

**摘 要**

计算机博弈是计算机科学与人工智能交叉领域中的重要研究方向，旨在开发能够在多方竞争或合作的博弈环境中作出智能决策的算法和系统。对抗搜索算法是计算机博弈中常见的方法，通过模拟可能的决策序列，评估每个决策的收益，以选择最优策略。六子棋是一种双人对弈搏杀的一种游戏，交战双方棋子数均为六颗。因为其先后手优势相对不明显的特点在机器博弈领域被广泛应用。

六子棋优化的算法有很多，我们在棋盘状态表示，棋路状态评估，必胜招搜寻等方面做了探究与优化。基于路表的静态函数评估与基于实时状态的威胁空间搜索的使用使得搜寻算法有比较好的适应能力，能够在多种情况下迅速给出应对。

对静态评估函数权重的寻找，我们使用了遗传算法，找到了一个比较好的参数值，并使用α-β剪枝进一步提高搜索效率。在威胁空间的搜索中，我们寻找不同的威胁数并做出不同应对，使得程序在反应时间和搜索开销上都有很大的提升。

在多个智能体的测试中，每一个智能体相较于上阶段都有很大的提升，最终的综合体也获得了极高的胜率，并提出了对接下来的优化方案。

**关键词：** 六子棋，计算机博弈，α-β剪枝，威胁空间搜索

目录

[**1.机器博弈概述** 6](#_Toc153367061)

[**1.1 机器博弈发展历程** 6](#_Toc153367062)

[**1.2机器博弈流程** 7](#_Toc153367063)

[**2.六子棋介绍** 8](#_Toc153367064)

[**2.1**  **K子棋和六子棋** 8](#_Toc153367065)

[**2.1.1 K子棋** 8](#_Toc153367066)

[**2.2 六子棋** 9](#_Toc153367067)

[**2.2.1 六子棋介绍** 9](#_Toc153367068)

[**2.2.2 六子棋规则** 10](#_Toc153367069)

[**2.2.3 六子棋的公平性问题** 10](#_Toc153367070)

[**2.2.4 六子棋的复杂度** 12](#_Toc153367071)

[**3.六子棋棋路与棋局状态** 12](#_Toc153367072)

[**3.1**  **棋路的定义** 12](#_Toc153367073)

[**3.2**  **棋路与路表的代码实现** 14](#_Toc153367074)

[**3.3**  **评估函数** 20](#_Toc153367075)

[**4.搜索算法级代码** 22](#_Toc153367076)

[**4.1 α-β剪枝算法** 22](#_Toc153367077)

[**4.1.1 极小化极大搜索** 22](#_Toc153367078)

[**4.1.2 α-β剪枝** 23](#_Toc153367079)

[**4.1.3 代码与解析** 25](#_Toc153367080)

[**4.2 威胁空间搜索** 29](#_Toc153367081)

[**4.2.1简介** 29](#_Toc153367082)

[**4.2.2代码实现** 30](#_Toc153367083)

[**4.2.3 优缺点分析** 32](#_Toc153367084)

[**4.3 \*扩展：蒙特卡洛树搜索** 33](#_Toc153367085)

[**4.3.1简介** 33](#_Toc153367086)

[**4.3.2 代码实现** 34](#_Toc153367087)

[**5. 着法生成详解** 38](#_Toc153367088)

[**5.1 最佳着法搜索概述** 38](#_Toc153367089)

[**5.2 计算威胁数** 39](#_Toc153367090)

[**5.3 双威胁进攻着法** 42](#_Toc153367091)

[**5.4 双威胁防御着法** 43](#_Toc153367092)

[**5.5 单威胁防御** 44](#_Toc153367093)

[**5.6 三威胁防御** 48](#_Toc153367094)

[**5.7 一般着法** 50](#_Toc153367095)

[**6. 棋力与性能对比** 50](#_Toc153367096)

[**6.1 α-β剪枝和威胁空间搜索对比** 50](#_Toc153367097)

[**6.2 棋力对比** 52](#_Toc153367098)

[**6.3 各种智能棋力对比** 56](#_Toc153367099)

[**7. 优化方案及展望** 57](#_Toc153367100)

[**7.1动态调整搜索深度** 57](#_Toc153367101)

[**7.2 动态调整静态估值权重** 57](#_Toc153367102)

[**7.3 多线程并行搜索** 58](#_Toc153367103)

[**7.4 深度学习** 58](#_Toc153367104)

[**8. 总结及成员感想** 59](#_Toc153367105)

[**8.1 周忠康** 59](#_Toc153367106)

[**8.2 王颢凯** 60](#_Toc153367107)

[**8.3 田晶怡** 60](#_Toc153367108)

[**8.4 曹文汀** 61](#_Toc153367109)

[**9. 参考文献** 62](#_Toc153367110)

**1.机器博弈概述**

**1.1 机器博弈发展历程**

机器博弈早在上个世纪中期就被提出研究，经过几十年的研究在算法，模型，应用等方面都取得了长久进步，下面是机器博弈的发展阶段。

**早期研究（20世纪中叶）：**

20世纪50年代，机器博弈的早期研究主要集中在国际象棋等棋类游戏。Claude Shannon被认为是机器博弈的奠基人，他在1950年代提出了使用Minimax算法进行博弈的思想。虽然当时的计算机性能有限，但这些早期工作为后来的研究奠定了基础。

**Minimax算法的引入（1950s - 1960s）：**

在这一时期，Minimax算法成为机器博弈的主流算法。Arthur Samuel的国际象棋程序是早期使用Minimax算法的代表性项目。这个算法基于博弈树搜索，通过枚举所有可能的移动，寻找最优的决策。

**Alpha-Beta剪枝（1960s - 1970s）：**

1960年代末，Alpha-Beta剪枝技术的引入使得Minimax算法的效率大幅提升。这一技术通过消除不必要的搜索分支，减少了搜索空间，使计算机能够在更短的时间内找到最优解。

**计算机国际象棋比赛兴起（1980s）：**

20世纪80年代，计算机国际象棋比赛开始引起广泛关注。Deep Thought等计算机程序在国际象棋领域取得了显著成就，尽管它们的水平还不足以与世界冠军竞争，但为后来的深度学习和计算能力提升奠定了基础。

1997年，IBM的Deep Blue计算机在国际象棋比赛中击败了世界冠军加里·卡斯帕罗夫。这一事件标志着机器博弈领域的重大突破，显示出计算机在某些领域已经能够超越人类顶尖选手。

**蒙特卡洛树搜索和AlphaGo（2000s - 至今）：**

2000年代以后，蒙特卡洛树搜索（MCTS）等算法开始在机器博弈中崭露头角。AlphaGo项目在2016年以MCTS为基础成功击败围棋世界冠军，展示了计算机在极其复杂的博弈中的强大能力。

**多智能体系统和深度学习（2010s - 至今）：**

进入2010年代，机器博弈的研究逐渐转向多智能体系统和深度学习。深度学习方法在博弈领域表现出色，如AlphaGo Zero展示了在没有人类先验知识的情况下学习下棋策略的能力。这个阶段还见证了多智能体系统的兴起，其中涉及多个智能体共同参与协作或竞争的博弈问题。

六子棋在计算机博弈中扮演着重要的角色，具有一系列独特的挑战和研究方向。在这个领域，有许多方法和技术被应用来提高计算机在六子棋中的性能。

搜索算法在六子棋中起着关键作用。Minimax算法是常见的选择，它通过博弈树搜索来评估每个可能的决策，并选择最优的策略。另一种方法是蒙特卡洛树搜索（MCTS），它使用随机模拟和统计学习来进行决策。这些算法能够帮助计算机有效地探索决策空间，找到最优解。

为了辅助搜索算法，计算机需要一个评估函数，用于估计当前局面的好坏。这个函数通常考虑诸如棋盘上的位置、棋子的连线情况以及对手的威胁等因素。设计有效的评估函数对于计算机在六子棋中做出准确决策至关重要。

**1.2机器博弈流程**

在博弈问题中（比如六子棋）：搜索是在博弈者双方之间进行的。任何一方在搜索时，都必须要考虑对方可能要采用的走步 。 对于一个优秀的博弈者来说，应考虑的不只是对方一步的走法，而是若干步的走法。这一过程一般来说是动态进行的。在考虑若干步走法以后 ，下了一步棋；而在对方走棋之后，还要再次考虑若干步走法，决定下一步的走法，而不是一劳永逸，搜索一次就决定了所有的走法。

在整个博弈智能的构建过程中，需要考虑智能体的响应时间，搜索步数以及对当前局面是否可解的判断，在这个过程中，需要使用过程建模、状态表示、着法生成、棋局评估、博弈树搜索、开局库与残局库开发、系统测试与参数优化等核心技术要点不断优化智能体。同时需要引入人类已经找到的一些规律与着法，提高智能体的搜索效率。

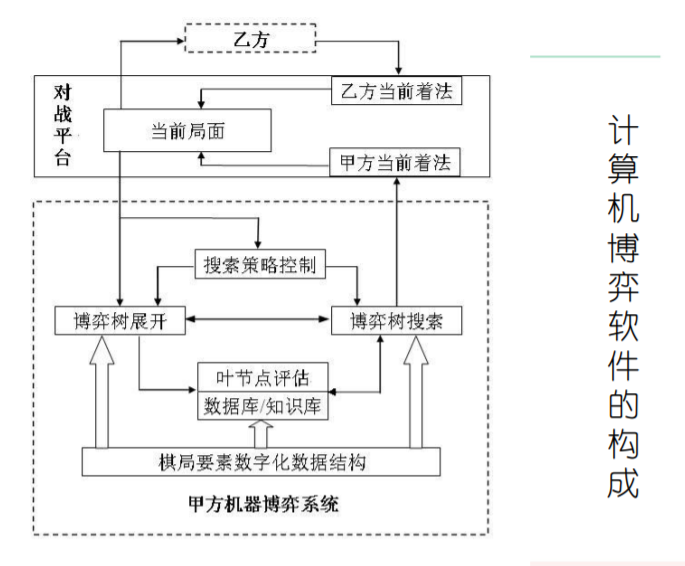


图１　计算机博弈软件的构成

**2.六子棋介绍**

**2.1**  **K子棋和六子棋**

**2.1.1** **K子棋**

台湾交通大学咨询工程系吴毅成教授在台湾交通大学资讯工程系的吴毅成教授在五子棋的基础上发展出一系列K子棋。

K子棋被定义为：

其中：

：大小为的棋盘。

：某一方首先在横向、竖向、斜向任一方向上形成连续颗自己的棋子便可获取游戏的胜利，下满棋盘后还未分出胜负，则为和局。

：比赛开始时第一方默认为黑方)落下颗棋子。

：此后黑白双方每次落下颗棋子。

同时还定义了：，即在一个无限大的棋盘上进行的K子棋。

**2.2 六子棋**

**2.2.1 六子棋介绍**

K子棋中最有趣的就是六子棋，英文名是Connect6，按照上述K子棋的定义可表示为，吴教授推荐的适合竞赛的棋盘大小为或，我们选择前一种棋盘大小，也是比赛标准棋盘，所以可表示为。

六子棋是一种二人零和完备信息博弈，具有以下三大特点：

* **规则简单**：除了黑方第一手下一子外，黑白双方轮流下两子。类似于五子棋，最先连成相连的六子者胜。
* **变化复杂**：由于一次下两子，组合非常多，可说是千变万化。复杂度已被评估为仅次于围棋及日本将棋，远高于五子棋及国际象棋，与中国象棋相当或略高。
* **游戏公平**：由于各方每次下完一手后，盘面都比对方多一子，因此赛局可自然达成平衡的状态，这使得公平性大为提升。不若许多棋种如五子棋、象棋、国际象棋，先下者具有一些优势。

**2.2.2 六子棋规则**

六子棋的规则非常简单，与传统的五子棋非常类似。

* **玩家**：如五子棋及围棋，有黑白两方，各持黑子与白子。
* **玩法**：除了第一次黑方下一颗子外，之后黑白双方轮流每次各下两子，连成6子或以上者获胜。没有禁手，例如长连仍算赢。若全部棋盘填满仍未分出胜负，则为和局。
* **棋盘**：因为公平性不是问题，棋盘可以任意地大，甚至是无限大亦可。

**2.2.3 六子棋的公平性问题**

**2.2.3.1公平的定义**

Van den Herik 等人于 2002 年给了“公平”一个适当的定义，若该游戏是平手的游戏，且双方犯错机率是相等的，则可称此游戏是公平的。然而，“双方犯错机率是相等的”数学模式很难建立，因为若有新的下棋策略被发明后，则犯错机率算法就会不同，从而影响公平性。因此，很难用建立数学模式来证明公平。反过来，要证明不公平则比较容易且可行。以下是吴毅成教授给出的定义：

**定义2.1：明确不公平性 (definite unfairness)。**若已经证明出一方必胜，则此游戏可称为明确不公平。例如用一般规则的五子棋为明确不公平的。

**定义2.2：单调不公平性(monotonical unfairness)。**若已经证明出一方必然不会必胜，但尚无法证明另一方必然不会必胜，则此游戏可称为单调不公平。例如K子棋中的 Connect(m,n,k,p,p)可用策略盗用论点(Strategy-stealing arguments)，证明白方必然不会必胜；因此 Connect(6,1,1)，Connect(7,1,1)，Connect(6,2,2)等皆为单调不公平的。然而 Connect(8,1,1)已被证明双方平手，所以不是单调不公平的。

**定义2.3：经验上不公平性 (empirical unfairness)。**若大多数棋士，尤其是专业棋士，经过实际的下棋经验认定一方必胜或有极高胜率，则此游戏可称为经验上不公平。例如在早期使用一般规则的五子棋及连珠棋，已被一般棋士认定是黑方必胜，因此在当时可称为经验上不公平。

**定义2.4：潜在公平性(potential fairess)。**若一游戏尚未被证明出或论证为明确不公平、单调不公平、经验上不公平，则此游戏可称为潜在公平的。

依据此定义，一个目前为潜在公平的游戏，不一定能持续到未来仍为潜在公平的；一个游戏能持续为潜在公平愈久，则成为公平的机会就愈高。

**2.2.3.2 六子棋的公平性**

对六子棋来说，每当一方下出一步（两子） 时，该方一定比对方多出一颗子直观上，这很自然地使得六子棋具有相当的公平性；当然如上所述，目前仍然不能依此论证六子棋是绝对公平的。但是，至少可以依据以下论点，来论证六子棋仍是潜在公平的：

1.目前，尚无人能证明六子棋是明确不公平。

2.目前，尚无人能证明六子棋是单不公平。吴毅成教授已在他们的论文中证明白方不能采用初始脱离战场策略，否则黑方必胜。这一理论暗示双方必须从中心点开始缠斗:且我们不能依上述初始脱离战场理论，推论出单调不公平性。

