第5章　博　　弈

知己知彼，百战不殆。

——孙武[①]

按照本书惯例，还是从博弈这个名字说起。博弈，中文原意就是下棋。但是今天博弈这个词引申的含义非常广，现在一般说到博弈，都是指在双方或者多方指定的规则下，选择合适的行为或者策略，从中获取各自最优的收益。一般用于描述运筹学、管理学、经济学等学科中的博弈行为。

博弈又分为零和博弈、正和博弈、负和博弈。零和博弈就是在双方博弈过程中，一方输掉的正是另一方所赢得的。例如，零和博弈过程中，你输了5元，那对手就赢了5元，棋类游戏就是最典型的零和博弈。在博弈过程中，如果加入增量，输赢之和大于零，就属于正和博弈，人们所说的“双赢”就是一种正和博弈（正和博弈也可能一方赢一方输，例如，一方输了5元，另一方赢了7元，也是一种正和博弈）。类似的，如果输赢之和小于0，就是负和博弈。

另外，按照博弈双方掌握信息的情况，博弈也可以分为完全信息博弈和不完全信息博弈。所谓完全信息博弈是指在博弈过程中，博弈双方均知道对方的博弈动作、策略等相关信息，例如双方下棋；不完全信息是指在博弈过程中，至少有部分信息是博弈对方不知道的，打扑克、打麻将就是典型的不完全信息博弈，因为并不知道对方手中有什么牌。

本章主要介绍计算机博弈，就是学习如何让计算机下棋。让计算机下棋一直是很多天才的梦想，事实上，如果以人机对弈为主线写一部人工智能的发展史，和本书第1章的内容应该有很大重合。计算机的博弈水平一直随着人工智能技术的发展而进步。人工智能最近一次大热，也是AlphaGo战胜人类围棋顶尖高手，从那时候开始，人工智能才从学术界走向大众。机器在围棋上打败人类之后，可以认为棋类问题已经被人工智能解决了。

本章从最简单的棋类游戏——井字棋开始。

5.1　棋类游戏框架

井字棋，英文叫作tic-tac-toe，规则是在3×3的棋盘上，双方轮流下棋，谁能够首先把横线、竖线或者斜线连成一排，谁就赢了。图5.1是一个井字棋游戏示例。井字棋的规则如此之简单，大家很快就能上手下棋。那么，如何编写一个下棋程序呢？

一个下棋程序需要有一个数据结构描述棋局的状态，这个数据结构一般是一个一维数

① 这句话来自《孙子兵法·谋攻篇》。博弈是不见硝烟的战争。博弈胜利的关键在于不仅要知道自己的想法，而且要知道对方的想法。

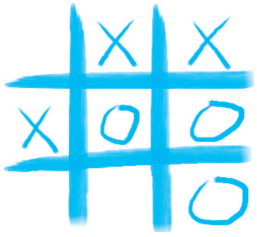

图 5.1　井字棋

（井字棋有个必赢的走法，但是本书不会剧透给你）

组或者二维数组（香农很早就提出用二维数组表示棋局局面）。对于井字棋来说，只需要一个 3×3 的二维数组即可（或者有 9 个元素的一维数组也可以），这个数组中每个元素的值分别代表当前局面下每个空格的状态。数组中每个元素的取值有三种，例如，可以用－1 表示一个（X）玩家的棋子落子在对应的位置上，用 1 表示另一个（O）玩家，用 0 表示当前位置还没有落子。例如，图 5.1 中的棋局局面对应的数组是[0，－1，－1][－1，1，1][0，0，1]。

其实中国象棋和围棋在维护棋局局面方面，原理上和井字棋是一样的。围棋和井字棋一样，只有黑白两种棋子，有一个 19×19 的数组就可以描述围棋的棋局局面。中国象棋比较复杂，需要一个 9×10 的数组表示整个棋盘，因为中国象棋棋子的种类多一些，双方一共有 32 个子，其中又包括车、马、炮等不同的棋子，因此数组元素的可能值有 33 个（16 个表示红方，16 个表示黑方，1 个表示无子）。在真正实现棋类游戏的时候，很多程序使用位数组来表示，加快运行速度。

