



中国海洋大学
OCEAN UNIVERSITY OF CHINA

六子棋博弈程序研究开发报告

课程名： 人工智能

任课教师： 徐建良

组名： 傲天良辰

年级专业： 2015级计算机科学与技术

组员： 曲磊钢、潘迎港、苏念亿、
茆加余、蒙腾斌



年 月 日

目录

一、计算机博弈的历史及最新进展	2
二、六子棋及六子棋博弈算法研究综述	5
三、“傲天良辰”博弈程序的主要思路及特点	11
四、版本 1 的主要数据结构及关键算法	12
4.1 主要数据结构	12
4.2 关键算法及其实现	15
4.2.1 盘面估值	15
4.2.2 着法生成	16
4.2.3 盘面更新	17
4.2.4 博弈树搜索	17
五、对版本 1 的改进	19
5.1 主要数据结构的改进	19
5.2 关键算法的改进	21
5.3 博弈搜索效率及胜率的对比	23
六、总结与展望	27
七、参考文献	28
附录	29

一、计算机博弈的历史及最新进展

1. 引言

计算机博弈是人工智能领域一个极其重要且最具挑战性的研究方向之一, 它的研究为人工智能带来了很多重要的方法和理论, 产生了广泛的社会影响和学术影响以及大量的研究成果。计算机博弈又称机器博弈, 而机器博弈是博弈论应用的一个方面。那么什么是博弈论? 首先谈谈什么是博弈。博弈是指一些个人、团队或其他组织, 面对一定的环境条件, 在一定的规则约束下, 依靠所掌握的信息, 同时或先后, 一次或多次, 从各自允许选择的行为或策略进行选择并加以实施, 并从中各自取得相应结果或收益的过程。博弈论又称对策论, 就是系统研究各种各样博弈中参与人的合理选择及其均衡的理论。博弈论广泛地应用于经济、军事、航空调度、天气预报、资源勘探等各个方面。当然, 计算机博弈也是其应用之一, 同时, 计算机博弈又是人工智能的经典研究领域。那么计算机博弈就把计算机(机器)、博弈论、人工智能三者组成了一个有机整体, 对于计算机博弈的研究, 既能推动博弈论的发展, 又能推动人工智能的发展。

2. 计算机博弈简史

在过去的半个世纪里, 世界各地的学者花费了大量的心血对于计算机博弈包括奥赛罗、checker、国际象棋、中国象棋、五子棋、围棋进行研究。这是因为计算机博弈是人工智能的一块试金石, 其中二人零和完备信息博弈的技术性和复杂性较强, 是人们研究博弈的集中点。二人零和随机性研究的一个代表是 Backgammon, 并且产生了很大影响。桥牌是研究不完备信息下的推理的好方法。高随机性的博弈项目趣味性很强, 常用于娱乐和赌博, 是研究对策论和决策的好例子。博弈的一个标准性问题, 各种搜索算法、模式识别及智能方法在计算机博弈中都可以得到广泛的应用。在长时间的研究中, 涌现出大量令人震惊的成果。人机大战成为检验机器博弈水平最好的一个重要尺度, 以下列举了一些著名的事件。

1) 1950 年 C. Shannon 发表了两篇有关计算机博弈的奠基性文章 (Programming A Computer for Playing Chess 和 A Chess-playing Machine)。1951 年 A. Turing 完成了一个叫做 Turochamp 的国际象棋程序, 但这个程序还不能在已有的计算机上运行。1956 年 Los Alamos 实验室的研究小组研制了一个真正能够在 MANIAC-I 机器上运行的程序 (不过这个程序对棋盘、棋子、规则都进行了简化)。1957 年 Bernstein 利用深度优先搜索策略, 每层选七种走法展开对局树, 搜索四层, 他的程序在 IBM704 机器上操作, 能在标准棋盘上下出合理的着法, 是第一个完整的计算机国际象棋程序。

2) 1958 年, 人工智能届的代表人物 H. A. Simon 预言 “计算机将在十年内赢得国际象棋比赛的世界冠军。” 当然, 这个预言过分乐观了。

3) 1967 年 MIT 的 Greenblatt 等人在 PDP-6 机器上, 利用软件工具开发的 Mac Hack VI 程序, 参加麻省国际象棋锦标赛, 写下了计算机正式击败人脑的记录。

4) 从 1970 年起, ACM Association for Computing Machinery 开始举办每年一度的全美计算机国际象棋大赛。从 1974 年起, 三年一度的世界计算机国际象棋大赛开始举办。

5) 1981 年, CRA YBL ITZ 新的超级计算机拥有特殊的集成电路, 预言可以在 1995 年击败世界棋王。1983 年, Ken Thompson 开发了国际象棋硬件 BELL

E, 达到了大师水平。

6) 80 年代中期, 美国的卡内基梅隆大学开始研究世界级的国际象棋计算机程序——“深思” 1987 年, “深思” 首次以每秒 75 万步的思考速度露面, 它的水平相当于拥有国际等级分为 2450 的棋手。1988 年, “深思” 击败丹麦特级大师拉尔森。1989 年, “深思” 已经有 6 台信息处理器, 每秒思考速度达 200 万步, 但在与世界棋王卡斯帕罗夫进行的“人机大战”中, 0 比 2 败北。

7) 1989 年第一届计算机奥林匹克大赛在英国伦敦正式揭幕, 计算机博弈在世界上的影响日益广泛。

8) 1997 年, 由 1 名国际特级大师, 4 名电脑专家组成的“深蓝”小组研究开发出“更深的蓝”, 它具有更加高级的“大脑”, 通过安装在 RSö6000S 大型计算机上的 256 个专用处理芯片, 可以在每秒钟计算 2 亿步, 并且存储了百年来世界顶尖棋手的 10 亿套棋谱, 最后“超级深蓝”以 3.5 比 2.5 击败了卡斯帕罗夫。成为人工智能领域的一个里程碑,

9) 在全世界范围内引发了震动。在年, 中国举办了“浪潮杯”全国首届机器博弈暨人机大战, 以“浪潮天梭”服务器为载体的“棋天大圣”与中国象棋特级大师许银川之间的巅峰对决, 最终以两战皆和告终。赛后, 许银川说“电脑一步可以算十几个变化, 而我只能凭借经验和理解与它对抗”。还说“和计算机下棋压力很大, 可以用暗潮涌动、惊心动魄来形容。”

3. 计算机博弈最新进展

目前, 对于像五子棋、中国象棋等棋类游戏的计算机博弈算法研究已相对成熟。其他很多棋类的计算机水平都已达到了世界冠军的水平。但是仍然有些游戏的博弈程序没有达到人类满意的程度或者尚处于初级阶段, 比如牌类游戏、围棋等。机器博弈现阶段面临的主要问题是解决非完备信息博弈和如何解决搜索空间巨大的游戏。

国外: 国外的机器博弈发展较早, 成立于年的国际组织一直致力于推动该领域的发展。该领域的一部分重要研究成果发表在学术期刊上。还有相当大的一部分学术成果发表在重要国际期刊和会议上, 如, , 颇和等。在的深蓝战胜卡斯帕罗夫之前, 领域内的研究者一直把国际象棋作为该领域的“果蝇”。一些研究者不但提出了优秀的理论成果, 还将博弈软件开放源代码, 供其它研究者阅读和使用。许多开源软件已经达到了大师级水准, 如国际象棋的几、洲, 等。此外, 为了测试博弈引擎的性能, 国际象棋机器博弈有著名的接口程序, 可以提供不同引擎之间的对战, 还是一个开源的程序。自年卡氏败北之后, 围棋的研究就逐渐变为机器博弈的热点, 并取代了国际象棋成为新的果蝇。近年来, 涌现了相当数量的围棋开源软件, 最著名的如川等。在围棋机器博弈社区, 提供了博弈软件间竞赛的服务器, 如著名的’和’等。这些服务器提供接入的协议, 只要遵循协议就可以自动进行竞赛, 从而自动化地进行博弈引擎的棋力的测量。其它种类的棋类博弈软件也都有相应的开源软件和工具软件。这些资源为机器博弈的研究和普及打下了坚实的基础, 以此为基础开展相关的研究, 可大大地减少科研工作量和周期。

国内: 中国大陆的机器博弈研究起步较晚, 近年来发展较为迅速。经过若干年筹办, 在年中国人工智能学习机器博弈专业委员会正式成立。该委员会致力于推广和普及机器博弈知识, 提高国内机器博弈的研究水平, 举办每年一度的中国机器博弈锦标赛、人机大战与学术研讨会。历年的赛事和学术会议吸引了不少海内外的华人学者的积极参与。近年来, 国内的一些科研院所在机器博弈领域做出了重要贡献, 包括东北大学徐心和团队等对中国象棋的研究, 该团队的“棋天圣”

