摘 要

计算机博弈是博弈领域长期的奋斗目标，也是人工智能学科极具挑战性的研究课题之一。凭借计算机的高速运算能力和设计严谨的算法，计算机可以在人机对弈中表现出相当高的“智能”。

国际象棋已经有了“深蓝”这样震惊世界的成果，更加复杂的中国象棋计算机博弈水平却远远落后。

一款象棋博弈程序的实现主要被分为两大部分：界面（程序辅助）+引擎（人工智能）。本文将介绍如何实现一款中国象棋博弈程序。其中包括：搜索算法、局面评估、哈希表、历史启发、局面库以及中盘研究；打谱；复盘等功能的实现。

关键词：中国象棋；人工智能；博弈树；PVS 搜索算法；局面评估；

ABSTRACT

Computer Game is the aim of game fields for long, and is the one of the challenging topics in artificial intelligence research. With the computer's high-speed computing power and Well-structured algorithms, the performance of computer can be very "smart" in Man-machine chess.

Though we have chess like "Deep Blue" which shocked the world, the more complicated Chinese Chess levels are far behind.

The implementation of Chess game program shall be divided into two parts: interface (program assist) + engine (artificial intelligence). This article will introduce how to implement a Chinese Chess Game program. About: search Algorithm、Research of Situation Assessment、HashTable、Historical inspiration、Situation Library and Mid-Cap Research、Make Situation、Recovery disk as the realization of functions such .

Key Words: Chinese Chess; artificial intelligence; Game Tree; PVS search Algorithm; Research of Situation Assessment;

学位论文独创性声明

本人声明所呈交的学位论文是本人在导师指导下进行的研究工作及取得的研究成果。据我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得 南昌大学 或其他教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。

学位论文作者签名（手写）：罗潭　　签字日期：2009 年 12 月 31 日

学位论文版权使用授权书

本学位论文作者完全了解南昌大学有关保留、使用学位论文的规定，有权保留并向国家有关部门或机构送交论文的复印件和磁盘，允许论文被查阅和借阅。本人授权南昌大学可以将学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编本学位论文。同时授权中国科学技术信息研究所将本学位论文收录到《中国学位论文全文数据库》，并通过网络向社会公众提供信息服务。

（保密的学位论文在解密后适用本授权书）

学位论文作者签名（手写）：罗潭　　导师签名（手写）：

签字日期：2009 年 12 月 31 日　　签字日期：2009 年 12 月 31 日

第 1 章 引言

1.1 计算机博弈概述

计算机博弈是人工智能学科和计算机学科的交叉研究方向之一。自从计算机诞生以来，许多著名学者都曾经涉足于这一领域的研究工作。计算机之父冯·诺依曼（Won Neunann）就提出了用于博弈的极大极小定理[1]，信息论的创始人香浓（Claude E. Shannon）教授，又给出了极大极小算法[2]，著名的计算机学家阿兰·图灵（A.Turing）也曾做过机器博弈的研究[3]。伴随着计算机硬件的飞速发展，Alpha-Beta 剪枝算法的提出[4]、极小树的证明[5]、负极大值算法的给出[5]、迭代深化思想的实现[6]，都使得机器博弈有了长足进展。以 IBM“深蓝”为代表的划时代成果，表明计算机的弈棋能力已经可以和人类精英较量。如今，机器博弈已经成为一个独立而重要、颇有发展前途的学术研究领域，但是它在中国的发展还很落后。计算机博弈通常只被看作是一种科技活动，而常常被学术界所忽视，也正是由于缺少学术研究的有力支持，才使得这一方向的发展比较缓慢，尤其是在中国内陆。机器博弈作为计算机学科的一个分支，在吸收其他学科的成果方面明显不够，如对策论、系统论、控制论等。作为人工智能学科的“果蝇”①，机器博弈由于缺少计算机学科的有力支持，其研究成果也只能作为初级的游戏产品。

1.2 计算机棋类博弈分析

尽管目前有的棋类的计算机水平已经高出了人类大师，如国际象棋，但是计算机间的较量仍然持续不断，各种人机大战也接二连三。因为计算机的智能也需要不断地提高。而那些水平还不高的计算机博弈项目，如中国象棋、日本将棋和围棋，更是倍受人们的青睐。

计算机棋类博弈基本属于完全信息的动态博弈。也就是对弈双方不仅清楚当前的局面，了解对手以往的着数，而且了解对手接下来可能采取的着数。尽管双方可能采取的着法数以十计、百计，但毕竟还是有限的。计算机可以通过展开一颗根在上、叶在下的庞大的博弈树描述这一过程。再利用自身在时间、空间上的强大能力进行巧妙的搜索，从而找到可行解及近优解，即给出当前的着法。

①摩尔根与果蝇［PPT/OL］，Tomas Hunt Morgan and drosophila
http://basic.shsmu.edu.cn/jpkc/Marx_philosophy/yxyzx/256,1,Thomas Hunt Morgan

显然，计算机的搜索能力是计算机智力水平的重要体现。其实，人类在求解各种问题的时候，解析求解还是次要的。人们大量采用的方法，人类智能的体现，主要也是在头脑里搜索。脑袋转的快、主意多，智商就高。

棋类博弈软件的设计，首先遇到的难题便是数据结构的选择。其中包括棋盘、棋子、棋局和着法的编码与存储。良好的数据结构可以节省大量的存储空间，可以提高存取的效率。为了适应博弈树的展开与搜索，常常还要同时给出棋局的多种数据格式。如棋局状态、棋子位置、比特棋局和比特向量，还要用到哈希变换和哈希表等。

棋局的静态评估是机器博弈的另一个难点，它不仅需要棋类对弈的基本知识，而且用到直接量化、模式量化、随机评估、模糊评估等一系列手段。对于象棋可以给每个棋子和棋位打分，而对于围棋则要进行定式的抽取和模式的匹配。棋类博弈的高手主要应在这方面提供更多的专业知识。

搜索算法是机器“思维”的核心。包括着法生成，博弈树展开，各种剪枝搜索和各种启发式搜索。现有的成果非常丰富，也是今后研究工作的重点。

编程语言、程序设计方法、软件工程、并行计算等都是算法实现的重要技术，需要和博弈原理有机结合，才能编写出高水平的博弈软件。

第 2 章 中国象棋在计算机中的表示

2.1 基本功能模块

中国象棋要在计算机上实现，首先要解决的就是界面和操作问题。程序应该提供给用户一个基本的游戏界面并且能够实现下棋，悔棋等相关的操作。

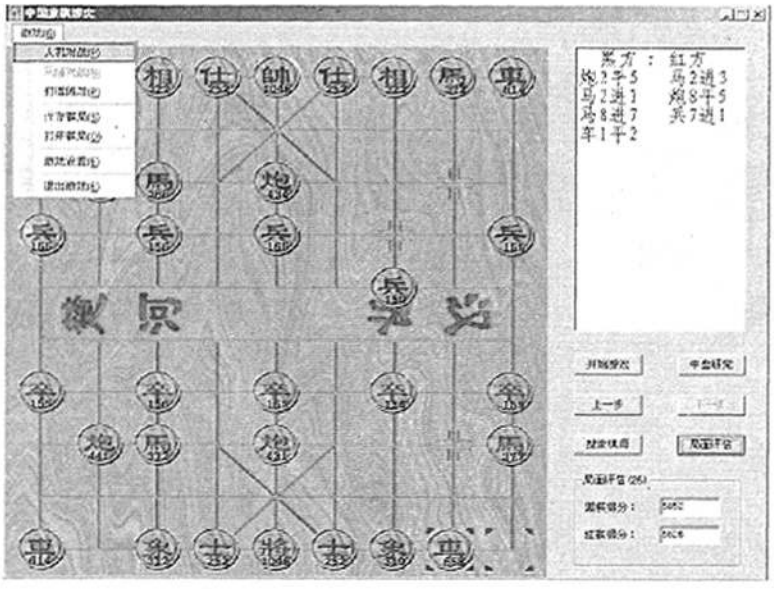


图 2.1 程序界面

由此，我们给程序规划了以下几个基本功能模块：

一、初始化部分

```
private void InitGame()
{
    ……;
}
```

InitGame()负责的是游戏的初始化。我们把有关中国象棋的棋局初始化过程放在了这里面。初始化的内容包括：1、引擎部分用到的相关变量初始化（包括棋子位置（棋盘数组），当前下棋方标志、选中棋子标志、游戏模式、棋局过程数组等）；2、程序辅助部分用到的相关控件的初始化（包括对上一步、下一步按钮的状态显示，棋局过程列表框显示等）。

二、绘图部分

```
void DrawQP()          void DrawKuang()       void DrawValue()
{                      {                      {
    ……;                    ……;                    ……;
}                      }                      }
```

DrawQP()函数和 DrawKuang()函数负责的是程序界面的绘图。在这里要完成棋盘、棋子的显示，下棋起始位置和目标位置的提示框的显示。

由于程序中的棋盘、棋子等都是以位图的形式给出的。所以在 DrawQP()函数里所做的工作主要都是在贴位图。

需要注意的是由于位图文件不能像 GIF 文件那样有透明的背景并且棋子是圆形的而位图文件只能是矩形的，所以如果直接贴图的话会在棋盘上留下一块白色的边框——棋子的背景。因此，要想让棋子文件的背景“隐藏”需要通过一些“与”和“异或”操作来屏蔽掉棋子的背景。在这里使用到了一些 API 函数：如 CreateCompatibleDC（）、TransparentBlt（）、SelectObject（）等。另外为了对颜色更好控制，还需要编写一个颜色转换函数，能够将颜色值（Color 对象）转换成 RGB 值。

DrawValue()函数负责的是游戏过程中局面分数的显示，它能够把当前局面每一个棋子的得分画在棋子的下方并且显示局面分差。这样方便我们观察棋局过程棋子分数的变化，来理解计算机选择下法的依据，从而判断局面评估函数参数是否设置得合理。这也是程序中重要的一个辅助部分。

三、行棋部分（鼠标事件响应）

程序中下棋的操作是通过鼠标在棋盘（panel_qipan 控件）上的点击实现的，我们为该控件添加鼠标事件的消息，可得到如下函数：

```
private void panel_qipan_MouseClick(object sender, MouseEventArgs e)
{
    ……;
}
```

当用户在窗体棋盘面板按下鼠标左键时,程序就会调用该函数来进行处理。其中第二个参数 e 是我们在程序中所要用到的，它给出了当鼠标被按下时鼠标指针相对于棋盘的位置坐标。通过对这一信息进行转换我们可以得知用户的走法，并且调用相关的规则函数判断该走法是否合法。

在 MouseClick 函数里我们处理以下几种游戏模式：

1、人机对战

2、打谱练习

3、复盘模式

4、局面评估

5、中盘研究

6、空模式

在大多数模式中，主要的操作为：

1、如果用户点击鼠标的位置落在己方的棋子上，表示用户选中了该棋子，下一步将移动该子进行走棋（也可能用户下一步将会选择己方另外的棋子，总之这一操作会记录下用户所选的将要走的棋子）。

2、如果之前用户已经选过了棋子，那么这一次的点击（如果不是另选本方的其它棋子的话）表达了用户的一次走棋过程。在收到用户传达的走棋信息后，我们先判断该着法是否合法（是否符合中国象棋的游戏规则），如果合法，则执行该下法。如此，在 MouseClick 函数里，我们实现了游戏的操作部分（当然每走一步棋，还还需要调用绘图函数来显示棋盘局面的更新）。

以上三部分并非程序所实现的全部，而仅仅是与我们的程序密切相关的部分。此外还有其它部分对程序同样必不可少，比如游戏规则函数、棋局过程显示等，但在此不多做介绍。

2.2 棋局表示

计算机下棋的前提是要让计算机读懂象棋。所谓读懂，即计算机应该能够清楚地了解到棋盘上的局面（棋盘上棋子的分布情况）以及下棋方所走的每一种着法。因而首先我们需要有一套数据结构来表示棋盘上的局面以及着法。

当前主要有两种棋盘表示法：一种是棋盘数组，另一种是位棋盘。位棋盘最初是国际象棋(8*8, 刚好 64bits)里引入的，其最大的优点就是超高的运算效率与空间节省。所以大多数博弈软件都是用位棋盘作为基本的数据结构。但中国象棋是 10*9 的棋盘，不易用位棋盘表示。这里采用了最传统的同时也是最为简单的棋盘数组。即用一个 10*9 的数组来存储棋盘上的信息，数组的每个元素存储棋盘上相应位置是何种棋子。这种表示方法简单易行（缺点是效率不是很高）。按此方法棋盘的初始情形如下所示：

```
public byte[,] ChessBoard = new byte[10, 9]
{
    { 11, 10, 12,  13, 14, 13,  12, 10,  11 },
    { 0,  0,  0,  0,  0,  0,  0,  0,  0  },
    { 0,  9,  0,  0,  0,  0,  0,  9,  0  },
    { 8,  0,  8,  0,  8,  0,  8,  0,  8  },
    { 0,  0,  0,  0,  0,  0,  0,  0,  0  },
    { 0,  0,  0,  0,  0,  0,  0,  0,  0  },
    { 1,  0,  1,  0,  1,  0,  1,  0,  1  },
    { 0,  2,  0,  0,  0,  0,  0,  2,  0  },
    { 0,  0,  0,  0,  0,  0,  0,  0,  0  },
    { 4,  3,  5,  6,  7,  6,  5,  3,  4  }
};
```

其中“0”表示无棋子；“1”表示黑卒，“2”表示黑炮，“3”表示黑马，“4”表示黑车，“5”表示黑象，“6”表示黑士，“7”表示黑将；“8”表示红兵，“9”表示红炮，“10”表示红马，“11”表示红车，“12”表示红相，“13”表示红仕，“14”表示红帅。

此外这个数组也表明了我们对棋盘进行了如图 2.1 所示的显示，并约定红方棋子总处于棋盘的上方。

对于着法的表示，我们直接借用棋盘数组的下标来记录着法的起点和目标点。至于是什么棋子在走，以及是否吃子、吃的是什么子，我们在着法结构中并不记录。这些信息由外部读取棋盘上起点、终点的数据获得。着法结构定义如下，其中还包含了对着法的历史得分的记录项，以供后面要用到的“历史启发”所使用。

```
public struct chessPoint/棋子位置结构
{
    public int x;
    public int y;
}
```

```
public struct chessMove//棋子走法的结构
{
    public chessPoint from; //起点
    public chessPoint to;    //目标点
    public int score; //该走法的历史得分
}
```

有了对棋盘局面和着法的表示之后，程序才能够完成以下操作：

1、生成所有合法着法；

2、执行着法、撤销着法；

3、针对某一局面进行评估。

因而，棋局表示好比是整个程序（计算机下棋引擎部分）的地基，之后所有的操作都将建立在其基础上。

2.3 悔棋（还原）

悔棋和还原是棋类软件中较为基本的功能。要实现悔棋和还原功能，首先我们要明确哪些信息应当被保存以供悔棋和还原所使用。

在我们的程序中保存了如下信息：

```
public int stepNum, tempNum;                    //棋局步骤数&&中盘步骤数
public byte[] stepQZ = new byte[MAXNUM];        //记录某步棋吃掉的棋子
public chessMove[] mList = new chessMove[MAXNUM]; //记录棋局的过程数组
```

在对弈过程中，每一回合我们都将棋局信息保存至 mList 数组及 stepQZ 数组中，以供后续所用。并且在 stepNum 变量中记录当前的棋局步骤数。另外考虑到人机对战时棋局的完整性，悔棋的时候需要撤销 2 次着法；还原的时候则只需要执行 1 次着法。

在悔棋中我们主要完成了以下工作：

1、下棋步骤数减

2、从 mList 数组和 stepQZ 数组中取出上一回合的棋局信息，恢复到前一局面；

3、将显示着法名称的列表框中的本回合的着法名称删除。

而在还原中我们所做的刚好和悔棋相反：

1、下棋步骤数加一；

2、从 mList 数组和 stepQZ 数组中取出下一回合的棋局信息，恢复到后一局面；

3、将被还原的着法名称重新添加到列表框中。

以上便是悔棋和还原功能所完成的具体操作，其代码分别写入上一步和下一步按钮的事件处理函数中。此外，根据游戏模式的不同（人机对战、中盘研究、复盘模式），对应函数的处理方式也有所区别。