当然，大家见到的棋类游戏软件都是图形界面的，这些软件中维护棋局局面的依然是数组，只不过在软件开发过程中，背景是棋盘图片，每个棋子也是图片，双方数组的元素值也和相应的棋子图片对应。双方每落一个子之后，都要刷新一下这个局面，对于图形界面，就是把棋子图片显示到正确的位置上。

另外，一个下棋程序还需要能够判断棋局的胜负。任何棋类游戏都有一个胜负规则，谁先满足了这个规则，就可以认为他胜利了。井字棋的规则最简单，任何一方只要有三个子连成一行、一列或者对角线，就可以认为该方胜利。围棋的胜负判定是数“气”，中国象棋是判断主帅是否被杀（象棋还要判断是否处于“将军”局面），其他的棋类游戏各有各的判定方法。

下棋的核心就是要选择正确的位置落子。对于人类来说，落子决定是由人做出来的，因此程序写起来简单。例如，井字棋在设计程序的时候，只要考虑对应位置上有无棋子即可。但是其他棋类，例如象棋，还要考虑合理的走法，“马走日、象走田”，围棋有“禁着点”，程序只要保证人类下棋应该遵守这些规则就可以。机器的落子方法是本章的核心，后文将会介绍。

有了以上基本概念之后，下棋程序就是双方轮流落子了。当然，不管这个程序是人机对弈还是人人对弈，写法都比较简单，双方轮流落子，每次落子之后，都刷新一下局面，判断是否一方胜利，如果没有判断出胜、负、平局状态，另一方接着落子即可……循环一直进行，直到能够判断出胜、负、平局状态。

程序 5.1 是一个井字棋游戏的框架，这个框架提供了人机博弈的界面，这里机器落子是随机的，这个框架中机器能下棋，但是没有棋力。如果人类和这个程序下棋，应该能做到百战百胜。

