棋手在下定一手棋后所能获得的收益，只与当前棋盘的状态和即将选取的行动有关，而与以往的历史没有关系。所以围棋落子的过程应被看成马尔科夫决策过程。

我们知道， 马尔科夫决策过程可以用五重族$\{T,S,A(i),P(\cdot|i,a),r(i,a)\}$表示，围棋落子过程也不例外：

决策时刻T：显然地，围棋是一个有限阶段的决策问题，在有限步对弈后，就能看到决策的结果。设一盘棋的总行棋步数为N ，则在$[1,N]$的时间内，黑白双方交替进行决策。由于黑方先行，所以在奇数时刻黑方进行决策，而在偶数时刻白方进行决策。

状态空间S：记$s = (B(m),W(n))$为状态，其中向量$B(m) = (p_{b1},p_{b2},\dots,p_{bm})$描

述了到目前为止盘面上所有黑棋的位置，向量$W(n)=(p_{w1},p_{w2},\dots,p_{wn})$描述了到目前为止盘面上所有白棋的位置。从前面的解释我们可以知道，围棋的状态空间S是相当大的。

可用行动集$A(s)$： 定义为在盘面s下的所有可落子点的集合，如果无任何可落子点，则$A(s)=\emptyset$。

转移概率$P(\cdot|s,a)$：在给定状态和行动集（可落子点）下，转移概率决定了每一个行动（选择哪个落子点）被选择的概率，原则上其定义方式没有绝对限制，但是其定义与每个落子点的价值紧密相关。如前所述，围棋中每一个落子的潜在价值较为难以估计，为转移概率的定义带来了一定的难度。简单地，我们可以定义如下的等概率模型：

$$P(\cdot|s,a)=\begin{cases}1，\text{ 如果}A(s)=\emptyset，\text{即没有任何可落子点}\\ \dfrac{1}{M}，\text{如果}|A(s)|=M，\text{即有}M\text{个可落子点}\end{cases}$$

在该模型中，我们认为每一个可落子点被选中的概率是相等的，这样的假设前提是下棋者完全没有领域内的经验知识。实际上，经验可以指导我们以更高的概率选择更容易获胜的点作为最终的行棋。但是，由于围棋经验的好坏难以定量衡量，因此我们很难给出加入经验后各可行状态的转移概率。所以，我们在建立马尔科夫决策模型时，只简单的考虑从当前状态等概地转移到下一个可行状态的情况。

报酬：R_b表示到目前为止黑棋所占领地域的大小，R_w表示到目前为止白棋所占领地域的大小。围棋落子模型是一类较特殊的马尔科夫决策模型，因为在整个决策过程中所有的报酬并不累加为最后的总报酬，而只有最后一次决策后双方获得的报酬才是最后的总报酬，但这不影响决策时刻争取较高报酬的重要性。

8.2 蒙特卡罗方法及模拟评估理论

蒙特卡罗算法以及基于蒙特卡罗随机模拟的局面评估方法构成了蒙特卡罗博弈理论的基础。在本部分，我们将首先介绍蒙特卡罗算法，并以计算机围棋博弈为例介绍其在计算机博弈系统中的具体应用。

8.2.1 蒙特卡罗方法

蒙特卡罗(Monte-Carlo)方法也称为随机模拟方法，有时也称作随机抽样技术或统计试验方法。它的基本思想是，为了求解数学、物理、工程技术以及生产管理等方面的问题，首先建立一个概率模型或随机过程，使它的参数等于问题的解，然后

通过对模型或过程的观察或抽样试验来计算所求参数的统计特征，最后给出所求解的近似值。

蒙特卡罗方法可以解决各种类型的问题，但总的来说，用蒙特卡罗方法处理的问题视其是否涉及随机过程的性态和结果可以分为两类：

第一类是确定性的数学问题，用蒙特卡罗方法求解这类问题的步骤是，首先建立一个与所求解有关的概率模型，使所求的解就是我们所建立模型的概率分布或数学期望，然后对这个模型进行随机抽样观察，即产生随机变量，最后用其算术平均值作为所求解的近似估计值。计算多重积分、求解逆矩阵、解线性代数方程组、解积分方程、解某些偏微分方程的边值问题和计算微分算子的特征值等都属于这一类。