3.目前，尚无人能证明六子棋是经验上不公平。目前，已有许多六子棋高手研究许多定石及诘棋，尚无人能认定对某方有利。

当然，此游戏的公平性，需要更多的证据和更长的时间来验证。

**2.2.4 六子棋的复杂度**

对的六子棋而言，其 State-Space 复杂度（所有棋局状态数）与围棋相当，可达 。假设平均每盘棋双方共下30 手（五子棋每盘平均约下30手）。由于每手均走两个子，以每个子有 300 种走法估算，每手有种走法，则其 Game-Tree 复杂度（博弈树展开的大小）可达（按总共30 手计算），远大于五子棋，与象棋相当。随着许多高段棋士对此游戏的逐渐了解，常会下到 40 多手，若以此推算可达到 ，远大于象棋的复杂度。表2-1给出了一些常见棋类游戏的 State-Space 复杂度和 Game-Tree 复杂度。

表1 一些常见棋类游戏的复杂度

|  |  |  |
| --- | --- | --- |
| 棋种 | 状态空间复杂度 | 博弈树复杂度 |
| (10为底数) | (10为底数) |
| 国际跳棋(100格) | 30 | 54 |
| 海克斯(11×11) | 57 | 98 |
| 国际象棋 | 46 | 123 |
| 中国象棋 | 48 | 150 |
| 亚马逊(10×10) | 40 | 212 |
| 将棋 | 71 | 226 |
| 六子棋 | 172 | 140 |
| 19路围棋 | 172 | 360 |

**3.六子棋棋路与棋局状态**

**3.1**  **棋路的定义**

目前 ,大部分六子棋博弈系统的结构都是基于棋形分类的结构 ,这种结构对于准确判断棋形的要求非常高 ,因此 ,有效和准确地判断棋形就成为衡量系统棋力高低的一个决定性因素。六子棋常见棋形有 19种如活三、眠三、跳三、活四、眠四、跳四等 ,每种棋形又存在多种情况 ,且还存在这些棋形交叉和不好分割的情况 ,如图 2所示。所以 ,如何有效和准确地判断、搜索棋形 ,已成为六子棋机器博弈系统的难题。

为破解这个难题 ,本模型中拟采用“路”的思想 , 构建六子棋的决策模型。所谓“路 ”就是指在棋盘上存在连续 6个可能连成一线的点位。由于每条 “路”上有 6个连续点位 ,这样对棋形的判断就变得非常简单。例如 ,某“路 ”中已存在 4颗子,就不用再去判断它到底是活四、眠四、跳四等棋形。同时,利用面向对象程序设计 思想 ,将“路 ”定义为类 ,就可以包括很多的属性 , 如代号、颜色、是否有效等信息 ,能方便地予以实 现。比如 ,通过对“路”中同种颜色棋子数目的判 断 ,即可获知该“路”是否有效。

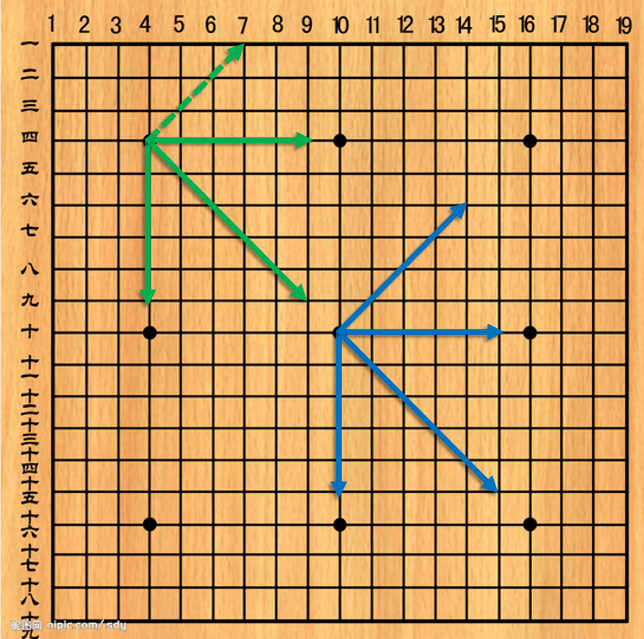


图2 六子棋

在棋盘上，连续的四个方向（横向，竖向，斜下，斜上）的六个点可以被认为是一个棋路，可以分别统计计算。计算规则 :

① 横向上 , 19行 × ( 19 - 6 + 1 )路 /行 = 266路 ;

② 纵向上 , 19列 × ( 19 - 6 + 1 )路 /列 = 266路 ;

③ 左斜上 , 14行 × (19 - 6 + 1)路 /行 = 196路

④ 右斜上 , 14列 × ( 19 - 6 + 1 )路 /列 = 196路。

因此 ,在 19 ×19围棋棋盘上 ,总共有 266 × 2 + 196 × 2 = 924 (路)。

以路表为单位，在之后的状态评估以及威胁空间计算中都极大简化了整体的复杂程度，并使棋盘状态能有一个较好的表征，同时减小了描述的复杂程度。

**3.2**  **棋路与路表的代码实现**

**3.2.1 路和路表的实现**

对于路表，我们使用了如下定义：

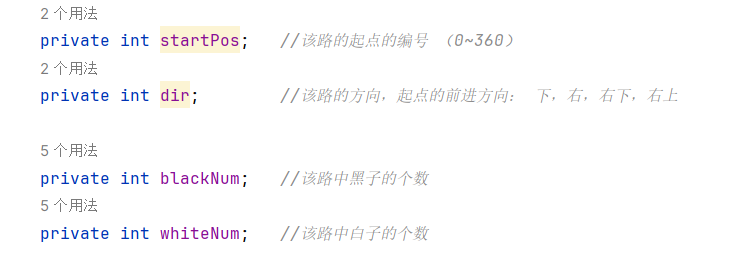


图3 路表中变量的定义1

将全局360个位置进行编号，startpos为路的起始点，dir为路的方向，blackNum和whiteNum分别为该路中黑子和白子的个数。



图4 路表中变量的定义2

IsIfleage变量通过判断路表中起始和末位点的位置判断是否越界， \_FORWARD[]变量通过坐标变换定位路表中不同点的位置。

对于路表，我们在RoadTable.java中实现了它的所有功能，具体函数和定义如下



图5 RoadTable定义



图6 Road和RoadList的定义

我们将路表分为基本路表和黑白路表，分别存储每一个点位不同位置的路和路中具有不同黑白子数的路。

对于对于整个棋盘，我们又定义了一些使用的函数，其中findWinMove与getAffectedRoads函数比较常用，具体代码实现如下



图7 findWinMove函数的实现



图8 getAffectedRoads函数的实现

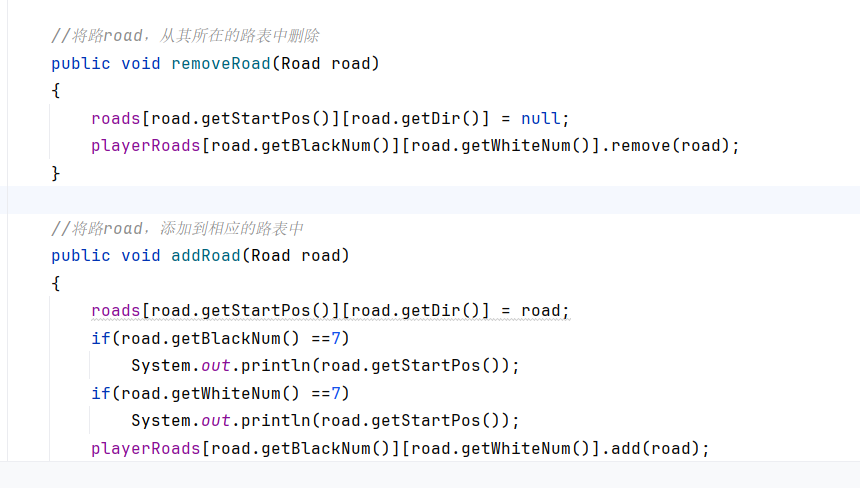


图9 removeRoad和addRoad函数的实现

通过以上函数，可以表示出棋局当前所有棋子的状态。

**3.2.2 Move操作**

对于每一次棋局的移动，我们将其抽象为一次Move操作，主要有以下几个操作

1. FindMove ：这是每个智能体最重要的一个函数，用来寻找在当前状态下下棋方最优的下一步落子，其会因不同的算法而不同，例如：



图10 findMove函数

1. ChangeRoads：每个移动的预演，用以评估寻找到的点落子后的状态。

图11 ChangeRoads函数

1. UnchageRoads：用以撤销模拟的操作，一般与change一起使用。



图12 UnchageRoads函数

**3.3**  **评估函数**

**3.3.1 评估函数简介**

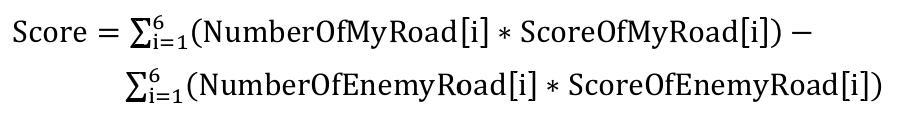
对于人类棋手来说，人们会根据当前棋局的状态，从脑中往下推演，对下面几种可能的情况进行评估，最终选择一条对自己最有利的走法。人们在下棋的过程中，对出现的局面会有一个大致的评估判断，人们会分辨出那种棋局对自己有利，那种棋局对自己不利。同样对于博弈机器人来说，也需要一个这样评估的方法，评估函数就是起着这样的作用。

博弈机器人在下棋的过程中，也是经过比较才得到较好的落子位置。评估函数就是对棋局的状态给出一个评分，给出具体的数值从而根据数值的高低决策取舍。估值大小决定落子的位置，是产生着法的基础，其准确性将直接决定决策系统的“棋力”。

六子棋的评估函数大致分为两类，基于棋型的评估函数与基于路的评估函数，各自有其优缺点。基于棋型的评估函数规定了六子棋中常见的几种棋型，六子棋的每种局面都是六子棋棋型的组合。这样通过把局面分割成棋型就可以评估出当前局面的态势。基于棋型的评估函数可以较为准确的评估局面，但棋型分隔起来比较复杂且耗时，没有一种很好的解决方案。基于路的评估函数把局面拆解成不同的路，拆解起来较为方便，但由于一组路只用一组参数经行评估，不利于应付不同的局面。

**3.3.2 六子棋评价函数**

采用“路 ”思想 ,对棋局状态的评估就不再进行棋形的判定 ,只需要对博弈双方“路”的情况进行计算。将建立的评估函数由 和2个部分组成。初始化函数。该函数的作用是在估值前先对当前局面进行状态扫描，洞察当前局面中含有本方1～6颗棋子的有效“路”数 ,假设用 表示之 ,而对方的有效“路”数假设用表示之。



根据相关论文遗传算法可知，Score数组的权重可为：

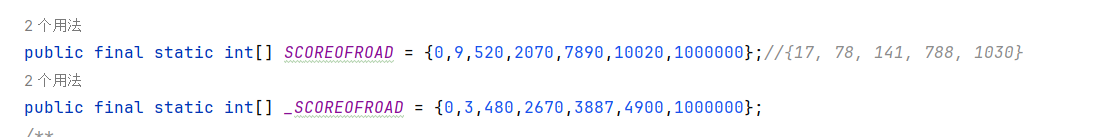


图13 Score数组的权重

具体计算如下：

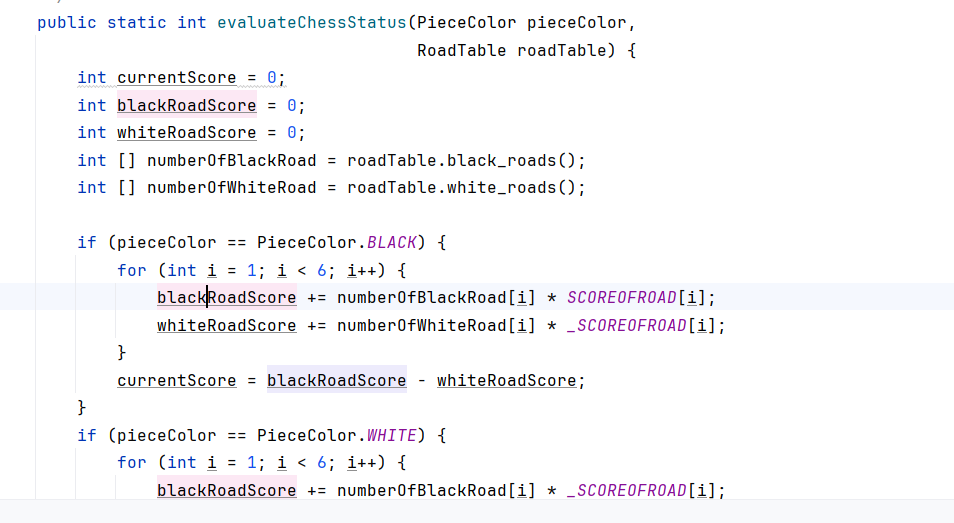


图14 evaluateChessStatus函数的实现

**4.搜索算法级代码**

**4.1 α-β剪枝算法**

**4.1.1 极小化极大搜索**

Minimax 是一种回溯算法，用于决策和博弈论，为玩家找到最佳移动，假设您的对手也发挥最佳作用。它广泛用于井字游戏、西洋双陆棋、曼卡拉、国际象棋等双人回合制游戏中，在Minimax中，两个玩家被称为最大化者和最小化者。最大化者试图获得尽可能高的分数，而最小化者则试图做相反的事情并获得尽可能低的分数。

每个板状态都有一个与之关联的值。在给定状态下，如果最大化者占上风，那么棋盘的分数将趋向于某个正值。如果最小化器在该板状态下占上风，那么它往往会是某个负值。棋盘的值是通过一些启发式方法计算的，这些启发式方法对于每种类型的游戏都是唯一的。

假设我们是参与博弈的一方。我们用静态估计函数来估计博弈双方的态势：

* 有利于我方的态势：
* 有利于敌方的态势：
* 双方均衡的态势：

显然，我方希望最大化，敌方希望最小化。因此称我方为Max方，敌方为Min方。

在Max方的角度，因为是我们自己做决策，我们可以选择任意一种方案，所以我们只需选择收益最大的方案，也就是说每种方案之间是“或”的关系。

而对于Min方而言，因为是敌方做决策，我们无法控制敌方选择哪种策略，假设敌方足够聪明，我们应该假设敌方选择对他最有利的方案，也就是对我们最不利的方案、使我们收益最小的方案，所以对他而言每种方案之间是“与”的关系。