怎么才能让机器也有棋力呢？

5.2 对抗搜索

5.2.1 博弈树

所有的下棋程序都可以用一棵树表示，称之为博弈树，下棋程序就是在这棵树上进行搜索。博弈树描述了下棋的每个步骤。图 5.2 是井字棋的博弈树（部分）。

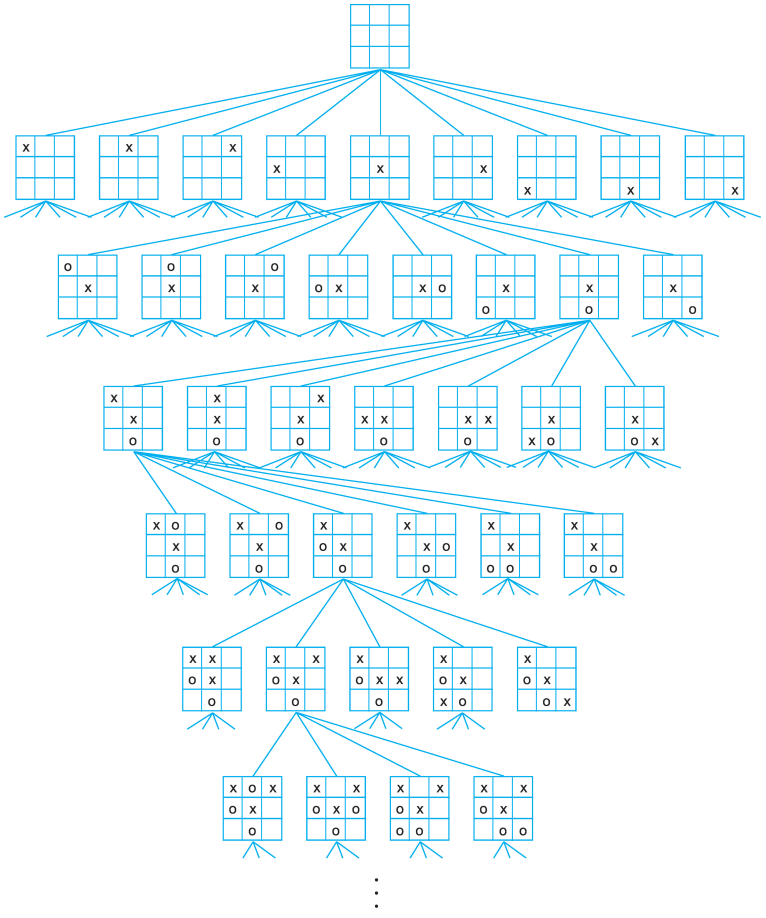

图 5.2 井字棋的博弈树（部分）

图 5.2 中，第零层（最上层）是开局状态，假设第一步是 X 落子，X 可能的落子点有 9 个，这样第一层就有 9 个可能的棋局；然后第二步是 O 落子，O 此时落子的可能位置有 8 个（因为某一个位置已经被 X 落子了），因此，第一层每个棋局后续都有 8 种可能（也可以说这棵树中第一层的每个节点都有 8 个孩子节点），图中只选取一个局面绘制其孩子节点，假设第一步 X 落到棋盘最中间的位置，随后 O 的 8 种可能落子位置如图中第二层所示；交替进行，现在下到第三步，X 落子，以此类推，一直到博弈结束。

理论上，这棵树应该有 9!（=362880）个节点。但是实际上，这里面很多节点是相同的（也就是棋局是重复的），另外，很多时候棋局没有到最后一层可能就判出胜负（例如在倒数

第二层的最后一个节点，此时 X 玩家已经赢了，这个节点就不应该有后续节点），实际的博弈树节点会少一些。

井字棋是一个非常简单的游戏，因为与更复杂的棋类游戏相比，该游戏的最大走棋步数非常少（双方加起来最多只能走 9 步）。因此，井字棋游戏非常适合计算机博弈入门。

本章的目标就是搜索这棵博弈树，找到下棋的最佳方案。

理论上，任何一种棋类游戏均可以创建一棵博弈树，并且，如果能够搜索整棵博弈树，一定会找到一种必胜的走法。然而实际上，很多棋类游戏的博弈树太庞大，根本无法完全搜索，即使是复杂如围棋，也有一个必胜的走法，只不过这个必胜走法可能永远不会被人类发现。

博弈树的想法十分自然，如果有下棋经验就会知道，下棋过程中经常要考虑，如果对方这么走，自己应该那样走，如果对方继续走这步，自己应该怎样应对。实际上，这样的思考过程就是一棵博弈树。

5.2.2　静态估值

在设计棋类游戏的时候，也有启发式的想法，一个最简单的启发式就是给每一个棋局状态打分（也叫估值），分数越高，表明该棋局越容易胜利。然后搜索分数最大的分支，现在的关键就是，如何去打分？