2.4 保存（打开）棋局

当我们完成一盘经典的棋局，或者在研究人机博弈时棋局过程的合理性时，都会需要把该棋局的过程保存下来。这里需要说明的是通常象棋程序处于程序效率的考虑并不保存所有棋局的过程信息，而只是保存之前一步的下棋信息。然而我们在编写程序时考虑到程序的可读性和稳定性以及对现在计算机的配置来说这点开销基本上不会对程序的效率产生什么影响。因此保存了全部的信息。

针对现实中很多经典名局都是以中残局开局这一特点，再结合打谱功能，我们定义了以下一些数据结构来实现：

```
public byte[,] BeginBoard = new byte[10, 9];          //起始棋盘备份数组
public string[] tempstep = new string[MAXNUM];        //复盘棋局的临时数组
public string[] step = new string[MAXNUM];            //记录棋局的步骤数组
```

在执行保存操作时，把棋局的初始状态预先保存到 BeginBoard 数组中，并将棋局的过程数组 mList 转换成象棋棋局的标准格式（如车 3 平 4、炮 2 进 1）保存到 step 数组，然后将初始下棋方，初始棋盘 BeginBoard 数组，棋局步骤 step 数组等相关数据写入到对应的文件中去。

在执行打开（复盘）操作时，则从对应文件中读取棋局的初始状态保存到 BeginBoard 数组中，读取棋局的过程保存到 step 数组并对其进行数据转换再保存到 mList 数组，然后通过上一下、下一步按钮操作来实现复盘的功能。

在程序中，保存（打开）棋局通过菜单操作和一个专门的 ChessFile 类来实现，相关函数都以类成员函数的形式存在。

2.5 打谱、中盘研究

检测一个象棋博弈程序是否优秀一个很重要的手段就是将其与一些经典名局进行对比，观察其搜索到的最佳下法是否和大师们的评断相近似。考虑到很多经典名局开局的特殊性，在本程序中我们增加了打谱功能，能够在棋盘上实现棋子的摆放和取消。

此外很多比赛中都有大师现场评棋。期间大师们会对棋局的走势做一个推演以此来判断未来的棋局孰优孰劣。所以我们也添加了一个中盘研究功能：在棋局过程及复盘中可以随时切入中盘模式，完成推演后回到棋局的中断处继续棋局。

这两个功能也是本程序中重要的两个创新点。然后结合程序的搜索棋局功能我们可以很方便的测试评估函数的合理性。

第 3 章 算法设计与实现

3.1 着法生成

博弈的规则决定了哪些走法是合法的。对有的游戏来说这很简单（如五子棋：任何空白的位置都是合法的落子点）。但对于象棋来说，存在马走日，象走田等一系列复杂的规则。

计算机要搜索出它下一步要走的着法，首先就必要列出当前局面它所有可以走的着法。这就需要我们首先判断当前轮到谁下棋（黑方或者红方），然后通过对整个棋盘同色棋子的扫描，结合落子规则函数的判断，归纳出所有的着法并加以保存。对于象棋游戏来说，每一个局面所能够产生的着法大概是 40~60 种。考虑到计算机搜索时需要往下搜索若干步，我采用了一个 chessMove 结构的二维数组负责存储：public chessMove[,] moveList = new chessMove[8, 100]; 其中 8 为最大搜索层数，100 是每步棋最多 100 种可能的下法。这样就保证了程序不会有溢出异常。相关代码在 MoveGenerator 类中实现。

走法生成是博弈程序中一个相当复杂而且耗费运算时间的模块。不过通过良好的数据结构可以显著的提高生成的速度。

3.2 局面评估

局面评估是本文中最主要的研究方向。对于计算机来说，直接通过棋盘信息判别走法的好坏并不精确。除了输赢这两种局面可以可靠的判断外，其他的判断都只能做到大致估计。而现在计算机性能普遍提高较快，绝大多数机器通过合理的搜索算法都能够很轻易的搜索到当前局面的 5~7 层之后。这时评估函数是否合理，细腻，准确就成了衡量一个博弈程序优劣的最重要的指标。在象棋程序中如果说搜索算法是心脏，负责驱动整个程序；那么局面评估就是大脑。负责对搜索的内容进行判断和评价。因而搜索算法与局面评估构成整个下棋引擎的核心。

首先，先介绍一下在局面评估中需要考虑的因素。不同的棋类可能要考虑的因素略有差异。在中国象棋中所要考虑的最基本的几个因素包括如下 5 点：

1、棋子的基本价值

棋子的基本价值是指某一棋子本身所具有的价值。通俗地讲就是一个棋子它

值多少分。例如，车价值 500 分的话，那可能马价值 350，卒价值 100 等等。所以在评估局面时，我们首先要考虑双方的子力总和的对比。比如红方拥有士象全加车马炮，而黑方只有残士象加双马，则红方明显占优。

2、棋子的控制区域价值

棋子的控制区域，是指某一方的棋子在棋盘上所占据（控制）的位置（范围）。例如：卧底炮、过河卒、以及巡河车等都是较好的棋子位置状态，价值增加；而边马、离将等则属较差的棋子位置状态，价值降低。

3、棋子的灵活性价值

棋子的灵活性指的是棋子的可移动性。例如：起始位置的车机动性较差，所以我们下棋讲究早出车。同样被拌马腿的马机动性也较差（对于一步也不能走的棋子，可以认为其机动性为零）。棋子灵活性越强，其价值越高。

4、棋型价值

棋型指的是棋子之间的相互作用构成的特殊阵型。如连环炮，连环马等。其无论是攻或守都能够起到特殊的效果，价值增加。

5、棋子的相互关系（包括攻击（被攻击）关系和保护（被保护）关系）

这一点的分析最为复杂，因为一个棋子与其它子之间往往存在多重关系。例如一个马可能在对方的炮的攻击之下同时它又攻击着对方的车并且受到己方的保护。能够对局面做出准确的判断这点尤为关键。

在我的程序中，估值函数最后返回的是每一方的总分的差值，而各方的总分就是上面所提到的几个因素的打分的总和。

对于棋子的基本价值打分和控制区域打分，只要遍历棋盘，当遇到棋子时简单地去查事先定义好的“棋子基本价值表”和“控制区域价值表”，取出相对应的值进行累加即可（这些值的具体设定参考了前人的程序并作了适当的调整,今后仍应根据电脑下棋所反映出的实际问题对这些值作适当修改）。

对于灵活性打分，需要求出各个棋子总共有多少种走法，然后根据各个棋子所不同的灵活性价值每多一种走法就加一次相应的分数。

对棋子间相互关系以及棋型的打分，我定义了如下的关系数组进行描述：

```
byte[, ,] attackQZ = new byte[10, 9, 5];          //统计受到的攻击(棋子类型)
byte[, ,] defenseQZ = new byte[10, 9, 5];         //统计受到的保护(棋子类型)
int[,] attackNum = new int[10, 9];                //统计受到的攻击次数
int[,] defenseNum = new int[10, 9];               //统计受到的保护次数
```

chessPoint[, ,] attackPoint = new chessPoint[10, 9, 5]; //受到的攻击(棋子位置)
chessPoint[, ,] defensePoint = new chessPoint[10, 9, 5]; //受到的保护(棋子位置)

因为考虑到实战中几乎不可能出现同时有超过四个棋子攻击/保护一个子的情况，故数组第三维下标设定为 5 并且从 1 开始使用（贴近现实使用习惯）。

当遍历一遍棋盘之后，基本价值打分、控制区域打分和灵活性打分都可以完成，而关系数组也保存了相关数据。之后，再根据关系数组来具体考察棋子的相互关系和棋型，进行附加值打分。分析关系时：

首先，对王的攻击保护应分离出来单独考虑。因为对王的保护没有任何意义，一旦王被吃掉整个游戏就结束了。并且一些特殊的攻击要加以标记（如双炮将军、马后炮等）。

其次，对一个普通棋子进行分析的时候要注意如下几个问题：

1、当前轮到那方下棋（对方还是己方）。

2、攻防实力对比，当攻击者子力之和小于被攻击者子力之和时,攻击方将愿意换子。例如：一个车正遭受一个炮的攻击，那么任何对车的保护都将失去意义——对方肯定乐意用一个炮来换一个车。

3、程序中是否需要考虑双方兑子之后的控制区域及机动性变化情况（搜索继续展开节点后会考虑这些因素，只是在临界（叶子接点）时效果受影响。例如：程序设定的搜索深度为 5 层，而在第 6 层可能会出现一个将死的局面。

综上所述，局面评估的时候需要使用大量的棋类知识。但与此同时也会使程序的运行速度减慢。在高层搜索的时候，所产生的叶子节点数量以百万，千万计，这就意味着评估函数要被反复而大量的调用。合理的选择是我们要考虑的重要问题。一般来说：

Performance（性能）=Speed（速度） × Knowledge（知识）[②]

此外，评估函数中所使用的参数应反复调试以求合理。通常可以采用机机对战等各种形式加以验证。

3.3 搜索算法

搜索算法是下棋引擎的核心之一。它如同程序的心脏，驱动着整个程序。搜索算法的好坏直接影响着程序执行的效率。而如何使计算机在单位时间内完成尽可能多的运算是衡量一个搜索算法好坏的标志。

---

② D.Machie 于 1977 年提出（A Theory of Advice,Machine Intelligence 8,1977）. 转自参考文献[38]

关于博弈程序中的搜索算法，前人已经形成了的 Alpha-Beta[③] 、PVS[④]、SSS*[⑤] 等成熟的搜索算法（还有众多的增强、变种算法）。鉴于目前我的知识储备、时间、精力等均达不到推陈出新、另开炉灶的要求，再加之前人的算法已相当完善，所以在我的程序中直接借鉴了 PVS 搜索算法并配合哈希表和历史启发等优化手段。算法的基本原理如下：

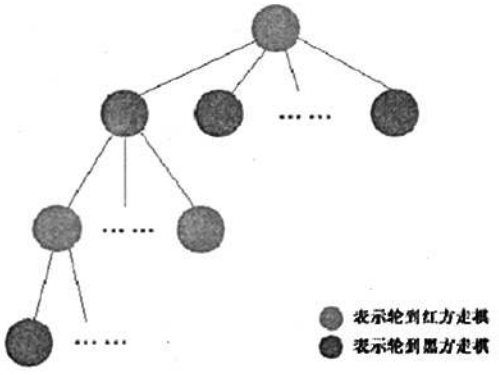


图 3.1 博弈树

在中国象棋里我们可以用一棵“博弈树”（一棵 n 叉树）来表示下棋的过程（树中每一个节点代表棋盘上的一个局面），对每一个节点根据不同的走法又产生不同的新节点，如此递推直到再无可选择的走法（棋局结束）。博弈树的模型如图 3.1 所示，我们可以把其中连接节点的线段看作是着法，不同的着法产生不同的局面。

现在让计算机来下棋，它应当选择一步对它最有利的着法（最终导致它取胜的着法）。获得最佳着法的方法就是“试走”每一种可能的着法，比较它们所产生的不同后果，然后从中选择能够产生对自己最有利的局面的着法。

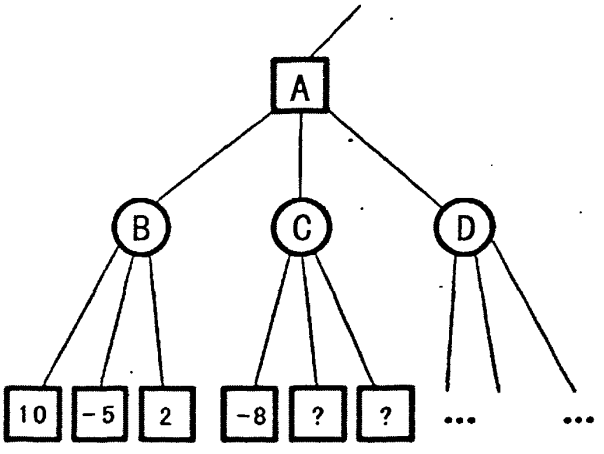
结合前面所讲的局面评估，那么我们可以通过比较局面分值的大小来判断局面的优劣。假定甲乙两方下棋，甲胜的局面是一个极大值（一个很大的正数），那么乙胜的局面就是一个极小值（极大值的负值），和棋的局面则是零值（或是接近零的值）。如此，当轮到甲走棋时他会尽可能地让局面上的分值大，相反轮到乙走棋时他会选尽可能地让局面上的分值小。反映到博弈树上，即如果我们假设奇数层表示轮到甲方走棋，偶数层表示轮到乙方走棋。那么由于甲方希望棋盘上的分值尽可能大，则在偶数层上我们会挑选分值最大的节点（偶数层的节点是

③ Alpha-beta 算法，该算法是由匹兹堡大学的三位科学家 Newell，Shaw and Simon 于 1958 年提出的。
④ A. Reinefeld 声称他于 1982 年发明了该算法，并证明它在展开的节点数目上优于 Alpha-Beta 算法。
⑤ Stockman 最初提出 SSS*算法但存在一些缺陷，Campbell 和 Marsland 对 SSS*算法做了一些修正。

甲走完一步棋之后的棋盘局面，反映了甲方对棋局形势的要求）。同样，由于乙方希望棋盘上的分值尽可能小，那么在奇数层上我们会选择分值最小的节点。这就是“极小-极大”（Minimax）[6]的基本思想。这样搜索函数在估值函数的协助下可以通过在奇数层选择分值最大（最小）的节点，在偶数层选择分值最小（最大）的节点的方式来搜索以当前局面为根节点、限定搜索层数以内的整棵树来获得一个最佳的着法。

然而不幸的是，博弈树相当庞大（成指数增长），因而搜索限定层数内的整棵树是一件相当费时的工作。对于中国象棋而言，每步棋着法数目大约是 40~60 种左右。假定为 40 种的话，那么搜索 4 层需要计算 250 万个节点，搜索 5 层需要计算 1 亿个节点，搜索 6 层需要计算 40 亿个节点！幸运的是，Alpha-Beta 搜索使得我们能在不影响搜索精度的前提下大幅减少节点数量。

Alpha-Beta 算法的基本思想是：某些局面如果确定能够产生出一个很糟糕的局面因而根本没有再继续考虑的价值，则应当立刻停止对其剩余子节点的分析（如果选择了它则必将得到那个糟糕的局面）这样一来便可以在很大程度上减少搜索的工作量，提高搜索效率，这称为“树的剪枝”。



图中□表示要取子节点中的最大值；○表示要取子节点中的最小值

图 3.2 Alpha-Beta 剪枝

[6] “极小-极大”（Minimax）最早是由 John von Nuoma（1903-1957，美籍匈牙利数学家）在 60 多年前完整描述的：

1、假设有对局面评分的方法，来预测棋手甲（我们称为最大者）会赢，或者对手（最小者）会赢，或者是和棋。评分用数字表示，正数代表最大者领先，负数代表最小者领先，零代表谁也不占便宜；

2、最大者的任务是增加棋盘局面的评分（即尽量让评分最大）；

3、最小者的任务是减少棋盘局面的评分（即尽量让评分最小）；

4、假设谁也不会犯错误，即他们都走能让使局面对自己最有利的着法。

下面用图 3.2 来进一步说明“树的剪枝”。为了简便，将博弈树进行了简化（每个节点只有三个分支）。假定棋盘上的局面发展到了节点 A，现在轮到红方走棋。

首先，我们考察 A 节点的子节点 B。B 节点要取子节点中的最小值。依次考察 B 节点的各个子节点，查看它们的分值。我们会发现 B 节点的第一个子节点返回 10，第二个子节点返回-5，第三个子节点返回 2。则 B 节点一定会选择返回值为-5 的那个节点。即如果 A 节点选择了产生 B 节点的走法，那么下一步，B 节点就会选择使得棋局发展成为分值为-5 的那个节点所表示的局面。

我们再来分析 A 节点的第二个子节点 C，C 节点与 B 节点同属一层，它依然要取子节点中的最小值。依次查看节点 C 的各个子节点的值，其第一个子节点返回了-8……这时已经不必再继续考察 C 节点的剩余子节点了，因为不管 C 节点的剩余子节点有怎样的分值，它只能传回-8 或者比-8 更小的值。而与前面已经分析过的 B 节点所传回-5 相比较，作为 A 节点要取子节点中的最大值，显然更不愿意看到-8 的局面。所以 C 节点肯定不会被选择，因此就没有必要继续搜索。由此，我们就在不影响搜索质量的前提下避免了搜索“无价值的” C 节点剩余子节点的工作，从而节省了大量时间，为在同样机器配置下搜索更多的层数提供了可能。