假设我们在进行动态博弈——你一步，我一步，且一方做完决策之后另一方知晓他所做的决策，那么我们可以把双方的行动展开成一棵树——博弈树。

在博弈树中，每个节点代表一种格局，每条边代表Max方或Min方的一步操作。那些下一步该Max方走的节点称为Max节点，下一步该Min方走的节点称为Min节点。

极大极小搜索过程比较简单。然而，实际问题中的博弈树往往庞大复杂，无法通过暴力搜索找到最佳解法。因此，需要引入剪枝技术，其中一种常见的方法是α-β剪枝。

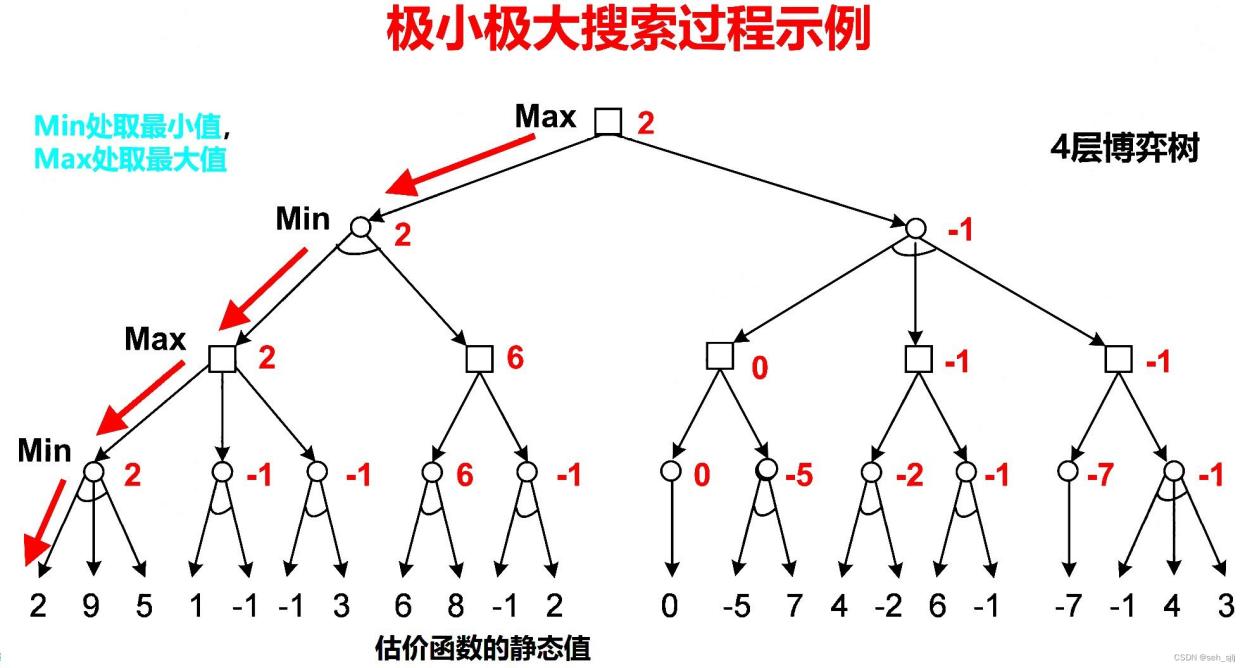


图15 极小极大法确定非端结点n的值

**4.1.2 α-β剪枝**

Alpha-beta 剪枝是一种搜索算法，旨在减少其搜索树中 minimax 算法评估的节点数。它是一种对抗性搜索算法，通常用于双人组合游戏（井字游戏、国际象棋、Connect 4 等）的机器游戏。当发现至少一种可能性证明该移动比先前检查的移动更糟糕时，它会停止评估移动。这些举动无需进一步评估。当应用于标准 minimax 树时，它返回与 minimax 相同的移动，但会修剪掉不可能影响最终决策的分支。

亚历山大·布鲁德诺（Alexander Brudno）独立构思了alpha-beta算法，并于1963年发表了他的结果。Donald Knuth 和 Ronald W. Moore 在 1975 年改进了该算法。 Judea Pearl在两篇论文中证明了其在随机分配叶子值的树木的预期运行时间方面的最优性。1986 年，Michael Saks 和 Avi Wigderson 证明了 alpha-beta 随机版本的最优性。

Max方：取当前子节点中效用值的极大值为该节点效用值的下界，称为α（α≥该极大值），只有当下一个节点的值大于α才会被选择

Min方：取当前子节点中效用值的极小值为该节点效用值的上界，称为β（β≤该极小值），只有当下一个节点的值小于α才会被选择

α：目前Max方可以搜索到的最好值，初始值为

β：目前Min方可以接受的最坏值，初始值为

α剪枝：当当前节点是极大节点（Max节点）时，如果发现当前节点的值（下界 α）已经大于等于其祖先节点的β值（上界），那么就可以放心地剪掉该节点的子树，因为对于父节点来说，已经有更好的选择了，而当前节点不再影响选择。

β剪枝：当当前节点是极小节点（Min节点）时，如果发现当前节点的值（上界 β）已经小于等于其祖先节点的α值（下界），那么就可以安全地剪掉该节点的子树。这是因为对于父节点来说，已经有更好的选择，而当前节点不再影响选择。

α-β剪枝的条件：

1. 在搜索过程中，如果某个节点的值更新为α或β，就可以根据该节点的角色进行剪枝。
2. 对于极大节点，如果α（下限）大于等于β（上限），就可以剪枝，因为后续的极小节点不会选择这个分支。
3. 对于极小节点，如果β（上限）小于等于α（下限），也可以剪枝，因为后续的极大节点不会选择这个分支。

α-β剪枝的搜索顺序：

1. 通常在实际搜索时，按照深度优先或广度优先的方式生成博弈树的节点。
2. 在生成节点的同时，根据α和β的值进行剪枝，避免不必要的搜索。

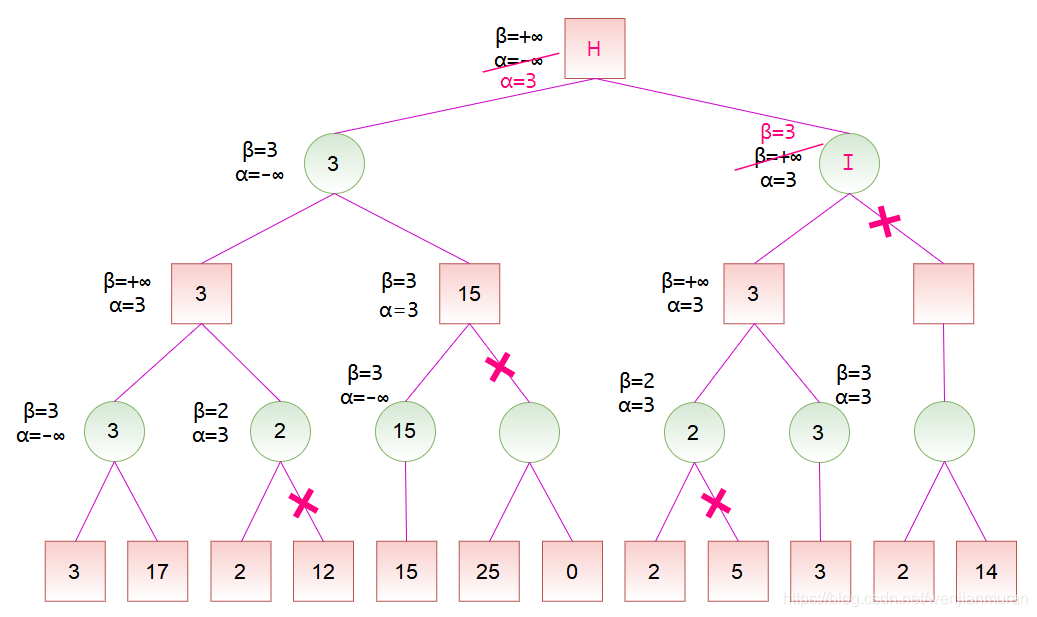


图16 使用α-β剪枝进行搜索

**4.1.3 代码与解析**

alphaBeta 方法：

**参数：**

1. alpha：极大节点的估值下界。
2. beta：极小节点的估值上界。
3. nw：标志，表示当前是极大节点（1）还是极小节点（0）。
4. depth：当前搜索的深度。

**返回值：**

返回当前局面的估值。在这个实现中，估值是通过调用evaluateChessStatus方法得到的。

**主要逻辑：**

1. 如果游戏结束或者达到搜索深度，则返回当前局面的估值。
2. 获取当前局面的所有可能走步，并按照启发式评估对它们进行排序。
3. 遍历所有可能的走步：

执行当前走步。

递归调用 alphaBeta 方法，更新局面估值。

撤销当前走步。

根据当前节点是极大还是极小，更新 alpha 或 beta。

执行 α-β剪枝。





图17 alphaBeta 方法的实现

getMovesAccordingToThreats 方法：

**返回值：**

1. 返回根据威胁数量选择的走步列表。

**主要逻辑：**

1. 获取当前局面的威胁数量。
2. 根据威胁数量选择相应的走步生成方法。
3. 返回生成的走步列表。

updateBestMove 方法：

**参数：**

1. move：要更新的最佳走步。

**主要逻辑：**

1. 执行给定的走步。
2. 切换棋手颜色。
3. 清空走步排序列表。
4. 如果深度为最大深度且不满足某个条件（!DTSS(7)），则更新最佳走步。
5. 再次切换棋手颜色。
6. 撤销走步。



图18 getMovesAccordingToThreats方法和updateBestMove方法的实现

**4.2 威胁空间搜索**

**4.2.1简介**

威胁空间搜索(Threat Space Search, TSS)是一种基于搜索树的游戏搜索算法,通过考虑所有可能的进攻和防守策略,来寻找最优解并避免搜索过程中的无效走法。它是模拟人对于“威胁”这个概念的对策以及从六子棋本身的特点入手进行优化，将棋手下棋的套路公式化提高搜索效率。威胁的定义：一个玩家，比如说 W，不能连接 6个。当且仅当 W需要放置 t块石头来阻止 B在 B的下一步行动中获胜时，B被认为有 t个威胁。

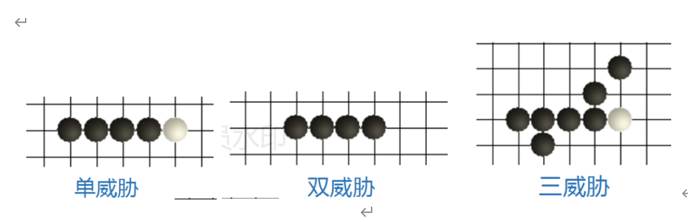


图19 几种威胁举例

威胁空间搜索的实现过程如下：

1.建立威胁空间：首先，需要建立一个威胁空间，用于存储所有可能的威胁。在Connect6中，威胁空间是一个二维数组，其中每个元素表示一个威胁，包括其ID、优先级和类型等属性。

2.扩展威胁空间：对于每个威胁，需要扩展其威胁空间。在Connect6中，扩展威胁空间的方法是使用启发式知识库来评估每个可能的行动，并选择一个最佳的行动。

3.评估行动：对于每个威胁，需要评估其当前状态下的可用行动。在Connect6中，这通常涉及到计算每个威胁的攻击范围以及其可能的行动。

4.选择行动：基于评估的结果，为每个威胁选择一个最佳的行动。在Connect6中，这通常是通过使用启发式知识库来评估每个行动的后果，并选择一个具有最优后果的行动。

5.更新状态：在每次评估和行动之后，需要更新威胁空间和状态。在Connect6中，这通常包括更新威胁、评估和行动属性。

6.搜索解决方案：在多次迭代之后，使用MCTS算法或alpha-beta剪枝搜索解决方案。在Connect6中，这通常是通过使用启发式知识库来加速搜索过程，并评估每个可能的解决方案。

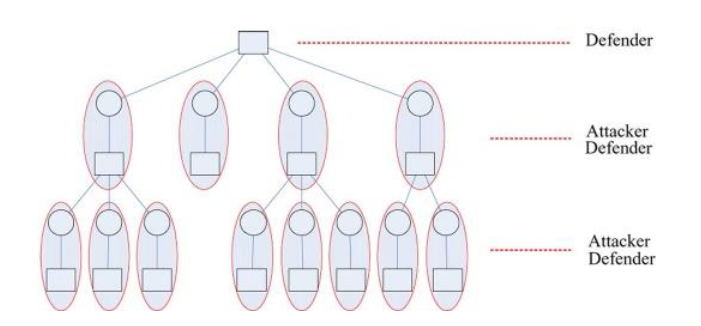


图20威胁空间搜索树

**4.2.2代码实现**

双威胁搜索算法的主体实现函数DTSS()，使用迭代的方法实现。**各个函数的具体实现与逻辑将在下一小节介绍。**

因为威胁空间搜索需要考虑所有可能的进攻和防守策略，所以分为两部分：我方行棋时和对方行棋时。但核心还是考虑我方的双威胁情况。因为TSS存在可能会无限搜索的问题，所以对DTSS()函数的搜索深度进行限制。

当我方行棋时，在对对方存在威胁的情况下通过findDoubleThreats()函数找到我方行棋能形成双威胁的着法；再对这些着法依次模拟，再用DTSS()函数判断这样下后上一层是否具有双威胁，如果有则返回true;模拟后还原棋盘。

当对方行棋时，如果威胁数大于2直接返回true,否则通过findDoubleBlocks()函数找到对方可能用来堵我方威胁的着法；再对这些着法依次模拟，再用DTSS()函数判断对方这样下后上一层是否具有双威胁，如果没有则代表威胁解除，返回false; 模拟后还原棋盘。