假设现在井字棋到了图 5.3 这一局面，X 刚刚落子完毕，此时轮到 O 落子，O 有 5 种选择（图 5.3 中的 5 个孩子节点）。

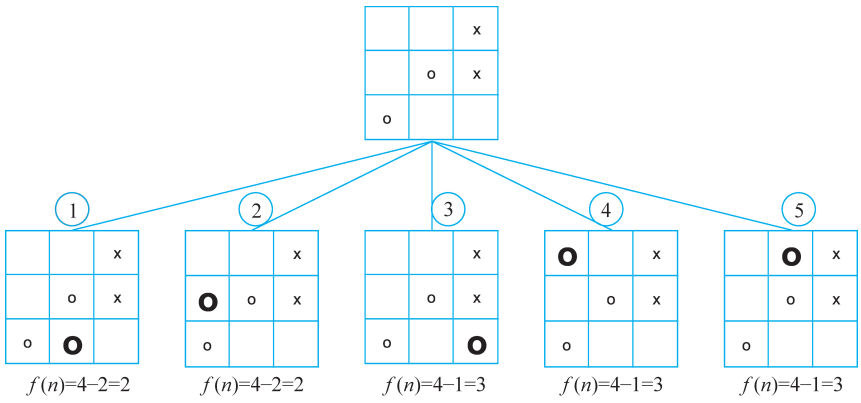

图 5.3　博弈树的静态估值

因为在 X 落子之后，只有 5 个空位，那么 O 的落子，只能落在剩余的 5 个空位上（粗体的 O 表示这一步的落子）。在这 5 个分支中，选择哪个分支最好呢？

在这棵树中，可以使用一个静态的估值函数，来评估下一步走棋的优良程度值。井字棋中一个可能的静态估值函数 $f(n)$ 为

$$f(n)=\text{自己可能获胜的位置数}-\text{对方可能获胜的位置数}$$

例如，在分支①中，O 如果落子在这个位置，那么，第一列、第二列、第三行、主对角线都可能使自己获胜，自己可能获胜的位置数是 4，故而分值是 4，而第一行和第三列有可能使得对方获胜，对方可能获胜的位置数为 2，因此，这个局面的估值分数是 4−2=2。利用这个估

值函数可以得到下一步走棋的分数，该估值函数的值越高，意味越有可能使玩家获胜。这样，在搜索的时候沿着估值高的分支前进，获胜的可能性就大一些。

静态估值函数虽然可以得到一个分数，但是这个估值函数存在一些问题。例如，这里有三种走棋方案的估值函数值等于3(③、④、⑤分支)，但是只有一种方案能够使O玩家获胜(分支③)。因为虽然有另外两种方案(分支④和分支⑤)，但是O玩家如果走分支④或分支⑤，那么玩家X在随后的走棋中马上就可获胜。因此，虽然静态估值函数有用，但是只靠静态估值函数是远远不够的。

在博弈这种对抗搜索中，minimax算法(极小极大算法)是最基础的算法。

5.2.3 minimax算法

普通人和高手下棋的一个重要区别就是普通人在下棋的时候只关注自己，只想着自己快走几步并杀掉对方，而不太关注对方的用意，这样很容易陷入对方的圈套，反而输掉棋局。

换句话说，在下棋的时候，你不光要考虑自己怎么走的，还要考虑对方为什么这么走，而不能把对方当成一个笨蛋。

那么，如何能做到既关注自己又关注对方？冯·诺依曼和摩根斯特恩在《博弈论与经济行为》一书中提出了minimax算法[1]。事实上，minimax算法并不是专门为人机对弈所设计，它能够解决博弈论中的很多问题。不过这个算法在人机对弈中发挥了巨大的作用，所有的棋类设计软件都能看到它的身影。

minimax算法是在轮流行动游戏中，选择最优行动的一种方法。轮流行动游戏是指一个玩家与另一玩家相互对抗，具有相同且相互排斥的目标，也就是说是一个零和博弈。在已给定的游戏状态下，每个玩家都知道下一步可能的行动方案，因此对于每一次行动，玩家都可以考察随后的所有行动。下棋就是标准的轮流行动游戏。

双人游戏轮流走棋，玩家会在走棋时选择对己方最有利而对对方最不利的走法。如何做到“损人利己”？原则非常简单，对于自己走棋，选择最大评估值的那个分支，如果是对方走棋，选择最小的评估值分支。因为双方交替走棋，因此，算法交替地在博弈树中寻求最大评估值和最小评估值。

在博弈树中的每一个节点都可以保存一个值，这个值即为评估值。该值用来定义对应行动在帮助玩家获胜方面的优良程度，前文介绍的静态函数估值即是计算该值的一种方法。

在设计人工智能棋类游戏中，节点评估值对于计算机下棋落子位置有指导性作用，该值越大，获胜的可能性越大。事实上，不只是人机对弈，人人对弈的双方其实也都有一个评估值，只不过对人类来说，这个评估值是隐性的，下棋的时候人们可能不会注意。对于计算机来说，需要有一个显性的评估值。

图5.4给出了在一个特定的井字棋棋局的残局博弈树中，minimax算法的运行过程。

图5.4是一棵博弈树的一部分，图中的树共有4层，根节点所在的那层为MAX层，这一层所有的节点都是MAX节点，也就是说，这一层的每个节点的评估值是其所有孩子节点的评估值的最大值；图中标记MIN的那两层的节点为MIN节点，这两层中每一个节点的评估值是其所有孩子节点的评估值的最小值。这里的终局状态有三个可能的评估值：−1表示玩家X赢，0表示平局，1表示玩家O赢。