“极小-极大”的思想再加上“树的剪枝”，这就是 Alpha-Beta 搜索算法的核心。Alpha-Beta 算法的基本代码如下：

```
int AlphaBeta(int depth, int alpha, int beta)
{
    if (depth == 0)   return Evaluate();   //如果是叶子节点则返回估值
    CreateAllMove();   //产生所有合法着法
    while (each move m)   //遍历所有着法
    {
        MakeMove ();   //执行着法产生新局面
        int score= -AlphaBeta(depth - 1, -beta, -alpha);   //递归调用搜索新局面
        UnMakeMove();   //撤销着法恢复到前一局面
        if (score > alpha)     alpha = score;   //保留最大值
        if (score >= beta)     break;   //剪枝
    }
    return alpha;   //返回极大值
}
```

PVS（也称极小窗口 Minimal Windows Search）搜索算法是 Alpha-Beta 的一种变体。此算法基于如下假设：假定第一步是最佳的下法，搜索其后继者，直到更好的另一节点。在每一个中间节点中，第一个分枝都以完整窗口（a,b）进行搜索并产生一个值 v，后继的分枝则以一个极小的窗口（v,v+1）进行搜索。如果猜测被证明不准确（failhigh），随后就需要以（v+1,b）为窗口进行搜索；如果猜测是（faillow）则剪枝；否则表示猜测命中。以此达到高效搜索的目的。PVS 算法的基本代码如下

```
int NegaScout(int depth, int alpha, int beta)
{
    if (depth == 0)   return Eveluate();
    CreateAllMove();      MakeMove();
    best = -NegaScout(depth - 1, -beta, -alpha); //调用全窗口搜索第一个结点
    UnMakeMove();
    for (each move m)
    {
        if (best < beta) //如果不能 Beta 剪枝
        {
            if (best > alpha)   alpha = best;
            MakeMove();
            score = -NegaScout(depth - 1, -alpha - 1, -alpha); //极窄窗口搜索
            if (score > alpha && score < beta)//fail-high
                best = -NegaScout(depth - 1, -beta, -score); //重新搜索
            else if (score > best)   best = score; //窄窗搜索命中
            UnMakeMove();
        }
    }
    return best;
}
```

第 4 章 算法的优化

通过设计好的搜索算法和局面评估，我们已经可以让计算机产生搏弈的过程。但如果能加上一些附加的手段进行优化，计算机所体现出来的智商会大大的提高。

4.1 哈希表应用

计算机在搜索过程中时有可能会出现以下局面，如图 4.1：

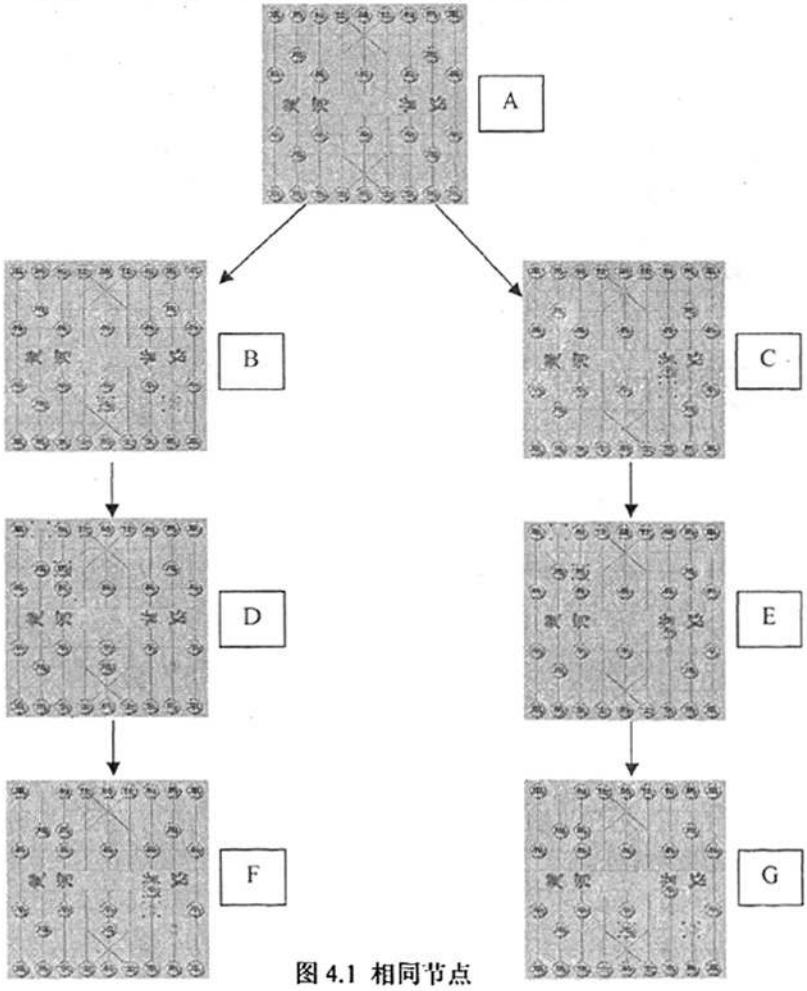


图 4.1 相同节点

从图中我们可以看出：从 A 节点开始，有两个分枝（B 节点和 C 节点）。B

节点又有子节点 D，C 节点有子节点 E。在再往下 D 节点又存在子节点 F，E 节点存在子节点 G。节点 F 和 G 在同一层，却又是两个完全相同的局面。这时我们在搜索 G 节点的时候完全可以利用已经搜索过的 F 节点的结果，而没有必要浪费时间重新搜索一次 G。

将已经搜索过的节点结果保存起来，在以后的搜索过程中先在表中进行查找，如果找到了则直接使用该值，找不到则正常搜索。可以提高搜索的效率。

考虑到存放数据和查找数据速度要尽可能的快（否则将得不偿失），哈希表是一种很好的选择。其基本算法如下：

```
int AlphaBeta(int depth, int alpha, int beta)
{
    score =SearchHashTable();//到哈希表中查找该局面是否出现过
    if (搜索命中)   return score;
    if (depth == 0)
    {
        AddHashItem(score);//添加结果到哈希表节点
        return Evaluate();  //由局面评估函数返回估值
        ……;
    }
}
```

为了能更快速的进行局面的存放和查找，每一个保存的节点应该是散列存放的。C#中有一个现成的 HashTable 类可以使用，我们要解决的问题是：如何迅速确定该节点在哈希表中的位置。为此我定义了一个 32 位的 10*9 二维数组和一个 64 位的 10*9 二维数组，它们的值都由随机函数产生。通过对 32 位二维数组的计算得到该节点在哈希表中的下标，再通过对 64 位二维数组的计算生成一个验证码，确定节点与局面是否相同。在搜索的过程中，每一个搜索过的节点都把得分保存到哈希表中去，而在搜索节点之前则先在表中查找一下该节点是否存在：如果存在，直接取出结果；否则正常搜索。

4.2 历史启发

在各类搜索算法中，改进搜索算法的目的在于将不必搜索的分枝从搜索过程中排除（剪枝）。而剪枝的效率几乎完全取决于节点排列的顺序。在节点排列顺

序处于理想状态下，Alpha-Beta 搜索算法需要遍历的节点数仅为完全遍历节点数平方根的两倍左右。也就是说如果一般情况搜索 5 层局面需要遍历 1 亿个节点，那么理想状态下的 Alpha-Beta 搜索只需要遍历约 2 万个节点（狭义理解速度提高了 5000 倍）。如何调整节点的顺序是提高搜索效率的关键。

我们可以将一些具体的棋类知识加入其中，例如：优先展开车马炮所产生的节点，优先展开攻防关键棋子所产生的节点等等。但这一方法需要通过一系列复杂的判断。

我们还可以使用历史启发[7]（History Heuristic）进行增强排序。该方法脱离了对具体棋类知识的依赖：在搜索过程中，每找到一个好的走法，就将该走法对应的历史得分增加。这样一个被多次搜索且被确认好的走法，其历史得分会较高。当搜索中间节点时，将产生的节点集合（步骤数组）按历史得分预先做一个排序，从而在搜索时引发更多的剪值以实现高效。好的走法可以定义如下：

1、由其产生的节点引发了剪枝。

2、是同层节点中最好的走法

这一方法的实现需要我们解决两个问题：

首先，如何将一个走法映射到历史得分的数组中？考虑到整个象棋棋盘是 10 行 9 列的显示，我们可以使用起始和目标坐标作为存取这个数组的两个下标。这样就只需要 90*90 的一个二维数组即可。

另一个问题是如何确定历史得分的权重？也就是说当一个好的走法被发现时，应该给它加多少分合适呢？一般而言，当该节点越靠近根节点的时候，所对应的局面相似程度就越高，搜索得到的值也就越可靠。所以在程序中设定每发现一个好走法就给其历史得分加上 2^{depth}（depth 是当前层数）。

4.3 开局库与残局库

中国象棋经过千百年的演绎和发展，涌现出许多千锤百炼的经典招数。无论我们的搏弈程序如何优秀，由于其受到诸多因素的影响，搜索出来的大部分着法都很难与之相媲美。特别是开局和残局。如果我们能将这些开局方法和残局手段加以引入，可以大大的提高博弈的质量。

大多数棋谱类书籍中都能找到几十种常用的开局方式。将这些经典开局搜集起来集中保存，我们可以建立一个 5~10 步的开局数据库。这样博弈时就可以先在开局库中找出类似的局面，直接获得其着法，加快开局的速度和质量。

开局库的数据库不必很大，我们也可以在其中使用哈希方法，只存放某一局面的哈希值以及对应的走法。有时候某一局面有着多种不错的走法，这时我们可以把一系列开局走法归为一组，前面的走法随机选择一种，后继的走法则在该组中优先选择。这样可以保持一个一致的策略。此外还可以增加一个胜率统计的字段，通过统计不同开局的胜率来自动选择成功率较高的开局。

残局与开局不同，残局的局面极少相同。我们建立的残局库可以使用回朔分析（Retrograde Analysis）的方法生成一个数据库。将所有 6 子（6 子以下）残局的结果保存在数据库中。在棋局的过程中一但出现了棋子数量少于或等于 5 个的局面，就可以直接在库中取出其最终的结果。这样搜索的精度就大大的提高，同时也可以提早发现一些必胜/必败的结局。但 7 子（7 子以上）残局库受存储空间和数据库查询速度等因素的影响，较难实现。

残局库的具体实现上依然可以使用哈希技术。唯一不同的是哈希表所使用的随机数组应该做为常量保存在残局库中，在程序启动时不再生成该数组，而是从库中取出使用。这样在搜索过程中生成的哈希值才能和残局库中保存的局面相契合。

第 5 章 算法分析与比较

在程序完成后测试中发现，编写此类的 Windows 应用程序还是应该选择更贴近底层的 C/C++语言。C#开发环境在运行速度上比 VC++开发环境要慢许多。据本人测试（Pentium(R)4 CPU 2.80GHz+512M 内存），在 C++环境下搜索 7 万个节点只需要 2~3 秒，但 C#.net 环境下却要花费 30~40 秒。这使得我的程序无法快速的搜索到更高层。表 5.1 是几种搜索引擎在开局时搜索不同深度的平均节点数。

表 5.1 算法分析比较

	1	2	3	4	5
Alpha-Beta 搜索	45	523	10689	72954	
PVS 搜索	45	552	5302	46361	
综合搜索	45	164	1567	6895	55623

与我所搜集到其他同类程序相比，相同搜索深度下，搜索的节点数大约减少了 17%左右。而且由于在程序中添加了多种辅助手段用来测试评估函数，所以局面评估更为精细。特别是开局时脱离开局库也能准确的模仿名家开局。

但由于我对在 C#使用 WindowsAPI 函数的不熟悉，导致我在界面及辅助功能部分花费了大量的时间和精力。因而没能够对计算机引擎部分作更深一步的挖掘和研究。对于诸如机器学习（Machine Learning）、位棋盘（BitBoard）、迭代加深（Iterative Deepening）、水平效应（Horizon Effect）、并行搜索等当今棋类对弈程序中所采用的先进技术和思想，在我的程序中并未涉及。这在一定程度上影响了程序中引擎的工作效率。

而在局面评估模块中又缺乏对面向对象思想的贯彻和使用。由于本人最初在编写局面评估部分时一心只想尽快看到成果，因而从一开始就按照自己先前的程序习惯（面向过程）来编写代码。到了开发后期，又缺少时间和精力来用面向对象思想重新改写程序，最终导致局面评估部分与类“无缘”。这也是我的一大遗憾。

第 6 章 结论与展望

6.1 结论

首先在导师的热心帮助下，最终顺利实现了程序。整个程序近 5000 行代码，内容涉及人工智能的基本理论以及开发 C#应用程序的一些基础知识。无论是在程序难度上还是规模上都是之前所未遇到过的。该程序的顺利完成为我积累了相当可观的运用理论知识解决实际问题的经验。特别是当程序运行过程中发现错误的时候，找出问题所在并解决问题更是一个提高实际编程能力的过程。得益于此次编写中国象棋对弈程序，我对人工智能有了一个较全面的认识——这为我以后进一步学习或研究的方向也提供了一定的参考。此外，我还实践了用 C#编写基本的 Windows 应用程序，为今后从事类似应用程序开发也起到了一定的帮助作用。

6.2 进一步工作的方向

本文的研究虽然取得了初步的成果，但依然任重道远，尚有许多需要进一步深入进行的研究工作，研究内容主要包括以下三个方面：

1、局面库的存储及管理方式；2、搜索树的并行处理方式的研究；3、使用面向对象的思想对局面评估部分做更深入的细化，优化各项参数。

在程序内借鉴了国际象棋中采用各种方法和取得的成果，包括开局库、残局库等。这些库的运用给程序节省了很多时间，但也吃掉了很多存储资源。6 子残局库需要大于 10G 的硬盘空间。我拟定的一项研究内容是对局面库的存储方式进行研究，以确定是否存在更好的方案来有效地降低局面库的存储空间。

另外，可以提高处理器利用率的树结构深度并行搜索方法也非常值得研究，也是我们所面临的一个关键问题。一般的小型机服务器大都配有多个处理器，从成本角度来看，可以使用多台微机来进行局面处理，这样就很容易搭建多处理器环境，给并行搜索的研究带来方便。

此外，使用 VC++开发环境重新改写程序。从面向对象出发，首先定义一个棋子类，然后派生出具体的车马炮等子类，以此来做个体/整体的评估。从而实现在类的对象中对局面做出准确的描述。

致 谢

工作六年后再进校园学习已是一件十分值得庆幸的事，更为庆幸的是，我遇到了一位好导师。我对计算机博弈的兴趣始于平时生活中的一些爱好，我的学位论文也是源于导师的指引。

在论文完成之际，首先，要特别感谢的是我的导师白似雪教授。最终我能够顺利完成论文，离不开导师对我的指导与帮助。谢谢您的支持！

其次，要感谢对我再教育的各位任课老师。在读研究生的期间，你们也给了我很多的教育和启发，让我能够进一步完善计算机学科的知识体系。

此外还要感谢学院领导李勇院长，黄海哨书记，蔡泽光院长对我的关心和爱护。让我尽可能的减少了一些学院工作安排，我才有更多的时间和精力去完成本次论文。

读研三年期间，班主任陈小红老师也给予了我很多的帮助。让我能更顺利的安排好各项学习时间，获得最新的通知。在这里也对您说一声谢谢，您辛苦了！

最后感谢胡彩明老师、肖雄斌老师等各位同事。谢谢你们在百忙之中抽空帮助我测试程序，我才能更快的修正程序中的各种 bug。感谢 WWW.CSDN.NET 论坛上各位大虾们，每次当我遇到问题的时候总能在第一时间得到你们的帮助。

罗涛

2009 年 10 月

参考文献

[1] J. von Neumann and O. Morgenstern. Theory of Games and Economic Behavior[M]. Princeton: Princeton University Press, 1944

[2] Shannon, Claude E. Programming a computer for playing chess[J]. Philosophical Magazine, 1950, 41:256-275