第二类是随机性问题的模拟，例如中子在介质中的扩散等问题，这是因为中子在介质内部不仅受到某些确定性的影响，而且更多的是受到随机性的影响。对于这类问题，虽然有时可表示为多重积分或某些函数方程，并进而可考虑用随机抽样方法求解，然而一般情况下都不采用这种间接模拟方法，而是采用直接模拟方法，即根据实际物理情况的概率法则，用电子计算机进行抽样试验。原子核物理问题、运筹学中的库存问题、随机服务系统中的排队问题、动物的生态竞争和传染病的蔓延等都属于这一类。

在应用蒙特卡罗方法解决实际问题的过程中，往往需要重点考虑如下几个方面的内容：

1. 对求解的问题建立简单而又便于实现的概率统计模型，使所求的解恰好是所建立模型的概率分布或数学期望；
2. 根据概率统计模型的特点和计算实践的需要，尽量改进模型，以便减小方差和降低费用，提高计算效率；
3. 建立对随机变量的抽样方法，其中包括建立产生伪随机数的方法和建立对所遇到的分布产生随机变量的随机抽样方法；
4. 给出获得所求解的统计估计值及其方差或标准误差的方法。

下面，我们将通过蒙特卡罗方法的几个典型应用对其有一个更深的理解。

蒲丰投针问题

1777 年法国科学家蒲丰提出的一种计算圆周率π的方法，即随机投针法，这就是著名的葡丰投针问题。这一方法的步骤是，取一张白纸，在上面画上许多条间距为d的等距平行线，另取一根长度为$l(l<d)$的针，随机地向画有平行直线的纸上投掷n次，并观察针与直线相交的次数，记为m。

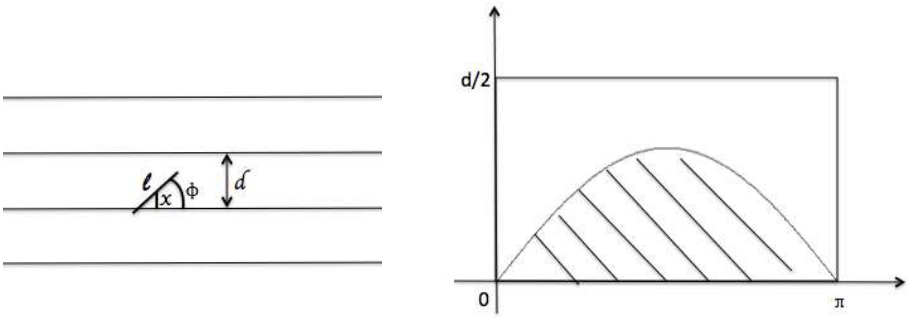

图 8.1 蒲丰投针示意图及其概率模型

设针与平行线的夹角为φ，针的中点到平行线的最近距离为x，如图 8.1 左所示，可知针与平行线相交的充要条件为$x \le \frac{l}{2}sin\varphi$。另一方面，我们有$0 \le x \le \frac{d}{2}$和$0 \le \varphi \le \pi$，从而全概率空间为$x-\varphi$平面上的一个矩形，如图 8.1 右所示，其中满足相交性的部分为$x \le \frac{l}{2}sin\varphi$部分，如阴影所示。因而阴影部分的面积所占的比例即为针和平行线相交的概率，即:

$$P = \frac{\int_0^{\pi} \frac{l}{2} sin\varphi d\varphi}{\frac{d}{2}\pi} = \frac{2l}{\pi d}$$

在投针实验中，针与平行线相交的频率为$\hat{P} = \frac{m}{n}$，由$P \approx \hat{P}$，可得π的估计值为：

$$\pi \approx \frac{2nl}{md}$$

一维定积分的计算

在单位正方形内等概率随机投点N次，其中第i次投点的坐标为(x_i, y_i)，如果投点满足不等式$y_i \le f(x_i)$，即该点落在曲线$y = f(x)$下，则认为实验成功，并记录下投点次数，反之，则认为该次实验失败。