现在棋局走到根节点这个局面，O玩家和X玩家各走了三个子(根节点状态)，下一步

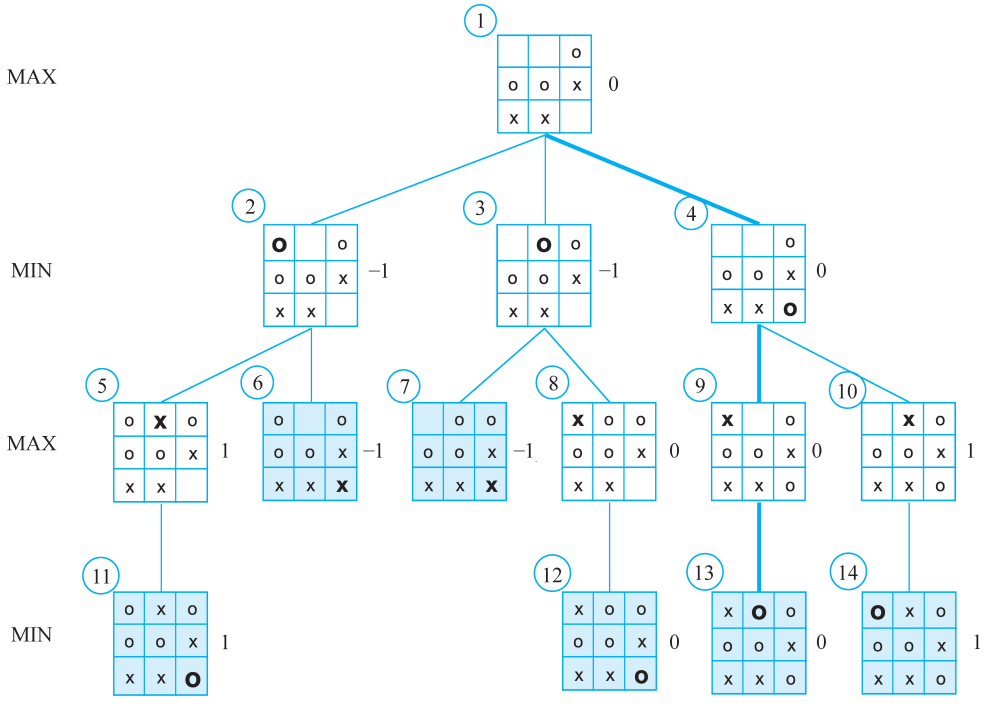

图 5.4　minimax 算法示例

轮到 O 玩家走棋。minimax 算法可以帮助 O 玩家找到最优的下棋路径，这个算法需要计算当前节点的评估值，如果当前节点是 MAX 节点，那么就取所有孩子的最大评估值；如果是 MIN 节点，就取所有孩子节点的最小评估值。

在图 5.4 中，灰色的节点为叶子节点，表示终局，也就是达到了胜、平、负三种状态之一，在井字棋中，只有这个时候才能得到评估值。在图 5.4 这棵博弈树中，O 玩家最后根节点的评估值是 0（也就是说，O 玩家最好的结果就是平局）。看看这个 0 是怎么得到的，对于根节点局面①来说，它是一个 MAX 节点，因此，它会取它三个孩子节点的最大评估值（分别是 −1，−1，0），结果是 0，来自局面④。局面④的评估值 0 是怎么得到的？这一层是 MIN 层，所以它会取它的两个孩子节点评估值（0 和 1）的最小值，得到 0。同理，局面⑨在一个 MAX 层，所以它会取所有的孩子节点的最大值，因为只有一个孩子节点，所以它的评估值为 0。局面⑬是一个终局局面，也就是在这个局面下，双方平局，因此，局面⑬的估值是 0。其他局面也通过同样的过程得到评估值。从这个过程可以看出，minimax 算法非常适合使用递归程序来实现。

根据 minimax 算法，对于当前局面来说，O 玩家最好的结果就是平局，走棋方式如黑色粗线所示，即按照局面①→④→⑨→⑬走棋，这样的走法对于 O 玩家来说是最优的。如果不这么走，O 玩家很可能的结局就是输掉比赛。