中国象棋机器博弈软件曾数次获得国际国内的冠军,还在人机大战中屡次战胜或战和人类大师此外,东北大学的六子棋、点点连格等棋类的机器博弈研究也较为活跃北京邮电大学的刘知青团队等对围棋机器博弈做了大量的研究已逝的中山大学的陈志行先生曾多次夺得国际围棋机器博弈冠军,一度处于世界领先水平北京理工大学黄鸿教授的团队对中国象棋、围棋、六子棋、亚马逊棋类的研究也有出色的成果南京航空航天大学的夏正友、哈尔滨工业大学的王轩、重庆理工大学的张小川、桂林电子科技大学的郝卫东、大连理工大学的高强等也做出了重要贡献。值得一提的是,许多专业的程序员和爱好者也加入研究和实践中,在缺乏强有力的资金支持和良好的科研环境的条件下,仍然开发出了具有较高水平的程序。总之,虽然起步较晚,但国内的机器博弈研究正呈现出星火燎原之势,研究的范围也在逐步扩大,包括四国军棋、兵棋、桥牌等的研究也正在如火如荼地进行。

划时代进展:2016年3月,阿尔法围棋与围棋世界冠军、职业九段棋手李世石进行围棋人机大战,以4比1的总比分获胜;2016年末2017年初,该程序在中国棋类网站上以“大师”(Master)为注册帐号与中日韩数十位围棋高手进行快棋对决,连续60局无一败绩;2017年5月,在中国乌镇围棋峰会上,它与排名世界第一的世界围棋冠军柯洁对战,以3比0的总比分获胜。围棋界公认阿尔法围棋的棋力已经超过人类职业围棋顶尖水平,在GoRatings网站公布的世界职业围棋排名中,其等级分曾超过排名人类第一的棋手柯洁。

二、六子棋及六子棋博弈算法研究综述

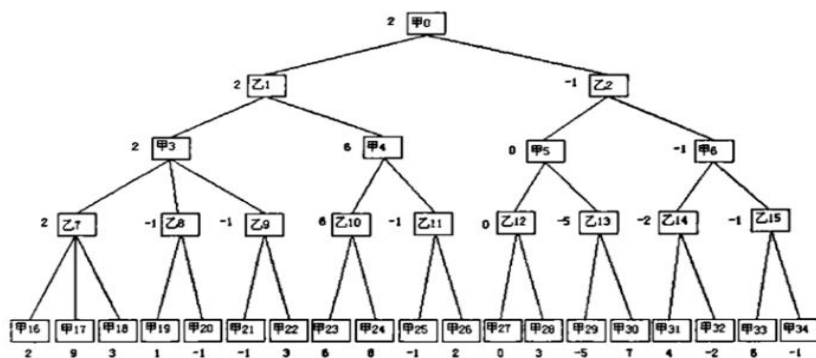
六子棋由台湾吴毅成教授于两年前发明，现在逐渐开始兴起，作为一个刚刚兴起不久的棋类游戏，其计算机博弈算法的研究还相对较少。即使目前已经出现六子棋的论坛以及比赛的平台，但只限于人人对弈。真正对于六子棋计算机博弈算法以及系统的研究还不多。吴毅成教授给出了六子棋的公平性问题以及基于迫著(Threats)的胜利策略，但是对于其计算机博弈问题没有给出更加深刻的阐述，同时也没有全面解决六子棋计算机博弈问题。

六子棋计算机博弈系统包括四个主要部分：搜索引擎、走法生成、评估函数和开局库。在搜索引擎模块中采用什么样的搜索算法，在走法生成模块中怎样确定走法，在评估函数模块中如何确定参数以及怎样确定适应度函数，在开局库中存储哪些开局模板，这都是我们研究六子棋计算机博弈需要解决的问题。其中，搜索算法和评估函数是六子棋计算机博弈中我们需要深入研究和解决的核心问题。

1. 博弈树

博弈树是把计算机和用户所有可能的走法和局面罗列出来的一颗树。黑白双方交替地按合理走法把树展开，树的每一个节点都表示某一个特定局面。根节点表示的是当前需要计算的局面，中间节点表示的是对弈过程中的某一个局面，叶子节点是树的最底端，称为搜索深度。整个博弈树描述的是从当前局面出发，包含所有可能的对弈过程的搜索树。

六子棋计算机博弈问题也就转化为寻求最佳路径的问题。表示可以推导的局面。叶子节点和根节点之间的最大距离博弈树：任何棋类游戏都要定义一棵有根的（即博弈树），一个结点就代表棋类的一个局面，子结点就是这个局面走一步可以到达的一个局面。例如下图是五子棋的极大极小搜索博弈树



一般搜索树中有三种类型的结点：

- (1) 偶数层的中间结点，代表棋手甲要走的局面；
- (2) 奇数层的中间结点，代表棋手乙要走的局面；
- (3) 叶子结点，代表棋局结束的局面，即棋手甲或棋手乙获胜，或者是和局。

2. 搜索引擎

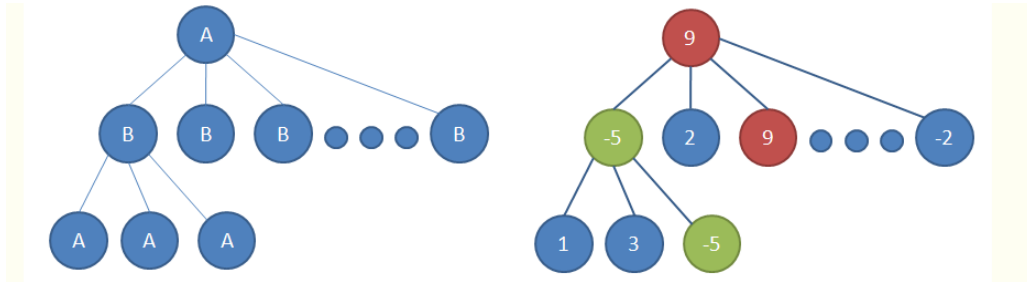
搜索算法是博弈树求解的灵魂，只有有了有效的搜索算法才能在有限的时间找到正确的解。搜索法是求解此类图模型的基本方法。我们无法搜索到最终的

胜负状态，于是搜索的目标便成为如何在有限深度的博弈树中找到评估值。以下是我们小组了解到的一些相关算法。

3. 极大极小搜索

在搜索树中，表示 A 走棋的节点即为极大节点，表示 B 走棋的节点为极小节点。

如下图：A 为极大节点，B 为极小节点。称 A 为极大节点，是因为 A 会选择局面评分最大的一个走棋方法，称 B 为极小节点，是因为 B 会选择局面评分最小的一个走棋方法，这里的局面评分都是相对于 A 来说的。这样做就是假设 A 和 B 都会选择有限的搜索深度内，得到的最好的走棋方法。



伪代码如下（来自维基百科）：

```
function minimax(node, depth) // 指定当前节点和搜索深度
    // 如果能得到确定的结果或者深度为零，使用评估函数返回局面得分
    if node is a terminal node or depth = 0
        return the heuristic value of node
    // 如果轮到对手走棋，是极小节点，选择一个得分最小的走法
    if the adversary is to play at node
        let  $\alpha := +\infty$ 
        foreach child of node
             $\alpha := \min(\alpha, \text{minimax}(\text{child}, \text{depth}-1))$ 
    // 如果轮到我们走棋，是极大节点，选择一个得分最大的走法
    else {we are to play at node}
        let  $\alpha := -\infty$ 
        foreach child of node
             $\alpha := \max(\alpha, \text{minimax}(\text{child}, \text{depth}-1))$ 
    return  $\alpha$ 
```

4. $\alpha - \beta$ 剪枝搜索

基本思想：根据倒推值的计算方法，或中取大，与中取小，在扩展和计算过程中，能剪掉不必要的分枝，提高效率。

(1) 定义：

α 值：有或后继的节点，取当前子节点中的最大倒推值为其下界，称为 α 值。节点倒推值 $\geq \alpha$ ；

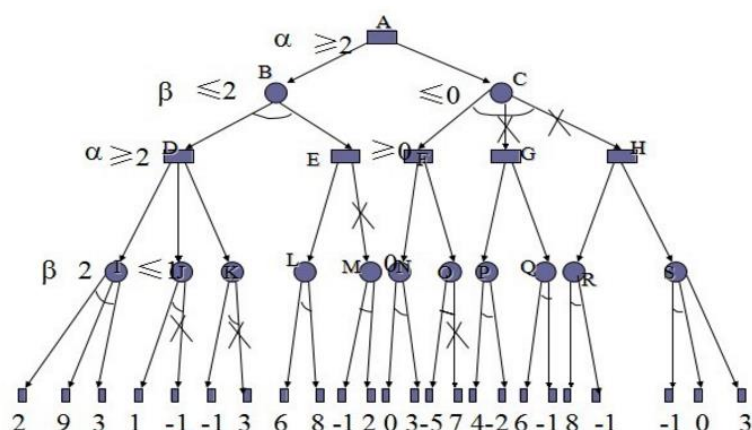
β 值：有与后继的节点，取当前子节点中的最小倒推值为其上界，称为 β 值。节点倒推值 $\leq \beta$ ；