用蒙特卡罗的语言来讲，就是产生均匀随机数ξ_1, ξ_2，如果$\xi_1 \le f(\xi_2)$，则认为实验成功，如果$\xi_1 > f(\xi_2)$，则认为实验失败。设实验成功的概率为I，若在N次试验中有n次成功，则比值$\frac{n}{N}$给出I的一个无偏估计值：

$$I \approx \frac{n}{N}$$

设随机变量：

$$\eta(\xi_1, \xi_2) = \begin{cases} 1, \xi_1 \le f(\xi_2) \\ 0, \xi_1 > f(\xi_2) \end{cases}$$

则有$I = E[\eta(\xi_1, \xi_2)]$，而与此同时：

$$E[\eta(\xi_1, \xi_2)] = \int_0^1 \int_0^1 P\{\xi_1 \le f(\xi_2)\} d\xi_1 \, d\xi_2 = \int_0^1 f(\xi_2) d\xi_2 = \int_0^1 f(x) dx$$

从而：

$$I = \int_0^1 f(x)dx \approx \frac{n}{N}$$

当抽样点数为N时，使用此种方法所得近似解的统计误差只与N有关（与$\frac{1}{\sqrt{N}}$正相关），不随积分维数的改变而改变。因此当积分维度较高时，蒙特卡罗方法相对于其他数值解法更优。

8.2.2 蒙特卡罗评估

基于蒙特卡洛模拟的局势评估方法，可以与传统的静态局面评估方法相比对。在计算机围棋中，我们常常利用类似于一维定积分中的掷点法完成对围棋盘面的评估。具体来讲，当我们给定某一个棋盘局面时，算法在当前局面的所有可落子点中随机选择一个点摆上棋子，并不断重复这个随机选择可落子点（掷点）的过程，直到双方都没有可下点（即对弈结束），再把这个最终状态的胜负结果反馈回去，作为评估当前局面的依据。

当然简单的随机掷点可以保证良好的随机效果，但是由于随机性很强会影响做出正确评估的收敛速度；特别地，由于围棋的状态空间非常巨大，我们所能搜索的是状态空间的一个小部分，所以如果能在随机策略中加入启发式知识将加快收敛速度，因此，在随机选点的过程中，一些策略也可被加入进来，并可以根据不同策略的紧要程度来排列这些策略在随机过程中作用的先后顺序，这些策略本身也可以根据其重要程度被赋予不同的随机性。

通过对蒙特卡罗评估的介绍我们可以发现，以蒙特卡罗方法为基础的评估方式更加适用于围棋博弈过程。静态评估方法尽管可以达到一定的效果，但由于围棋棋局情况相当复杂且规模庞大，单单采用一套固定的静态的规则去覆盖所有的情况必然会由于各种例外情况的出现而产生出各种不足。

更为重要的是，围棋规则的描述不可能非常的准确。围棋中每个棋子的作用都差不多，而棋子的影响力更多的取决于它周围的环境，以及该棋子连接或距离较近的己方棋子和对方棋子都是决定其影响力的因素，这使得围棋中的很多情况甚至连职业棋手也很难精确的描述，而只能具体情况具体处理。另外，过度复杂的规则不仅会让机器处理起来效率低下，而且也会超出人们对系统实际运行效果的理解程度。

因此，对于围棋、五子棋及类似棋类博弈游戏，蒙特卡洛评估相比于静态评估方法具有明显的优势。它通过对当前局面下的每个的可选点进行大量的模拟来得出相应的胜负的统计特性，在简单情况下，胜率较高的点就可以认为是较好的点予以选择。

8.3 蒙特卡罗树搜索

蒙特卡罗树搜索是解决马尔科夫决策问题的有效方法之一，它是以蒙特卡罗方法的思想为基础的一种搜索方法。在蒙特卡罗树搜索中，可以将马尔科夫决策过程中可能出现的状态转移过程用状态树建立并表示出来。不同于普通搜索树的建立，蒙特卡罗树搜索通过从初始状态开始重复给出随机模拟事件，并逐步扩展搜索树中的每个节点，而每一次模拟事件都是“状态-行为-回报”的三元序列。因此，相对于在评估之初就将（隐含）博弈树进行展开的静态方法而言，蒙特卡罗树搜索的过程是一种动态的搜索过程，蒙特卡罗树搜索的基本思想如图 8.2 所示。