[3] A. Turing. Digital computers applied to games. Father than Thought[C] (B. Bowden, editor), Pitman, 1953, 286-295

[4] S.H. Fuller, J.G. Gasching and J.J. Gillogly, An analysis of the alpha-beta pruning algorithm[D]. Department of Computer Science Report, Caregie-Mellon University, Pittsburg, 1973

[5] D.E Knuth and R.N Moore, An analyze of alpha-beta pruning[J]. Artificial Intelligence, 1975,6: 293-326

[6] R.Korf. Iterative deepening: An optimal admissible tree search[J]. Artificial Intelligence, 1985, 27(1):97-109

[7] Jonathan Schaeffer,The history heuristic and alpha-beta search enhancements in practice. IEEE Transactions on Patton Analysis and Machine Intelligence,November 1989, PAMI-11(1): 1203~1212

[8] Breakthrough of the year, S, cience[J], 2007, Vol. 318, 1842-1849

[9] T.A. Marsland , COMPUTER CHESS AND SEARCH[D], Computing Science Department, University of Alberta

[10] Berliner H J. An Examination of Brute Force Intelligence[C]. International Joint Conference on Artificial Intelligence, 1981:581-587

[11] E. Heinz, "How DarkThought plays chess," Journal of the International Computer Chess Association, Vol. 20, No. 3, pp. 166-176 (1997)

[12] Cracraft, S.M. (1984). Bitmap move generation in chess. ICCA Journal, Vol. 7, No. 3, pp. 1984: 146-153

[13] Andreas Junghanns, J. Schaeffer, Search Vesus Knowledge in Game- Playing Program Revisited. Technical Report, Dept. of Computer Science, University of Albama, 1998

[14] Hsu, S.C. and Tsao, K.M. (1991). Design and Implementation of an Opening Game Knowledge-Base System for Computer Ch Endgame Database by Retrograde Analysis

[15] A.Zobrist. 'A New Hashing Method with Application for Game Playing'. Technical Report 88, Computer Science Department, University of Wisconsin, Madison. 1970

[16] Haw-ren Fang, Tsan-sheng Hsu, and Shun-chin Hsu, Construction of Chinese Chess

[17] REN WU, Donald F.Beal, A Memory Efficient Retrograde lgorithm and Its Application To Chinese Chess Endgames, More Games of No Chance MSRI Publications Volume 42, 2002

[18] Holland J H.Adaptation in nature and artificial system[M].Ann Arbor:The University of Michigan Press,1975.inese Chess[J]. Bulletin of the College of Engineering, N.T.U., No. 53, pp.

75-86
[19] 王小春：《PC 游戏编程（人机博弈）》[M]，2002 年，第一版，重庆大学出版社
[20] 网冠科技：《Visual C++.NET 小游戏开发时尚编程百例》[M]，2004 年，第一版，机械工业出版社
[21] 陈建春：《Visual C++ 高级编程技术——开发实例剖析》[M]，1999 年，第一版，电子工业出版社
[22] 涂光平，黄敏，钟坚成等：《Visual C++.NET 基础教程与上机指导》[M]，2005 年，第一版，清华大学出版社
[23] 伍红兵：《Visual C++ 编程深入引导》[M]，2002 年，第一版，中国水利水电出版社
[24] 何斌，马天予，王运坚等编著：《Visual C++ 数字图像处理》[M]，2002 年（第二版），人民邮电出版社
[25] 范如国，韩民春：《博弈论》[M]，武汉大学出版社，2006.1
[26] 李光久：《博弈论基础教程》[M]，化学工业出版社，2005
[27] 徐心和，徐长明，邓志立：积极开展民间棋类的计算机博弈活动[G]，《2007 中国自动化教育年会论文集》，机械工业出版社
[28] 徐心和，王骄：中国象棋计算机博弈关键技术分析[J]，《小型微型计算机系统》，2006 年第 6 期
[29] 徐心和，王浩，孔凡禹：事件对策理论及在棋类游戏中的应用[G]，《2007 年中国智能自动化会议论文集》
[30] 徐阳东，刘弘：遗传算法在机器博弈中的创新应用[J]，《电脑知识与技术》，2008 年第 1 卷第 7 期
[31] 何大华，陈传波：弈棋程序的设计思路[G]，《2004 年全国理论计算机科学学术年会计算机科学》
[32] 韩逢庆：基于选手风格的中国象棋博弈树搜索[G]，《2006 中国机器博弈学术研讨会》
[33] 王晓鹏，王骄，徐心和等：中国象棋与国际象棋比较分析[G]，《2006 中国机器博弈学术研讨会》
[34] 徐心和，郑新颖：棋牌游戏与事件对策[G]，《2006 中国机器博弈学术研讨会》
[35] 褚泽永，黄鸿，黄远灿：浅析中国象棋计算机博弈关键技术[G]，《2006 中国机器博弈学术研讨会》
[36] 叶品星：一种博弈树静态估值算法--△Feature 状态估值[J]，《计算机工程与设计》，2004 年第 25 卷第 7 期
[37] 周玮，张颐，周静怡：基于对弈局势的二次估值方法[J]，《系统仿真学报》，2006 年第 9 期
[38] 张越，芦东昕：面向目标的博弈搜索策略及其应用[J]，《计算机技术与发展》，2007 年第 3 期
[39] 王骄，王涛，罗艳红等：中国象棋计算机博弈系统评估函数的自适应遗传算法实现[J]，《东北大学学报（自然科学版）》，2005 年 第 10 期
[40] 周玮，王水涛，孙旸：中国象棋计算机博弈中的一种数据结构方法[J]，《计算机工程与应用》，2006 年第 35 期

[41] 陈光： 遗传算法在人工智能中的应用[J]，《福建电脑》，2005 年第 11 期
[42] 付强,陈焕文： 基于 RL 算法的自学习博弈程序设计及实现[J],》长沙理工大学学报(自然科学版)》， 2007 年 第 4 期
[43] ICGA 赛：电脑棋类程序竞赛 [EB/OL]。http://www.grappa.univ-lille3.fr/icga/?lang=4
[44] 象棋百科全书论坛 [EB/OL]。http://www.elephantbase.net

附录 A 搜索算法代码

```
namespace 中国象棋博奕{
class NegaScoutEngine : SearchEngine//搜索引擎派生类:极小窗口搜索引擎类
{
public override int SearchAGoodMove(ref byte[,] position, ref byte qz, ref chessMove qzMove, ref string step, int now)//搜索函数{
int qzNum = 0;
for (int i = 0; i <= 9; i++)
for (int j = 0; j <= 8; j++)
if (position[i, j] != 0) qzNum++;//统计棋子个数
MaxDepth = SearchDepth;      //设置搜索层数
if (qzNum < 20) MaxDepth++;//棋子较少时自动增加搜索深度
if (qzNum < 10) MaxDepth++;//棋子极少时自动增加搜索深度
int score = NegaScout(ref position, MaxDepth, now, -9999, 9999);
if(score==9999 || score == -9999) { System.Windows.Forms.MessageBox.Show("您获得了胜利！"); return 1; }//游戏结束
qz = position[bestMove.to.x, bestMove.to.y];//将吃掉的棋子传出
step = new ChessFile().ParseStep(position[bestMove.from.x, bestMove.from.y], bestMove.from.x, bestMove.from.y, bestMove.to.x, bestMove.to.y, new MoveGenerator().IsFront(position, position[bestMove.from.x, bestMove.from.y], bestMove.from.x, bestMove.from.y)); //转换该步走法为标准格式传出
qzMove = bestMove;                    //将最佳走法传出
MakeMove(ref position, bestMove);      //使用最佳走法修改棋盘
return 0;}

protected int NegaScout(ref byte[,] position, int d, int now, int alpha, int beta) //PVS 递归搜索函数,n 为距离叶子结点的层数{
int score;                //当前局面得分
int count;                //当前局面走法总数
```

```
byte nowQZ;              //当前操作的棋子
int best;                //极窄局面得分
score = IsGameOver(position, now);
if (score != 0) return score;//棋局结束,返回极大/极小值
if (d == 0)    //叶子结点,最后一层搜索  return p_E.Eveluate(position, now);
count = p_MG.CreateAllMove(position, d, now);
nowQZ = MakeMove(ref position, p_MG.moveList[d, 0]); //产生第一个节点
best = -NegaScout(ref position, d - 1, 3 - now, -beta, -alpha);
UnMakeMove(ref position, p_MG.moveList[d, 0], nowQZ);
if (d == MaxDepth)   bestMove = p_MG.moveList[d, 0]; //根部时保存最佳走法
for (int i = 1; i < count; i++){
if (best < beta) //如果不能 Beta 剪枝{
if (best > alpha)   alpha = best;
nowQZ = MakeMove(ref position, p_MG.moveList[d, i]); //产生新局面
score = -NegaScout(ref position, d - 1, 3 - now, -alpha - 1, -alpha);
if (score > alpha && score < beta){
best = -NegaScout(ref position, d - 1, 3 - now, -beta, -score); //重新搜索
if (d == MaxDepth)   bestMove = p_MG.moveList[d, i]; //根部时保存最佳走法
else if (score > best){
best = score; //窄窗搜索命中
if (d == MaxDepth)   bestMove = p_MG.moveList[d, i]; //根部时保存最佳走法
UnMakeMove(ref position, p_MG.moveList[d, i], nowQZ); //恢复原来局面}}
return best;}}}
```

附录 B　局面评估代码

```
namespace 中国象棋博奕
{
class Eveluation          //估值核心基类
{
int[] baseValues = new int[15] { 0, 150, 400, 350, 600, 300, 250, 1000, 150, 400, 350, 600, 300, 250, 1000 };//棋子的基本价值数组
int[] moveValues = new int[15] { 0, 5, 3, 5, 3, 2, 2, 10, 5, 3, 5, 3, 2, 2, 10 };//棋子的灵活性价值数组
int[,] PositionValue_B = new int[10, 9]//兵：过河分值+1~5，攻击九宫分值+5
{//棋子的控制区域价值数组
{ 0,   0,   5,   5,   10,   5,   5,   0,   0 },
{ 5,   10,  21,  30,  42,   30,  21,  10,  5 },
{ 4,   8,   18,  26,  37,   26,  18,  8,   4 },
{ 3,   6,   10,  17,  22,   17,  10,  6,   3 },
{ 2,   4,   7,   8,   12,   8,   7,   4,   2 },
{ 1,   2,   4,   4,   7,    4,   4,   2,   1 },
{ 0,   0,   1,   0,   2,    0,   1,   0,   0 },
{ 0,   0,   0,   0,   0,    0,   0,   0,   0 },
{ 0,   0,   0,   0,   0,    0,   0,   0,   0 },
{ 0,   0,   0,   0,   0,    0,   0,   0,   0 }
};
int[,] PositionValue_P = new int[10, 9]//炮：九宫内分值-5，卡自己象眼分值-2，在象位或士位分值-3　　卡对方象眼分值+2，河内挡将帅面分值+1
{
{  0,   0,  -3,  -3,   0,  -3,  -3,   0,   0 },
{  0,   2,  0,   2,   -3,  2,   0,   2,   0 },
{  -3,  0,  0,   -3,  -3,  -3,  0,   0,   -3 },
{  0,   2,  0,   2,   0,   2,   0,   2,   0 },
{  0,   0,  -3,  0,   0,   0,   -3,  0,   0 },
{  0,   0,  0,   1,   1,   1,   0,   0,   0 },
```

```
{  0,  -2,   0,  -1,   1,  -1,   0,  -2,   0  },
{  0,   0,   0,  -4,  -4,  -4,   0,   0,   0  },
{  0,  -2,   0,  -6,  -4,  -6,   0,  -2,   0  },
{  0,   0,   0,  -5,  -5,  -5,   0,   0,   0  }
};
```

int[,] PositionValue_M = new int[10, 9]//马：边马分值-5，九宫内分值-5，象位士位分值-3，卡自己象眼分值-2　过河分值+10，卡对方象眼分值+2，攻击九宫分值+5，河内挡将帅面分值+1

```
{
{  5,  10,  15,  12,  15,  12,  15,  10,  5  },
{  5,  22,  20,  22,  7,   22,  20,  22,  5  },
{  5,  15,  20,  17,  20,  17,  20,  15,  5  },
{  5,  17,  20,  22,  20,  22,  20,  17,  5  },
{  5,  10,  15,  15,  20,  15,  15,  12,  5  },
{  -5,  0,  -3,   1,   1,   1,  -3,   0,  -5  },
{  -5,  -2,  0,  -1,   1,  -1,   0,  -2,  -5  },
{  -8,  0,   0,  -4,  -7,  -4,   0,   0,  -8  },
{  -5,  -2,  0,  -6,  -4,  -6,   0,  -2,  -5  },
{  -5,  -5,  -8, -10, -10, -10,  -8,  -5,  -5  }
};
```

