

题 目： 能人机对弈的五子棋的设计与实现

姓 名： 石贇

学 号： 201808170043

院 系： 计算机科学与技术学院

专 业： 渗透与测试方向

年级班级： 2018级1班

指导教师： 周文刚

2022年 3月30 日

**毕业论文（设计）作者声明**

本人郑重声明：所呈交的毕业论文是本人在导师的指导下独立进行研究所取得的研究成果。除了文中特别加以标注引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写的成果作品。

本人完全了解有关保障、使用毕业论文的规定，同意学校保留并向有关毕业论文管理机构送交论文的复印件和电子版。同意省级优秀毕业论文评选机构将本毕业论文通过影印、缩印、扫描等方式进行保存、摘编或汇编；同意本论文被编入有关数据库进行检索和查阅。

本毕业论文内容不涉及国家机密。

论文题目：能人机对弈的五子棋的设计与实现

作者单位：计算机科学与技术学院

作者签名： （学号：201808170043）

2022年4月13日

**目 录**

[引言 2](#_Toc375)

[1 前言 2](#_Toc6795)

[2. 相关技术介绍 2](#_Toc13526)

[2.1 五子棋规则简单介绍 2](#_Toc5776)

[2.2 使用的技术的简单介绍。 3](#_Toc31741)

[2.3 系统思路的简单介绍 3](#_Toc5783)

[2.4 系统结构的简单介绍： 4](#_Toc26139)

[2.5 关键点介绍 5](#_Toc22985)

[2.5.1 五元组 5](#_Toc22403)

[2.5.2 估值评分。 6](#_Toc5559)

[2.5.3 搜索算法 7](#_Toc12262)

[2.5.4 剪枝算法 9](#_Toc12843)

[2.5.5 胜利判定 10](#_Toc2902)

[3 系统测试 12](#_Toc10389)

[3.1基本功能测试 13](#_Toc15495)

[3.1.1 测试界面运行 13](#_Toc19415)

[3.2系统不同模式智能性测试 14](#_Toc20962)

[3.2.1 选择黑棋 14](#_Toc9086)

[3.2 选择白棋 14](#_Toc17510)

[3.3 测试结论 15](#_Toc7588)

[参考文献 16](#_Toc9845)

[致谢 17](#_Toc13590)

**能人机对弈的五子棋的设计与实现**

摘 要：目前为止，市面上已经有了很多的五子棋游戏，但多为残局和PVP，少数的PVE模式棋局也大多存在着运算时间长、对计算机要求高、不够智能等缺点，无法实现在一个简单的个人计算机上做到高速有效反应。该系统基于C/C++实现在个人计算机上可以人机对弈的五子棋游戏。通过设立五元组、定价估值、深度搜索等方法。并以是否进行深度搜索为区别设立了可供对比的两个模式。就整体来说，该程序体积小，环境要求低，在快速反映的基础上还有不低的智能。

关键词：人机对弈、快速反应、深度搜索、剪枝。

**Design and Realization of Backgammon Capable of Man-machine Game**

**Abstract**:So far, there are many backgammon games on the market, but most of them are endgames and PVP, and most of the few PVE games have the disadvantages of long computing time, high requirements on computers, and insufficient intelligence, which cannot be implemented in a simple individual. High-speed and effective response on the computer. The system is based on C/C++ to realize the backgammon game which can be played against human and machine on personal computer. Through the establishment of quintuple, pricing valuation, in-depth search and other methods. And set up two modes for comparison based on whether to conduct in-depth search or not. On the whole, the program is small in size, low in environmental requirements, and has a lot of intelligence on the basis of rapid response.