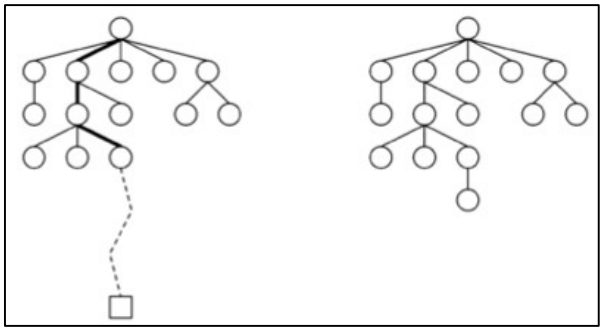

图 8.2 蒙特卡罗树搜索过程示意

在该过程中，算法首先根据一定的策略在搜索树上选择一个节点用于扩展，该策略被称为搜索树策略(Tree Policy)。该策略往往需要综合考虑两个方面的因素：一是对尚未充分了解的节点的探索(exploration)，以尽早发现潜在价值高的节点并尽早排除潜在价值低的节点；二是对当前有较大希望的节点的利用(exploitation)，以尽可能地抓住机会创造对己方更有利的局势。

当确定了扩展节点之后，算法以该节点为根进行大量的随机模拟，并根据模拟的结果确定在该根节点下如何落子，并更新祖先节点的估计值，该策略一般称为模拟策略或默认策略(Default Policy)。在平凡情况下，算法在每一次模拟中从根节点的状态开始，随机地选择一个落子点x进行落子，接下来不断地交替随机落子直至棋局结束，并根据结束时双方的胜负状态来确定当前状态下落子点x的估计值。当完成大量的模拟之后，对每一种落子点的估计值将变得更为准确，算法以此来决定最终的落子，并向上依次更新祖先节点的估计值。

更具体来讲，蒙特卡罗树搜索共包含四个基本步骤，分别为选择(selection)、扩展(Expansion)、 模拟(Simulation)和回溯(Back propagation)，如图 8.3 所示。图中的每

个节点表示博弈过程中的一个盘面状态，每一条边表示在父节点上采取一个行动，并得到子节点所对应的状态。这四个基本步骤依次执行，从而完成一次搜索：

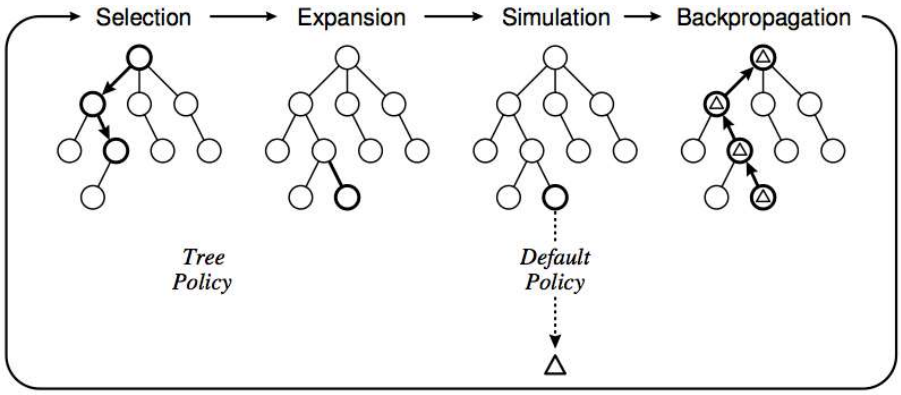

图 8.3 蒙特卡罗树搜索的流程

1. 选择：从根节点出发，在搜索树上自上而下迭代式执行一个子节点选择策略，直至找到当前最为紧迫的可扩展节点为止，一个节点是可扩展的，当且仅当其所对应的状态是非停止状态，且拥有未被访问过的子状态；
2. 扩展：根据当前可执行的行动，向选定的节点上添加一个（或多个）子节点以扩展搜索树；
3. 模拟：根据默认策略在扩展出来的一个（或多个）子节点上执行蒙特卡罗棋局模拟，并确定节点的估计值；
4. 回溯：根据模拟结果向上依次更新祖先节点的估计值，并更新其状态。