int[,] PositionValue_C = new int[10, 9]//车：河内挡将帅面分值+1，九宫内分值-5，卡自己象眼分值-2，卡对方象眼分值+2　象位士位分值-3，攻击九宫分值+5，巡河分值+10，车不出分值-10

```
{
{  15,  15,  12,  17,  20,   17,  12,  15,  15  },
{  15,  17,  15,  22,  17,   22,  15,  17,  15  },
{  12,  15,  15,  17,  17,   17,  15,  15,  12  },
{  0,   2,   0,   17,  15,   17,  0,   2,   0   },
{  10,  10,  7,   25,  25,   25,  7,   10,  10  },
{  10,  10,  10,  26,  26,   26,  10,  10,  10  },
{  0,  -2,   0,   14,  14,   14,  0,  -2,   0   },
```

```
{  0,   0,   0,   11,  11,   11,  0,    0,   0   },
{  0,  -2,   0,   9,   11,   9,   0,  -2,   0   },
{ -10,  0,   0,   10,  10,   10,  0,   0,  -10   }
};
int[,] PositionValue_X = new int[10, 9]//象
{
{ 0,   0,  0,   0,  0,   0,  0,   0,  0  },
{ 0,   0,  0,   0,  0,   0,  0,   0,  0  },
{ 0,   0,  0,   0,  0,   0,  0,   0,  0  },
{ 0,   0,  0,   0,  0,   0,  0,   0,  0  },
{ 0,   0,  0,   0,  0,   0,  0,   0,  0  },
{ 0,   0,  0,   0,  0,   0,  0,   0,  0  },
{ 0,   0,  0,   0,  0,   0,  0,   0,  0  },
{-10,  0,  0,   0,  20,  0,  0,   0, -10  },
{ 0,   0,  0,   0,  0,   0,  0,   0,  0  },
{ 0,   0,  0,   0,  0,   0,  0,   0,  0  }
};
int[,] PositionValue_S = new int[10, 9]//士
{
{ 0,  0,  0,  0,   0,   0,   0,  0,  0 },
{ 0,  0,  0,  0,   0,   0,   0,  0,  0 },
{ 0,  0,  0,  0,   0,   0,   0,  0,  0 },
{ 0,  0,  0,  0,   0,   0,   0,  0,  0 },
{ 0,  0,  0,  0,   0,   0,   0,  0,  0 },
{ 0,  0,  0,  0,   0,   0,   0,  0,  0 },
{ 0,  0,  0,  0,   0,   0,   0,  0,  0 },
{ 0,  0,  0,  0,   0,   0,   0,  0,  0 },
{ 0,  0,  0,  0,   10,  0,   0,  0,  0 },
{ 0,  0,  0,  0,   0,   0,   0,  0,  0 }
};
int[,] PositionValue_K = new int[10, 9]//将
```

```
{
{ 0,  0,  0,  0,   0,   0,   0,  0,  0 },
{ 0,  0,  0,  0,   0,   0,   0,  0,  0 },
{ 0,  0,  0,  0,   0,   0,   0,  0,  0 },
{ 0,  0,  0,  0,   0,   0,   0,  0,  0 },
{ 0,  0,  0,  0,   0,   0,   0,  0,  0 },
{ 0,  0,  0,  0,   0,   0,   0,  0,  0 },
{ 0,  0,  0,  0,   0,   0,   0,  0,  0 },
{ 0,  0,  0,-15, -10, -15,   0,  0,  0 },
{ 0,  0,  0,-10, -20, -10,   0,  0,  0 },
{ 0,  0,  0, 10,  30,  10,   0,  0,  0 }
};

public int[,] valueBoard = new int[10, 9];          //当前局面得分
private byte[,] tempBoard = new byte[10, 9];        //棋盘副本
private int grade = 3; //估值等级(1:初级估值 2:中级估值 3:高级估值)
public int black, red;                              //当前双方得分
private int nowTurn;                                //当前局面轮到谁下棋
private chessMove cm = new chessMove();//记录关系棋子的位置

public int[,] baseValue = new int[10, 9];           //基本价值
public int[,] posValue = new int[10, 9];            //区域价值
public int[,] attackValue = new int[10, 9];         //攻击价值
public int[,] defenseValue = new int[10, 9];        //保护价值
public int[,] moveValue = new int[10, 9];           //灵活性价值
public int[,] appendValue = new int[10, 9];         //棋形附加价值

public int[,] old_baseValue = new int[10, 9];       //基本价值
public int[,] old_posValue = new int[10, 9];        //区域价值
public int[,] old_attackValue = new int[10, 9];     //攻击价值
public int[,] old_defenseValue = new int[10, 9];    //保护价值
```

```
public int[,] old_moveValue = new int[10, 9];        //灵活性价值
public int[,] old_appendValue = new int[10, 9];     //棋形附加价值

byte[, ,] attackQZ = new byte[10, 9, 5]; //统计受到的攻击(棋子类型)
byte[, ,] defenseQZ = new byte[10, 9, 5]; //统计受到的保护(棋子类型)
int[,] attackNum = new int[10, 9];              //统计受到的攻击次数
int[,] defenseNum = new int[10, 9];             //统计受到的保护次数
chessPoint[, ,] attackPoint = new chessPoint[10, 9, 5]; //受到的攻击(棋子位置)
chessPoint[, ,] defensePoint = new chessPoint[10, 9, 5]; //受到的保护(棋子位置)

//public void setGrade(int g) { grade = g; }            //设置估值等级
public void GetValueBoard(ref int[,] valueChess) { valueChess = valueBoard; }
//返回估值得分数组
public double attackGrade = 1.0 / 15; //攻击等级系数
public double defenseGrade = 1.0 / 20;//防御等级系数

public int Eveluate(byte[,] chessBoard, int now) //估值函数{
tempBoard = chessBoard; nowTurn = now;

Array.Copy(attackValue, old_baseValue, 90);
Array.Copy(defenseValue, old_posValue, 90);
Array.Copy(attackValue, old_attackValue, 90);
Array.Copy(defenseValue, old_defenseValue, 90);
Array.Copy(moveValue, moveValue, 90);
Array.Copy(appendValue, old_appendValue, 90);

baseValue = new int[10, 9];
posValue = new int[10, 9];
attackValue = new int[10, 9];
defenseValue = new int[10, 9];
moveValue = new int[10, 9];
```

```
appendValue = new int[10, 9];
attackQZ = new byte[10, 9, 5];
defenseQZ = new byte[10, 9, 5];
attackNum = new int[10, 9];
defenseNum = new int[10, 9];
attackPoint = new chessPoint[10, 9, 5];
defensePoint = new chessPoint[10, 9, 5];
valueBoard = new int[10, 9];

int qzNum = 0;
for (int i = 0; i <= 9; i++)
for (int j = 0; j <= 8; j++)
if (tempBoard[i, j] != 0) qzNum++;//统计棋子个数
if (qzNum < 20) { attackGrade = 1.0 / 10; defenseGrade = 1.0 / 15; }
if (qzNum < 10) { attackGrade = 1.0 / 15; defenseGrade = 1.0 / 10; }
for (int i = 0; i <= 9; i++)
for (int j = 0; j <= 8; j++)  //对局面打分并保存在数组 valueBoard 中{
if (tempBoard[i, j] == 0) continue;
baseValue[i, j] = baseValues[tempBoard[i, j]];
switch (tempBoard[i, j]){
case 1: Value_B_B(i, j); break;
case 2: Value_B_P(i, j); break;
case 3: Value_B_M(i, j); break;
case 4: Value_B_C(i, j); break;
case 5: Value_B_X(i, j); break;
case 6: Value_B_S(i, j); break;
case 7: Value_B_K(i, j); break;
case 8: Value_R_B(i, j); break;
case 9: Value_R_P(i, j); break;
case 10: Value_R_M(i, j); break;
case 11: Value_R_C(i, j); break;
```

```
case 12: Value_R_X(i, j); break;
case 13: Value_R_S(i, j); break;
case 14: Value_R_K(i, j); break;
default: break;}}
SortValueArray(ref attackQZ, ref attackPoint);
SortValueArray(ref defenseQZ, ref defensePoint);
for (int i = 0; i <= 9; i++)
for (int j = 0; j <= 8; j++){
if (tempBoard[i, j] == 0) continue;
appendValue[i, j] += GetAppendValue(i, j); //统计棋形分
attackValue[i, j] = Convert.ToInt16(attackValue[i, j] * attackGrade);
defenseValue[i, j] = Convert.ToInt16(defenseValue[i, j] * defenseGrade);
valueBoard[i, j] = baseValue[i, j] + posValue[i, j] + attackValue[i, j] + defenseValue[i, j] + moveValue[i, j] + appendValue[i, j];//统计棋子总分}
black = 0; red = 0; //统计局面总分
for (int i = 0; i <= 9; i++)
for (int j = 0; j <= 8; j++){
if (tempBoard[i, j] >= 1 && tempBoard[i, j] <= 7) black += valueBoard[i, j];
if (tempBoard[i, j] >= 8 && tempBoard[i, j] <= 14) red += valueBoard[i, j];}
if (nowTurn == 1) return black - red;    else return red - black;}

private void Value_B_B(int i, int j) //对某位置的黑卒进行价值评估{
if (i < 5)                          //兵已经过河，搜索左和右{
if (j - 1 >= 0) { cm.from.x = i; cm.from.y = j; cm.to.x = i; cm.to.y = j - 1; GetRelationValue(cm); }
if (j + 1 <= 8) { cm.from.x = i; cm.from.y = j; cm.to.x = i; cm.to.y = j + 1; GetRelationValue(cm); }}
{ cm.from.x = i; cm.from.y = j; cm.to.x = i - 1; cm.to.y = j; GetRelationValue(cm); }//搜索前方
posValue[i, j] = PositionValue_B[i, j];}
```

```
    private void Value_R_B(int i, int j)  //对某位置的红兵进行价值评估{
    if (i >= 5)                           //兵已经过河，搜索左和右{
    if (j - 1 >= 0) { cm.from.x = i; cm.from.y = j; cm.to.x = i; cm.to.y = j - 1;
GetRelationValue(cm); }
    if (j + 1 <= 8) { cm.from.x = i; cm.from.y = j; cm.to.x = i; cm.to.y = j + 1;
GetRelationValue(cm); }}
    { cm.from.x = i; cm.from.y = j; cm.to.x = i + 1; cm.to.y = j;
GetRelationValue(cm); }//搜索前方
    posValue[i, j] = PositionValue_B[9 - i, 8 - j];}

    private void Value_B_P(int i, int j)   //对某位置的黑炮进行价值评估{
    int x, y, f;
    for (f = 0, x = i - 1; x >= 0; x--)//向上搜索
    if (tempBoard[x, j] != 0) { f = 1; break; }
    else moveValue[i, j] += moveValues[tempBoard[i, j]];
    if (f == 1)
    for (x--; x >= 0; x--)//炮隔子攻击搜索
    if (tempBoard[x, j] != 0) { cm.from.x = i; cm.from.y = j; cm.to.x = x; cm.to.y = j;
GetRelationValue(cm); break; }
    for (f = 0, x = i + 1; x <= 9; x++)//向下搜索
    if (tempBoard[x, j] != 0) { f = 1; break; }
    else moveValue[i, j] += moveValues[tempBoard[i, j]];
    if (f == 1)
    for (x++; x <= 9; x++)//炮隔子攻击搜索
    if (tempBoard[x, j] != 0) { cm.from.x = i; cm.from.y = j; cm.to.x = x; cm.to.y = j;
GetRelationValue(cm); break; }
    for (f = 0, y = j - 1; y >= 0; y--)//向左搜索
    if (tempBoard[i, y] != 0) { f = 1; break; }
    else moveValue[i, j] += moveValues[tempBoard[i, j]];
    if (f == 1)
    for (y--; y >= 0; y--)//炮隔子攻击搜索
```

```
if (tempBoard[i, y] != 0) { cm.from.x = i; cm.from.y = j; cm.to.x = i; cm.to.y = y;
GetRelationValue(cm); break; }
for (f = 0, y = j + 1; y <= 8; y++)//向右搜索
if (tempBoard[i, y] != 0) { f = 1; break; }
else moveValue[i, j] += moveValues[tempBoard[i, j]];
if (f == 1)
for (y++; y <= 8; y++)//炮隔子攻击搜索
if (tempBoard[i, y] != 0) { cm.from.x = i; cm.from.y = j; cm.to.x = i; cm.to.y = y;
GetRelationValue(cm); break; }
posValue[i, j] = PositionValue_P[i, j];}

private void Value_R_P(int i, int j)    //对某位置的红炮进行价值评估{
int x, y, f;
for (f = 0, x = i - 1; x >= 0; x--)//向上搜索
if (tempBoard[x, j] != 0) { f = 1; break; }
else moveValue[i, j] += moveValues[tempBoard[i, j]];
if (f == 1)
for (x--; x >= 0; x--)//炮隔子攻击搜索
if (tempBoard[x, j] != 0) { cm.from.x = i; cm.from.y = j; cm.to.x = x; cm.to.y = j;
GetRelationValue(cm); break; }
for (f = 0, x = i + 1; x <= 9; x++)//向下搜索
if (tempBoard[x, j] != 0) { f = 1; break; }
else moveValue[i, j] += moveValues[tempBoard[i, j]];
if (f == 1)
for (x++; x <= 9; x++)//炮隔子攻击搜索
if (tempBoard[x, j] != 0) { cm.from.x = i; cm.from.y = j; cm.to.x = x; cm.to.y = j;
GetRelationValue(cm); break; }
for (f = 0, y = j - 1; y >= 0; y--)//向左搜索
if (tempBoard[i, y] != 0) { f = 1; break; }
else moveValue[i, j] += moveValues[tempBoard[i, j]];
if (f == 1)
```

```
for (y--; y >= 0; y--)//炮隔子攻击搜索
if (tempBoard[i, y] != 0) { cm.from.x = i; cm.from.y = j; cm.to.x = i; cm.to.y = y;
GetRelationValue(cm); break; }
for (f = 0, y = j + 1; y <= 8; y++)//向右搜索
if (tempBoard[i, y] != 0) { f = 1; break; }
else moveValue[i, j] += moveValues[tempBoard[i, j]];
if (f == 1)
for (y++; y <= 8; y++)//炮隔子攻击搜索
if (tempBoard[i, y] != 0) { cm.from.x = i; cm.from.y = j; cm.to.x = i; cm.to.y = y;
GetRelationValue(cm); break; }
posValue[i, j] = PositionValue_P[9 - i, 8 - j];}

private void Value_B_M(int i, int j)  //对某位置的黑马进行价值评估{
if (i - 1 >= 0 && tempBoard[i - 1, j] == 0)//11 点和 1 点方向搜索{
if (i - 2 >= 0 && j - 1 >= 0) { cm.from.x = i; cm.from.y = j; cm.to.x = i - 2;
cm.to.y = j - 1; GetRelationValue(cm); }
if (i - 2 >= 0 && j + 1 <= 8) { cm.from.x = i; cm.from.y = j; cm.to.x = i - 2;
cm.to.y = j + 1; GetRelationValue(cm); }}
if (j - 1 >= 0 && tempBoard[i, j - 1] == 0)//10 点和 8 点方向搜索{
if (i - 1 >= 0 && j - 2 >= 0) { cm.from.x = i; cm.from.y = j; cm.to.x = i - 1;
cm.to.y = j - 2; GetRelationValue(cm); }
if (i + 1 <= 9 && j - 2 >= 0) { cm.from.x = i; cm.from.y = j; cm.to.x = i + 1;
cm.to.y = j - 2; GetRelationValue(cm); }}
if (i + 1 <= 9 && tempBoard[i + 1, j] == 0)//7 点和 5 点方向搜索{
if (i + 2 <= 9 && j - 1 >= 0) { cm.from.x = i; cm.from.y = j; cm.to.x = i + 2;
cm.to.y = j - 1; GetRelationValue(cm); }
if (i + 2 <= 9 && j + 1 <= 8) { cm.from.x = i; cm.from.y = j; cm.to.x = i + 2;
cm.to.y = j + 1; GetRelationValue(cm); }}
if (j + 1 <= 8 && tempBoard[i, j + 1] == 0)//2 点和 4 点方向搜索{
if (i - 1 >= 0 && j + 2 <= 8) { cm.from.x = i; cm.from.y = j; cm.to.x = i - 1;
cm.to.y = j + 2; GetRelationValue(cm); }
```

```
    if (i + 1 <= 9 && j + 2 <= 8) { cm.from.x = i; cm.from.y = j; cm.to.x = i + 1;
cm.to.y = j + 2; GetRelationValue(cm); }}
    posValue[i, j] = PositionValue_M[i, j];}

    private void Value_R_M(int i, int j)  //对某位置的红马进行价值评估{
    if (i - 1 >= 0 && tempBoard[i - 1, j] == 0)//11 点和 1 点方向搜索{