对于 O 玩家来说，当然希望能够赢棋，下到估值为 1 的局面，也就是局面⑪和⑭。但是别忘了博弈是双方进行的，O 玩家想要下到这些状态，X 玩家可不会同意，以最左侧的分支为例，如果 O 玩家走到局面②（也就是落子在最左上角），那么 X 玩家马上会落到最右下角位置，局面⑥，从而 X 取得胜利。因此，对于 O 玩家来说，是走不到局面⑪的。对手不是白痴，不会让你轻易胜利。

对于井字棋游戏来说，因为博弈树不会很深，最深的层数才是 9，所以可以搜索完整的博弈树。但是对于大部分棋类来说，都需要建立一棵庞大的博弈树，这样的博弈树是不可能搜索到全部局面节点的。在大多数棋类游戏中，包括国际象棋和中国象棋等，对于搜索都限制了一定的深度，例如从当前局面出发，只搜索下面 10 层的博弈树。这里搜索 10 层相当于人们平时说的下棋能看到后面多少步，例如，专业棋手能看到十步到二十步棋，一般的高手能看到三步到五步棋，普通人下棋只能走一步看一步。

博弈树太深意味着无法到达叶子节点。递归地搜索完整的博弈树是一个费时且费空间的过程。这意味着虽然 minimax 算法能够应用到简单的游戏中（例如井字棋游戏），但对于很多其他棋类来说应用起来却很困难，因为其他棋类游戏的博弈树太复杂而且难以搜索完全。这也是早期电脑下不过人类的原因之一，因为没有办法搜索那么大的博弈树。

minimax 算法的基本过程如下。

```
minimax(player,board)
    if judge(player,board)==end
        return
    for 当前局面之后的每一个局面
        if(player==computer)
            return max_score
        if(player==human)
              return min_score
```

在这个函数中，player、board 分别指当前玩家和当前局面，judge 用于评估当前局面的胜负情况。

对于如何得到每个局面的最大值和最小值，可以使用递归调用，因为在不同的层上（MAX 层和 MIN 层）需要分别计算极大值和极小值，因此，需要有两个函数 player_computer(board)和 player_human(board)。

以下是 player_computer()函数。

```
player_computer(board)
    if judge(board)==end
        return
    for every empty entry
        new_board=board
        mark entry in new_board as player_computer
        value=player_human(new_board)
        if value>max
            max = value
    return max
```

以下是 player_human()函数。

```
player_human(board)
    if judge(board)==end
        return
    for every empty entry
        new_board=board
        mark entry in new_board as player_human
        value=player_computer(board)
        if value<min
              min = value
    return min
```

注意，这两个函数互相递归调用，你中有我，我中有你。在这两个递归函数中，都模拟了其后所有可能的走棋局面（mark entry in new_board as player_computer/ player_human 这一行的功能），然后计算极大值或极小值。那么，有没有办法使用一个函数就完成上述两个函数的功能呢？答案就是使用 negamax 算法。

5.2.4　negamax 算法

minimax 算法在解决简单棋类问题中用处很大，但是在编程实现过程中，需要检查哪一方要取极大值而哪一方要取极小值，以执行不同的动作。高德纳等人于 1975 年提出了 negamax（负极大值）算法[2]。negamax 算法消除了 minimax 双方的差别，简洁优雅。使用 negamax 算法时，博弈双方都取极大值即可，这样，minimax 算法中的两个递归函数就可以写到一个递归函数里面了。negamax 算法其实和 minimax 算法本质上是一样的，它的核心思想在于：父节点的值是各子节点的负数的极大值，计算公式为

$$\max(a,b) = -\min(-a,-b)$$

在井字棋的设计中，可以把一个玩家用－1 表示，另一个用 1 表示，这样，negamax 算法的递归函数可以写为

score = －negamax(board,(player * (－1)))（参见程序 5.2 的第 10 行）

虽然写法不一样，但是本质上，minimax 和 negamax 算法是一样的。negamax 算法虽然把两个递归函数整合到一个函数中，但是因为它和 minimax 算法使用了同样的思想，并没有解决博弈树太庞大、无法搜索的问题，为了让算法能够搜索更深的层次，人们提出了 alpha-beta 剪枝算法。