(2) $\alpha - \beta$ 剪枝：

① β 剪枝：节点 x 的 α 值不能降低其父节点的 β 值，x 以下的分支可停止搜索，且 x 的倒推值为 α ；

② α 剪枝：节点 x 的 β 值不能升高其父节点的 α 值， x 以下的分支可停止搜索，且 x 的倒推值为 β ；

以下为根据剪枝原理进行的搜索树剪枝过程：



5. 启发式搜索 (Heuristic search)

具体问题的领域决定了初始状态、算符和目标状态，进而决定了搜索空间。

因此，具体问题领域的信息常常可以用来简化搜索。此种信息叫做启发信息，而把利用启发信息的搜索方法叫做启发性搜索方法。

6. 负极大值算法

前面谈到博弈树的搜索是一种“变性”搜索。在偶数层进行“Max 搜索”，而在奇数层进行“Min 搜索”。这无疑给算法的实现带来了一大堆麻烦。Knuth 和 Moore 充分利用“变性”搜索的内在规律，在 1975 年提出了意义重大的负极大值算法。它的思想是：父节点的值是各子节点值的变号极大值，从而避免奇数层取极小而偶数层取极大的尴尬局面。

$$F(v) = \max \{-F(v_1), -F(v_2), \dots, -F(v_n)\}$$

其中， v_1, v_2, \dots, v_n 为 v 的子节点。

5. 走法生成

走法生成是指将一个局面的所有可能的走法罗列出来的那一部分程序，也就是用来告诉其它部分下一步可以往哪里走的模块。各种棋类的规则不同，走法生成的复杂程度也有很大的区别。一般说来，在一种棋类游戏中，双方棋子的种类越多，各种棋子走法的规则也越多，则在程序中，走法生成的实现就越复杂。

在六子棋的对弈程序中，由于双方只有黑白各一种棋子，并且在走子的过程中，没有吃子、提子等规则的存在，双方轮流走子，只要有一方首先在棋盘的水平、垂直、斜线方向上形成连续的六颗棋子，就获得胜利。因此，对于六子棋的走法生成来说，棋盘上的任意空白位置都是合法的走法。但在实际中，并不一定要找到所有的空白位置，如果在开局时就把那些边界的空白位置当作下一步的走法，这样就会导致脱离战场的危险，对于走棋的一方是非常不利的。经常采用的着法生成方法包括：棋盘扫描法、模板匹配法、预置表法，有时还结合使用。

棋盘扫描法需根据不同棋类游戏的规则，定义可行区域才能使用。该着法的生成过程是反复在棋盘上扫描有效区域、制约条件和落子状况，确定有效的落子位置。不同的棋类游戏有不同的规则，比如五子棋，棋盘有效区域内的所有空白的交叉点都是可行落子点；这样在五子棋的走法产生模块里，只要扫描棋盘，寻找到所有的空白，就可以罗列出所有符合规则的下一步。中国象棋根据游戏规则

的不同，对落子位置有较多的限制，不适合用该方法。在着法的表达上，棋盘扫描法最为直观，在五子棋、围棋等棋类设计中经常使用，但时间开销巨大。

模板匹配法主要在中国象棋机器博弈系统中使用，比较适合使用模板匹配方法的棋子是马、相（象）。以中国象棋为例，当动子（某一回合中计算机方要移动的棋子）确定之后，其落址（根据中国象棋中兵种的走步规则，找到的可能的落子位置）与提址（该动子所在的棋盘位置）的相对关系便被固定下来。所以可以为某些动子设计“走子匹配模板”，只要与提址匹配，就能快速找到落址。如图 2.1 中国象棋的走马匹配模板。当马位于中心提址，○代表符合“马走日”这一规则的落址，×代表蹩马腿的制约条件，根据×的具体分布，很容易判断可能的落址。再通过单项比特矩阵比对，实现“本方子则止，对方子则吃”，完成“提-动-落-吃”内容的确定。

预置表法是最为常用的着法生成方法，该方法的本质是用空间换时间。针对当前棋局，给出动子可能的吃子走法和非吃子走法，以空间换取对弈过程中的生成着法的扫描时间。在中国象棋计算机博弈中，对于炮、车等可以利用预置表法进行定义。如下图 2.2 所示是炮的一个行着法预置表。

6. 评估函数

对于博弈树求解有了良好的搜索算法还只是问题的一个方面，问题的另一个方面就是评估函数。只有有了良好的评估函数才能保证较快地找到正解。而评估函数是对棋局的综合评估，该函数的好坏直接决定解题能力强与弱。通常一个优秀的棋手总有一个良好的对棋局的判断能力，能够协调各棋子的关系、取舍，有机地组织各棋子的进攻步调，控制棋局的发展。因此如果要把这一整套的思维物化成一个数值函数来评估，本身就是一个相当复杂的问题。根据不需要人工干预的方法，我们小组找到了以下方法：

爬山法(Hill-Climbing)。类似于交手法，每次对权重作很小的改变，测试改变后的表现，仅当成绩提高时才采纳这个改变，需要重复很多次。这个方法看上去很慢，并且只能找到“局部最优”的组合(即评价可能很差，但是任何很小的改变都会使评价更差)

模拟退火法(Simulated Annealing)。类似于爬山法，也是对权重做出改变来提高成绩的。但是如果改变没有提高成绩，有时候(随机地，给定一个几率)也采纳改变，试图跳出全局最优。这个方法需要给定一些几率，从几率高、梯度大的条件开始，然后逐渐减小。模拟退火法比爬山法更慢，但是最终可能得到比较好的值。

遗传算法(Genetic Algorithms)。爬山法和模拟退火法可以得到一组好的权重，它们是逐渐变化的。相反，遗传算法可以得到几组不同的好的权重，不断增加新的组合跟原来的做比较(取用某组中的某个权重，另一组中的另一个权重，互相交换得到新的)，通过淘汰坏的组合来控制种群的数量。

神经网络(Neural Networks)。实际上这更多地是一种评价函数的类型，而不是用来选择权重的：神经元是阈值(输入权重的和)的函数，第一层神经元输入的关于局面的性质(例如位棋盘表示中的某几个位)就可以构造网络，然后前一层的结果输入到后一层。因此单输入神经元的单层网络就等同于我们上次讨论过的一阶评价函数，但是接下来就可以构造更复杂的神经网络了，而且用这种方法作为评价函数是不难的(只要根据输入的改变来重新计算神经元的输出就可以了)。问题仍然像前面所说的，如何设置权重？除了前面的方法外，针对神经网络还发展出一些方法，例如“暂时差别学习”(Temporal Difference Learning)。其

基本思想是确定网络何时会作出坏的评价，并且让每个权重增加或减小看是否会评价得更好，这很类似于爬山法。跟其他自动学习的方法相比，神经网络的好处就在于它不需要很多人类的智慧：你不需要懂得太多的棋类知识，就可以让程序有个比较好的评价函数。但是根据目前我们掌握的情况，根据自己的智慧来做评价函数，要比机器学习做得好，并且做得快。

7. 开局库

目前，六子棋爱好者提出了多种开局命名规则，包括开局命名法, 山水开局命名法, DIF 开局命名法。现在通常采用 DIF 开局命名法，即利用坐标进行命名。统一坐标为：横坐标用 A-S 表示，纵坐标由下到上用 1-19 表示。黑方第一手棋子下于 J10，即围棋中所谓的天元。白方第一手两子需要重新定位，将依照以下规则做旋转对称

① 决定第一子。在所有八种旋转对称的棋型中，先寻找所有南南西方（约棋盘 1/8 区域）出现的棋型。在这 1/8 区域的棋型中，找寻距离天元最近者（先看纵轴，再看横轴）为第一子，另一子为第二子。

② 去除重复对称盘面。以第二子在右方或下方为先。当在上项中找到第一子后，若仍有对称致使无法区分时，则看第二子，以在右方或下方为先。

③ 重定位后，进行白第一手的命名。坐标命名：直接以第一手坐标及第二子坐标命名。可以按照下面的规则将命名简易化。第一子依据其距离天元情形，改用更简单的命名方式如下：第一子依照其坐标，如 J9，称之为 J9-珠型。简称的规则为：T，代表往下一格；X，代表往左下一格。