**Keywords**:**M**an-machine Game; Quick Response; Deep Search; Pruning.

引言

目前为止，虽然市面上已经有了很多的五子棋游戏，但是大多依旧是依靠残局和PVP来充数，虽然有可以实现人机对弈的，但是大多存在着运算时间长、对计算机要求高或者不够智能等缺点，无法实现在一个简单的个人计算机上做到高速有效反应。该系统通过设立五元组，从而对每个位置进行价值评估。在有多个位置可选的时候，进行深度搜索来提升其智能性。以是否进行深度搜索为区别设立了两个模式可供对比。整体来说，该程序体积小，环境要求低，在快速反映的基础上还有不低的智能，完全满足一般的游戏需求。

# 1 前言

五子棋游戏历史悠久，规则简单，老少皆宜，对环境要求又简单，这使得其受众不断增加，成为了如今最普遍的棋类游戏之一。对人机对弈五子棋的研究过程中，人们不断地对其搜索算法进行优化升级，已经发展出了极大极小值搜索、贪心算法、遗传算法、置换表技术、运用 MCTS 和卷积神经网络训练、使用α-β剪枝来减小搜索范围等各种方法，以此来优化搜索方法、提高程序的智能性，并拥有了许多的成果。然而在日益现代化的今天，五子棋游戏虽然早已在互联网上出现，但是大多却只是依靠残局和PVP来充数，实现人机对弈的虽然也有，但是却依旧存在着运算时间长、对计算机要求高、不够智能等缺点，无法实现在个人计算机上做到高速有效反应。

# 2. 相关技术介绍

## 2.1 五子棋规则简单介绍

双方一方执黑棋一方执白棋，交替落子，黑子先手。棋盘大小为15×15。该系统不考虑禁手等情况。当任意一方落子后，如果形成五子连珠则会立刻获得胜利，另一方落败。

## 2.2 使用的技术的简单介绍。

C/C++：C语言作为一种通用的高级语言，其最显著的特点就是运行速度快，几乎和汇编等同。而C++则是可以面向对象编程并且可以运行所有的C代码。

EasyX ：EasyX Graphics Library 是针对 Visual C++ 的免费绘图库，支持 VC6.0 ~ VC2022，简单易用，学习成本极低，应用领域广泛。

α-β剪枝技术：Alpha-beta剪枝是一种搜索[算法](https://baike.baidu.com/item/%E7%AE%97%E6%B3%95" \t "https://baike.baidu.com/item/Alpha-beta%E5%89%AA%E6%9E%9D/_blank)，用以减少极小化极大算法（Minimax算法）搜索树的节点数。在系统进行深度搜索，模拟棋手与系统交替落子的过程中便多次使用。其原理为通过剪掉博弈树的冗余枝杈从而减少系统开销，节约时间。如图1所示，便是通过剪掉B及其子节点来缩减了冗余计算的数量。具体算法思路详见[2.5.4 剪枝算法](#_2.5.4  剪枝算法)。该算法一种对抗性搜索算法，主要应用于机器游玩的二人游戏（如[井字棋](https://baike.baidu.com/item/%E4%BA%95%E5%AD%97%E6%A3%8B" \t "https://baike.baidu.com/item/Alpha-beta%E5%89%AA%E6%9E%9D/_blank)、[象棋](https://baike.baidu.com/item/%E8%B1%A1%E6%A3%8B" \t "https://baike.baidu.com/item/Alpha-beta%E5%89%AA%E6%9E%9D/_blank)、[围棋](https://baike.baidu.com/item/%E5%9B%B4%E6%A3%8B" \t "https://baike.baidu.com/item/Alpha-beta%E5%89%AA%E6%9E%9D/_blank)）。

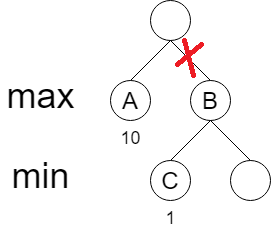


图 1 剪枝算法示意图

## 2.3 系统思路的简单介绍

五子棋游戏作为经典的零和博弈模型，博弈的每一方都假设对方的所有策略的根本目的是使自己最大程度地失利，并据此最优化自己的对策，那么，系统通过一定的线性运算，可使得每一次博弈过程，都能够找到一个“最优解”，最终达到胜利。

由于C语言的代码运行速度几乎与汇编代码等同，所以为了运行速度便使用C来编写程序。由于C++可以运行任意一行C代码，以及绘图时使用了EasyX Graphics Library，所以系统最终为“.cpp”格式。EasyX Graphics Library是针对 Visual C++ 的免费绘图库。