5.2.5　alpha-beta 剪枝

minimax 算法以及 negamax 算法是棋类游戏中的基本算法（因为这两个算法本质上是一致的，因此，后文不区分这二者，认为它们是一个算法），然而对于大部分棋类游戏，minimax 算法只能搜索有限的深度，因此，有学者提出了一种技术，付出很小的代价即可加深搜索的层次。这种技术在计算机内部经常使用，叫作剪枝。

剪枝在生活中常见，即对于一棵树木，剪去不必要的枝干。计算机的剪枝也是在树这种数据结构上进行。在博弈树上剪枝，就是在对博弈树进行搜索的过程中，如果发现哪个节点分支结果不好，那么就不再搜索该节点以及它的子孙后代，从而达到减少搜索量的作用，那些不被搜索的分支就好像被剪掉一样，因此被称为剪枝。不同的算法有不同的剪枝方法，对于 minimax 算法来说，因为要寻找某一个节点的 max/min 评估值，经过合理的算法设计，

可以提前发现一些节点走棋效果不好，那么就可以进行剪枝，提前终止搜索。在 minimax 算法中，这种有效的剪枝算法叫作 alpha-beta 剪枝。

alpha-beta 剪枝的基本思想是识别出不利于玩家的节点，然后将它们从博弈树中除去。博弈树中被剪枝的分支所在的层越高，最小化博弈树搜索范围的效果越明显。

在 alpha-beta 剪枝算法的具体实行过程中，需要计算并维护 alpha 和 beta 两个变量，按照深度优先搜索博弈树。变量 alpha 定义了能够用来实现极大化目标的最好走棋方案（即对己方来说最好的走棋方案），变量 beta 定义了能够用来实现极小化目标的最好走棋方案（即对对手来说最好的走棋方案），alpha 表示 max 值的下界（也就是所有可能的 max 值中最小的那个），beta 表示 min 值的上界（也就是所有可能的 min 值中最大的那个）。当搜索博弈树时，如果 alpha 大于或等于 beta，则在这样的分支上对手的走棋方案会把己方推向不利的境地，此时，应避免在这样的分支上进行更多的计算。

如果大家觉得上面的一段话比较绕，在 5.5 节有个简单例子，大家跟踪完这个例子，就会明白 alpha-beta 剪枝是如何工作的了。

5.3 其他棋类游戏

5.3.1 人机博弈小史

1928 年，冯·诺依曼利用 Brouwer 不动点定理证明了博弈论的第一个深刻定理，指出在所谓的零和、完全信息博弈中，存在一种策略可以使参赛双方的最大损失极小化，这就是 minimax 一词的由来。1944 年，冯·诺依曼和摩根斯特恩共著的划时代巨著《博弈论与经济行为》奠定了博弈论这门学科的基础和理论体系。

1950 年，香农在 *Programming a Computer for Playing Chess* 论文中[3]，使用 minimax 算法进行棋类游戏设计，他在论文中也提出了使用数组表示棋类游戏局面的做法。

20 世纪 50 年代中期，塞缪尔创建了一个跳棋游戏。很难说清楚谁首先发明了启发式算法，但是塞缪尔是第一个将启发式应用到人工智能程序中的，塞缪尔的跳棋程序也是第一个真正意义上的机器学习程序：他的程序自学了如何下跳棋。

1956 年，麦卡锡提出了 alpha-beta 剪枝算法，这个算法至今在各种棋类游戏中都可以看到。1997 年，IBM 公司制造的 DeepBlue 便使用了 alpha-beta 剪枝算法。

1975 年，高德纳等人提出了 negamax 算法，可以使用一个递归函数完成 minimax 算法中需要两个递归函数的工作。

minimax 算法可以说是博弈游戏的基础，在所有的棋类游戏中，都需要进行博弈树搜索，无论是井字棋，还是 1997 年战胜卡斯帕罗夫的 DeepBlue，抑或 2016 年战胜李世石的 AlphaGo，minimax 算法（或改进的 minimax 算法）随处可见。和 minimax 算法一样，alpha-beta 剪枝几乎在所有的棋类游戏中都会出现。