下面是几种较常用的定石：

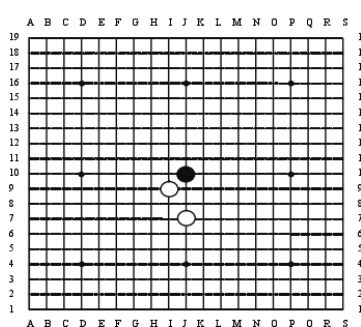


图 2.5 X-J8 定石

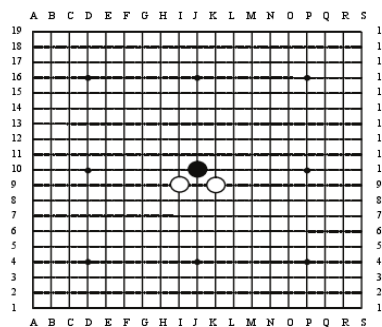


图 2.6 X-K9 定石

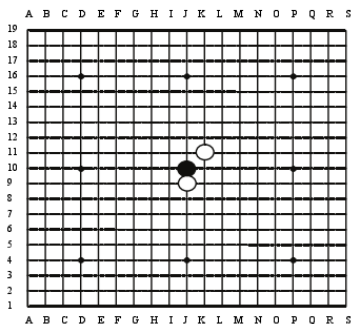


图 2.7 T-K11 定石

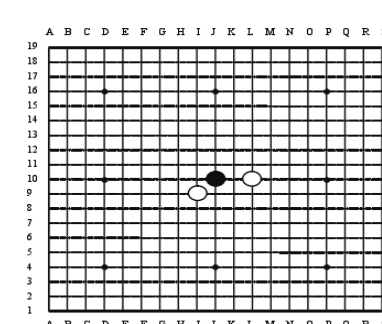


图 2.8 X-L10 定石

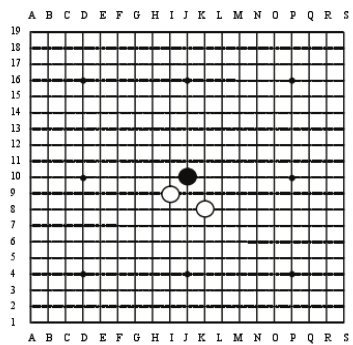


图 2.9 X-K8 定石

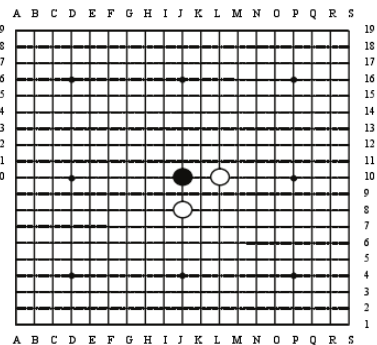


图 2.10 TT-L10 定石

三、“傲天良辰” 博弈程序的主要思路及特点

（一）主要思路

1. 首先脱离对战平台建立自己的六子棋对战程序，方便实践及调试。
2. 建立六子棋模型，将棋子作为一个类，此类中包含棋子坐标以及棋子颜色两个属性，0 表示黑，1 表示白。棋盘用 `Vector<Point>` 表示。
3. 利用评估函数计算得分，以得分决定下一步走法，相当于极大极小值算法。具体操作方法为：设置六子滑动窗口，计算每个可以形成六子的窗口，记录窗口中所包含的一方棋子数，设置各种棋子数应得的得分，从而计算出当前棋局某一方的得分。经过借鉴与实践之后确定棋子数得分分别为对手 {0, 0, 2, 20, 65535, 65535}，己方 {0, 0, 1, 10, 35, 35, 999999}。对弈时，传入算法的是当前棋局及行棋颜色，传出的是将要下的旗 point。当传入当前棋局后，辨别敌我执棋颜色，首先判断对手是否形成六子，若未形成六子则进行下一步，否则对局结束。搜索棋盘现在的空位置，搜索下两步可以着棋位置（搜索时保证 (x1, y1) 在 (x2, y2) 之前，以防出现两个点排列组合中重复计算的情况），为减少无用搜索，检测是否为空位置时同时判断此位置附近附近没有棋子，若附近没有棋子接下来则不进行考虑，搜索出合适空位置之后，尝试性下两个棋子，将棋局改变，计算改变后的得分，重复此过程，将所有合适情况遍历，找出得分最高的情况（如果得分相同则随机选择一个），将此作为输出点。
4. 用 java 建立简单界面，进行调试，加以改进。
5. 将引擎编写规范中的事例读懂，分析输入输出，用 java 语言解决 `fflush(stdin);` 等在 c++ 中独有的语句，生成最简单的可以随机下棋的引擎。
6. 将写好的引擎与六子棋算法结合，用 Move 类将算法运用到引擎中，判断当前是否为先手第一步行棋，区分第一手跟之后的行棋，形成最终 java 版本。
7. 将 java 文件打包成可执行 jar 包，然后用 exe4j 生成可执行 exe 文件，在对战平台上应用。



（二）特点

- 优点：
1. 相对于其他小组的程序耗时短，效率极高
 2. 进攻性强，在先手时，大多数情况下可以胜利
- 缺点：
1. 没有进行开局库搜索，前期布局略有劣势
 2. 没有人工智能学习策略，在跟其他组对弈几局之后，几率失败
 3. 比赛时间提前，导致一些功能未能完成完整优化

四、版本 1 的主要数据结构及关键算法

4.1 主要数据结构

1. Point 类——棋子类。



 Point
connect6_1_2
<ul style="list-style-type: none">• x: int• y: int• color: int
<ul style="list-style-type: none">•  Point(x: int, y: int, color: int)• getX(): int• getY(): int• getColor(): int

Point 类有三个成员变量：
① 棋子点的横坐标 (x)
② 棋子点的纵坐标 (y)
③ 该棋子的颜色 (color)

Point 类与 4 个成员函数：
① 构造函数 (Point)
② 获取横坐标 (getX)
③ 获取纵坐标 (getY)
④ 获取棋子颜色 (getColor)

2. Step 类——着法类。

Step 类有 5 个成员变量，用来表示每一步所落的两颗棋子的属性。以及 6 个成员函数来获取 Step 的属性。

 Step
connect6_1_2
<ul style="list-style-type: none">▪ x1: int▪ y1: int▪ x2: int▪ y2: int▪ score: int
<ul style="list-style-type: none">•  Step(x1: int, y1: int, x2: int, y2: int, score: int)• getX1(): int• getY1(): int• getX2(): int• getY2(): int• getScore(): int

3. SixPointsWindow 类——“路”类。





SixPointsWindow 类共有 3 个成员变量：

- ① 窗口 (window)：含有 6 个元素的数组，用来存储窗口中的 6 个棋子。
- ② 各类棋子的个数 (numOfPoint)：含有 3 个元素的数组，下标 0, 1, 2 分别表示黑棋、白棋、空位置。
- ③ 当前偏移量 (offset)：当前偏移量，在向窗口中添加新棋子时候会用到

SixPointsWindow 类共有 3 个成员函数：

- ① 构造方法 (SixPointsWindow)：初始化成员变量。
- ② 将新的棋子点添加到窗口中 (addPointToWindow)：用新元素替换最早加入的元素
- ③ 获得当前窗口中某颜色得分 (getWindowScore)：若在此窗口中没有另一方棋

子，则返回本方棋子个数；否则返回 0。

 SixPointsWindow
connect6_1_2
<ul style="list-style-type: none">▪ window: int[]▪ numOfPoint: int[]▪ offset: int
<ul style="list-style-type: none">•  SixPointsWindow()•  addPointToWindow(color: int): void•  getWindowScore(color: int): int







4..Priority_Queue 类——优先队列类。

Priority_Queue 类共有 2 个成员变量：

- ① 着法比较器 (stepComparator)：根据每一个 step 的 score 来进行比较。
- ② 着法优先队列 (stepPriorityQueue)：存储 Step 对象的优先队列。

Priority_Queue 类共有 5 个成员函数：

- ① 构造函数 (Priority_Queue)
- ② 获取优先队列的首元素 (getHeadStep)
- ③ 增加着法 (addStep)：向优先队列里添加新元素
- ④ 获取长度 (getSize)
- ⑤ 是否为空 (isEmpty)：判断优先队列是否为空

 Priority_Queue
connect6_1_2
<ul style="list-style-type: none">▪ stepComparator: Comparator<Step>▪ stepPriorityQueue: Queue<Step>
<ul style="list-style-type: none">•  Priority_Queue()•  getHeadStep(): Step•  addStep(stp: Step): void•  getSize(): int•  isEmpty(): boolean


5. Move 类——落子类。

Move 类共有 6 个成员变量：

- ① x1_AI, y1_AI, x2_AI, y2_AI 表示电脑的落子。
- ② 棋子容器 (points)：用以存储棋盘上的棋子。
- ③ 玩家棋子的颜色 (playerColor)