在findMove()函数中，若DTSS()函数在当前深度下返回true，则下棋。

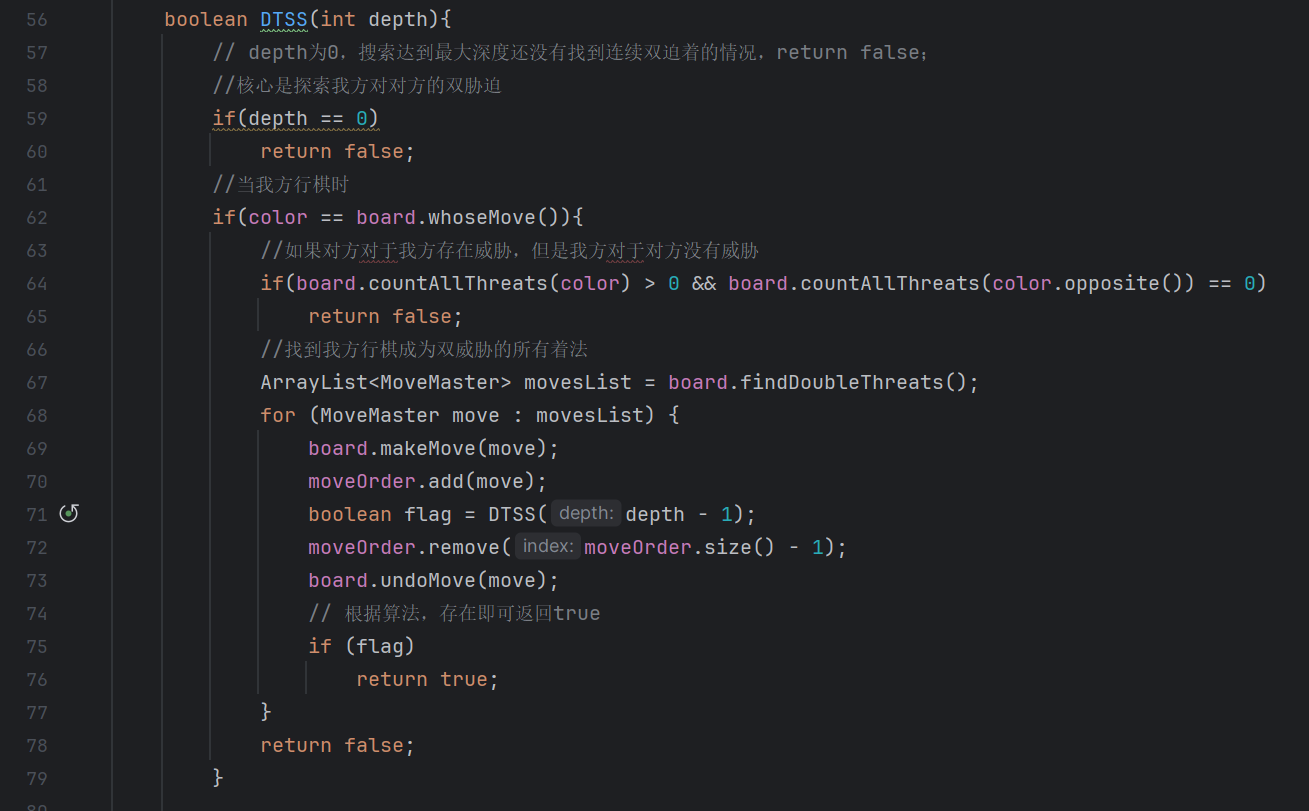


图21 DTSS函数我方行棋时的实现

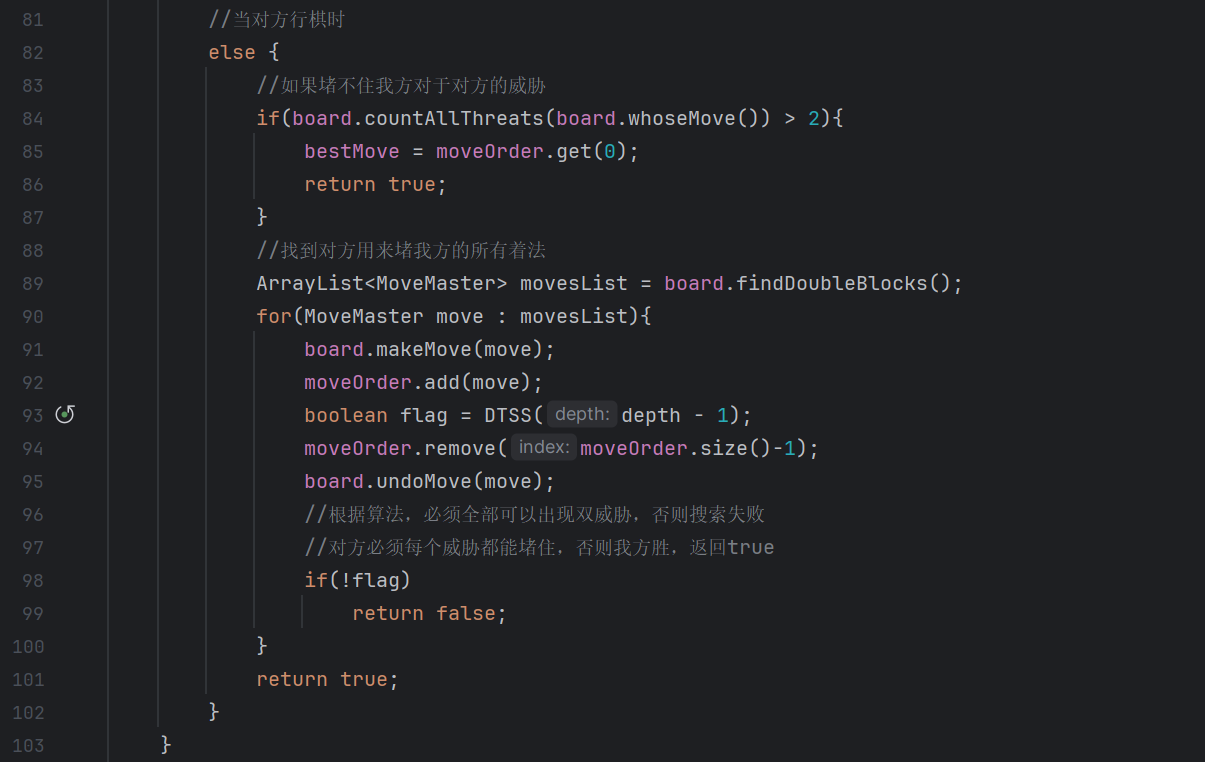


图22 DTSS函数对方行棋时的实现

**4.2.3 优缺点分析**

**优点：**

1.基于局部搜索:TSS通过在当前状态下搜索所有可能的下一步移动,来找到最优解。因此,它可以快速地找到局部最优解。

2.搜索空间小:TSS只搜索与当前状态相邻的节点,因此搜索空间比深度优先搜索(DFS)等算法小得多。

3.找到最优解:TSS可以找到最优解,因为它是基于局部搜索的,所以它可以快速地找到最优解。

4.避免重复搜索:TSS会避免重复搜索,因为它会从当前状态开始搜索所有可能的下一步移动,而不是从所有状态开始搜索。

**缺点：**

1.需要大量先验知识:TSS需要大量的先验知识,例如游戏规则和可能的下一步移动。如果没有足够的信息,TSS可能无法找到最优解。

2.搜索速度慢:TSS需要搜索所有可能的下一步移动,因此搜索速度可能比DFS等算法慢。

3.无法处理某些情况:TSS只能找到局部最优解,无法处理某些情况,例如多个可行解或无限搜索。

4.对棋子的数量敏感:TSS对棋子的数量敏感,因为每个棋子只能被搜索一次。因此,如果棋子数量很大,TSS的搜索时间可能会很长。

**4.3 \*扩展：蒙特卡洛树搜索**

**4.3.1简介**

MCTS也就是蒙特卡罗树搜索（Monte Carlo Tree Search），是一类树搜索算法的统称，可以较为有效地解决一些探索空间巨大的问题，例如一般的围棋算法都是基于MCTS实现的。蒙特卡罗树搜索是一种基于树数据结构、能权衡探索与利用、在搜索空间巨大仍然比较有效的的搜索算法。至于这个树搜索算法主要包括Selection、Expansion、Simulation和Backpropagation四个阶段。随着搜索树的自动生长，蒙特卡洛树搜索可以保证在足够长的时间后收敛到完美解（但可能需要极长的时间）。

MCTS的算法分为四步，第一步是Selection，就是在树中找到一个最好的值得探索的节点，一般策略是先选择未被探索的子节点，如果都探索过就选择UCB值最大的子节点。第二步是Expansion，就是在前面选中的子节点中走一步创建一个新的子节点，一般策略是随机自行一个操作并且这个操作不能与前面的子节点重复。第三步是Simulation，就是在前面新Expansion出来的节点开始模拟游戏，直到到达游戏结束状态，这样可以收到到这个expansion出来的节点的得分是多少。第四步是Backpropagation，就是把前面expansion出来的节点得分反馈到前面所有父节点中，更新这些节点的quality value和visit times，方便后面计算UCB值。

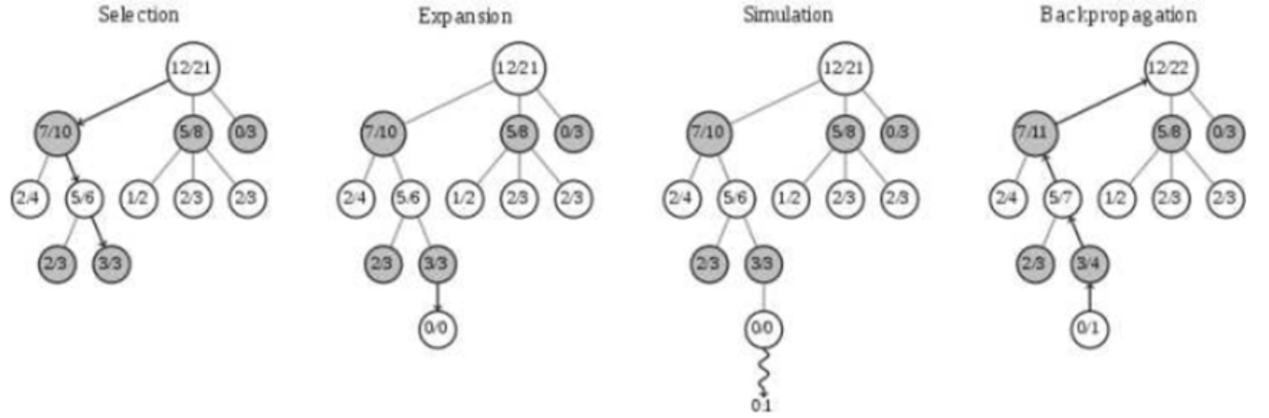


图23 蒙特卡洛树搜索示意图

基本思路就是这样的，通过不断的模拟得到大部分节点的UCB值，然后下次模拟的时候根据UCB值有策略得选择值得利用和值得探索的节点继续模拟，在搜索空间巨大并且计算能力有限的情况下，这种启发式搜索能更集中地、更大概率找到一些更好的节点。

AlphaGo也是基于MCTS算法，但做了很多优化。AlphaGo不仅替换了经典实现的UCB算法，而是使用policy network的输出替换父节点访问次数，同样使用子节点访问次数作为分母保证exploration，Q值也改为从快速走子网络得到的所有叶子节点的均值，神经网络也是从前代的CNN改为最新的ResNet，这样复杂的神经网络模型比一般的UCB算法效果会好更多。

**4.3.2 代码实现**

设置棋盘状态类：



图24 棋盘状态类

设置蒙特卡洛树的节点类：



图25蒙特卡洛树的节点类

蒙特卡洛树实现：

实现蒙特卡洛树搜索算法，传入一个根节点，在有限的时间内根据之前已经探索过的树结构expand新节点和更新数据，然后返回只要exploitation最高的子节点。蒙特卡洛树搜索包含四个步骤，Selection、Expansion、Simulation、Backpropagation。前两步使用tree policy找到值得探索的节点。第三步使用default policy也就是在选中的节点上随机算法选一个子节点并计算reward。最后一步使用backup也就是把reward更新到所有经过的选中节点的节点上。进行预测时，只需要根据Q值选择exploitation最大的节点即可，找到下一个最优的节点。



图26 蒙特卡洛树的实现

蒙特卡洛树的选择阶段：



图27 treePolicy的实现

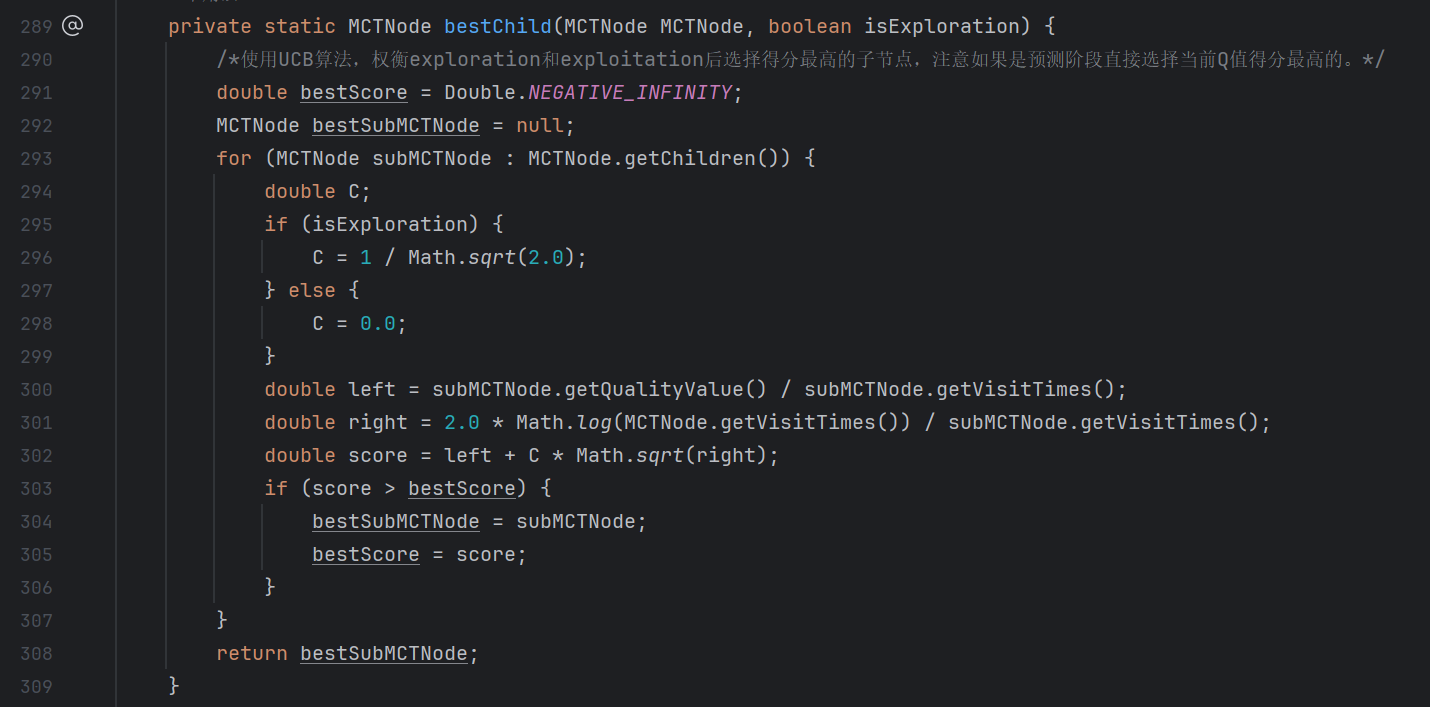


图28 bestChild的实现

蒙特卡洛树扩展阶段：



图29 expand的实现

蒙特卡洛树模拟阶段：



图30 defaultPolicy的实现

蒙特卡洛树回溯阶段：

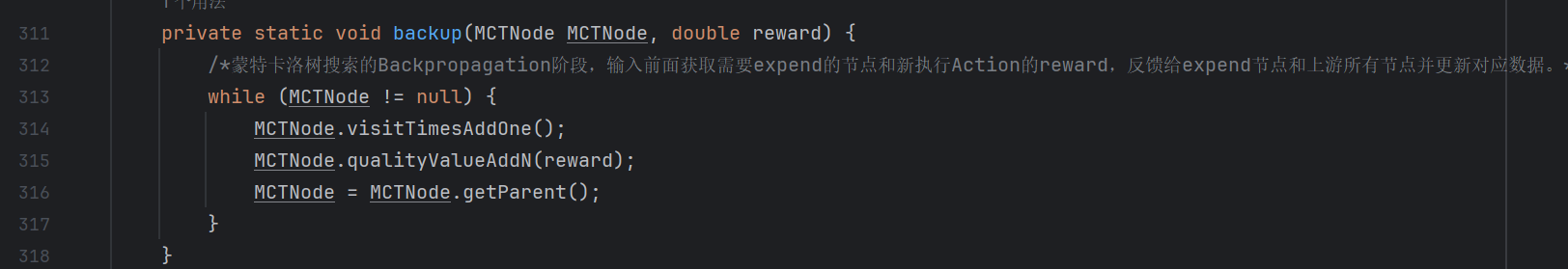


图31 backup的实现

我们项目中对于MCTS的创新点在于，评估节点和选择下一棋局情况的函数结合了本项目中的威胁空间搜索算法；当当前节点存在双威胁解时，直接选择该棋局作为下一节点，当不存在双威胁解时，再进行随机模拟通过UCB算法选择下一节点。该函数在State类中实现。