    if (i - 2 >= 0 && j - 1 >= 0) { cm.from.x = i; cm.from.y = j; cm.to.x = i - 2;
cm.to.y = j - 1; GetRelationValue(cm); }
    if (i - 2 >= 0 && j + 1 <= 8) { cm.from.x = i; cm.from.y = j; cm.to.x = i - 2;
cm.to.y = j + 1; GetRelationValue(cm); }}
    if (j - 1 >= 0 && tempBoard[i, j - 1] == 0)//10 点和 8 点方向搜索{
    if (i - 1 >= 0 && j - 2 >= 0) { cm.from.x = i; cm.from.y = j; cm.to.x = i - 1;
cm.to.y = j - 2; GetRelationValue(cm); }
    if (i + 1 <= 9 && j - 2 >= 0) { cm.from.x = i; cm.from.y = j; cm.to.x = i + 1;
cm.to.y = j - 2; GetRelationValue(cm); }}
    if (i + 1 <= 9 && tempBoard[i + 1, j] == 0)//7 点和 5 点方向搜索{
    if (i + 2 <= 9 && j - 1 >= 0) { cm.from.x = i; cm.from.y = j; cm.to.x = i + 2;
cm.to.y = j - 1; GetRelationValue(cm); }
    if (i + 2 <= 9 && j + 1 <= 8) { cm.from.x = i; cm.from.y = j; cm.to.x = i + 2;
cm.to.y = j + 1; GetRelationValue(cm); }}
    if (j + 1 <= 8 && tempBoard[i, j + 1] == 0)//2 点和 4 点方向搜索{
    if (i - 1 >= 0 && j + 2 <= 8) { cm.from.x = i; cm.from.y = j; cm.to.x = i - 1;
cm.to.y = j + 2; GetRelationValue(cm); }
    if (i + 1 <= 9 && j + 2 <= 8) { cm.from.x = i; cm.from.y = j; cm.to.x = i + 1;
cm.to.y = j + 2; GetRelationValue(cm); }}
    posValue[i, j] = PositionValue_M[9 - i, 8 - j];}

    private void Value_B_C(int i, int j)  //对某位置的黑车进行价值评估{
    int x, y;
    for (x = i - 1; x >= 0; x--)//向上搜索
    if (tempBoard[x, j] == 0) moveValue[i, j] += moveValues[tempBoard[i, j]];
```

```
    else { cm.from.x = i; cm.from.y = j; cm.to.x = x; cm.to.y = j;
GetRelationValue(cm); break; }
    for (x = i + 1; x <= 9; x++)//向下搜索
    if (tempBoard[x, j] == 0) moveValue[i, j] += moveValues[tempBoard[i, j]];
    else { cm.from.x = i; cm.from.y = j; cm.to.x = x; cm.to.y = j;
GetRelationValue(cm); break; }
    for (y = j - 1; y >= 0; y--)//向左搜索
    if (tempBoard[i, y] == 0) moveValue[i, j] += moveValues[tempBoard[i, j]];
    else { cm.from.x = i; cm.from.y = j; cm.to.x = i; cm.to.y = y;
GetRelationValue(cm); break; }
    for (y = j + 1; y <= 8; y++)//向右搜索
    if (tempBoard[i, y] == 0) moveValue[i, j] += moveValues[tempBoard[i, j]];
    else { cm.from.x = i; cm.from.y = j; cm.to.x = i; cm.to.y = y;
GetRelationValue(cm); break; }
    posValue[i, j] = PositionValue_C[i, j];}

    private void Value_R_C(int i, int j)  //对某位置的红车进行价值评估{
    int x, y;
    for (x = i - 1; x >= 0; x--)//向上搜索
    if (tempBoard[x, j] == 0) moveValue[i, j] += moveValues[tempBoard[i, j]];
    else { cm.from.x = i; cm.from.y = j; cm.to.x = x; cm.to.y = j;
GetRelationValue(cm); break; }
    for (x = i + 1; x <= 9; x++)//向下搜索
    if (tempBoard[x, j] == 0) moveValue[i, j] += moveValues[tempBoard[i, j]];
    else { cm.from.x = i; cm.from.y = j; cm.to.x = x; cm.to.y = j;
GetRelationValue(cm); break; }
    for (y = j - 1; y >= 0; y--)//向左搜索
    if (tempBoard[i, y] == 0) moveValue[i, j] += moveValues[tempBoard[i, j]];
    else { cm.from.x = i; cm.from.y = j; cm.to.x = i; cm.to.y = y;
GetRelationValue(cm); break; }
    for (y = j + 1; y <= 8; y++)//向右搜索
```

```
if (tempBoard[i, y] == 0) moveValue[i, j] += moveValues[tempBoard[i, j]];
else { cm.from.x = i; cm.from.y = j; cm.to.x = i; cm.to.y = y;
GetRelationValue(cm); break; }
posValue[i, j] = PositionValue_C[9 - i, 8 - j];}

private void Value_B_X(int i, int j)  //对某位置的黑象进行价值评估{
if (i - 1 >= 0 && j - 1 >= 0 && tempBoard[i - 1, j - 1] == 0)//左上方向搜索
if (i - 2 >= 5 && j - 2 >= 0) { cm.from.x = i; cm.from.y = j; cm.to.x = i - 2;
cm.to.y = j - 2; GetRelationValue(cm); }
if (i - 1 >= 0 && j + 1 <= 8 && tempBoard[i - 1, j + 1] == 0)//右上方向搜索
if (i - 2 >= 5 && j + 2 <= 8) { cm.from.x = i; cm.from.y = j; cm.to.x = i - 2;
cm.to.y = j + 2; GetRelationValue(cm); }
if (i + 1 <= 9 && j - 1 >= 0 && tempBoard[i + 1, j - 1] == 0)//左下方向搜索
if (i + 2 >= 5 && i + 2 <= 9 && j - 2 >= 0) { cm.from.x = i; cm.from.y = j;
cm.to.x = i + 2; cm.to.y = j - 2; GetRelationValue(cm); }
if (i + 1 <= 9 && j + 1 <= 8 && tempBoard[i + 1, j + 1] == 0)//右下方向搜索
if (i + 2 >= 5 && i + 2 <= 9 && j + 2 <= 8) { cm.from.x = i; cm.from.y = j;
cm.to.x = i + 2; cm.to.y = j + 2; GetRelationValue(cm); }
posValue[i, j] = PositionValue_X[i, j];}

private void Value_R_X(int i, int j)  //对某位置的红相进行价值评估{
if (i - 1 >= 0 && j - 1 >= 0 && tempBoard[i - 1, j - 1] == 0)//左上方向搜索
if (i - 2 >= 0 && i - 2 <= 4 && j - 2 >= 0) { cm.from.x = i; cm.from.y = j; cm.to.x
= i - 2; cm.to.y = j - 2; GetRelationValue(cm); }
if (i - 1 >= 0 && j + 1 <= 8 && tempBoard[i - 1, j + 1] == 0)//右上方向搜索
if (i - 2 >= 0 && i - 2 <= 4 && j + 2 <= 8) { cm.from.x = i; cm.from.y = j;
cm.to.x = i - 2; cm.to.y = j + 2; GetRelationValue(cm); }
if (i + 1 <= 9 && j - 1 >= 0 && tempBoard[i + 1, j - 1] == 0)//左下方向搜索
if (i + 2 <= 4 && j - 2 >= 0) { cm.from.x = i; cm.from.y = j; cm.to.x = i + 2;
cm.to.y = j - 2; GetRelationValue(cm); }
if (i + 1 <= 9 && j + 1 <= 8 && tempBoard[i + 1, j + 1] == 0)//右下方向搜索
```

```
if (i + 2 <= 4 && j + 2 <= 8) { cm.from.x = i; cm.from.y = j; cm.to.x = i + 2;
cm.to.y = j + 2; GetRelationValue(cm); }
posValue[i, j] = PositionValue_X[9 - i, 8 - j];}

private void Value_B_S(int i, int j)    //对某位置的黑士进行价值评估{
if (i - 1 >= 7 && i - 1 <= 9 && j - 1 >= 3 && j - 1 <= 5)//左上方向搜索
{ cm.from.x = i; cm.from.y = j; cm.to.x = i - 1; cm.to.y = j - 1;
GetRelationValue(cm); }
if (i - 1 >= 7 && i - 1 <= 9 && j + 1 >= 3 && j + 1 <= 5)//右上方向搜索
{ cm.from.x = i; cm.from.y = j; cm.to.x = i - 1; cm.to.y = j + 1;
GetRelationValue(cm); }
if (i + 1 >= 7 && i + 1 <= 9 && j - 1 >= 3 && j - 1 <= 5)//左下方向搜索
{ cm.from.x = i; cm.from.y = j; cm.to.x = i + 1; cm.to.y = j - 1;
GetRelationValue(cm); }
if (i + 1 >= 7 && i + 1 <= 9 && j + 1 >= 3 && j + 1 <= 5)//右下方向搜索
{ cm.from.x = i; cm.from.y = j; cm.to.x = i + 1; cm.to.y = j + 1;
GetRelationValue(cm); }
posValue[i, j] = PositionValue_S[i, j];}

private void Value_R_S(int i, int j)    //对某位置的红仕进行价值评估{
if (i - 1 >= 0 && i - 1 <= 2 && j - 1 >= 3 && j - 1 <= 5)//左上方向搜索
{ cm.from.x = i; cm.from.y = j; cm.to.x = i - 1; cm.to.y = j - 1;
GetRelationValue(cm); }
if (i - 1 >= 0 && i - 1 <= 2 && j + 1 >= 3 && j + 1 <= 5)//右上方向搜索
{ cm.from.x = i; cm.from.y = j; cm.to.x = i - 1; cm.to.y = j + 1;
GetRelationValue(cm); }
if (i + 1 >= 0 && i + 1 <= 2 && j - 1 >= 3 && j - 1 <= 5)//左下方向搜索
{ cm.from.x = i; cm.from.y = j; cm.to.x = i + 1; cm.to.y = j - 1;
GetRelationValue(cm); }
if (i + 1 >= 0 && i + 1 <= 2 && j + 1 >= 3 && j + 1 <= 5)//右下方向搜索
{ cm.from.x = i; cm.from.y = j; cm.to.x = i + 1; cm.to.y = j + 1;
```

```
GetRelationValue(cm); }
    posValue[i, j] = PositionValue_S[9 - i, 8 - j];}

    private void Value_B_K(int i, int j)        //对黑将进行价值评估{
    for (int x = 0; x <= 2; x++)
    if (tempBoard[x, j] == 14 && new MoveGenerator().IsValidMove(tempBoard, i, j,
x, j))
    { cm.from.x = i; cm.from.y = j; cm.to.x = x; cm.to.y = j;
GetRelationValue(cm); }//将帅会面
    if (i - 1 >= 7 && i - 1 <= 9) //上方向搜索
    { cm.from.x = i; cm.from.y = j; cm.to.x = i - 1; cm.to.y = j;
GetRelationValue(cm); }
    if (i + 1 >= 7 && i + 1 <= 9) //下方向搜索
    { cm.from.x = i; cm.from.y = j; cm.to.x = i + 1; cm.to.y = j;
GetRelationValue(cm); }
    if (j - 1 >= 3 && j - 1 <= 5) //左方向搜索
    { cm.from.x = i; cm.from.y = j; cm.to.x = i; cm.to.y = j - 1;
GetRelationValue(cm); }
    if (j + 1 >= 3 && j + 1 <= 5) //右方向搜索
    { cm.from.x = i; cm.from.y = j; cm.to.x = i; cm.to.y = j + 1;
GetRelationValue(cm); }
    posValue[i, j] = PositionValue_K[i, j];}

    private void Value_R_K(int i, int j)        //对红帅进行价值评估{
    for (int x = 7; x <= 9; x++)
    if (tempBoard[x, j] == 7 && new MoveGenerator().IsValidMove(tempBoard, i, j,
x, j))
    { cm.from.x = i; cm.from.y = j; cm.to.x = x; cm.to.y = j;
GetRelationValue(cm); }//将帅会面
    if (i - 1 >= 0 && i - 1 <= 2) //上方向搜索
    { cm.from.x = i; cm.from.y = j; cm.to.x = i - 1; cm.to.y = j;
```

```
GetRelationValue(cm); }
    if (i + 1 >= 0 && i + 1 <= 2) //下方向搜索
    { cm.from.x = i; cm.from.y = j; cm.to.x = i + 1; cm.to.y = j;
GetRelationValue(cm); }
    if (j - 1 >= 3 && j - 1 <= 5) //左方向搜索
    { cm.from.x = i; cm.from.y = j; cm.to.x = i; cm.to.y = j - 1;
GetRelationValue(cm); }
    if (j + 1 >= 3 && j + 1 <= 5) //右方向搜索
    { cm.from.x = i; cm.from.y = j; cm.to.x = i; cm.to.y = j + 1;
GetRelationValue(cm); }
    posValue[i, j] = PositionValue_K[9 - i, 8 - j];}


    private void GetRelationValue(chessMove cm) .//获得棋子的关系价值(灵活
性，攻击性，防御性){
    int i = cm.from.x;
    int j = cm.from.y;
    int x = cm.to.x;
    int y = cm.to.y;
    byte qz1 = tempBoard[i, j];
    byte qz2 = tempBoard[x, y];
    if (qz2 == 0) { moveValue[i, j] += moveValues[qz1]; return; } //灵活性分值增加
    if (new MoveGenerator().IsFriend(qz1, qz2)) //己方，防御性分值增加{
    if (qz2 != 7 && qz2 != 14)//保护将帅无意义{
    if (baseValues[qz2] > defenseValue[i, j])// 记录保护的最大子力
    defenseValue[i, j] = baseValues[qz2];
    for (int k = 1; k <= 4; k++) //目标棋子记录受到的保护
    if (defenseQZ[x, y, k] == 0){
    defenseQZ[x, y, k] = qz1;
    defensePoint[x, y, k].x = i;
```

```
defensePoint[x, y, k].y = j;
defenseNum[x, y]++;//目标棋子记录受到的保护次数
break;}}}
else    //敌方，攻击性分值增加{
if (baseValues[qz2] > attackValue[i, j])//纪录攻击的最大子力
attackValue[i, j] = baseValues[qz2];
for (int k = 1; k <= 4; k++)      //目标棋子记录受到的攻击
if (attackQZ[x, y, k] == 0){
attackQZ[x, y, k] = qz1;
attackPoint[x, y, k].x = i;
attackPoint[x, y, k].y = j;
attackNum[x, y]++;//目标棋子记录受到的攻击次数
break;}}}


private int GetAppendValue(int i, int j)  //获得棋子的附加价值(棋形){
if ((tempBoard[i, j] == 7 || tempBoard[i, j] == 14) && attackNum[i, j] != 0) {
if (new MoveGenerator().IsBlack(tempBoard[i, j]) && nowTurn == 1 || new
MoveGenerator().IsRed(tempBoard[i, j]) && nowTurn == 2)//轮到自己下{
int k, l;
for (k = 1; k <= 4; k++)
if (attackQZ[i, j, k] == 7 || attackQZ[i, j, k] == 14) //k 层将帅会面
return baseValues[tempBoard[i, j]] / 3;
for (k = 1; k <= 4; k++)
if (attackQZ[i, j, k] == 2 || attackQZ[i, j, k] == 9) break;//k 层受到炮攻击
if (tempBoard[i, j] == 7 && k < 5)//黑将被炮将军{
if (i == attackPoint[i, j, k].x)//水平受到攻击
if (j > attackPoint[i, j, k].y)//左侧受到攻击
for (l = attackPoint[i, j, k].y + 1; l < j; l++){
if (tempBoard[i, l] == 9) //炮炮将军{
for (int m = 0; m <= 9; m++)
```

```
for (int n = 0; n <= 8; n++)
for (int s = attackPoint[i, j, k].y + 1; s < l; s++)
if (tempBoard[m, n] >= 2 && tempBoard[m, n] <= 6 && new
MoveGenerator().IsValidMove(tempBoard, m, n, i, s))//有子可挡(炮炮间) return 0;
if (!(new MoveGenerator().IsValidMove(tempBoard, i, j, i + 1, j)) && !(new
MoveGenerator().IsValidMove(tempBoard, i, j, i - 1, j))) //将不能上下移动{
if (attackNum[attackPoint[i, j, k].x, attackPoint[i, j, k].y] == 0)//炮没有受到攻击
return -baseValues[tempBoard[i, j]] / 3;//将死
else{
if (defenseNum[attackPoint[i, j, k].x, attackPoint[i, j, k].y] == 0)  return 0;
if (defenseNum[attackPoint[i, j, k].x, attackPoint[i, j, k].y] != 0)
return -baseValues[tempBoard[i, j]] / 4;}}
else return -baseValues[tempBoard[i, j]] / 5;}
if (tempBoard[i, l] == 10) //炮马将军{
for (int m = 0; m <= 9; m++)
for (int n = 0; n <= 8; n++)
for (int s = attackPoint[i, j, k].y + 1; s < l; s++)
if (tempBoard[m, n] >= 2 && tempBoard[m, n] <= 6 && new
MoveGenerator().IsValidMove(tempBoard, m, n, i, s))//有子可挡(炮炮间) return 0;
if (!(new MoveGenerator().IsValidMove(tempBoard, i, j, i + 1, j)) && !(new
MoveGenerator().IsValidMove(tempBoard, i, j, i - 1, j))) //将不能上下移动{
if (attackNum[attackPoint[i, j, k].x, attackPoint[i, j, k].y] == 0)//炮未受到攻击{
for (int m = 0; m <= 9; m++)
for (int n = 0; n <= 8; n++)
for (int s = l + 1; s < j; s++)
if (tempBoard[m, n] >= 2 && tempBoard[m, n] <= 6 && new
MoveGenerator().IsValidMove(tempBoard, m, n, i, s))//有子可挡(马将间) return 0;
return -baseValues[tempBoard[i, j]] / 3;//将死}