算法 1：蒙特卡罗树搜索（MCTS）

function MCTSSEARCH(s_0)
　以状态s_0创建根节点v_0;
　while 尚未用完计算时长 **do**:
　　v_l ←TREEPOLICY(v_0);
　　Δ←DEFAULTPOLICY($s(v_l)$);
　　BACKUP(v_l, Δ);
　end while
　return a(BESTCHILD(v_0));

算法 1 描述了蒙特卡罗搜索的伪代码，在该算法中，节点v_0是与初始状态s_0相

对应的根节点，$s(v)$表示节点v所对应的状态。算法首先从根节点开始调用搜索树策略并返回该过程中所确定的待扩展节点v_l及其对应的状态s_l；接下来，算法从状态s_l开始调用默认策略进行随机模拟，返回报酬Δ，并更新v_l及祖先节点的估计值；算法最终采取行动a，并返回为根节点v_0计算得到的最优子节点 BESTCHILD(v_0)，当然，这里的“最优”与博弈的具体情况和算法的具体实现有关。

采用蒙特卡洛树搜索的一个重要原因是，它可以保证我们在从始至终的每一次随机模拟中随时得到行为的评价。因此，如果在模拟过程中某个状态被再次访问到，我们便可以利用之前的“行为-回报”信息来为接下来的选择做参考，借此便可以加速评价的收敛速度。一方面，如果被重复访问到的状态很少，那么蒙特卡洛树搜索就退化成非选择性的蒙特卡洛方法；另一方面，如果被重复访问到的后续状态在很大程度上都集中在少数几个状态上，那么相对于其他算法而言，蒙特卡洛树搜索则具有明显的优势。

具体到围棋问题中，就是属于上面谈到的第二类情况，在每个当前状态下，可选择的点最多不超过 361 个，而这些可选点中基本可以找到一些明显偏好的点。因此，当模拟次数达到几万次甚至几十万上百万次的时候，在这些较好的点上必然会聚集大量的模拟。下面将以围棋为例，给出一个蒙特卡罗树搜索过程的示例性实现。

当棋手面对一个待决策盘面时，他需要从多个可下点中选择一个行棋，因此，如果我们有足够的时间做模拟，则我们可以做以下的统计：设轮到一方进行决策时共有k个点可供选择,第i个节点具有参数n_i和w_i，分别表示到该次模拟为止第i个节点被选中的次数，以及在其所对应的这n_i次模拟中第i个节点获得胜利的次数。

当轮到算法给出落子时，我们将待决策的盘面作为某子树的根节点，然后在可下点中随机选择一点$p_i, i \in [i,k]$并进行以下模拟过程：

1. 如果该节点第一次被选中，即$n_i = 0$，则将填入该节点后的盘面作为叶节点加入搜索树中并采用随机投点的方式完成之后的行棋过程。在这一过程结束之后n_i自动加 1，如果模拟至终盘得到胜利的结果则w_i加△（△为收益，如果以是否获胜作为收益，则胜的收益为 1，负的收益为 0）。接着我们将叶节点的信息返回给上层节点，即让其父节点的访问次数加 1，收益加上其子节点收益变化值的负值 1-△（这是因为父节点对应于对方落子的情况）。如果父节点仍有父节点的话，则其父节访问次数加 1，获得的收益加上其子节点收益变化值的负值 1-（1-△），即△。依此类推，直至根节点为止，该次模拟结束并进行新一次的模拟；
2. 如果该点不是第一次被选中，即$n_i \neq 0$，亦即填入该节点后的盘面已作为叶节点加入搜索树中，则我们以该盘面为子树的根节点再一次执行构

建搜索树的过程。

在模拟时间结束后，我们可以简单地计算根节点的每个儿子可下点的模拟收益率：

$$P(win) = \frac{w_i}{n_i}$$

而在k个可下点中收益率最高的可下点即为该次决策的结果。通过以上方法，就可以简单的实现计算机围棋博弈的选点落子过程。当然，这种解决过程因为蒙特卡罗树搜索的收敛缺乏效率而并没在计算机围棋博弈中直接使用，我们在下一节将介绍以此为基础的更加优秀的算法。

8.3 UCB 和 UCT 算法

8.3.1 多臂老虎机模型

我们都知道赌场里有一种赌博机器名叫老虎机，当我们将赌注放入某个老虎机并拉动手臂后，就会出现或好或坏的收益结果。多臂老虎机拥有k个手臂，拉动每个手臂所能产生的回报互不相关，而赌博者的目的就是为了获得尽可能多的收益，在多臂老虎机模型中也是如此，我们希望找到一个合理的策略，可以使得拉动手臂的人获得最大的收益。