图32 getNextStateWithRandomChoice的实现

**5. 着法生成详解**

**5.1 最佳着法搜索概述**

如果对手的移动为null（即对方是第一步移动），则调用 firstMove() 方法进行首步移动，将返回的移动应用于游戏棋盘，并将该移动返回。

如果对手已经进行了移动，则将对手的移动应用于游戏棋盘。

调用 board.findwinMoves() 方法，寻找是否有可以直接获胜的移动（winning moves）。如果找到，则将该移动应用于游戏棋盘，并返回该移动。

使用循环在不同的搜索深度上执行动态时间步长搜索（DTSS）。如果找到一个移动，将其应用于游戏棋盘，并返回该移动。

如果以上步骤没有找到获胜的移动，且DTSS也没有找到移动，那么执行Alpha-Beta剪枝搜索（alphaBeta 方法）来寻找最优的移动。

如果找到最优移动，则将其应用于游戏棋盘。

如果以上步骤都未找到最优移动，那么通过 board.findGenerateMoves() 方法生成可能的移动列表，将其按照分数排序，然后选择列表中的第一个移动。

如果找到一个移动，则将其应用于游戏棋盘。返回最终选择的移动。

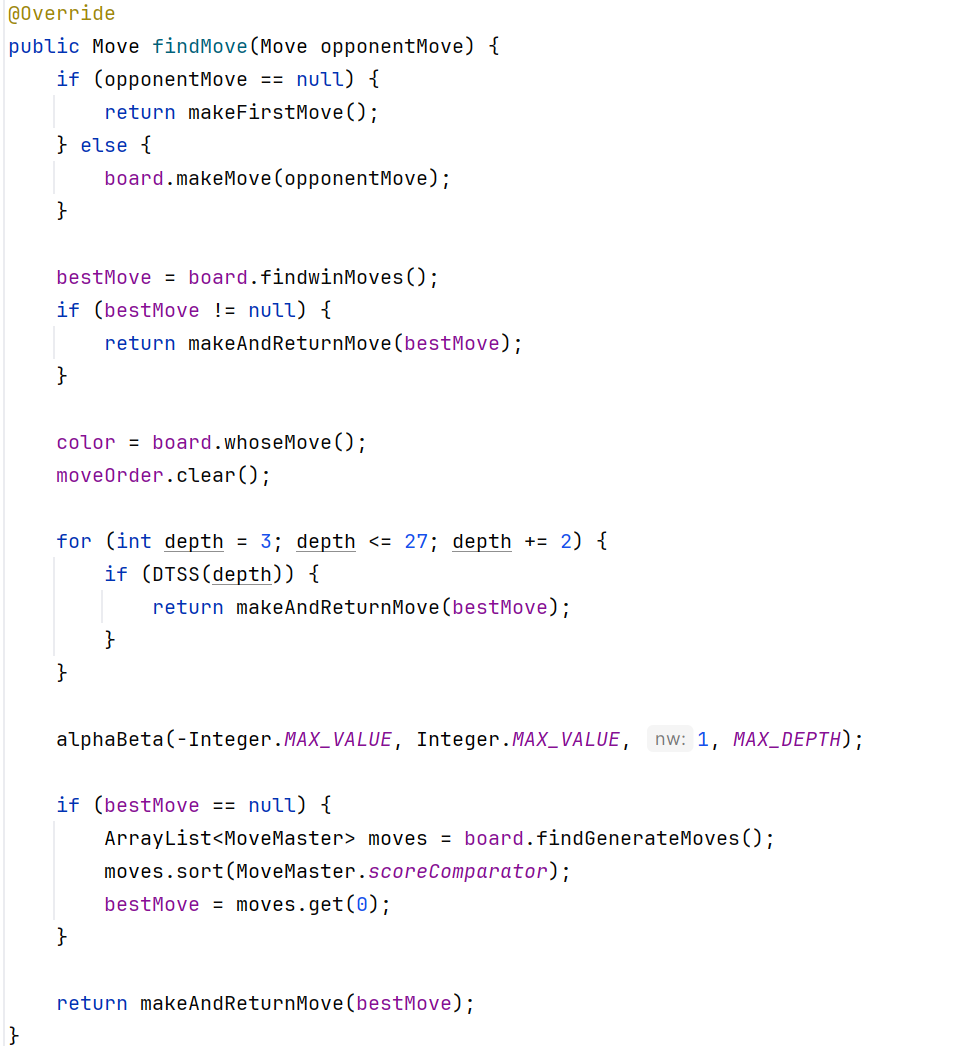


图33最佳着法搜索概述

**5.2 计算威胁数**

找到棋盘路表上所有威胁的通用函数countAllThreats()。

我们基于黑白路表实现，用二维数组存储黑白子的路数，第一个参数为一路中的白子个数，第二个参数为一路中的黑子个数；所以要找出威胁个数，需要找出黑白棋分别为四子和五子的路数，然后再下子进行模拟来判断威胁类型。countAllThreats()函数的返回值有0——无威胁，1——单威胁，2——双威胁，3——三威胁。

我们主要是实现双威胁空间搜索。

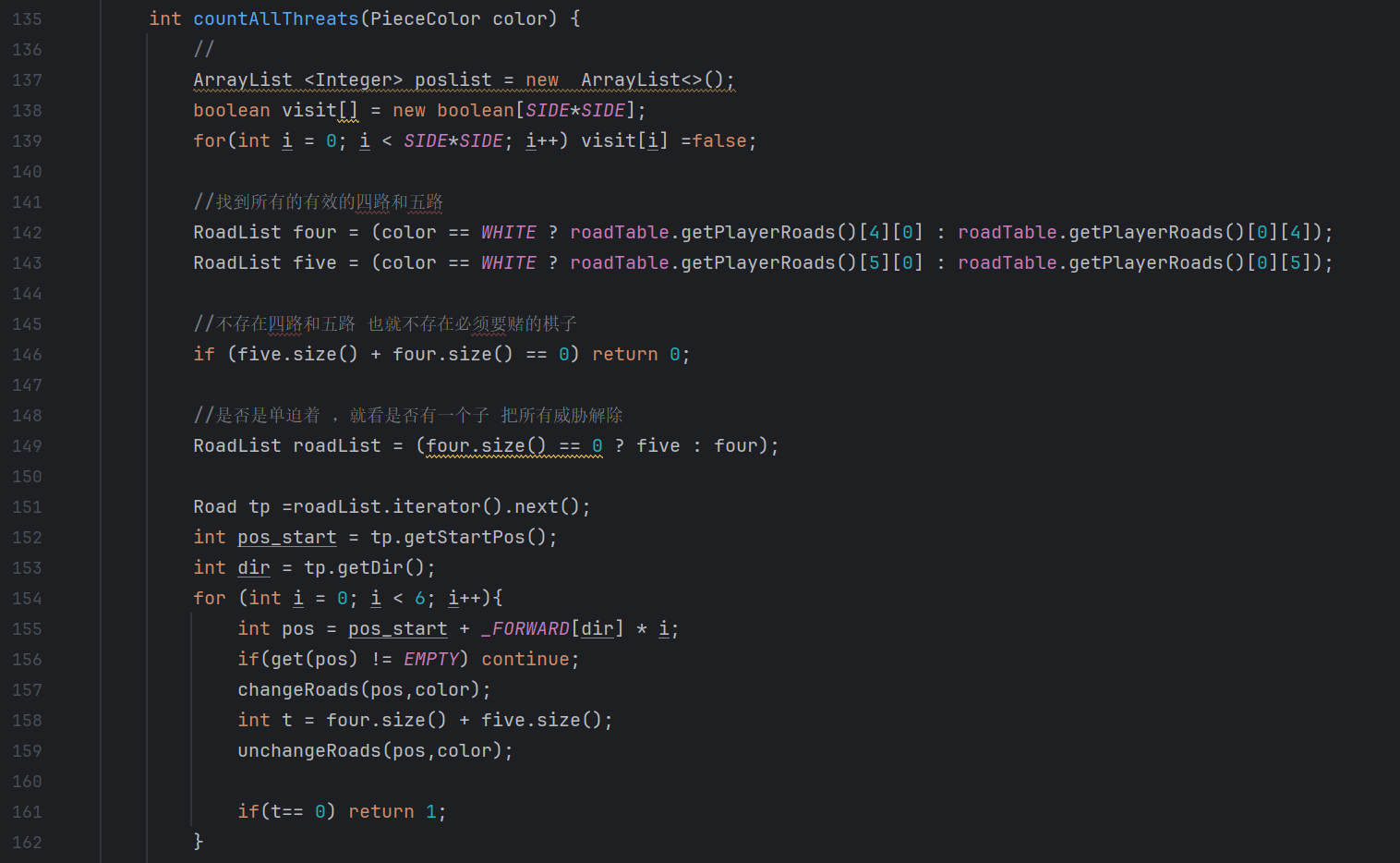


图34 countAllThreats的实现2



图35 countAllThreats的实现2

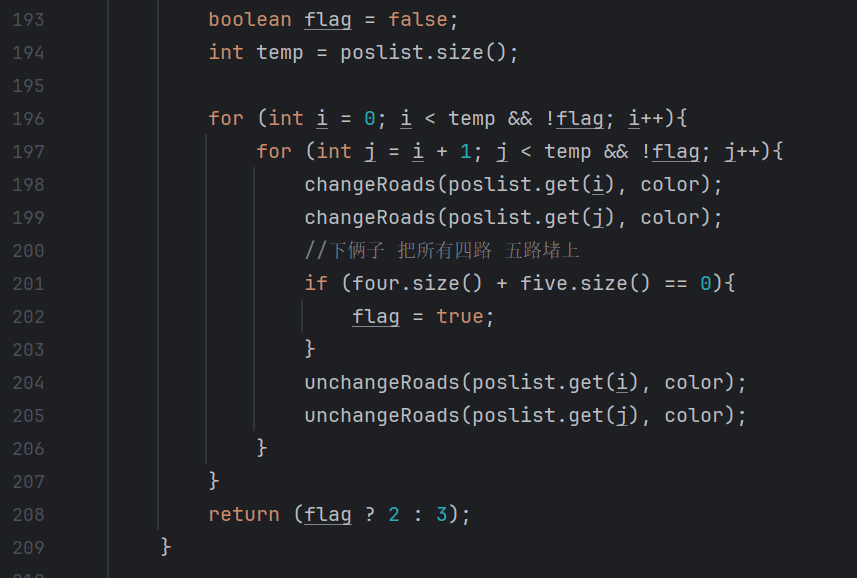


图36 countAllThreats的实现3

**5.3 双威胁进攻着法**

接着是作为进攻方时，我们需要找到能够形成双威胁的着法，实现该功能的函数为findDoubleThreats()。

双威胁为已连四子或五子且该路未被堵住的情况，所以通过路表查找出我方已经连二子或三子且未被堵住的路，即黑白路表中仅存在两个或三个我方颜色子的路，将它们存入myTwo和myThree的路表中，再通过findblanks()函数分别找到这两个路表中每个路中的空位置；在存储了这些空位置的路表中找查，进行下棋模拟，判断这一步下后是否能达成双威胁，如果能，则存入movelist中；如果不能，则通过unchangeRoad()函数还原棋盘。