程序使用EasyX 来创建窗口，绘制棋子，使用五元组来作为最基本的估值单位，使用估值函数计算单个点位和当下棋局的价值，使用搜索算法来确定系统落子位置。

在游戏过程中，系统使用全局变量num来记录步数，以此来判断轮到哪一方落子。用自己创建的coordinate类型来记录每一个位置的信息。使用[五元组](#_2.5.1  五元组)作为基础最基础的价值计算元素。使用value\_one（）函数来评估单个点位的五元组价值之和。使用value\_whole（）函数评价当下整个棋局的优劣。在不进行深度搜索时，系统通过比较每个空余位置的点位估价，寻找其最大值（当下最优解）来确定落子位置。在进行深度搜索时，则会依次遍历估价最大五个位置，模拟交替落子，并计算最后的棋局价值，选择最优解将棋局导向对系统方最有利的局面。每次真正落子后，都要根据不同情况去判断是否胜利，具体见[2.5.5 胜利判定](#_2.5.5  胜利判定)。

struct coordinate{//coordinate类，记录了某一点位的落子情况及其估价。

int x;

int y; //x,y为该点位在数组中的坐标。

int rt; //rt表示棋子颜色。

int score; //score表示该点的估价。

}

## 2.4 系统结构的简单介绍：

系统的简单组成：

开始界面 ==> 选择模式界面(以是否进行深度搜索为区别) ==> 游戏界面 ==> 是否结束界面 ==> 结束界面。

系统运行结构简单介绍：

系统运行大致流程如图1 系统运行流程图，其中num表示步数，value值为该点在当前棋局中的估价，表示在下一步中该点的重要程度，计算方法详见[2.5.2 估值评分](#_2.5.2  估值评分。)，深度搜索是考虑到有时候对当下而言的最优解并非是对整局游戏而言的最优解，系统需要考虑如果在此处落子后下一步对方如何落子，以及该如何应对，算法思路与详情见[2.5.3 搜索算法](#_2.5.3  搜索算法)。