else{
if (defenseNum[attackPoint[i, j, k].x, attackPoint[i, j, k].y] == 0) return 0;
if (defenseNum[attackPoint[i, j, k].x, attackPoint[i, j, k].y] != 0)
```

```
return -baseValues[tempBoard[i, j]] / 4;}}
else{
if (j - l == 2) return -baseValues[tempBoard[i, j]] / 3;//马后炮
else return -baseValues[tempBoard[i, j]] / 5;}}
if (tempBoard[i, l] == 11) //炮车将军 {
if (!(new MoveGenerator().IsValidMove(tempBoard, i, j, i + 1, j)) && !(new
MoveGenerator().IsValidMove(tempBoard, i, j, i - 1, j))) //将不能上下移动{
for (int m = 0; m <= 9; m++)
for (int n = 0; n <= 8; n++)
for (int s = l + 1; s < j; s++)
if (tempBoard[m, n] >= 2 && tempBoard[m, n] <= 6 && new
MoveGenerator().IsValidMove(tempBoard, m, n, i, s))//有子可挡
return -baseValues[tempBoard[i, j]] / 4;
return -baseValues[tempBoard[i, j]] / 3;//无子可挡}
else return -baseValues[tempBoard[i, j]] / 5;}}
else //右侧受到攻击
for (l = j + 1; l < attackPoint[i, j, k].y; l++){
if (tempBoard[i, l] == 9) //炮炮将军{
for (int m = 0; m <= 9; m++)
for (int n = 0; n <= 8; n++)
for (int s = l + 1; s < attackPoint[i, j, k].y; s++)
if (tempBoard[m, n] >= 2 && tempBoard[m, n] <= 6 && new
MoveGenerator().IsValidMove(tempBoard, m, n, i, s))//有子可挡(炮炮间) return 0;
if (!(new MoveGenerator().IsValidMove(tempBoard, i, j, i + 1, j)) && !(new
MoveGenerator().IsValidMove(tempBoard, i, j, i - 1, j))) //将不能上下移动{
if (attackNum[attackPoint[i, j, k].x, attackPoint[i, j, k].y] == 0)//炮没有受到攻击
return -baseValues[tempBoard[i, j]] / 3;//将死
else{
if (defenseNum[attackPoint[i, j, k].x, attackPoint[i, j, k].y] == 0) return 0:
if (defenseNum[attackPoint[i, j, k].x, attackPoint[i, j, k].y] != 0)
return -baseValues[tempBoard[i, j]] / 4;}}
```

```
else return -baseValues[tempBoard[i, j]] / 5;}
if (tempBoard[i, l] == 10) //炮马将军{
for (int m = 0; m <= 9; m++)
for (int n = 0; n <= 8; n++)
for (int s = l + 1; s < attackPoint[i, j, k].y; s++)
if (tempBoard[m, n] >= 2 && tempBoard[m, n] <= 6 && new
MoveGenerator().IsValidMove(tempBoard, m, n, i, s))//有子可挡(炮马间)  return 0;
if (!(new MoveGenerator().IsValidMove(tempBoard, i, j, i + 1, j)) && !(new
MoveGenerator().IsValidMove(tempBoard, i, j, i - 1, j))) //将不能上下移动{
if (attackNum[attackPoint[i, j, k].x, attackPoint[i, j, k].y] == 0)//炮未受到攻击{
for (int m = 0; m <= 9; m++)
for (int n = 0; n <= 8; n++)
for (int s = j + 1; s < l; s++)
if (tempBoard[m, n] >= 2 && tempBoard[m, n] <= 6 && new
MoveGenerator().IsValidMove(tempBoard, m, n, i, s))//有子可挡(马将间)  return 0;
return -baseValues[tempBoard[i, j]] / 3;//将死 }
else{
if (defenseNum[attackPoint[i, j, k].x, attackPoint[i, j, k].y] == 0) return 0;
if (defenseNum[attackPoint[i, j, k].x, attackPoint[i, j, k].y] != 0)
return -baseValues[tempBoard[i, j]] / 4;}}
else{
if (l - j == 2) return -baseValues[tempBoard[i, j]] / 3;//马后炮
else return -baseValues[tempBoard[i, j]] / 5;}}
if (tempBoard[i, l] == 11) //炮车将军{
if (!(new MoveGenerator().IsValidMove(tempBoard, i, j, i + 1, j)) && !(new
MoveGenerator().IsValidMove(tempBoard, i, j, i - 1, j))) //将不能上下移动{
for (int m = 0; m <= 9; m++)
for (int n = 0; n <= 8; n++)
for (int s = j + 1; s < l; s++)
if (tempBoard[m, n] >= 2 && tempBoard[m, n] <= 6 && new
MoveGenerator().IsValidMove(tempBoard, m, n, i, s))//有子可挡
```

```
return -baseValues[tempBoard[i, j]] / 4;
return -baseValues[tempBoard[i, j]] / 3;//无子可挡}
else   return -baseValues[tempBoard[i, j]] / 5;}}
if (j == attackPoint[i, j, k].y)//竖直受到攻击
if (i > attackPoint[i, j, k].x)//上方受到攻击
for (l = attackPoint[i, j, k].x + 1; l < i; l++){
if (tempBoard[l, j] == 9) //炮炮将军{
for (int m = 0; m <= 9; m++)
for (int n = 0; n <= 8; n++)
for (int s = attackPoint[i, j, k].x + 1; s < l; s++)
if (tempBoard[m, n] >= 2 && tempBoard[m, n] <= 6 && new
MoveGenerator().IsValidMove(tempBoard, m, n, s, j))//有子可挡  return 0;
if (!(new MoveGenerator().IsValidMove(tempBoard, i, j, i, j + 1)) && !(new
MoveGenerator().IsValidMove(tempBoard, i, j, i, j - 1))) //将不能左右移动{
if (attackNum[attackPoint[i, j, k].x, attackPoint[i, j, k].y] == 0)//炮没有受到攻击
return -baseValues[tempBoard[i, j]] / 3;//将死
else{
if (defenseNum[attackPoint[i, j, k].x, attackPoint[i, j, k].y] == 0) return 0;
if (defenseNum[attackPoint[i, j, k].x, attackPoint[i, j, k].y] != 0)
return -baseValues[tempBoard[i, j]] / 4;}}
else return -baseValues[tempBoard[i, j]] / 5;}
if (tempBoard[l, j] == 10) //炮马将军{
for (int m = 0; m <= 9; m++)
for (int n = 0; n <= 8; n++)
for (int s = attackPoint[i, j, k].x + 1; s < l; s++)
if (tempBoard[m, n] >= 2 && tempBoard[m, n] <= 6 && new
MoveGenerator().IsValidMove(tempBoard, m, n, s, j))//有子可挡(炮炮间)  return 0;
if (!(new MoveGenerator().IsValidMove(tempBoard, i, j, i, j + 1)) && !(new
MoveGenerator().IsValidMove(tempBoard, i, j, i, j - 1))) //将不能左右移动{
if (attackNum[attackPoint[i, j, k].x, attackPoint[i, j, k].y] == 0)//炮未受到攻击{
for (int m = 0; m <= 9; m++)
```

```
for (int n = 0; n <= 8; n++)
for (int s = l + 1; s < i; s++)
if (tempBoard[m, n] >= 2 && tempBoard[m, n] <= 6 && new MoveGenerator().IsValidMove(tempBoard, m, n, s, j))//有子可挡(马将间)  return 0;
return -baseValues[tempBoard[i, j]] / 3;//将死 }
else{
if (defenseNum[attackPoint[i, j, k].x, attackPoint[i, j, k].y] == 0)//炮受到攻击，对将的攻击无效  return 0;
if (defenseNum[attackPoint[i, j, k].x, attackPoint[i, j, k].y] != 0)
return -baseValues[tempBoard[i, j]] / 4;}}
else{
if (i - l == 2) return -baseValues[tempBoard[i, j]] / 3;//马后炮
else return -baseValues[tempBoard[i, j]] / 5;}}
if (tempBoard[l, j] == 11) //炮车将军 {
if (!(new MoveGenerator().IsValidMove(tempBoard, i, j, i, j + 1)) && !(new MoveGenerator().IsValidMove(tempBoard, i, j, i, j - 1))) //将不能左右移动{
for (int m = 0; m <= 9; m++)
for (int n = 0; n <= 8; n++)
for (int s = l + 1; s < i; s++)
if (tempBoard[m, n] >= 2 && tempBoard[m, n] <= 6 && new MoveGenerator().IsValidMove(tempBoard, m, n, s, j))//有子可挡
return -baseValues[tempBoard[i, j]] / 4;
return -baseValues[tempBoard[i, j]] / 3;//无子可挡}
else  return -baseValues[tempBoard[i, j]] / 5;}}}
if (tempBoard[i, j] == 14 && k < 5)//红帅被炮将军{
if (i == attackPoint[i, j, k].x)//水平受到攻击
if (j > attackPoint[i, j, k].y)//左侧受到攻击
for (l = attackPoint[i, j, k].y + 1; l < j; l++){
if (tempBoard[i, l] == 2) //双炮将军{
for (int m = 0; m <= 9; m++)
for (int n = 0; n <= 8; n++)
```

```
for (int s = attackPoint[i, j, k].y + 1; s < l; s++)
if (tempBoard[m, n] >= 9 && tempBoard[m, n] <= 13 && new MoveGenerator().IsValidMove(tempBoard, m, n, i, s))//有子可挡(炮炮间)  return 0;
if (!(new MoveGenerator().IsValidMove(tempBoard, i, j, i + 1, j)) && !(new MoveGenerator().IsValidMove(tempBoard, i, j, i - 1, j))) //将不能上下移动{
if (attackNum[attackPoint[i, j, k].x, attackPoint[i, j, k].y] == 0)//炮没有受到攻击
return -baseValues[tempBoard[i, j]] / 3;//将死
else{
if (defenseNum[attackPoint[i, j, k].x, attackPoint[i, j, k].y] == 0)//炮受到攻击，对将的攻击无效  return 0;
if (defenseNum[attackPoint[i, j, k].x, attackPoint[i, j, k].y] != 0)
return -baseValues[tempBoard[i, j]] / 4;}}
else return -baseValues[tempBoard[i, j]] / 5;}
if (tempBoard[i, l] == 3) //马炮将军{
for (int m = 0; m <= 9; m++)
for (int n = 0; n <= 8; n++)
for (int s = attackPoint[i, j, k].y + 1; s < l; s++)
if (tempBoard[m, n] >= 9 && tempBoard[m, n] <= 13 && new MoveGenerator().IsValidMove(tempBoard, m, n, i, s))//有子可挡(炮炮间)  return 0;
if (!(new MoveGenerator().IsValidMove(tempBoard, i, j, i + 1, j)) && !(new MoveGenerator().IsValidMove(tempBoard, i, j, i - 1, j))) //将不能上下移动{
if (attackNum[attackPoint[i, j, k].x, attackPoint[i, j, k].y] == 0)//炮未受到攻击{
for (int m = 0; m <= 9; m++)
for (int n = 0; n <= 8; n++)
for (int s = l + 1; s < j; s++)
if (tempBoard[m, n] >= 9 && tempBoard[m, n] <= 13 && new MoveGenerator().IsValidMove(tempBoard, m, n, i, s))//有子可挡(马将间)  return 0;
return -baseValues[tempBoard[i, j]] / 3;//将死 }
else{
if (defenseNum[attackPoint[i, j, k].x, attackPoint[i, j, k].y] == 0) return 0;
if (defenseNum[attackPoint[i, j, k].x, attackPoint[i, j, k].y] != 0)
```

```
return -baseValues[tempBoard[i, j]] / 4;}}
else{
if (j - l == 2) return -baseValues[tempBoard[i, j]] / 3;//马后炮
else return -baseValues[tempBoard[i, j]] / 5;}}
if (tempBoard[i, l] == 4) //车炮将军{
if (!(new MoveGenerator().IsValidMove(tempBoard, i, j, i + 1, j)) && !(new
MoveGenerator().IsValidMove(tempBoard, i, j, i - 1, j))) //将不能上下移动{
for (int m = 0; m <= 9; m++)
for (int n = 0; n <= 8; n++)
for (int s = l + 1; s < j; s++)
if (tempBoard[m, n] >= 9 && tempBoard[m, n] <= 13 && new
MoveGenerator().IsValidMove(tempBoard, m, n, i, s))//有子可挡
return -baseValues[tempBoard[i, j]] / 4;
return -baseValues[tempBoard[i, j]] / 3;//无子可挡}
else return -baseValues[tempBoard[i, j]] / 5;}}
else //右侧受到攻击
for (l = j + 1; l < attackPoint[i, j, k].y; l++){
if (tempBoard[i, l] == 2) //双炮将军{
for (int m = 0; m <= 9; m++)
for (int n = 0; n <= 8; n++)
for (int s = l + 1; s < attackPoint[i, j, k].y; s++)
if (tempBoard[m, n] >= 9 && tempBoard[m, n] <= 13 && new
MoveGenerator().IsValidMove(tempBoard, m, n, i, s))//有子可挡(炮炮间) return 0;
if (!(new MoveGenerator().IsValidMove(tempBoard, i, j, i + 1, j)) && !(new
MoveGenerator().IsValidMove(tempBoard, i, j, i - 1, j))) //将不能上下移动{
if (attackNum[attackPoint[i, j, k].x, attackPoint[i, j, k].y] == 0)//炮没有受到攻击
return -baseValues[tempBoard[i, j]] / 3;//将死
else{
if (defenseNum[attackPoint[i, j, k].x, attackPoint[i, j, k].y] == 0) return 0;
if (defenseNum[attackPoint[i, j, k].x, attackPoint[i, j, k].y] != 0)
return -baseValues[tempBoard[i, j]] / 4;}}
```

```
else return -baseValues[tempBoard[i, j]] / 5;}
if (tempBoard[i, l] == 3) //马炮将军{
for (int m = 0; m <= 9; m++)
for (int n = 0; n <= 8; n++)
for (int s = l + 1; s < attackPoint[i, j, k].y; s++)
if (tempBoard[m, n] >= 9 && tempBoard[m, n] <= 13 && new MoveGenerator().IsValidMove(tempBoard, m, n, i, s))//有子可挡(炮马间) return 0;
if (!(new MoveGenerator().IsValidMove(tempBoard, i, j, i + 1, j)) && !(new MoveGenerator().IsValidMove(tempBoard, i, j, i - 1, j))) //将不能上下移动{
if (attackNum[attackPoint[i, j, k].x, attackPoint[i, j, k].y] == 0)//炮未受到攻击{
for (int m = 0; m <= 9; m++)
for (int n = 0; n <= 8; n++)
for (int s = j + 1; s < l; s++)
if (tempBoard[m, n] >= 9 && tempBoard[m, n] <= 13 && new MoveGenerator().IsValidMove(tempBoard, m, n, i, s))//有子可挡(马将间) return 0;
return -baseValues[tempBoard[i, j]] / 3;//将死}
else{
if (defenseNum[attackPoint[i, j, k].x, attackPoint[i, j, k].y] == 0) return 0;
if (defenseNum[attackPoint[i, j, k].x, attackPoint[i, j, k].y] != 0)
return -baseValues[tempBoard[i, j]] / 4;}}
else{
if (l - j == 2) return -baseValues[tempBoard[i, j]] / 3;//马后炮
else return -baseValues[tempBoard[i, j]] / 5;}}
if (tempBoard[i, l] == 4) //车炮将军 {
if (!(new MoveGenerator().IsValidMove(tempBoard, i, j, i + 1, j)) && !(new MoveGenerator().IsValidMove(tempBoard, i, j, i - 1, j))) //将不能上下移动{
for (int m = 0; m <= 9; m++)
for (int n = 0; n <= 8; n++)
for (int s = j + 1; s < l; s++)
if (tempBoard[m, n] >= 9 && tempBoard[m, n] <= 13 && new MoveGenerator().IsValidMove(tempBoard, m, n, i, s))//有子可挡
```

```
return -baseValues[tempBoard[i, j]] / 4;
return -baseValues[tempBoard[i, j]] / 3;//无子可挡}
else   return -baseValues[tempBoard[i, j]] / 5;}}
if (j == attackPoint[i, j, k].y)//竖直受到攻击
if (i < attackPoint[i, j, k].x)//下方受到攻击
for (l = i + 1; l < attackPoint[i, j, k].x; l++){
if (tempBoard[l, j] == 2) //双炮将军{
for (int m = 0; m <= 9; m++)
for (int n = 0; n <= 8; n++)
for (int s = l + 1; s < attackPoint[i, j, k].x; s++)
if (tempBoard[m, n] >= 9 && tempBoard[m, n] <= 13 && new
MoveGenerator().IsValidMove(tempBoard, m, n, s, j))//有子可挡  return 0;
if (!(new MoveGenerator().IsValidMove(tempBoard, i, j, i, j + 1)) && !(new
MoveGenerator().IsValidMove(tempBoard, i, j, i, j - 1))) //将不能左右移动{