最后返回能形成双威胁的移动方法。

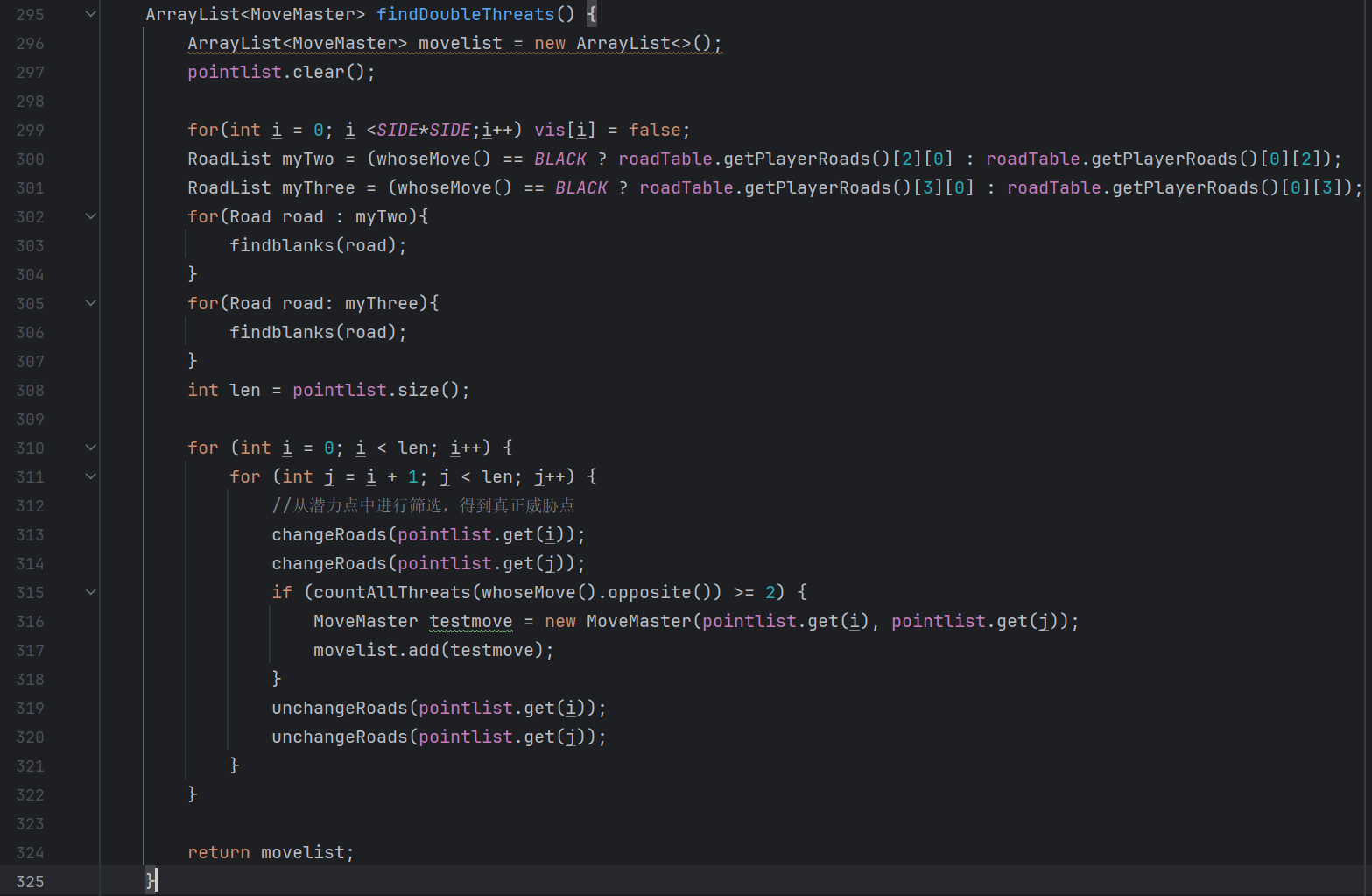


图37 findDoubleThreats的实现

**5.4 双威胁防御着法**

然后是作为防御方时，我们需要找到能堵住对方双威胁的着法，实现该功能的函数是findDoubleBlocks()。

同上述，为了堵住对方对我们形成双威胁，需要找出黑白路表中仅存在四个或五个对方子的路，然后同样把空白位置存储起来，在这些空白位上进行下棋模拟，当当前路表中存储的四子或五子路数为0时，威胁解除；如果不能，则利用unchangeRoads()函数还原棋盘，再继续模拟。

最后返回能接触对方双威胁情况的移动方法。

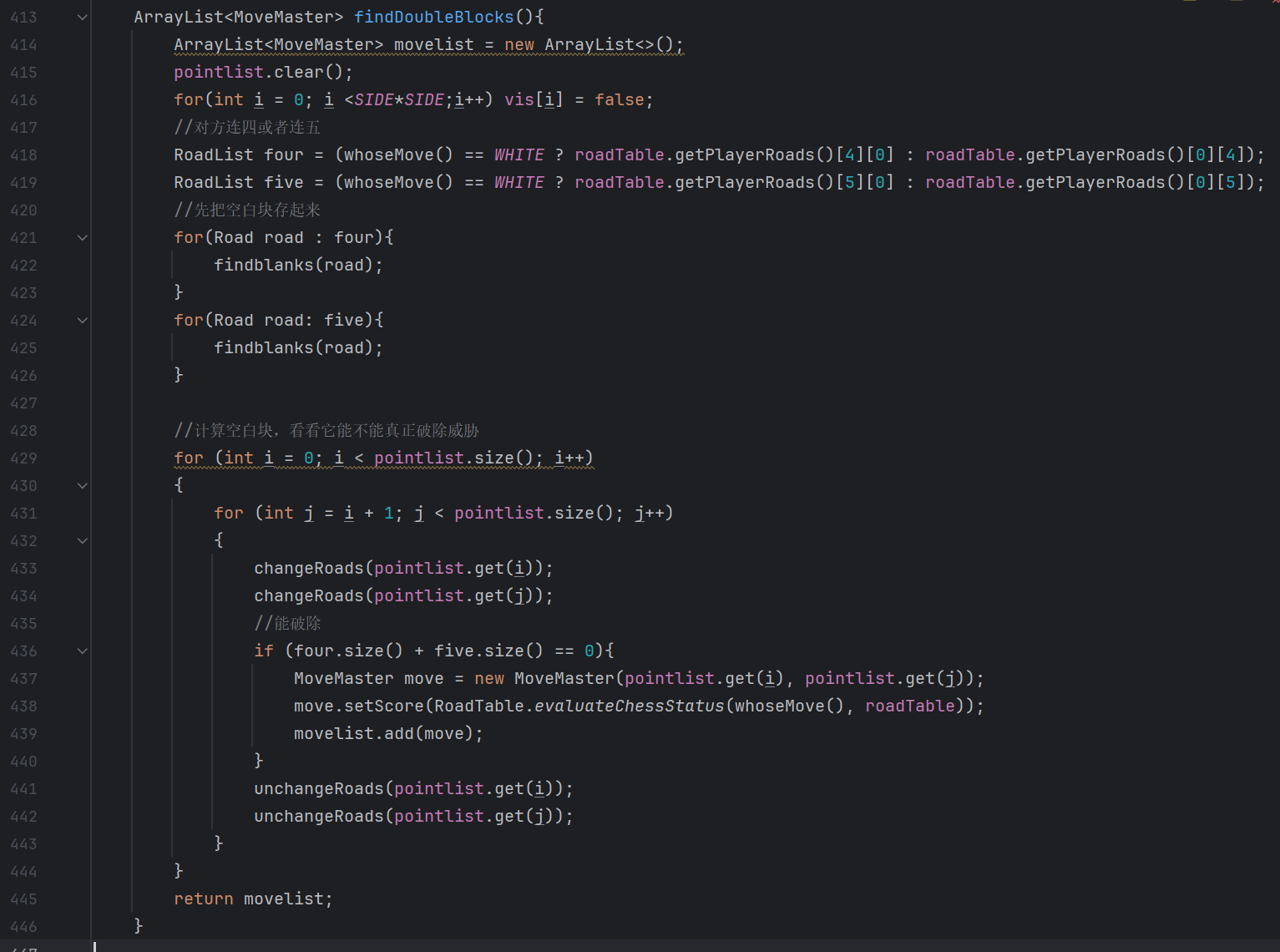


图38 findDoubleBlocks的实现

**5.5 单威胁防御**

findSingleBlocks 方法：

这是主要的方法，负责找到特定条件下的走步列表。首先，它初始化变量，然后获取两个不同类型的道路列表（R1 和 R2），并在这些道路上进行迭代检查。接着，对找到的空白点进行筛选和排序，并最后通过 updateRoadTable 和 generateMoveList 方法更新并返回走步列表。

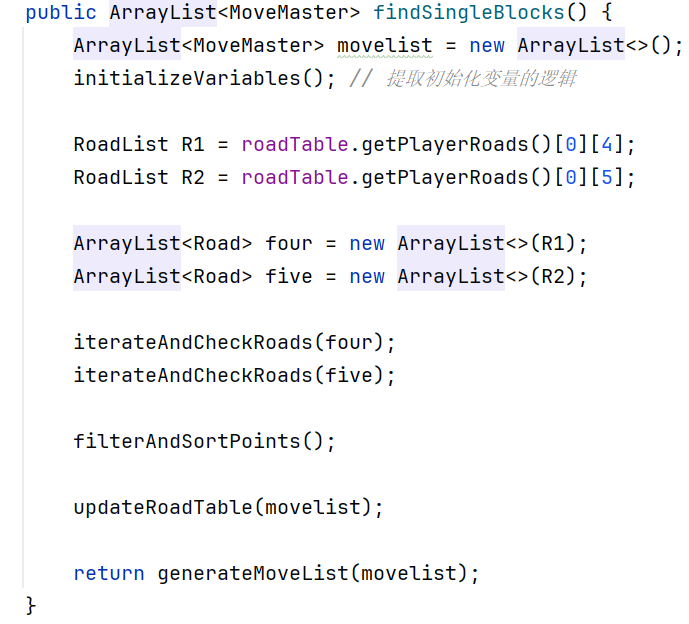


图39 findSingleBlocks方法实现

initializeVariables 方法：

初始化变量的方法，将 pointlist 清空，并将 vis 数组初始化为 false。

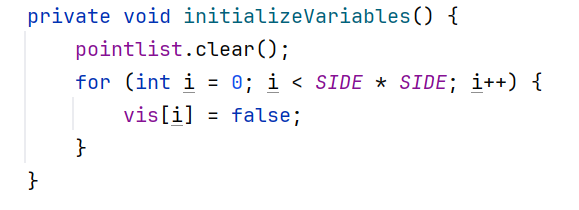


图40 initializeVariables方法实现

iterateAndCheckRoads 方法：

对给定的道路列表进行迭代，并检查每个道路上的点。如果某个点是空白而且没有被访问过，则调用 processPoint 方法。



图41 iterateAndCheckRoads方法实现

processPoint 方法：

处理单个点的方法。首先，修改路表（调用 changeRoads 方法），然后检查特定条件（R1 和 R2 的大小之和是否为0），如果满足条件，则将点添加到 pointlist 中，最后撤销对路表的修改，并将相应位置标记为已访问。

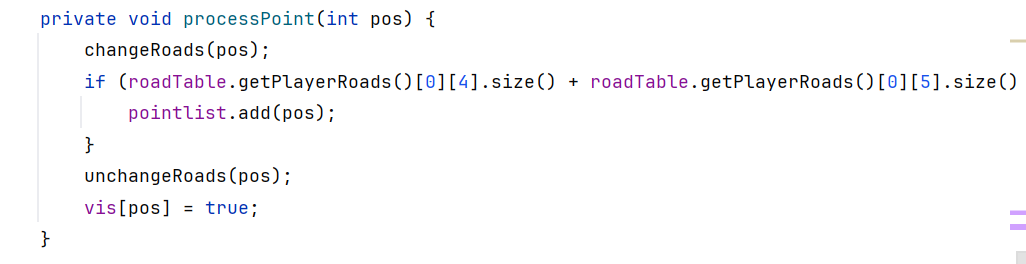


图42 processPoint方法实现

filterAndSortPoints 方法：

对点列表进行筛选和排序的方法。首先，清空 pointslist 并调用 findblanks 方法填充。然后，对 pointslist 进行按分数排序。接着，计算分数平均值，根据平均值筛选点，最终保留高于平均值减一的点。

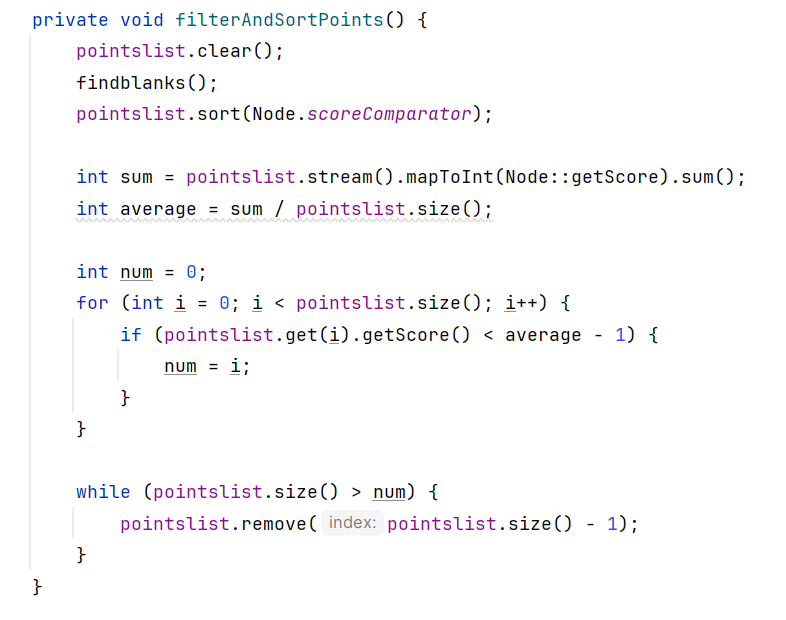


图43 filterAndSortPoints方法实现

updateRoadTable 方法：

更新路表的方法。对 pointlist 中的每个点，以及对 pointslist 中的每个节点，调用 processMoveAndUpdateRoadTable 方法。

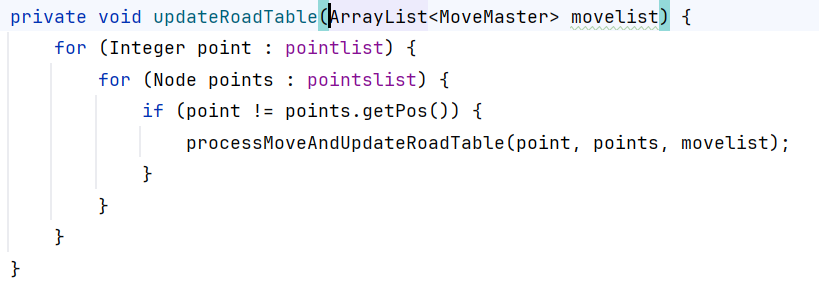


图44 updateRoadTable方法实现

processMoveAndUpdateRoadTable 方法：

处理移动并更新路表的方法。首先，创建一个移动（MoveMaster），然后修改路表，设置移动的分数，最后撤销对路表的修改，并将移动添加到传入的移动列表。

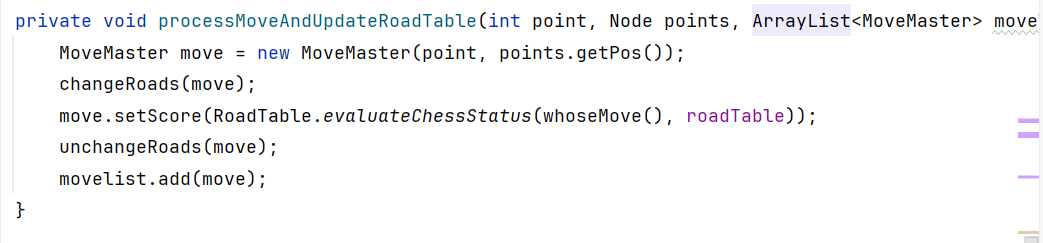


图45 processMoveAndUpdateRoadTable方法实现

generateMoveList 方法：

生成移动列表的方法。这个方法看起来在当前版本中并没有实际的操作，它只是返回传入的移动列表。

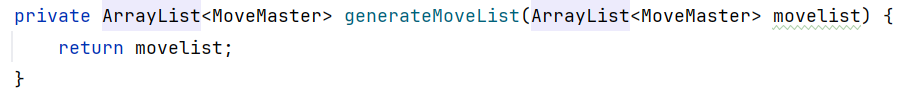


图46 generateMoveList方法实现

**5.6 三威胁防御**

通过双重循环遍历 pointlist 中的每一对点，表示在这两个位置下子。然后，对于每一对点，通过调用 changeRoads 修改路表，计算当前对手可能形成的威胁数量（nowthreats）。如果发现当前威胁数量小于之前记录的最小威胁数量（minthreats），则清空之前的走步列表，并将当前走步加入列表。如果当前威胁数量等于最小威胁数量，则将当前走步加入列表。