赌博者要从这k个手臂中选择一个手臂进行操作来获得回报，这个回报可能为正值、0 或者负值；每个手臂的回报所遵循的分布方式是不尽相同的，而拉动同一个手臂所获得的回报满足某一特定的分布；并且在某一个特定的时间段内，赌博者只能拉动有限次数，而我们要做的就是在有限的拉动次数中找到一个策略，使得赌博者可以根据这个策略在规定的最后一次拉动中获得尽可能大的回报。

可以想象，在游戏开始之前所有的手臂对玩家来说都是等价的，若想知道哪个手臂最好，就需要不断的去试探，并通过不断的试探发现规律，以此来推断出哪个手臂可能获得最大的回报。由于玩家所拥有的试探次数是有限的，因此不得不在探索和利用间寻求一个平衡点，探索就是通过进行更多的试探以获得更多的知识，这包括尽可能排除收益低的手臂和尽可能发现收益高的手臂，而利用则是充分利用当前已经获得的知识（对每个手臂所对应的收益高低的判断）获得尽可能大的回报。

该模型所蕴含的在搜索过程权衡探索和利用的基本思想是对蒙特卡罗树搜索方法进行改进的核心内容之一。在下面的内容中，我们将介绍该思想下的两个典型算法信心上限算法 (Upper Confidence Bound, UCB)和信心上限树算法 (Upper Confidence Bounds for Trees, UCT)，它们构成了现代蒙特卡罗博弈理论的基础算法。

8.3.2 UCB 算法

以上的多臂老虎机问题可以抽象为如下的数学模型：一个有k个臂的老虎机被一系列的随机收益值所表示$\{X_{1t}, X_{2t}, \dots, X_{it}, \dots, X_{kt}\}$，其中$1 \le i \le k, t \ge 1$。这里的$i$表示第$i$个臂的编号，拉动第$i$个臂，所得到的回报序列为$\{X_{i1}, X_{i2}, X_{i3}, \dots\}$，且不同的手臂的收益分布之间没有相互关系。

解决多臂老虎机问题实际上就是选择一个策略A去决定下一次将被拉动的手臂的编号，这个决策既取决于过去每个臂已经被拉动的次数，也受限于这些被拉动的次数中每个手臂上取得的收益值。假设$T_j(n)$表示当总的拉动次数为n时，第j号手臂被拉动的次数，那么当进行n次拉动后根据当前策略A所产生的损失可以表示为：

$$\rho = n\mu^* - \sum_{j=1}^{k} E[T_j(n)]\,\mu_j$$

其中μ^*表示收益最好的手臂的收益均值，μ_j为第j号手臂被拉动后所产生的收益均值。在 Lai 和 Robbins 的 1985 年的关于该问题的经典文章中讲到，对于某一特定的回报分布，拉动策略满足：

$$E\big[T_j(n)\big] \le \frac{\ln(n)}{D(P_j||P^*) + o(1)}$$

其中，当$n \to \infty$时，有$o(1) \to 0$,并且：

$$D(P_j||P^*) = \int P_j ln \frac{P_j}{P^*}$$

为在任一次尝试中，手臂j的回报分布概率密度P_j和当前回报最高的手臂的概率密度P^*之间的 KL 测度。因而在尝试的次数（亦即随机模拟的次数）足够多时，最优的手臂被拉动的次数和其它手臂被拉动的次数之间有较大的差距，具体而言，次优的手臂j被拉动的次数满足：

$$E\big[T_j(n)\big] \le \frac{\ln(n)}{D(P_j||P^*)}$$

我们还可以把损失表示为:

$$\rho = max_i E\left[\sum_{t=1}^{n} X_{it}\right] - E\left[\sum_{i=1}^{k}\sum_{t=1}^{T_i(n)} X_{it}\right]$$

从这个公式我们可以看出，造成损失的原因是我们选取的策略不一定会每次都选择收益最佳的臂。对于这样的一大类收益分布问题，已经证明没有任何策略能让损失的增长速度低于$O(\ln(n))$。对于这种收益分布，如果一个策略的损失增长速度不超过$Cln(n)$，其中C为一个常数，我们就认为它在探索和利用之间找到了一个平衡