事实上，一棵真正完全的博弈树很难构建起来，大部分棋类游戏只能构建部分博弈树，然后在这棵部分博弈树上搜索最优的走法。这棵部分博弈树的根节点即当前的棋局局面，树的深度即博弈双方未来可能走法的步数，树的叶子节点要提供一个局面评估值。假设一棵博弈树的树深 20 层，未剪枝的博弈树就表明了双方在之后 20 步内的可能走法，叶子节点

表明 20 步之后那个棋局的评估分数。

在真正的下棋过程中，要想得到某一个局面的评估值是非常困难的。假设你看到马路旁边有人下象棋，过去看一眼，如何判断双方谁占优？一般主要还是看双方主力（主要是车、马、炮这些“大子”）的剩余情况和位置，但是这种判断很多时候是不准确的。除非下棋已经到了很容易分出胜负的地步，否则只凭着当前局面就得出估值函数，会很不准确。然而到了很容易分出胜负的时候，就说明这个博弈树已经很深了，而程序很难搜索到如此深的博弈树。

只凭着带 alpha-beta 剪枝的 minimax 算法，在大多数时候，都不能设计一个棋力高超的计算机棋类游戏，否则，各种棋类游戏早就被攻破了。要想实现能下棋的人工智能，需要更多的技巧。

5.3.2　棋类软件设计技巧

那么，自己开发一个棋类游戏，需要的技巧有哪些？

针对不同的棋类游戏，当然各自有各自的技巧，事实上，很少有人掌握这些棋类的全部技巧。如果让围棋大师去下象棋，估计会比大部分普通人好。但是，围棋大师在象棋比赛中下不过专业棋手，这个一点也不让人奇怪。

虽然不同的棋类游戏有不同的技巧，但是，在开发棋类游戏中，还是有一些公用技巧的。这些公用技巧包括以下 4 种。

（1）开局库。“好的开始是成功的一半”。在不同的游戏当中，好的开局对于赢棋都非常重要。事实上，在开局过程中，已经形成了一些公认的定式。几乎所有的棋类游戏都有成熟的开局定式，例如在围棋中，有三连星、中国流、秀策流等开局定式，象棋有中炮开局、飞象开局、起马开局等，跳棋、国际象棋也都有各自经过验证的开局定式，这些棋局经过人们成百上千年的经验总结，大部分都得到了时间的验证。

（2）残局库。“编筐织篓，全在收口”。有开局库就有残局库。对于一些吃子类的游戏，例如中国象棋、国际象棋，下到后面的时候棋子越来越少，会形成很多经典的残局库。以象棋为例，有名字的残局就有好几百个（“七星聚会”“野马操田”“蚯蚓降龙”“千里独行”等，它们都有炫酷的名字）。残局看起来变幻莫测，但其实有固定套路。一般人很难记住那么多棋谱，但是对于计算机来说很容易。Lewis Stiller 曾在国际象棋方面构建了多达 6 个棋子的完整残局数据库[4]，不要小看 6 个棋子，只剩 6 个棋子的残局局面超过 60 亿个。所以那个时代在下国际象棋的时候，如果只剩 6 个子，人类是赢不了计算机的。当然现在任何棋类，人类对计算机都没有优势了。

（3）棋局估值函数。棋局估值函数是棋类游戏中最难设计的部分，除了像井字棋这种游戏很容易得到棋局的估值函数（很容易就知道胜、平、负的状态）之外，大部分棋类游戏对于一个棋局的估值都很难得到。要想得到估值函数，需要评估的东西太多了。例如对于中国象棋来说，己方的将帅或者一个重要的棋子有危险吗？有一个子被吃掉了吗？如果被吃掉了，这个棋子的价值有多大？一般来说，兵（卒）的价值要小于马和炮，马和炮的价值要小于车；兵（卒）是否过河，它们的价值也不一样。不止要看棋子是什么，棋子的位置也很重要，它们的移动空间有多大，是否受到被吃掉的威胁，它们是否能够吃掉对方的哪些子。如果评估值计算准确，对于搜索算法就具有非常强的指导作用。同样，如果估值函数设计不好，那