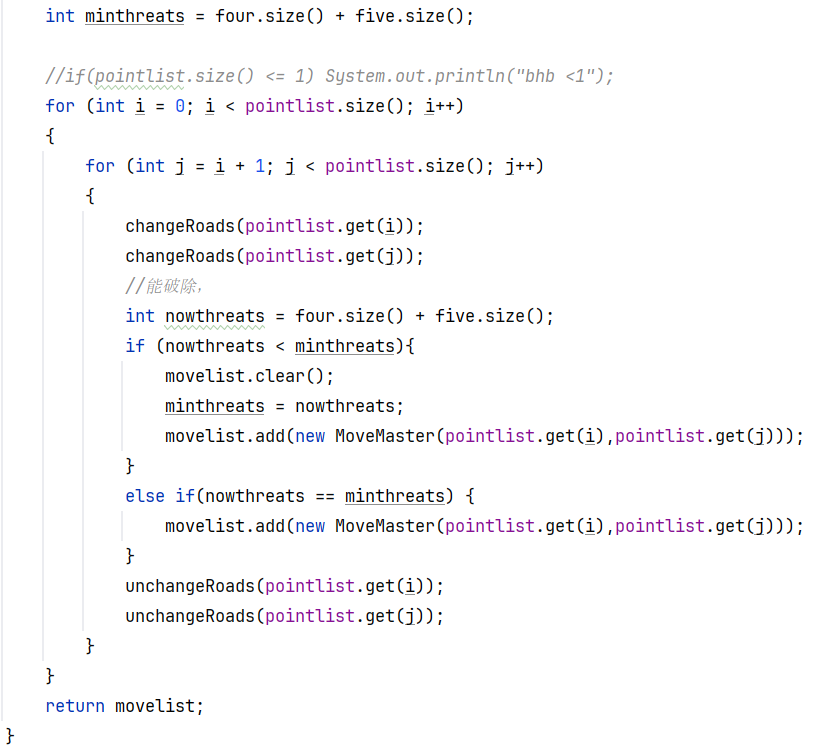


图47 三威胁防御方法实现

**5.7 一般着法**

此段代码是通过使用启发式排序和有限搜索深度，从所有可能的移动中选择出一个较小的子集。这样做是为了在游戏中提高性能，同时仍然能够考虑到当前局面的一定复杂性。

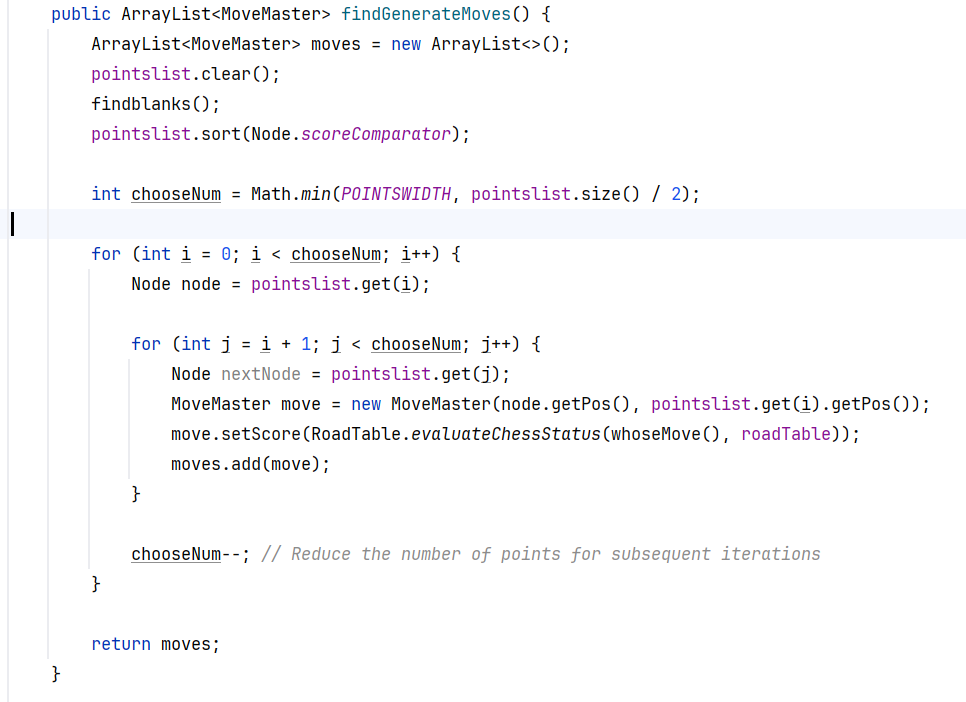


图48 一般着法实现

**6. 棋力与性能对比**

**6.1 α-β剪枝和威胁空间搜索对比**

我们在phase1和phase3中实现了α-β剪枝，phase2和phase4在此基础上增加了DTSS搜索。

phase1和phase2是在α-β剪枝为先手的情况下进行对弈的，phase3和phase4是在DTSS为先手的情况下进行对弈的。

两个阶段的胜率如下：

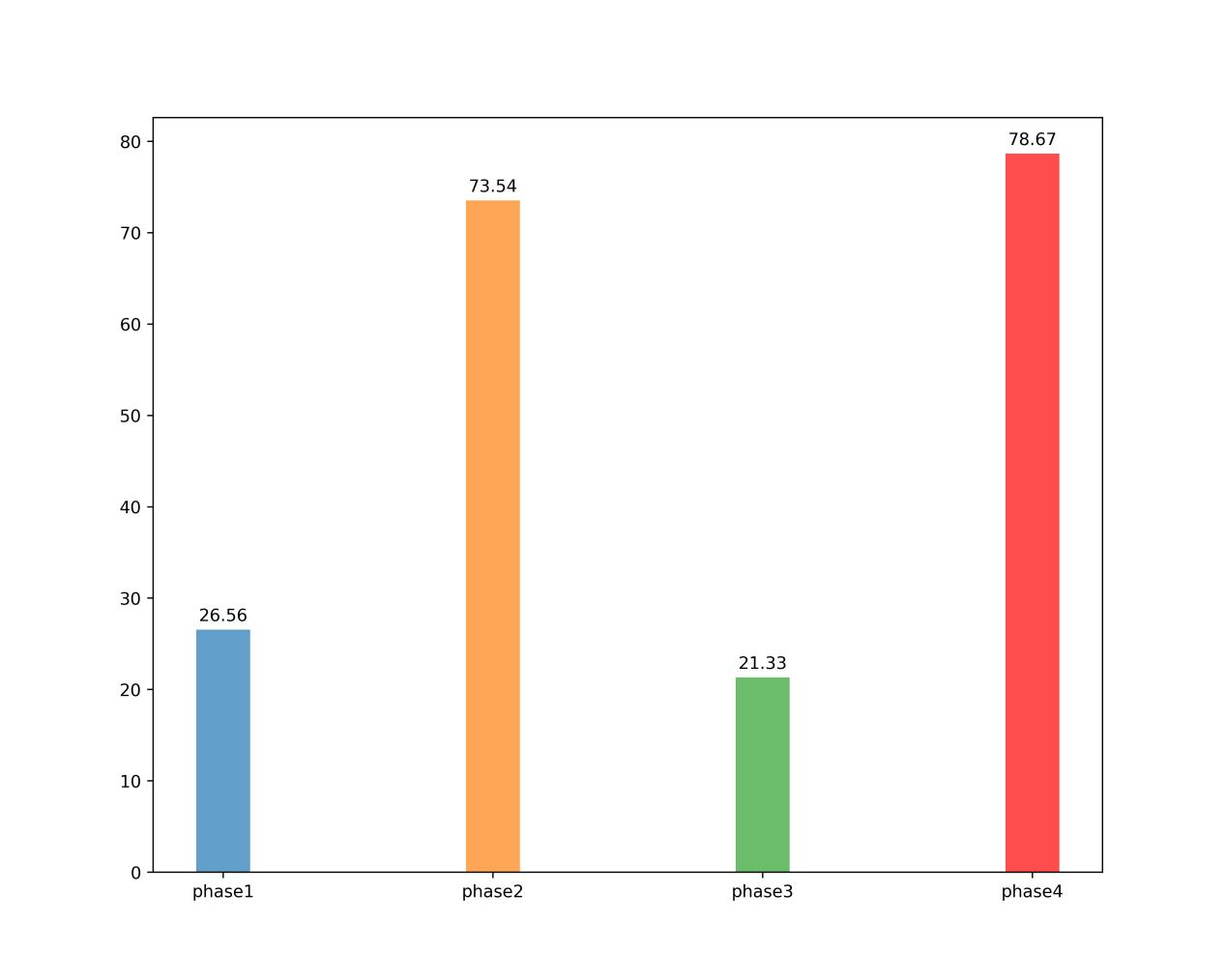


图49性能对比

威胁空间搜索主要专注于寻找可能的杀棋，其目标是在当前局面中找到一种可能的获胜策略。相比之下，AlphaBeta剪枝算法侧重于通过在搜索树中排除不必要的分支，以更高效地确定当前局面下的最佳着法。由于AlphaBeta搜索往往受到搜索深度的限制，可能无法准确获取当前局面的实际博弈理论值。

为了充分利用两者的优势，我们可以采用一种协同的策略。首先，利用威胁空间搜索来快速检测当前局面是否存在可能的杀棋。如果找到了杀棋，那么这就是一个潜在的最佳着法，无需进一步搜索。然而，如果威胁空间搜索未能找到杀棋，那么我们可以转而使用AlphaBeta剪枝算法，以深入搜索更多层次，找到当前局面下的最佳着法。

通过这种组合使用，我们既能在短时间内捕捉到可能存在的杀棋，又能在需要更深入搜索的情况下，利用AlphaBeta剪枝来提高搜索效率。这样的策略在实际博弈中能够更全面地应对不同类型的局面，并取得更好的性能。