方法。

UCB 算法即信心上界算法，是一类解决多臂老虎机问题的算法的总称，其中，我们将在本节介绍是 UCB1 算法是该类算法的代表，可以用如下伪码表示：

算法 2：信心上限算法（UCB1）

function UCB1

for each 手臂j:

访问该手臂并记录收益

end for

while 尚未达到访问次数限制 **do**:

计算每个手臂的 UCB1 信心上界I_j（如下所述）

访问信心上界最大的手臂

end while

UCB1 算法的核心在于解决继续探索和利用当前状态之间的平衡问题。它为所有臂记录了平均收益$\bar{X}_{i,T_i(t-1)}$，并选择具有最大置信上界的臂：

$$I_t = argmax_{1\le i\le k}\left\{\bar{X}_{i,T_i(t-1)} + c_{t-1,T_i(t-1)}\right\}$$

这里的偏移序列$c_{t,s}$定义为：

$$c_{t,s} = \sqrt{\frac{2\ln(t)}{s}}$$

在$X_{i,t}$独立同分布的条件下，偏移序列$c_{t,s}$满足如下的概率收敛不等式：

$$P\left(\bar{X}_{is} \ge \mu_i + c_{t,s}\right) \le t^{-4}$$

$$P\left(\bar{X}_{is} \le \mu_i - c_{t,s}\right) \le t^{-4}$$

在搜索的过程中，UCB1 被用来确定内部节点将要选择的下一个尝试的行为，我们可以用更简单具体的形式表示 UCB1 的信心上界索引：

$$I_j = \bar{X}_j + \sqrt{\frac{2\ln(n)}{T_j(n)}}$$

其中，$\bar{X}_j$是手臂j所获得的回报的均值，n是到当前这一时刻为止所访问的总次数，$T_j(n)$是手臂j到目前为止总共所访问的次数。

在 UCB1 算法中，当手臂数目$k > 1$时，如果k个手臂的回报在$[0,1]$区间上分别服从分布$P_1, P_2, \dots, P_k$，那么当总共拉动n次手臂之后，该算法的损失不超过以下上限：

$$\left\lceil 8 \cdot \sum_{i:\mu_i<\mu^*} \frac{\ln(n)}{\Delta_i} \right\rceil + \left(1+\frac{\pi^2}{3}\right) \cdot \left(\sum_{j=1}^{k} \Delta_j\right)$$

其中$\Delta_j = \mu^* - \mu_j$，$\mu_i (1 \le i \le k)$分别为分布P_i的数学期望，μ^*为$\mu_i (1 \le i \le k)$中的最大值。此外，次优手臂j的访问次数的数学期望满足以下上限：

$$E[T_j(n)] \le \frac{8}{\Delta_j^2} \ln(n)$$

观察 UCB1 算法中上限信心索引的计算公式$I_j = \bar{X}_j + \sqrt{\frac{2\ln(n)}{T_j(n)}}$，我们可以更加直观的理解其两部分的含义，其中前一部分$\bar{X}_j$就是对到目前为止已经搜集到的知识的价值，而后一部分则可以看作是尚未充分探索过的节点需要继续探索的必要性。

8.3.3 UCT 算法

虽然利用构造蒙特卡罗树搜索可以解决围棋博弈问题，但是由于蒙特卡罗树搜索在没有知识的指导时树的扩展层数较少，不利于最优解的获取，因此我们将 UCB1 算法加入蒙特卡罗树搜索的构建过程中，形成了我们即将要介绍的信心上限树算法(UCT)。

UCT 算法是将 UCB1 算法思想用于蒙特卡罗树搜索的特定算法，其与只利用蒙特卡罗方法构建搜索树的方法的主要区别如下：

1. 可落子点的选择不是随机的，而是根据 UCB1 的信心上界值进行选择，如果可落子没有被访问，则其信心上限值为正无穷大，如果可落子点已经被访问过，则其信心上限索引值可以根据 UCB1 算法给出的值来确定。在实际应用中，我们采用 UCB1 给出的信心上限值，当我们需要在众多可下点中选取一个时，我们选择信心上限值最大的那一个。
2. 当模拟结束后确定最终选择结果时，我们不再仅仅根据胜率做出判断，而是兼顾信心上限所给出的估计值。在实际应用中，选择方法也是多种多样的，一些策略中选择估计值最大的子节点为落子点，另外由于选择可落子点进行访问的过程已经兼顾了极小极大算法的思想，所以在另外一些策略中也经常直接选择那个被访问了最多次的可落子点，等等。