Move 类共有 8 个成员函数：

- ① 两个有参和无参构造函数 (Move)：
- ② 对手落子 (playerMoves)：获取对手的落子，并将其记录到 points 中。
- ③ 着法生成 (generateComputerMoves)：生成电脑的落子。
- ④ 四个获取电脑落子的属性函数。

 Move connect6_1_2
<ul style="list-style-type: none"> ▪ x1_AI: int ▪ y1_AI: int ▪ x2_AI: int ▪ y2_AI: int ▪ points: Vector<Point> ▪ playerColor: int
<ul style="list-style-type: none"> • Move() • Move(playerColor: int) • playerMoves(x1: int, y1: int, x2: int, y2: int): void • generateComputerMoves(): void • getX1_AI(): int • getY1_AI(): int • getX2_AI(): int • getY2_AI(): int

6. AI 类——智能类，该类通过将评估函数与着法生成模块结合，实现博弈树搜索并确定最终的落子。

AI 类共有 11 个成员变量：

- ① 落子点 (points)：存储落子点的容器。
- ② 棋盘 (chessBoard)：19×19 的棋盘。
- ③ 电脑落子 (computerMoves)：记录电脑的落子。

其中还包括另外 8 个静态成员变量：

落子点 (points)：存储落子点的容器。

- ① 对方权重 (rivalWight)：用以计算每一路的得分。
- ② 我方权重 (selfWight)：用以计算每一路的得分。
- ③ int 类型所能表示的最大值 (MAX)，int 类型所能表示的最小值 (MIN)、 $\alpha - \beta$ 剪枝的次数 (ABcut)、游戏胜负的标记 (win)、博弈树节点总数 (total_node)、博弈树叶子节点数 (leaf)

AI	
connect6 1 2	
<ul style="list-style-type: none"> points: Vector<Point> chessBoard: int[][] computerMoves: int[] rivalWight: int[] selfWight: int[] MAX: int MIN: int ABcut: int win: int total_node: int leaf: int 	
<ul style="list-style-type: none"> max(color: int, deep: int, alpha: int, beta: int): Step min(color: int, deep: int, alpha: int, beta: int): Step maxmin(color: int, deep: int): Step getComputerMoves(computerColor: int): int[] AI(points: Vector<Point>) setData(points: Vector<Point>): void getPoints(): Vector<Point> resetChessBoard(): void placeTwoStones(color: int): void hasStonesAround(x: int, y: int): boolean getTotalScore(color: int): int hasSix(color: int): boolean hasSix(): boolean canSix(): boolean 	

4.2 关键算法及其实现

4.2.1 盘面估值

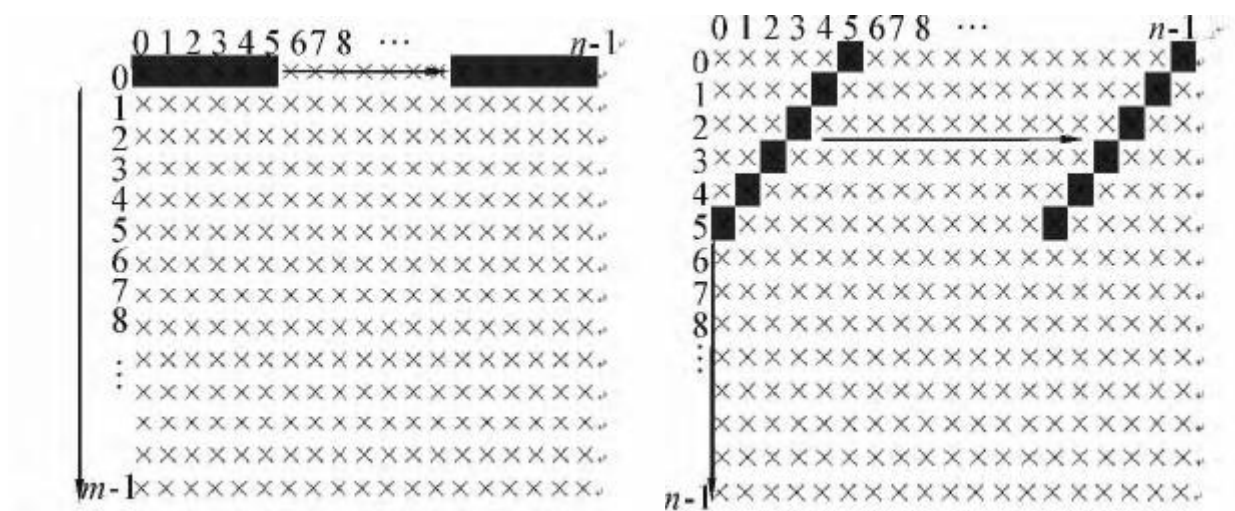
盘面估值函数是对棋局的综合评估，该函数的好坏直接决定解题能力的强弱。同时，合理的盘面估值也是博弈树搜索的前提。

评估函数综合了大量跟具体棋型有关的知识，我们从以下两个基本假设开始：

(1) 我们能把局面的性质量化成一个数字。例如，这个数字可以是对取胜概率做出的估计。

(2) 我们衡量的这个性质应该跟对手衡量的性质是一样的，如果我们认为我们处于优势，那么反过来对手认为他处于劣势。真实情况并非如此，但是这个假设可以让我们的搜索算法正常工作，而且在实战中它跟真实情况非常接近。

我们的盘面估值基于“路”的扫描方式，六子棋 19×19 方格全局扫描的总路数为 924。



水平方向扫描

左斜方向扫描

在扫描过程中，每移动一步，求出当前“路”的得分 $Score_i$ ，最后棋局的得分：

$$Score = \sum_{i=0}^{923} Score_i,$$

其中：

当前“路”中既有黑子又有白子时， $Score_i=0$ ；

当前“路”中只有己方棋子时， $Score_i \in \{0, 0, 1, 10, 35, 35, 99999\}$ ，该数组下标为“路”中己方棋子数；

当前“路”中只有敌方棋子时， $Score_i \in \{0, 0, -2, -20, -65535, -65535\}$ ，该数组下标为“路”中己方棋子数。

评估函数伪代码为：

```

Procedure Evaluation(color)
  for(四个扫描方向):
    for(从左到右扫描):
      score = score + 当前“路”得分”
    end
  end for
end Procedure

```

4.2.2 着法生成

计算机博弈中，着法生成方法一般有棋盘扫描法、模板匹配法和预置表法。

围棋、五子棋、六子棋等属于添子类走法，棋盘任意空白处皆可行棋，对走法没有复杂的限制。这类棋多采用棋盘扫描法。

着法生成函数伪代码如下：

```

Procedure Generator()
  for(搜索第一个 Point1)
    for(搜索第二个 Point2)
      if(Point2 在 Point1 之前或两者重合)
        continue;
      end if
      if(Point1 或 Point2 位置已有棋子)
        continue;
      end if
      if(Point1 附近没有棋子或 Point2 附近没有棋子)
        continue;
      end if
      生成着法;
    end for
  end for
end Procedure

```

4.2.3 盘面更新

盘面更新的过程如下：

- (1) 轮到己方下棋时，Move 调用其成员方法 generateComputerMoves ()。
- (2) generateComputerMoves () 启动主引擎 AI 通过博弈树生成最佳着法。
- (3) 最佳着法—>AI 的成员变量 computerMoves。
- (4) 通过 AI 的成员函数 getComputerMoves () 返回最佳着法给 Move 中的成员变量——x1_AI, y1_AI, x2_AI, y2_AI。
- (5) 将最佳着法输出用以平台交互。

4.2.4 博弈树搜索

极大极小搜索结合 $\alpha - \beta$ 剪枝算法伪代码如下：

极大搜索：

```

Procedure Max-Search(color, depth, alpha, beta)
  best = MIN;
  if(depth≤0 或连成六子)
    调用盘面估值以获取当前局面的评分 score;
    return score;
  end if
  调用着法生成函数，生成着法并将其存到优先队列 priority_queue 中;
  for(step in priority_queue 的前 50%)
    尝试下一子 Move ( ) ;
    一步棋 step1 ← Min-Search(color, depth-1, alpha, best > beta ?
best : beta);
    撤销落子 unMove();
    if(step1.score > best)
      更新最佳结点 step 的评分与坐标;
    end if
  end for
end Procedure

```



```

        if(step1.score > alpha)
            break; // 进行 alpha 剪枝操作
        end if
    end for
end Procedure