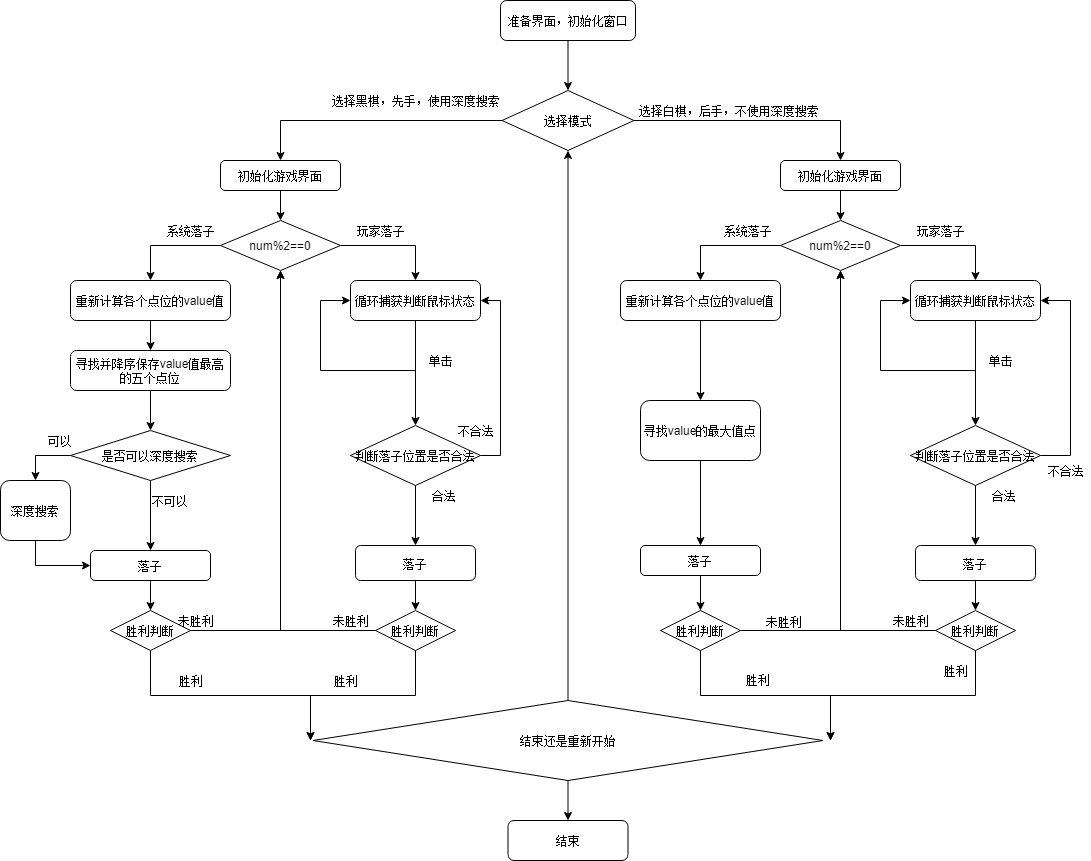


图 2 系统运行流程图

## 2.5 关键点介绍

### 2.5.1 五元组

在游戏中，对于连续的五个点位，可以称其为一个五元组。五元组共有四种方向，即左右方向，上下方向，左下-右上方向，左上-右下方向。棋局中的每一个点位最多属于20个五元组（边界位置的点位所属的五元组少于20），这些五元组可以按方向分为4组。

如果系统想要取得胜利，则必须有一个五元组中含有且只能含有五颗己方颜色的棋子。对于任意一方而言，当一个五元组出现了对方的棋子，那么自己将无法在该五元组中取得胜利。此时，这种五元组可以称其为无效五元组。同理，如果一个五元组中只含有一方的棋子，那么这种五元组可以称为有效五元组。

五元组作为最基础的评分估值单位，系统对于每一个有效五元组，都将根据其中包含的棋子的数量及敌我双方的身份来给其估价，估价方法详见[2.5.2 估值评分](#_2.5.2  估值评分。)。

五元组也是胜利判定的单位。如果系统判定某一方获得胜利，则必定是该方拥有了一个含有五颗己方棋子的五元组。

### 2.5.2 估值评分。

这个系统中共有两个估值函数，一个是评价一个点的价值的点位估价函数value\_one（），一个是评价整个棋局的棋局估价函数value\_whole（）。

在点位估价函数中，系统把需要评分估价的点的五元组按照方向分为四组，每组单独求和，存放在数组value[]中。所以有value\_one\_end = value[0] + vlaue[1] + value[2] + value[3];

在任意一个方向中，其对应的value值等于这个点位在该方向上的所有有估价的五元组的估值之和。

对于每一个有效五元组，系统都应该按照敌方或己方棋子数量给其一个估价。而对于无效五元组，己方虽然不能在该五元组上取得胜利，但是却可以通过落子去阻止对方在该五元组中取得胜利，因此它对己们而言也是有具有价值的。所以对任一有效五元组而言，系统都要对该五元组进行估价。对于无效五元组，因其对敌我双方都毫无价值，所以不予估价。

而且，在合理范围内设计者可以通过调节估价来使系统变得偏向进攻或这偏向防御，但就一般而言，估价依据则是：系统组成五连的价值>阻止敌方组成五连的价值>系统组成四连>阻止敌方组成四连>系统组成三连>阻止敌方组成三连。该系统使用的函数的估价表如表1 点位估价函数估价表所示：

表 1 点位估价函数估价表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 棋子数 | 0 | 1 | 2 | 3 | 4 |
| 己方 | 1 | 10 | 100 | 1000 | 100000 |
| 敌方 | 0 | 5 | 50 | 500 | 5000 |

而在棋局估价函数中，函数得到的value值表明了棋局对己方而言的优劣。使用该函数并进行深度搜索的目的是为了避免系统通过点位估价函数寻找到的当下最优解为敌方的陷阱。