**6.2 棋力对比**

我们将g02智能体分别与dtss智能、random02智能体和random03智能体进行了500场比赛，结果如下(剩余未展示出的为平局)：

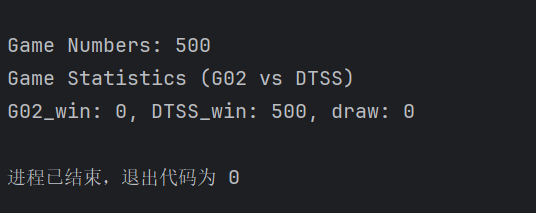


图50 dtss与g02比赛结果

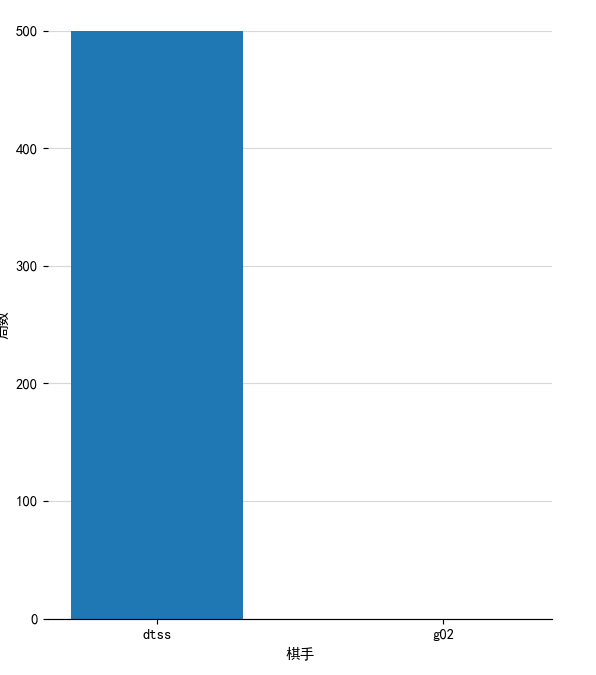


图51 dtss与g02胜利局数对比

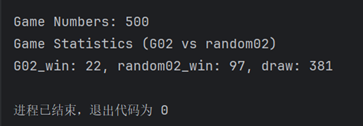


图52 g02与random02比赛结果

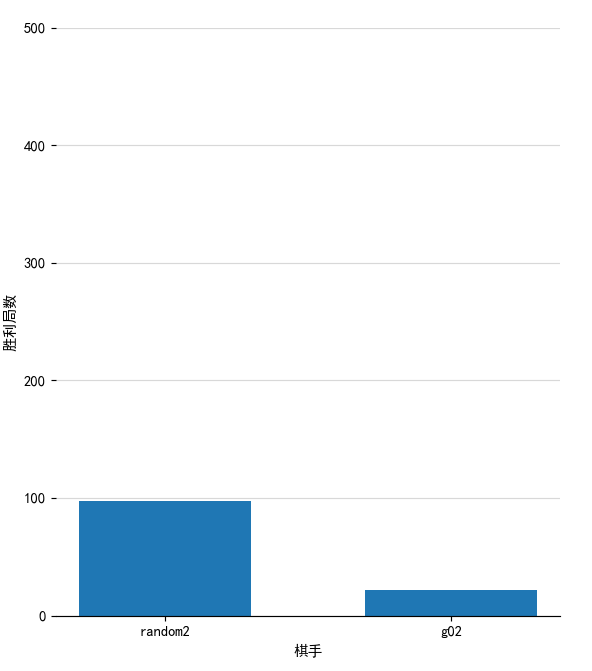


图53 random02与g02胜利局数对比

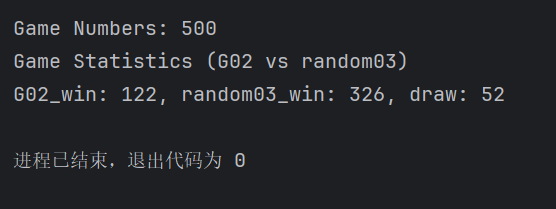


图54 g02与random03比赛结果

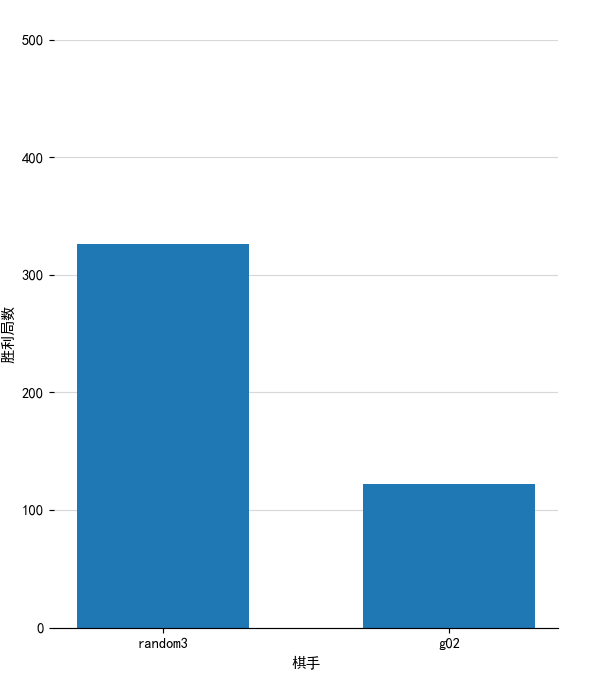


图55 random03与g02胜利局数对比

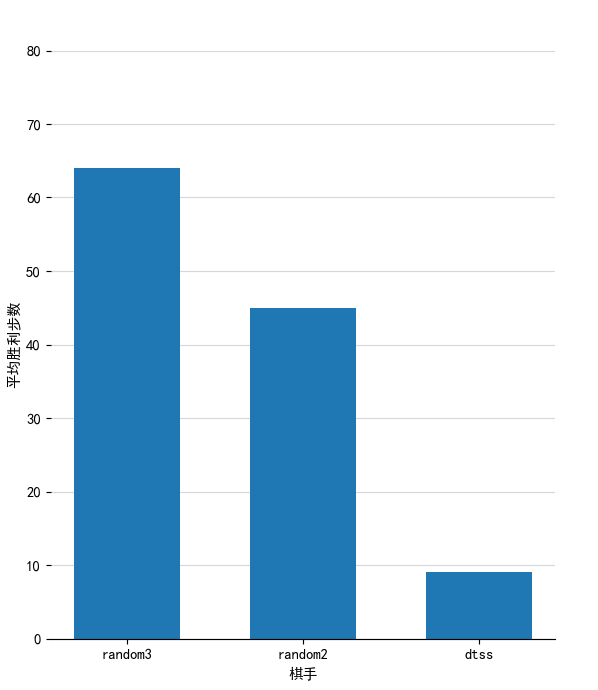


图56 dtss和random02、random03平均胜利步数对比

由上可见，威胁空间算法明显比随机走子算法更加具有智能，胜率更高。

**6.3 各种智能棋力对比**

下方两幅图体现了各种智能的胜率对比，各个算法各两两对弈100盘，统计出胜率并画图。

* ab-DTSS: AlphaBeta和DTSS对弈
* ab-DTSS\_ab: AlphaBeta和DTSS与AlphaBeta相结合智能
* ab-static: AlphaBeta和静态智能
* DTSS-DTSS\_ad: DTSS和DTSS与AlphaBeta相结合智能
* DTSS-static: DTSS和静态智能
* DTSS\_ab-static: DTSS与AlphaBeta相结合智能和静态智能

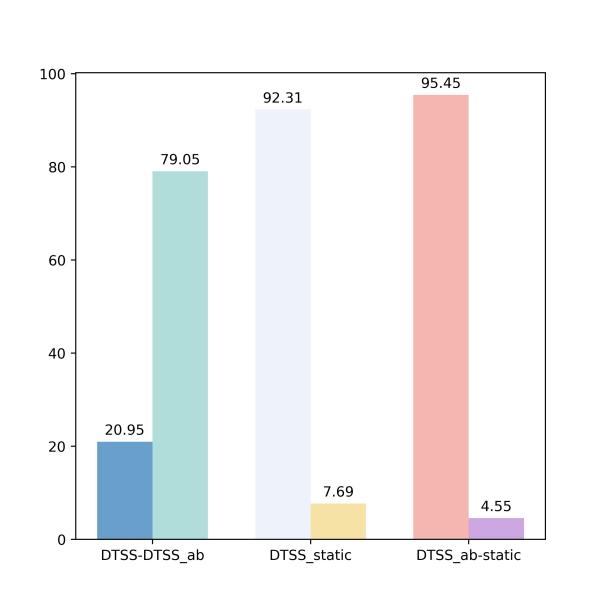


图57各种智能胜率对比

**7. 优化方案及展望**

**7.1动态调整搜索深度**

在搜索算法中，动态调整搜索深度可以是一个有效的优化策略。通过在游戏的不同阶段或关键局面下调整搜索深度，可以提高算法的效率和性能。博弈树搜索深度的增加，会使得搜索的准确程度不断提升，然而也会带来搜索时间和搜索空间的接近指数倍的增长，如何平衡二者，是我们面临的问题。我们使用以下的方案：

1.设置最大搜索深度，并且根据当前时间节点（开局，终局，残局）设置不同搜索深度

2.根据当前局面威胁数，选取不同深度搜索

3.考虑时间限制：

在博弈中，时间通常是一个重要的因素。考虑设置一个最大搜索时间，确保算法在规定的时间内完成决策。动态调整搜索深度时，可以根据剩余时间来进一步调整深度。

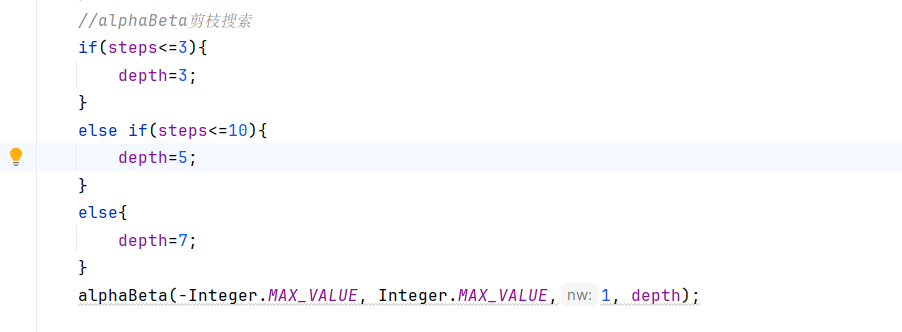


图58 调整深度

**7.2 动态调整静态估值权重**

在六子棋AI中，动态调整静态估值权重是一种关键的优化策略，以确保静态估值函数能够更准确地反映当前局面的特性。通过分解静态估值函数为多个组成部分，并选择需要调整的权重参数，AI系统可以更灵活地适应不同的博弈情况。设定合适的权重调整规则，监控对局的进展，并根据实时观察的局面特点进行自适应调整，有助于提高AI在复杂六子棋局面下的决策准确性。这种自适应的权重调整机制考虑了时间因素，可以在过去经验的基础上不断优化，使得AI在不同阶段和局面下表现更为出色。

**7.3 多线程并行搜索**

在六子棋AI的优化中，多线程并行搜索是一项关键策略，用于充分利用多核计算机的性能，提高搜索算法的效率。通过将搜索算法进行并行化，系统能够同时搜索多个分支，加速对局决策的过程。这种并行搜索策略不仅在提高搜索效率上具有显著优势，而且能够在处理较大的搜索空间时更为高效。通过合理设计并管理多线程，六子棋AI能够更快速地探索决策空间，提高对复杂棋局的分析深度，从而提升整体性能。这种并行搜索的方法有助于使六子棋AI更加适应性强，能够在有限时间内做出更为智能和高效的决策。

**7.4 深度学习**

在六子棋中应用深度学习意味着利用神经网络模型来提升AI的博弈能力。通过深度学习，六子棋AI可以学习并优化棋盘状态到最佳决策的映射关系。神经网络通过多层次的隐藏层来学习抽象的棋局特征，使得AI能够更准确地评估当前局势，预测最佳的下棋位置。训练过程中，神经网络通过大量对弈数据不断优化权重，从而提升对于不同棋局的泛化能力。这种深度学习方法不仅能够使AI更智能地应对复杂的六子棋局面，还有望发展出更强大、适应性更强的博弈策略。

在六子棋中，深度学习的具体应用技术主要包括卷积神经网络（CNN）、强化学习和端到端学习。

**7.4.1卷积神经网络（CNN）**

使用卷积神经网络是六子棋中深度学习的典型应用。CNN能够有效地处理图像数据，而棋盘状态可以被看作是一个二维的图像。通过卷积层和池化层，CNN能够学习到局部的棋局特征，提高对局面的理解和评估。

**7.4.2 强化学习**

强化学习是另一个关键的深度学习技术，用于训练AI在六子棋中做出更优决策。通过与自身或其他对手进行对弈，深度学习模型可以通过强化学习框架来学习优化策略。这使得AI能够逐步改进并在实际对局中取得更好的表现。

**7.4.3 端到端学习**

端到端学习是指直接从输入数据到最终输出结果的学习方式，无需手动设计特征或规则。在六子棋中，可以采用端到端学习的方法，将棋盘状态作为输入，让神经网络直接学习到最佳的下棋策略，从而更好地适应不同的局面和对手。

**7.4.4 蒙特卡洛树搜索与深度学习的结合**

一种常见的技术是将深度学习与蒙特卡洛树搜索相结合。通过使用深度学习模型评估局面，结合蒙特卡洛树搜索进行决策，可以在提高搜索效率的同时保持对局面的准确评估。

**8. 总结及成员感想**

**8.1 周忠康**

在本次的实验中我主要负责**前期路表结构的设计以及整体项目的运行**，主要实现了**项目一二阶段的编写运行**，在这个过程中我借鉴了项目一中静态函数估值的思路，参考了论文中给出的权重值完成了第一阶段。同时由于对于威胁数防御的需要，我也学习了**威胁空间的算法**，如单威胁，双威胁等，并在阶段一中使用了该技术。在阶段二中我和王颢凯一起学习了**α-β剪枝**算法，这极大的减少了计算量。基于整个项目的需要，我学习**java中的多线程**技术方便同步测试，同时**优化了威胁空间和博弈树**，这让我对整个项目更加了解，也对对抗博弈有了新的认识。在实现整个项目的过程中，我的机器博弈的知识有了极大扩充。这个项目是我们对抗博弈的一个尝试，在完成后我认识到对于对抗博弈存在着很多我们可以采用的方法，比如深度学习等等，这也是我以后继续学习的方向。

**8.2 王颢凯**

本次项目中我主要负责了一二阶段的编写，协助完成项目基本类的搭建。主要完成了 **α-β剪枝**、**最佳着法搜索**、**单威胁防御**、**三威胁防御**以及**一般着法搜索**的实现。并在项目最后阶段对α-β剪枝和威胁空间搜索进行了性能对比，并做了**可视化展示**。同时对各个阶段的智能，两两各进行了100盘对弈，并统计各方胜率，最后进行可视化展示。

在完成项目过程中学会了如何高效管理棋盘上的路表信息，以便快速找出双方的有效路。这对于机器人的决策过程至关重要，提高了整体性能。

了解并实现了α-β剪枝，通过有效地剪掉不必要的搜索分支，提高了搜索效率，使得机器人能够更快速地找到最优解。

总体而言，通过这个项目，我不仅学到了六子棋的基本规则和对弈策略，还深入理解了博弈树、α-β剪枝和威胁空间搜索等核心算法的实际应用。在代码编写过程中，我注重了代码的规范性和稳定性，并通过实验对比进一步加深了对各算法的理解。这次实验为我提供了丰富的实践经验，使我在人工智能领域的学习和探索取得了实质性的进展。

**8.3 田晶怡**

在本次项目中我主要负责**威胁空间搜素算法的设计和尝试了实现蒙特卡洛树搜索与威胁空间搜素相结合，**并将四个智能体AI进行了**棋力对比**。网络上对于威胁空间搜索算法的介绍和教程少之又少，所以通读了论文《Two-Stage Monte Carlo Tree Search for Connect6》了解到了威胁空间搜索算法的一些基本知识，同时论文中提到的两阶段蒙特卡洛树搜索拓展了我的思路，并想要尝试将它复现于项目中，遗憾的是，到目前为止我的两阶段MCTS算法代码仍有异常，未能进行展示，后续我将继续尝试优化争取将它进行实现。通过本次项目，我深入了解并学习了机器博弈的相关知识，在项目推进的过程中我也逐步理解了曾经在我看来很神秘的AlphaGo的实现原理和方法，并且对机器学习模型训练的相关知识也有了较为直观的学习和了解。本次实验项目让我受益匪浅。

**8.4 曹文汀**

在本次项目中我主要负责六子棋实验部分的编程实现和报告撰写，实现了随机走步的三种走法，并对三种走法进行了分析比较。后续我还探究了两个参数对走法3胜率的影响，并进行了可视化，更加深刻地理解了六子棋的着法及其公平性。

这次的实验，让我学习了六子棋的基本规则和对弈策略。机器博弈需要根据各个棋种的特点，选择合适的搜索算法和评估函数。在和其他成员的合作中，我学习了博弈树、极小极大搜索、α-β剪枝、威胁空间搜索等算法。同时，通过阅读论文，我理解了蒙特卡洛树搜索算法，包括Selection, Expansion, Simulation, Back Propagation四个过程，是一种Exploitation和Exploration之间的平衡，适用于分支很大的博弈树搜索，它也是成就AlphaGo的重要算法之一。同时，我也了解到机器博弈还可以进行很多优化，例如多线程并行能够加速搜索，深度学习能够提高精度和速度等。机器博弈是检验人工智能发展的重要领域之一，我们应对未来人工智能领域在机器博弈领域的发展充满信心。

在编写代码的过程中，我对Java编程语言更加熟练，对面向对象编程封装、继承、多态的实现更加深刻。本次项目是我们机器博弈的入门尝试，我希望通过学习人工智能的算法和深度学习的方法，未来能够在人工智能领域继续探索前进。

**9. 参考文献**

[1] Wu I, Huang D Y, Chang H C. Connect6[J]. ICGA Journal, 2005, 28(4): 235-242.

[2] 张小川, 陈光年, 张世强, 等. 六子棋博弈的评估函数[J]. 重庆理工大学学报: 自然科学, 2010 (2): 64-68.

[3] [蒙特卡罗树搜索（MCTS）【转】 - 彼岸花杀是条狗 - 博客园 (cnblogs.com)](https://www.cnblogs.com/yifdu25/p/8303462.html)

[4] Shi-Jim Yen, Jung-Kuei Yang, IEEE,2011.Two-Stage Monte Carlo Tree Search for Connect6.