```

极小搜索：

```

Procedure Min-Search(color, depth, alpha, beta)
    best = MAX;
    if(depth ≤ 0 或连成六子)
        调用盘面估值以获取当前局面的评分 score;
        return score;
    end if
    调用着法生成函数，生成着法并将其存到优先队列 priority_queue 中;
    for(step in priority_queue 的前 50%)
        尝试下一子 Move ();
        一步棋 step1 ← Max-Search(color, depth-1, alpha, best > beta ?
best : beta);
        撤销落子 unMove();
        if(step1.score < best)
            更新最佳结点 step 的评分与坐标;
        end if
        if(step1.score < beta)
            break; // 进行 beta 剪枝操作
        end if
    end for
end Procedure

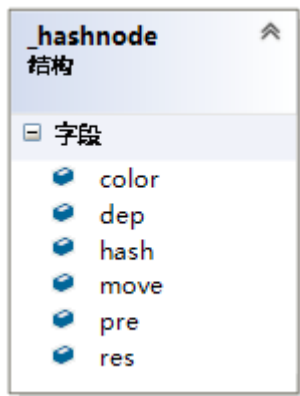
```

五、对版本 1 的改进

经过前期的优化，版本 1 可以进行 3 层的 alpha-beta 搜索，时间消耗也在可控范围内。然而版本 1 的棋力还是无法达到预期水平，本小组通过阅读国内外优秀文献以及与老师、同学间的交流，对六子棋的程序设计也有了更深层次的理解。我们对版本 1 进行全方面的改进，引入新的搜索算法 TSS (Threat Space Search)，进一步提升我们程序的博弈水平。

5.1 主要数据结构的改进

(1) 在威胁空间中进行搜索时，为了节省时间，我们引用 Hash 机制，新增 _hashnode 结构体，具体内容如下：

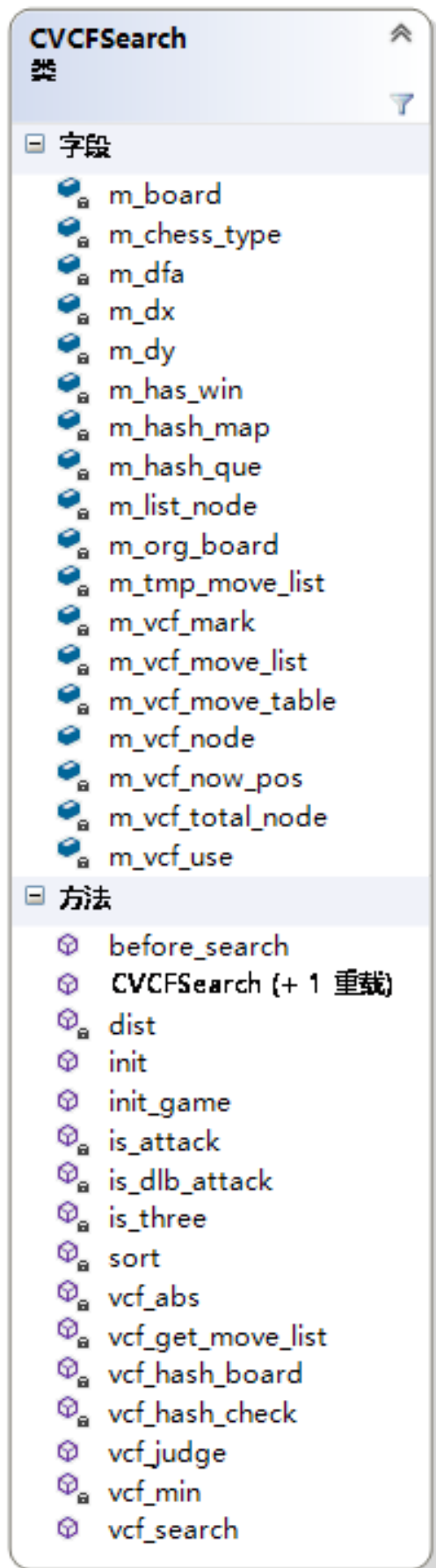


```
typedef struct _hashnode
{
    int dep; // 深度
    unsigned long hash; // Hash 键值
    int pre; // 前面的结点
    move_t move; // 着法
    bool res; // 结果
    char color; // 棋子的颜色
} HashNode; // Hash 节点
```

(2) 引入 CVCFSearh 类，实现 TSS 算法：

五子棋术语 VCF: (Victory of Continuous Four 的缩写) 五子棋中引入的英文名称，即利用连续不断冲四这种绝对先手，直至最终成五而取得胜利的一种技巧。简称“连续冲四胜”或“连冲胜”。常为五子棋残局排局解题的一种取胜技巧。

这里我们利用 VCF 的思想设计 CVCFSearh 类来实现 TSS 搜索，以提高棋力。



其中,CVCFSearh类共有16个成员变量:

- ① 存储棋盘 (m_board): 19×19 的二维数组
- ② 棋子类型(m_chess_type): 黑、白、空
- ③ 有穷自动机(m_dfa) : 存储状态间的转化
- ④ 横坐标(m_dx) : 存储棋子的位置
- ⑤ 纵坐标(m_dy) : 存储棋子的位置
- ⑥ 是否取胜(m_has_win): bool 类型, 判断胜与否
- ⑦ 哈希映射 (m_hash_map) : 存储 Hash 映射的数组
- ⑧ 结点表 (m_list_node) : 存储威胁空间中的节点
- ⑨ 临时着法表(m_temp_move_list) : 存储临时的着法
- ⑩ VCF 标记 (m_vcf_mark) : TSS 搜索时对棋盘的标记
- ⑪ VCF 着法表(m_vcf_move_list): 存储 TSS 搜索产生的着法
- ⑫ VCF 结点表 (m_vcf_move_table) : 存储搜索到的 TSS 节点
- ⑬ VCF 结点 (m_vcf_node) : 威胁空间中的节点
- ⑭ 当前位置 (m_vcf_now_pos) : 记录当前在 TSS 空间中的位置
- ⑮ 结点总数 (m_vcf_total_node) : TSS 空间中的节点总数
- ⑯ 使用 VCF (m_vcf_use) : 是否使用 TSS 算法

CVCFSearh 类共有 15 个成员函数:

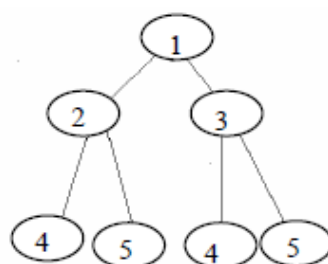
- ① 搜索前(before_search) : 进行内存拷贝, 属性赋值
- ② 构造函数 (CVCFSearh) : 对象初始化
- ③ 计算距离 (dist) : 计算威胁空间中节点距离
- ④ 初始化 (init) : 有穷自动机初始化
- ⑤ 初始化游戏 (init_game) :

m_has_win = 0, 未获胜

- ⑥ 是否进攻 (is_attack) : 判断是否形成致胜威胁序列
- ⑦ 是否成三 (is_thress) : 判断是否形成 3 个威胁
- ⑧ 排序 (sort) : 对节点表进行排序

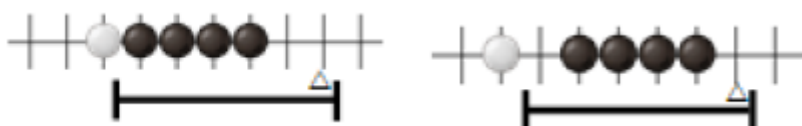
- ⑨ 取绝对值 (vcf_abs) :获取绝对值
- ⑩ 获取着法列表 (vcf_get_move_list)
- ⑪ Hash 棋盘 (vcf_hash_board) :将棋局用哈希数表示
- ⑫ Hash 判重 (vcf_hash_check)
- ⑬ 判断进攻 (vcf_judge) :判断能否进攻
- ⑭ 判断大小 (vcf_min)
- ⑮ 进行 TSS 搜索 (vcf_search)

(3) alpha-beta 搜索过程中, 当搜索深度 $\text{depth} \leq 2$ 时, 不会存在节点间的重复。但是, 当 $\text{depth} > 2$, 出现重复评估节点的情况。将棋盘的信息保存起来以备后续使用, 其中, 棋盘的每一个状态实际就对应着一个随机数。

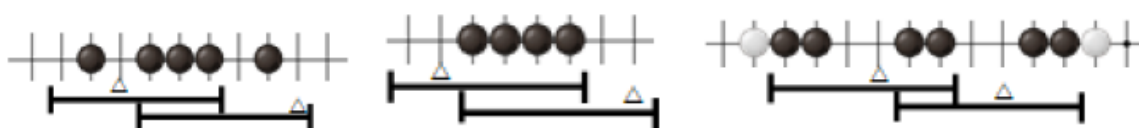


5.2 关键算法的改进

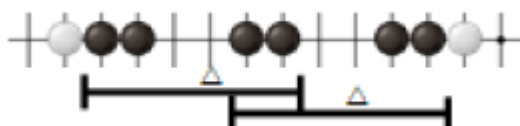
(1) “威胁”的定义: 若黑方有 t 个威胁, 当且仅当白方需要放置 t 个白棋才能阻止黑方成功连成六子。