在该函数中，系统通过将每一个有效五元组估值之和减去每一个无效五元组估值之和所得的结果作为评价当下整个棋局对系统方的优劣。

很明显，如果棋局中有更多的四连或者三连则更容易胜利。

在value函数中，系统将整个棋局的价值进行评估。这里函数用己方存在的有效五元组价值之和减去敌方的有效五元组价值之和得到的结果作为标准来评价棋局。即扫描棋局，寻找其中的有效的三连四连（指有效五元组内的三连或者四连）。因为是评价当下整个棋局的对己方而言的优劣，所以这里的估价标准变为了：五连价值>所有四连的价值，一个四连的价值>多个三连价值。该系统中使用的估价表如表2 棋局估价函数估价表所示：

表 2 棋局估价函数的估价表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 棋子数 | 3 | 4 | 5 | 其他 |
| 己方 | 1 | 100 | 100000 | 0 |
| 敌方 | 1 | 100 | 100000 | 0 |

### 2.5.3 搜索算法

在不进行深度搜索的情况下，算法思路很简单，只需要将没有落子的位置遍历一次分别计算该位置的价值。然后寻找其中的value值最大值点位并在此落子就可以了。

在进行深度搜索的情况下，算法则会变得比较复杂、耗时。为了做到节省时间、快速响应，在这里系统只进行3层搜索。

算法运行的流程图如图2所示。

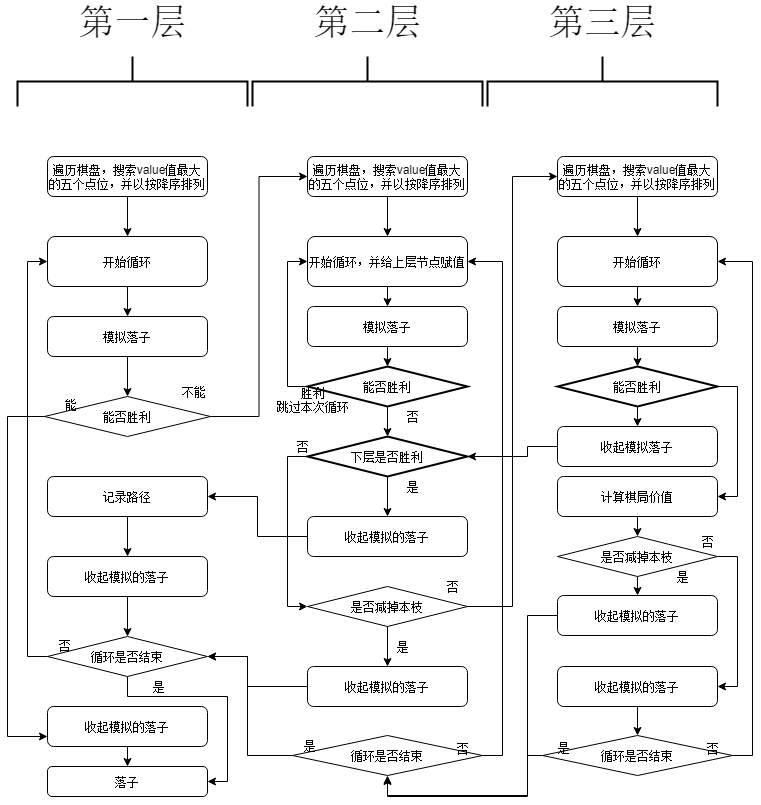


图 3 深度搜索算法运行流程图

算法会优先进行深度搜索。第一层搜索的作用是模拟系统落子，如果可以取得胜利则直接收起模拟的落子，然后在该位置落子取得胜利。如果不能取得胜利，则进行第二层模拟。第二层搜索的作用是模拟棋手落子，如果棋手可以取得胜利，那么减掉这一枝。如果不能取得胜利，则进行第三层模拟。第三层搜索的作用是模拟系统落子，然后寻找最优势的棋局。