if (attackNum[attackPoint[i, j, k].x, attackPoint[i, j, k].y] == 0)//炮没有受到攻击
return -baseValues[tempBoard[i, j]] / 3;//将死
else{
if (defenseNum[attackPoint[i, j, k].x, attackPoint[i, j, k].y] == 0) return 0;
if (defenseNum[attackPoint[i, j, k].x, attackPoint[i, j, k].y] != 0)
return -baseValues[tempBoard[i, j]] / 4;}}
else return -baseValues[tempBoard[i, j]] / 5;}
if (tempBoard[l, j] == 3) //马炮将军{
for (int m = 0; m <= 9; m++)
for (int n = 0; n <= 8; n++)
for (int s = l + 1; s < attackPoint[i, j, k].x; s++)
if (tempBoard[m, n] >= 9 && tempBoard[m, n] <= 13 && new
MoveGenerator().IsValidMove(tempBoard, m, n, s, j))//有子可挡(炮炮间)  return 0;
if (!(new MoveGenerator().IsValidMove(tempBoard, i, j, i, j + 1)) && !(new
MoveGenerator().IsValidMove(tempBoard, i, j, i, j - 1))) //将不能左右移动{
if (attackNum[attackPoint[i, j, k].x, attackPoint[i, j, k].y] == 0)//炮未受到攻击{
for (int m = 0; m <= 9; m++)
```

```
for (int n = 0; n <= 8; n++)
for (int s = i + 1; s < l; s++)
if (tempBoard[m, n] >= 9 && tempBoard[m, n] <= 13 && new MoveGenerator().IsValidMove(tempBoard, m, n, s, j))//有子可挡(马将间) return 0;
return -baseValues[tempBoard[i, j]] / 3;//将死 }
else{
if (defenseNum[attackPoint[i, j, k].x, attackPoint[i, j, k].y] == 0) return 0;
if (defenseNum[attackPoint[i, j, k].x, attackPoint[i, j, k].y] != 0)
return -baseValues[tempBoard[i, j]] / 4;}}
else{
if (l - i == 2) return -baseValues[tempBoard[i, j]] / 3;//马后炮
else return -baseValues[tempBoard[i, j]] / 5;}}
if (tempBoard[l, j] == 4) //车炮将军 {
if (!(new MoveGenerator().IsValidMove(tempBoard, i, j, i, j + 1)) && !(new MoveGenerator().IsValidMove(tempBoard, i, j, i, j - 1))) //将不能左右移动{
for (int m = 0; m <= 9; m++)
for (int n = 0; n <= 8; n++)
for (int s = i + 1; s < l; s++)
if (tempBoard[m, n] >= 9 && tempBoard[m, n] <= 13 && new MoveGenerator().IsValidMove(tempBoard, m, n, s, j))//有子可挡
return -baseValues[tempBoard[i, j]] / 4;
return -baseValues[tempBoard[i, j]] / 3;//无子可挡}
else return -baseValues[tempBoard[i, j]] / 5;}}}
return -baseValues[tempBoard[i, j]] / 4; //轮到自己走，威胁较大}
else return -baseValues[tempBoard[i, j]] / 3; //轮到对方走，死棋局面}
if (attackNum[i, j] == 0 && defenseNum[i, j] == 0) //无攻击和无保护 return 0;
if (attackNum[i, j] == 0 && defenseNum[i, j] >= 1) //无攻击和多保护{
for (int k = 1; k <= 4; k++)//无攻击时保护得分减半{
if (baseValue[i, j] == defenseValue[defensePoint[i, j, k].x, defensePoint[i, j, k].y])
defenseValue[defensePoint[i, j, k].x, defensePoint[i, j, k].y] /= 2;} return 0; }
if (attackNum[i, j] >= 1 && defenseNum[i, j] == 0) //多攻击和无保护{
```

```
if (new MoveGenerator().IsBlack(tempBoard[i, j]) && nowTurn == 1 || new
MoveGenerator().IsRed(tempBoard[i, j]) && nowTurn == 2)
return -Convert.ToInt16(baseValues[tempBoard[i, j]] * attackGrade);
else //轮到对方走，威胁很大{
moveValue[i, j] = 0; attackValue[i, j] = 0; defenseValue[i, j] = 0;//清空相关分值
return -baseValues[tempBoard[i, j]];}}
if (attackNum[i, j] == 1 && defenseNum[i, j] >= 1) //1 攻击和多保护{
if (new MoveGenerator().IsBlack(tempBoard[i, j]) && nowTurn == 1 || new
MoveGenerator().IsRed(tempBoard[i, j]) && nowTurn == 2){
if (baseValues[tempBoard[i, j]] > baseValues[attackQZ[i, j, 1]])
return -Convert.ToInt16(baseValues[tempBoard[i, j]] * attackGrade);
else  //己方的价值比对方小或相等
return Convert.ToInt16(baseValues[tempBoard[i, j]] * defenseGrade); }
else{
if (baseValues[tempBoard[i, j]] > baseValues[attackQZ[i, j, 1]]) {
moveValue[i, j] = 0; attackValue[i, j] = 0; defenseValue[i, j] = 0;//清空相关分值
return -baseValues[tempBoard[i, j]]; }
else     //己方的价值比对方小或相等
return Convert.ToInt16(baseValues[tempBoard[i, j]] * defenseGrade); }}
if (attackNum[i, j] >= 2 && defenseNum[i, j] == 1) //多攻击和 1 保护{
if (new MoveGenerator().IsBlack(tempBoard[i, j]) && nowTurn == 1 || new
MoveGenerator().IsRed(tempBoard[i, j]) && nowTurn == 2) {
if (baseValues[tempBoard[i, j]] + baseValues[defenseQZ[i, j, 1]] >
baseValues[attackQZ[i, j, 1]]) //己方的价值比对方大
return -Convert.ToInt16(baseValues[tempBoard[i, j]] * attackGrade);
else  //己方的价值比对方小或相等
return Convert.ToInt16(baseValues[tempBoard[i, j]] * defenseGrade); }
else{
if (baseValues[tempBoard[i, j]] + baseValues[defenseQZ[i, j, 1]] >
baseValues[attackQZ[i, j, 1]]) //己方的价值比对方大{
moveValue[i, j] = 0; attackValue[i, j] = 0; defenseValue[i, j] = 0;//清空相关分值
```

```
return -baseValues[tempBoard[i, j]]; }
else     //己方的价值比对方小或相等
return Convert.ToInt16(baseValues[tempBoard[i, j]] * defenseGrade); }}
if (attackNum[i, j] == 2 && defenseNum[i, j] >= 2) //2 攻击和多保护{
if (new MoveGenerator().IsBlack(tempBoard[i, j]) && nowTurn == 1 || new
MoveGenerator().IsRed(tempBoard[i, j]) && nowTurn == 2) {
if (baseValues[tempBoard[i, j]] > baseValues[attackQZ[i, j, 1]] ||
baseValues[tempBoard[i, j]] + baseValues[defenseQZ[i, j, 1]] > baseValues[attackQZ[i,
j, 1]] + baseValues[attackQZ[i, j, 2]])
return -Convert.ToInt16(baseValues[tempBoard[i, j]] * attackGrade);
else  return Convert.ToInt16(baseValues[tempBoard[i, j]] * defenseGrade); }
else{
if (baseValues[tempBoard[i, j]] > baseValues[attackQZ[i, j, 1]] ||
baseValues[tempBoard[i, j]] + baseValues[defenseQZ[i, j, 1]] > baseValues[attackQZ[i,
j, 1]] + baseValues[attackQZ[i, j, 2]]) {//己方的价值比对方大
moveValue[i, j] = 0; attackValue[i, j] = 0; defenseValue[i, j] = 0;//清空相关分值
return -baseValues[tempBoard[i, j]]; }
else//己方的价值比对方小或相等
return Convert.ToInt16(baseValues[tempBoard[i, j]] * defenseGrade); }}
if (attackNum[i, j] >= 3 && defenseNum[i, j] == 2) //多攻击和 2 保护{
if (new MoveGenerator().IsBlack(tempBoard[i, j]) && nowTurn == 1 || new
MoveGenerator().IsRed(tempBoard[i, j]) && nowTurn == 2) {
if (baseValues[tempBoard[i, j]] + baseValues[defenseQZ[i, j, 1]] >
baseValues[attackQZ[i, j, 1]] || baseValues[tempBoard[i, j]] + baseValues[defenseQZ[i,
j, 1]] + baseValues[defenseQZ[i, j, 2]] > baseValues[attackQZ[i, j, 1]] +
baseValues[attackQZ[i, j, 2]])
return -Convert.ToInt16(baseValues[tempBoard[i, j]] * attackGrade);
else  return Convert.ToInt16(baseValues[tempBoard[i, j]] * defenseGrade); }
else{
if (baseValues[tempBoard[i, j]] + baseValues[defenseQZ[i, j, 1]] >
baseValues[attackQZ[i, j, 1]] || baseValues[tempBoard[i, j]] + baseValues[defenseQZ[i,
```

```
j, 1]] + baseValues[defenseQZ[i, j, 2]] > baseValues[attackQZ[i, j, 1]] +
baseValues[attackQZ[i, j, 2]]){//己方的价值比对方大
    moveValue[i, j] = 0; attackValue[i, j] = 0; defenseValue[i, j] = 0;//清空相关分值
    return -baseValues[tempBoard[i, j]]; }
    else//己方的价值比对方小或相等
    return Convert.ToInt16(baseValues[tempBoard[i, j]] * defenseGrade); }}
    if (attackNum[i, j] == 3 && defenseNum[i, j] >= 3) //3 攻击和多保护{
    if (new MoveGenerator().IsBlack(tempBoard[i, j]) && nowTurn == 1 || new
MoveGenerator().IsRed(tempBoard[i, j]) && nowTurn == 2) {
    if (baseValues[tempBoard[i, j]] > baseValues[attackQZ[i, j, 1]] ||
baseValues[tempBoard[i, j]] + baseValues[defenseQZ[i, j, 1]] > baseValues[attackQZ[i,
j, 1]] + baseValues[attackQZ[i, j, 2]] || baseValues[tempBoard[i, j]] +
baseValues[defenseQZ[i, j, 1]] + baseValues[defenseQZ[i, j, 2]] >
baseValues[attackQZ[i, j, 1]] + baseValues[attackQZ[i, j, 2]] + baseValues[attackQZ[i,
j, 3]])
    return -Convert.ToInt16(baseValues[tempBoard[i, j]] * attackGrade);
    else  return Convert.ToInt16(baseValues[tempBoard[i, j]] * defenseGrade); }
    else{
    if (baseValues[tempBoard[i, j]] > baseValues[attackQZ[i, j, 1]] ||
baseValues[tempBoard[i, j]] + baseValues[defenseQZ[i, j, 1]] > baseValues[attackQZ[i,
j, 1]] + baseValues[attackQZ[i, j, 2]] || baseValues[tempBoard[i, j]] +
baseValues[defenseQZ[i, j, 1]] + baseValues[defenseQZ[i, j, 2]] >
baseValues[attackQZ[i, j, 1]] + baseValues[attackQZ[i, j, 2]] + baseValues[attackQZ[i,
j, 3]]) {//己方的价值比对方大
    moveValue[i, j] = 0; attackValue[i, j] = 0; defenseValue[i, j] = 0;//清空相关分值
    return -baseValues[tempBoard[i, j]]; }
    else//己方的价值比对方小或相等
    return Convert.ToInt16(baseValues[tempBoard[i, j]] * defenseGrade); }}
    if (attackNum[i, j] == 4 && defenseNum[i, j] == 3) //4 攻击和 3 保护{
    if (new MoveGenerator().IsBlack(tempBoard[i, j]) && nowTurn == 1 || new
MoveGenerator().IsRed(tempBoard[i, j]) && nowTurn == 2) {
```

```
if (baseValues[tempBoard[i, j]] + baseValues[defenseQZ[i, j, 1]] > baseValues[attackQZ[i, j, 1]] || baseValues[tempBoard[i, j]] + baseValues[defenseQZ[i, j, 1]] + baseValues[defenseQZ[i, j, 2]] > baseValues[attackQZ[i, j, 1]] + baseValues[attackQZ[i, j, 2]] || baseValues[tempBoard[i, j]] + baseValues[defenseQZ[i, j, 1]] + baseValues[defenseQZ[i, j, 2]] + baseValues[defenseQZ[i, j, 3]] > baseValues[attackQZ[i, j, 1]] + baseValues[attackQZ[i, j, 2]] + baseValues[attackQZ[i, j, 3]])
return -Convert.ToInt16(baseValues[tempBoard[i, j]] * attackGrade);
else return Convert.ToInt16(baseValues[tempBoard[i, j]] * defenseGrade); }
else{
if (baseValues[tempBoard[i, j]] + baseValues[defenseQZ[i, j, 1]] > baseValues[attackQZ[i, j, 1]] || baseValues[tempBoard[i, j]] + baseValues[defenseQZ[i, j, 1]] + baseValues[defenseQZ[i, j, 2]] > baseValues[attackQZ[i, j, 1]] + baseValues[attackQZ[i, j, 2]] || baseValues[tempBoard[i, j]] + baseValues[defenseQZ[i, j, 1]] + baseValues[defenseQZ[i, j, 2]] + baseValues[defenseQZ[i, j, 3]] > baseValues[attackQZ[i, j, 1]] + baseValues[attackQZ[i, j, 2]] + baseValues[attackQZ[i, j, 3]]) {//己方的价值比对方大
moveValue[i, j] = 0; attackValue[i, j] = 0; defenseValue[i, j] = 0;//清空相关分值
return -baseValues[tempBoard[i, j]]; }
else return Convert.ToInt16(baseValues[tempBoard[i, j]] * defenseGrade); }}
if (attackNum[i, j] == 4 && defenseNum[i, j] == 4) //4 攻击和 4 保护{
if (new MoveGenerator().IsBlack(tempBoard[i, j]) && nowTurn == 1 || new MoveGenerator().IsRed(tempBoard[i, j]) && nowTurn == 2) {
if (baseValues[tempBoard[i, j]] > baseValues[attackQZ[i, j, 1]] || baseValues[tempBoard[i, j]] + baseValues[defenseQZ[i, j, 1]] > baseValues[attackQZ[i, j, 1]] + baseValues[attackQZ[i, j, 2]] || baseValues[tempBoard[i, j]] + baseValues[defenseQZ[i, j, 1]] + baseValues[defenseQZ[i, j, 2]] > baseValues[attackQZ[i, j, 1]] + baseValues[attackQZ[i, j, 2]] + baseValues[attackQZ[i, j, 3]])
return -Convert.ToInt16(baseValues[tempBoard[i, j]] * attackGrade);
else return Convert.ToInt16(baseValues[tempBoard[i, j]] * defenseGrade); }
```

```
else{
if (baseValues[tempBoard[i, j]] > baseValues[attackQZ[i, j, 1]] ||
baseValues[tempBoard[i, j]] + baseValues[defenseQZ[i, j, 1]] > baseValues[attackQZ[i,
j, 1]] + baseValues[attackQZ[i, j, 2]] || baseValues[tempBoard[i, j]] +
baseValues[defenseQZ[i, j, 1]] + baseValues[defenseQZ[i, j, 2]] >
baseValues[attackQZ[i, j, 1]] + baseValues[attackQZ[i, j, 2]] + baseValues[attackQZ[i,
j, 3]] || baseValues[tempBoard[i, j]] + baseValues[defenseQZ[i, j, 1]] +
baseValues[defenseQZ[i, j, 2]] + baseValues[defenseQZ[i, j, 3]] >
baseValues[attackQZ[i, j, 1]] + baseValues[attackQZ[i, j, 2]] + baseValues[attackQZ[i,
j, 3]] + baseValues[attackQZ[i, j, 4]]) {//己方的价值比对方大
moveValue[i, j] = 0; attackValue[i, j] = 0; defenseValue[i, j] = 0;//清空相关分值
return -baseValues[tempBoard[i, j]]; }
else//己方的价值比对方小或相等
return Convert.ToInt16(baseValues[tempBoard[i, j]] * defenseGrade); }}
return 0; }

private void SortValueArray(ref byte[, ,] arrayQZ, ref chessPoint[, ,] arrayPt) {
byte t; chessPoint p;
for (int i = 0; i <= 9; i++)
for (int j = 0; j <= 8; j++)
for (int k = 1; k <= 3; k++)
for (int s = k + 1; s <= 4; s++)
if (arrayQZ[i, j, s] != 0 && arrayQZ[i, j, s] < arrayQZ[i, j, k]) {
t = arrayQZ[i, j, s]; p = arrayPt[i, j, s];
arrayQZ[i, j, s] = arrayQZ[i, j, k]; arrayPt[i, j, s] = arrayPt[i, j, k];
arrayQZ[i, j, k] = t; arrayPt[i, j, k] = p; }}}}
```

攻读学位期间的研究成果

已发表论文：

1. 罗涛，蔡虔. 多校区高校专网方案的研究与分析[J]. 《科技广场》，2009 年第 7 期.