威胁数: 1



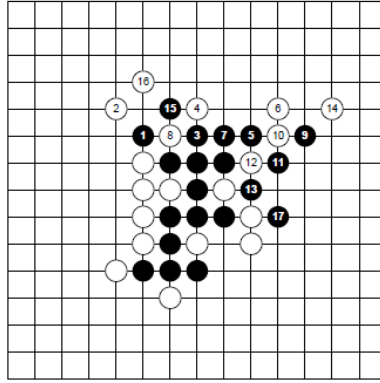
威胁数: 2



威胁数: 3

(2) TSS (Threat Space Search), 通过在威胁空间中进行搜索, 找到可以达到威胁数 ≥ 3 的路径。

TSS 搜索目的: 找到致胜威胁序列, 如下图所示:



(3) TSS 伪代码:

```

Procedure Threat-Space-Search(进攻方)
  if (防守方连成六子 or 防守方搜索到威胁序列) then
    return FALL;
  end if
  if (进攻方连成六子 or 进攻方形成 3 个威胁) then
    return SUCCESS;
  end if
  着法生成函数生成落子空间表 list;
  if (list is empty) then
    return FALL;
  end if
  result = FALL;
  for move in list :
    Move(); // 落子
    result = Threat-Space-Search(进攻方)
    Undo(); // 撤销落子
    if ( result == SUCCESS ) then
      return SUCCESS
    end if
  end for
  return FALL
End Procedure

```

(4) 滑动窗口算法用以在棋盘上扫描时计算威胁数目
滑动窗口算法伪代码:

```

Procedure Sliding-Window-Algorithm(lpattern)
  for (从左向右滑动窗口)
    if (窗口中有 6 个进攻方棋子) then
      threat = 正无穷;
      break;
    else if (无 marked points or 防守方棋子) and
      (进攻方棋子 ≥ 4) then

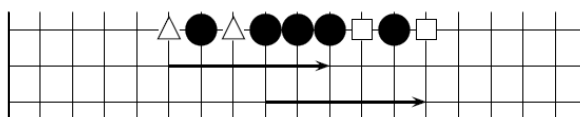
```



```

        标记所有空位置;
        threat      threat + 1;
    end if
end for
return threat;
End Procedure

```



5.3 博弈搜索效率及胜率的对比

博弈搜索效率:

测试 alpha-beta 剪枝, 和基于 alpha-beta 剪枝的优化 (预估排序, 进一步限定落子空间)

搜索树深度为 2, 开局前 6 步的各项指标的测试结果如下:

step	first_layer_node	leaf	$\alpha - \beta$ cut	time/ms
1	378	279128	0	9720
2	1035	1618407	0	62939
3	1378	2731272	0	95849
4	1653	3870163	0	173719
5	1953	5343598	0	268691
6	2926	10938585	0	554252

未剪枝

step	first_layer_node	leaf	$\alpha - \beta$ cut	time/ms
1	378	273172	2081	9638
2	1035	1385783	84274	55298
3	1378	683690	472053	28000
4	1653	3619587	143224	165151
5	1953	5238219	70139	264869
6	2926	10251616	193131	527572

alpha-beta 剪枝

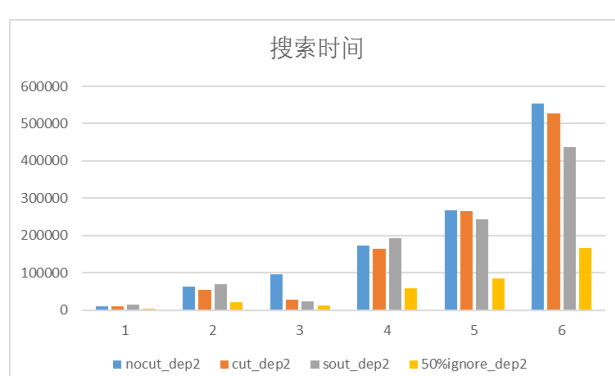
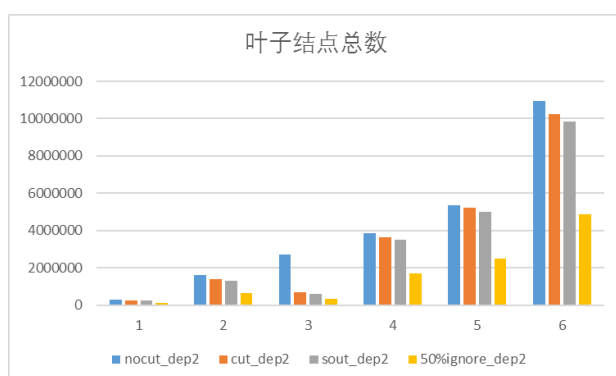
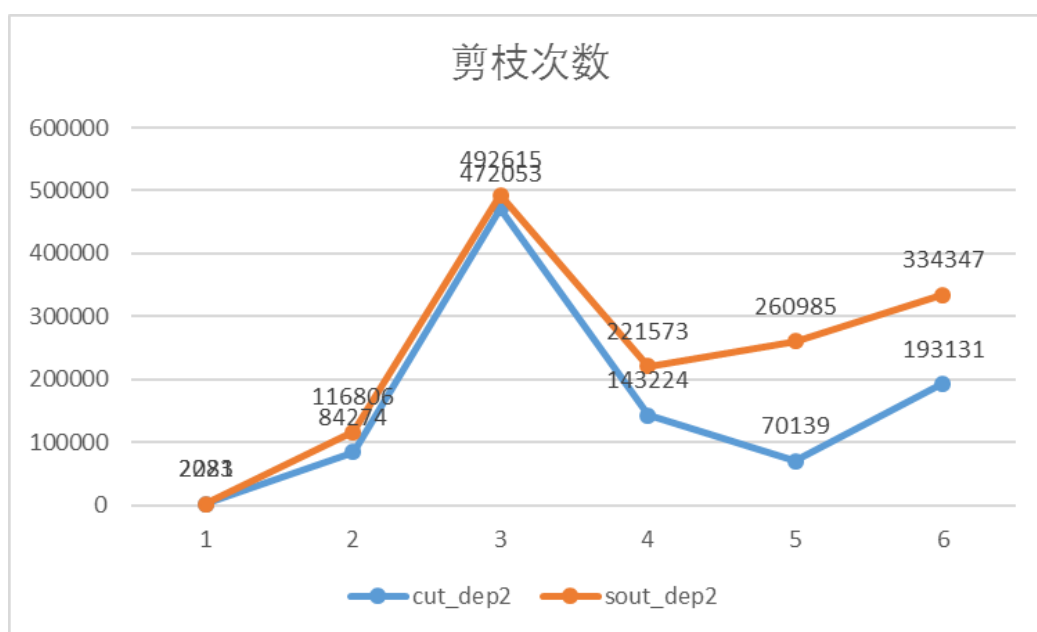
step	first_layer_node	leaf	$\alpha - \beta$ cut	time/ms
1	378	273046	2223	14091
2	1035	1298422	116806	70100
3	1378	590224	492615	22881
4	1653	3494262	221573	193935
5	1953	5009569	260985	244128

6	2926	9822075	334347	437729
---	------	---------	--------	--------

alpha-beta 剪枝 + 预估排序

step	first_layer_node	leaf	$\alpha - \beta$ cut	time/ms
1	189	138643	761	4826
2	517	646120	47455	21926
3	689	327206	229597	12882
4	826	1726399	93163	58308
5	976	2503398	86090	84480
6	1463	4886026	128657	165811

alpha-beta 剪枝 + 预估排序 + 剔除后 50%



实验结果显示：

- ② 预排序可以增加剪枝次数，提高效率。
- ③ 通过优化，在保证最终落子结果不变的前提下，缩短了搜索时间，为搜索深度的加深提供了可能。

各版本所用到的算法与提升：

版本号	用到的算法	提升与不足
Version 0.0	一层搜索、静态评估	效率低、棋力差
Version 0.1	一层搜索、限定落子空间、基于“路”扫描	速度提升较大
Version 1.0	极大极小搜索、 $\alpha - \beta$ 剪枝	只能搜索到 2 层、棋力提升不大
Version 1.1	预估排序优化 $\alpha - \beta$ 、限定搜索空间	速度提升较大, 可以搜索到 3 层
Version 1.2	置换表	速度提升较大, 可以搜索到 3 层
Version 2.0	TSS 算法	棋力获得一定提升, 搜索层数较少
Version 2.1	优化 TSS 算法, 减小搜索空间	效率提高
Version 2.2	Hash 线性模式表	搜索层数进一步加深