由于多层的搜索生成的博弈树有着大量的子节点，所以在其中嵌入了剪枝算法来减少运算，节约时间。剪枝算法详见[2.5.4 剪枝](#_2.5.4  剪枝)。

这棵由深度搜索生成的博弈树，其父节点的值为符合要求的子节点的极值。算法会在第三层中根据棋局评估函数计算得到一个value。那么第二层的节点值为在第三层中各自的子节点中的最大值。第一层的节点值为第二层他们各自的子节点中的极小值。假如博弈树如图3所示，则B=MIN(D,E);D=MAX(H,I);

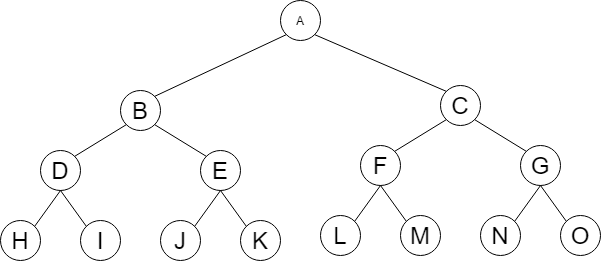


图 4 博弈树示例

### 2.5.4 剪枝算法

算法生成的博弈树有着众多的子节点，如果一项一项计算完全则会耗费大量时间，并且这些子节点中只有一个是系统真正需要的。

为了加快计算速度，就需要减少冗余计算。由于这些节点中第一层、第三层取的是最大值，第二层取最小值，所以这里使用α-β剪枝来缩减搜索范围。Alpha-beta剪枝是一种搜索[算法](https://baike.baidu.com/item/%E7%AE%97%E6%B3%95" \t "https://baike.baidu.com/item/Alpha-beta%E5%89%AA%E6%9E%9D/_blank)，用以减少极小化极大算法（Minimax算法）搜索树的节点数。其原理为通过剪掉博弈树的冗余枝杈从而减少系统开销，节约时间。

在搜索算法中，使用了深度优先的搜索方法。如图3，D节点会比J、K等更早获得赋值，B也会比F、G等节点更早获得赋值。

那么，假设通过计算得到D=1，J=5。那么由于E=MAX(J,K);即E>=J。所以有D<E。又因为B=MIN(D,E)。所以B！=E。也就是说E和他的子节点为冗余数据，这样就可以直接将其减掉不计算。同理如果B=5,F=1。所以C=MIN(F,G)，A=MAX(b,c)。由于B>F,所以B>C。所以C及他的子节点也将会是冗余数据，可以直接剪掉。

剪枝算法的代码大概描述如下：

for(x in value\_1){ //遍历第一层，vlaue=max（value\_1）。

for(y in vlaue\_2){ //遍历第二层,value=min(value\_2)。

for(z in vlaue\_3){ //遍历第三层,value=max(value\_3)。

if(z>y) break; //即上文中D<J的情况。

}

if(y<x) break; //即上文中B>F的情况。

}

if(value<x)value=x; //value=max(x)。

}

### 2.5.5 胜利判定

系统中共在四种种情况下使用了胜利判定。

前两种都未发生在深度搜索中。

第一种是棋手一方落完子后判断是否棋手胜利。这里使用的胜利判定函数会以运行棋手落子函数时使用的坐标(a,b)。以这一点为中心，想五元组的四个方向逐个判断是否胜利。这里以判断在左右方向上是否胜利为例。函数会先寻找连子的左右边界，然后计算两边界之差，这个差值再加一就是这串连子的长度。如果这个长度超过5，那么就说明棋手在这个方向上取得了胜利。示例代码如下（以判断左右方向为例，其他方向同理。）：

int judge\_lr(int x,int y,int color){ //仅判断在左右方向上是否胜利。

x0=x;y0=y; //记录原本点位

x1,y1=search\_left\_uncolor(x,y); //寻找左边界，边界为地方棋子或空

x=x0;y=y0; //复位

x2,y2=search\_right\_uncolor(x,y); //寻找右边界，边界为地方棋子或空

x=x0;y=y0; //复位

if((x2-x1+1)>=5) return win; //判断是否胜利

}

第二种是系统落完子后的胜利判定。但是系统在落子则是依据点位估价函数来判断的，在这个函数中，系统会搜索每个点位所属的五元组，如果发现他的某个五元组里已经有四个己方棋子，那么程序就可以直接在这里落子并停止继续搜索，然后直接取得胜利。这样不仅可以直接省掉一次单独的胜利判定，还省掉了一些冗余的搜索。进一步加快了程序的运行速度。