UCT 算法的基本过程可以用算法 3 中的伪代码来描述，在该算法中，每一个节点v包含四项基本信息，分别为其所对应的状态$s(v)$，所对应的来自父节点的行为$a(v)$，随机模拟收益$Q(v)$（例如获胜次数），以及节点的被访问次数$N(v)$。

在算法 3 所示的伪代码中，最终将返回具有最高收益的子节点所对应的行动，这是因为我们在返回值$a(\text{BESTCHILD}(v_0, 0))$中将 UCB1 中的常数$c$设为 0。在实际系

统中，算法也经常可以直接返回访问次数最多的子节点所对应的行动。需要指出的是，在实际系统中具有最高收益的子节点和访问次数最多的子节点往往是同一个节点，但是这一点并不能保证。在基于蒙特卡罗博弈的围棋博弈程序 ERICA 中，算法在最高收益子节点和最多访问子节点不一致时继续进行博弈树搜索，直至得到相同的结果。这一策略使 ERICA 的胜率从 47%提高到 55%。

算法 3：信心上限树算法（UCT）

function UCTSEARCH(s_0)
以状态s_0创建根节点v_0;
while 尚未用完计算时长 **do**:
v_l ←TREEPOLICY(v_0);
Δ←DEFAULTPOLICY(s(v_l));
BACKUP(v_l, Δ);
end while
return a(BESTCHILD(v_0, 0));

function TREEPOLICY(v)
while 节点v不是终止节点 **do**:
if 节点v是可扩展的 **then**:
return EXPAND(v)
else:
$v \leftarrow$ BESTCHILD(v, c)
return v

function EXPAND(v)
选择行动$a \in A(state(v))$中尚未选择过的行动
向节点v添加子节点v'，使得$s(v')= f(s(v), a), a(v') = a$
return v'

function BESTCHILD(v, c)
return $argmax_{v' \in children\ of\ v}\left(\frac{Q(v')}{N(v')} + c\sqrt{\frac{2ln(N(v))}{N(v')}}\right)$

function DEFAULTPOLICY(s)
while s不是终止状态 **do**:
以等概率选择行动$a \in A(s)$
$s \leftarrow f(s, a)$
return 状态s的收益

function BACKUP(v, Δ)
while $v \neq NULL$ **do**:
$N(v) \leftarrow N(v) + 1$
$Q(v) \leftarrow Q(v) + \Delta$
$\Delta \leftarrow 1 - \Delta$
$v \leftarrow v$的父节点

由于 UCB 算法本身对于探索和利用的兼顾，所以利用 UCB 算法作为选点指导的 UCT 算法也具有该特点，它在探索和利用之间找到平衡，使得在模拟过程中，那些表现良好的节点所在支路能够被更多次的走到，而一些不甚理想节点在少量访问后就不在被访问。这样做的一个重要优势在于，在有限的计算时间内，对那些较好的节点我们可以更深入的进行探索以保证我们的选择更加接近最优解。因此，我们在模拟过程中往往以 UCT 算法代替单纯的蒙特卡罗树搜索来实现蒙特卡罗博弈。

8.4 本章小结

在本章中，我们首先介绍了马尔科夫决策过程，并探讨了围棋落子模型的马尔科夫性。在此基础上，介绍了蒙特卡罗算法的基本思想及其在相关领域中的具体应用，并进一步引入了博弈中的蒙特卡罗评估方法及其与传统的静态评估方法之间的区别。在计算机博弈理论的具体应用方面，本章介绍了蒙特卡罗树搜索的基本思想，以多臂老虎机模型为基础，介绍了在实际应用中平衡探索与利用两方面因素的典型算法信心上限(UCB)算法，并进一步将其应用到蒙特卡罗随机模拟博弈树中，详细介绍了信心上限树算法(UCT)。