Version 0.1

对手	先手	后手
HelloW	1	1
开拓者	1	1
葫芦娃	1	1
红鲤鱼	1	1
人工智障	1	1
梦之云	1	1
$\alpha \beta \gamma$	0	0
AI++	0.5	1
轻症患者	1	1
Seals	1	1

先手胜率：85%

后手胜率：90%

总胜率：87.5%

第二次对战：version 1.2

对手	先手	后手
HelloW	1	0
开拓者	1	1
葫芦娃	1	0
红鲤鱼	1	0
人工智障	1	1
梦之云	1	0
α β γ	0	1
AI++	1	0
轻症患者	1	1
Seals	1	0

先手胜率：90%

后手胜率：40%

总胜率：65%

第三次对战（对手为上一次对战时的版本）：Version 2.2

	version 2.2	
对手	先手	后手
HelloW	1	1
开拓者	1	1
葫芦娃	1	1
红鲤鱼	1	1
人工智障	1	1
梦之云	1	1
α β γ	1	0
AI++	1	1
轻症患者	1	1
Seals	1	1

先手胜率：100%

后手胜率：90%

总胜率：95%

六、总结与展望

在本次实验中，我们采取的创新之处是模拟人类思维模式提出空位估值方法。人类棋手对弈过程中，会思考空位置对当前局势可能产生的影响。在抑制对方局势的前提下，如果某空位置对我方有利，棋手会选择最为有利的位置落子。本文提出的空位估值方法就是实现了这一思想，在后期加入了 TSS 迫着搜索，提升了棋力。该方法实验过程中得到了比较理想的效果。

1. 本系统与其他六子棋程序进行多次对弈，实验结果较理想。但目前仍存在以下不足：

- ① 利用棋形估值方法时，选用棋型个数有限，应当适当增加棋型。
- ② 评估函数由空位置的估值，棋形估值两方面组成，可以适当增加棋型距离之间的估值，使评估函数性能更好。
- ③ 采用了剪枝和极大极小搜索，在加深搜索深度的同时极大的缩短了计算的时间，但是深度超过 4 会使计算超过时间限制

④

2. 后续工作：

在已有的工作基础上，我们小组将继续六子棋计算机博弈系统的研究工作。

① 将棋形估值方法做进一步的改进，添加棋型库，使系统能自动更新棋型。

⑤ 增加定石库，初始状态根据定石库进行落子，使局势更有利于我方。

六子棋规则是除了第一次落一子外，黑白双方轮流下两子。改进系统中的博弈树，使“两步”合并为“一步”。

3. 展望

机器博弈被称为是人工智能的“果蝇”，其对人工智能的发展有很大的推动作用。机器博弈各项技术的研究成果有些也适用于其他方面。随着我国各高校、研究所以及业余棋类爱好者对机器博弈的重视，国内出现了机器博弈研究的热潮。六子棋由于其简单的规则，更高的公平性，在发明之初就受到大家的喜爱，相关领域的研究人员也在逐渐增加。借助对机器博弈的大力研究，人工智能也将得到前所未有的发展。

七、参考文献

- [1] 刘雅靖,《基于 Alpha_Beta 搜索算法的计算机博弈的研究与实现》, 硕士学位论文, 大连交通大学, 2012 年 6 月。
- [2] 王志水,《基于搜索算法的人工智能在五子棋博弈中的应用研究》, 工程硕士学位论文, 中国石油大学, 2006 年 10 月
- [3] 李果,《六子棋计算机博弈及其系统的研究与实现》, 硕士学位论文, 重庆大学, 2007 年 4 月
- [4] 李学俊, 王小龙, 吴蕾, 刘慧婷,《六子棋中基于局部_路_扫描方式的博弈树生成算法》, 智能系统学报: 第 10 卷第 2 期, 2015 年 4 月
- [5] 杜思翰, 李 铭,《一种改进的威胁空间搜索算法》, 湖南文理学院学报: 自然科学版, 第 22 卷第 3 期, 2010 年 9 月
- [6] 刘云青,《 A Defensive Strategy Combined with Threat-Space Search for Connect6 》, 硕士论文, 国立台湾师范大学资讯工程研究所, 2007 年 6 月
- [7] T Czenave,《A Generalized Threats Search Algorithm》, In《Lecture Notes in Computer Science》, 2002, 2883:75-87
- [8] LV Allis, HJ Herik, MPH Huntjens,《Go-Moku and Threat-Space Search》, In《Interview Questions》, 1993
- [9] Wu-Huang, Dei-Yen Huang,《A New Family Of K-in-a-Row Games》, 2006 年

附录：

1. 小组各成员的分工及贡献度

组员	分工	贡献度
曲磊钢	进展规划与任务分工、TSS 算法编程	25%
潘迎港	着法生成算法实现、程序优化、算法改进	25%
苏念亿	程序与博弈平台间的交互、程序测试	16.7%
蒙腾斌	极大极小搜索、 $\alpha - \beta$ 剪枝算法实现	16.7%
茆加余	数据结构设计与改进、盘面估值算法的设计与实现	16.7%

2 小组成员感言

曲磊钢：本次六子棋博弈程序的开发历时两个月，从开始接触时阅读文献寻找思路，到之后编程实现第一个版本，再到最后通过我们的努力使胜率和博弈效率的整体提升，收获颇丰。这次开发经历，一方面，通过应用以往课程中的知识加深了自己对于计算机学科基础知识、编程思想等的理解与掌握；另一方面，拓宽我的知识面的同时，也培养了我自主学习、团队组织与合作、计划的制定与实施、文献的阅读等各个方面的能力，使我受益匪浅。当今世界，计算机行业的知识更新迭代迅速，只有不断学习，才能不被时代所抛弃，才有希望在这一领域留下自己的足迹。

潘迎港：当初老师刚布置小组课题时，我们对六子棋、极大极小搜索、威胁空间搜索、阿尔法-贝塔剪枝一无所知，于是我们进入了漫长的看论文的历程，从无到有，对算法从陌生到熟悉，从伪代码到 Java 实现，一步步的大家一起协同作战，共同面对这一“困难”，齐心协力，互相沟通，将大困难分解成一个个小问题，再逐一解决。面对强敌 $\alpha \beta \gamma$ 队和人工智障队，我们一步步改进算法，从原来的单层局部最优决策搜索到三层搜索，再借助置换表用空间换时间大大缩短了搜索时间，最后打败人工智障队，先手打败 $\alpha \beta \gamma$ 队，最后取得了优异的成绩，这一切的一切都是大家一起努力的结果。

在以后的工作学习中，我也会发扬这种不畏困难、迎难而上的精神，提升自我，完善自我。

苏念亿：这次人工智能课程，让我更加注重思考，真正去了解算法，研究算法。一开始很期待写出的程序，但随着一步步的探索，发现离着真正的智能还是有很大差距的，尤其是知道程序所使用的算法之后，先手就能轻而易举的打败机器，还不能真正的用电脑打败人脑，也可能是自己在算法上太菜了都原因，所以更加钦佩研究出 α 狗的大神。今后一定要更加注重算法，注重程序真正的核心。

蒙腾斌：第一次接触人工智能，是 2016 年听说阿尔法狗打败李胜石的时候。作为一个下棋爱好者，我接触过很多不同棋类的 AI，认为大部分 AI 棋力很弱，

于是便抱着兴趣选了这门课程。这门课让我体会到了人工智能能够优于人脑的原因之一，就是“计算”。这是我通过这门课程中的两次实验感受到的，不论是滑动积木块或是六子棋，都是通过我们在程序中设计算法和提供样本，然后计算机进行有条不紊的重复计算和人脑所达不到的大量计算，最后达到与人脑等同，甚至超越人脑的水平，这是我第一次在比较专业的层面上了解人工智能，也学到了很多关于机器博弈的算法如 $\alpha - \beta$ 剪枝，TSS 迫着空间搜索等算法，受益匪浅。

茆加余：通过一学期的学习，逐渐也从对人工智能毫无认知的状态到了小有见解。当今时代的人工智能，如果要以科幻电影中的标准来衡量，那么必会大失所望，但虽说并不强大，却在飞速发展中，就以棋类游戏为例，从一开始的无法入职业选手法眼，到现在可以说已经到了超越人类的地步，其中的曲折历程自然不必多说，但我看到的是人工智能的飞速发展。机器学习的诞生为人工智能又添了一把火，人脸识别系统的投入使用就可见这把火的旺盛程度。纵观历史再展望未来，如今虽说人工智能已经达到前所未有的高度，但不难发现，仍然只是其发展前期，还是一个未长成人的少年或是青少年，未来几十年中，人工智能必定会不断有所创新，有所提高，发展到一个人类无法想象的高度，科幻电影中的炫目技术也将化为现实。