以判断在上下方向上是否胜利为例。

x0=x;y0=y; //记录原本点位

while(color(x,y)==uncolor&&y-1>=0)y--; //寻找上方边界，边界为敌方棋子

x1=x;y1=y;x=x0;y=y0; //记录、复位

while(color(x,y)==uncolor&&y+1<=14)y++; //寻找下方边界，边界为敌方棋子

x2=x;y2=y;x=x0;y=y0; //记录、复位

for(y=y1;y<y2-4;y++){ //遍历两边界内的每一个五元组

num=0; //用以记录五元组内己方棋子的数量。

for(l=0;l<5;l++){ //遍历无元组内部

if(color(x,y)==color)num++; //记录数量

}

if(num==4){ //某个五元组中已经有了四个落子

down(x0,y0,color); //落子

win\_color; //取得胜利

break;

}

value+=value\_num(num); //在无法取得胜利时记算该点的价值。

}

其中点位估价计算的方法详见[2.5.2 估价评分](#_2.5.2  估值评分。)中的点位估价函数介绍。

第三种和第四种发生在深度搜索中。

第三中则是在深度搜索的过程中，如果发现棋手一方将会获胜。那么这种落子方法必然不是程序最终要找的方法。所以直接需要跳出循环，将其父节点减去。这里以判断在左上-右下方向上是否胜利为例。

x0=x;y0=y; //保存原本点位

x1,y1=search\_lu\_uncolor(x,y); //寻找左上方边界

x=x0;y=y0; //复位

x2,y2=search\_rd\_uncolor(x,y); //寻找右下方边界

x=x0;y=y0; //复位

if((x2-x1+1)>=5) { //如果胜利

seat\_1\_max[seat\_1\_i];=-100000000; /\*给其父节点赋一个特别小的值，表示 非常不推荐系统在其父节点代表的位置落子。\*/

break; //跳出循环，防止父节点被再次覆盖

}

第四种则是在模拟中发现系统可以取得胜利，那么就记录下这个路径，然后跳出循环，在此落子。

循环遍历前：

win\_flag=0; 初始化 win\_flag

第一层中：

x=seat\_1\_max[seat\_1\_i].x;

y=seat\_1\_max[seat\_1\_i].y;

if(flag\_win==1){down\_color(x,y,color);black;} //如果收到下层传来的消息则落子并跳出循环。，

第二层中：

if(flag\_win==1){load=seat\_1\_i;black;} //如果收到下层传来的消息则记录路径，跳出循环。

第三层中：

int judge=judge\_lu\_rd(int x,int y,int color); //判断是否能取得胜利，其判断思路与第一种情况类似

if(judge==win){win\_flag=1;black;} //如果胜利则给win\_flag赋值。并跳出循环。

else{value=value\_whole(int color);} //未取得胜利则计算棋局价值

# 3 系统测试

系统编写完成之后，还需要进行测试，以此来判断系统是否达到既定目标，是否出现错误等。

## 3.1基本功能测试

### 3.1.1 测试界面运行

目标：测试开始、选择模式、游戏、是否结束、结束等界面是否可以正常运行。

时间：2022年4月2日

结果：可以正常运行，字体变色、页面跳转、落子位置、页面变淡效果、结束游戏等功能可以正常运行。

部分截图：

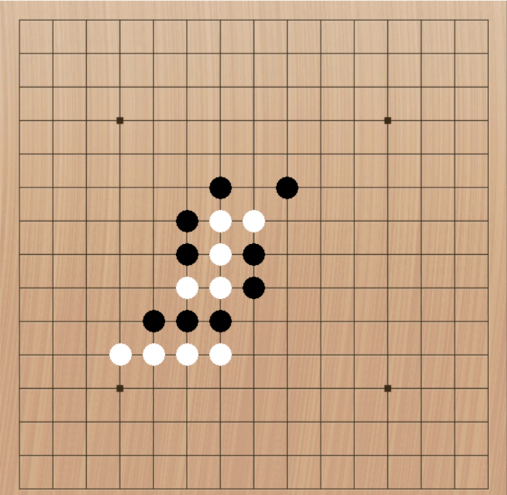


图 5 系统选择模式界面 图 6 游戏过程



图 7 页面变淡效果

## 3.2系统不同模式智能性测试

### 3.2.1 选择黑棋

黑棋先手，机器方不进行深度搜索。

一般业余玩家已经很难取得胜利，但是落子方法较为僵硬，偶尔还会出现可以让人一眼看出想要在那个五元组中取得胜利的落子。但是即便如此，通过凭借对各个点的估价比较，选出最优解。系统落子依旧算得上迅速、高效，一般人依旧很难招架，属于以力压人。

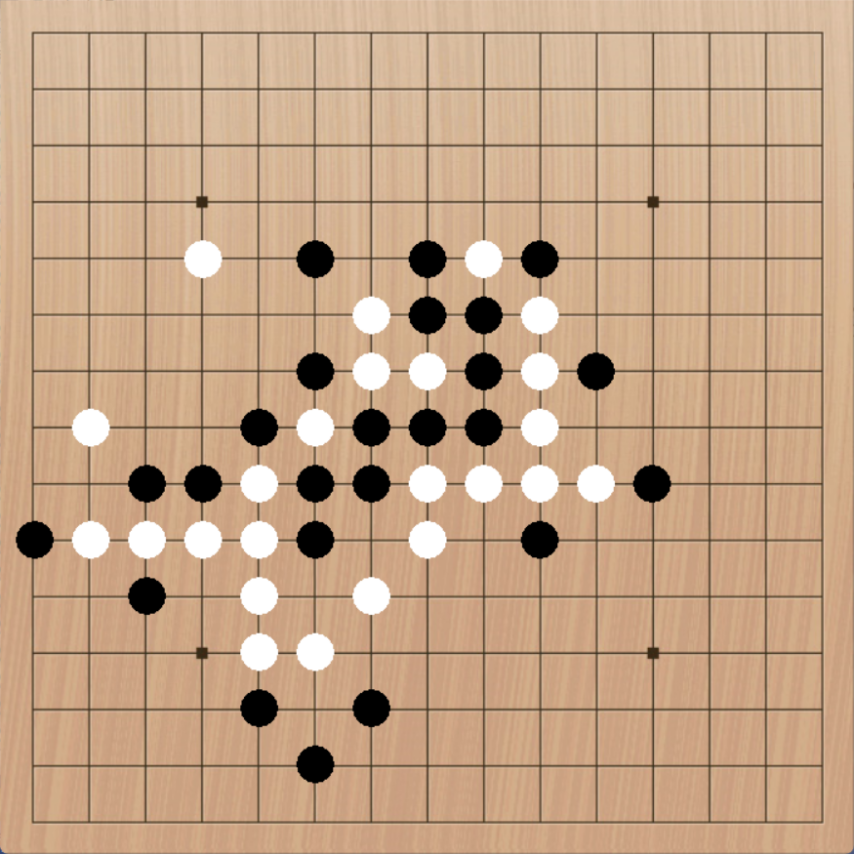


图 8 选择黑棋的游戏截图

### 3.2 选择白棋

白棋后手，机器方进行深度搜索。

对广大业余的业余选手来讲，单就胜率来讲，虽有提高，但是一百局里赢三次还是两次带来的体验差别并不大。但是可以明显感受到的是，系统的落子更加“拟人化”。每一次落子都好像有极多后手，会让人在不知不觉中尽失先机。缺点是，有的时候会为了更长远的布局，而没有选择对当下而言更好的选择。

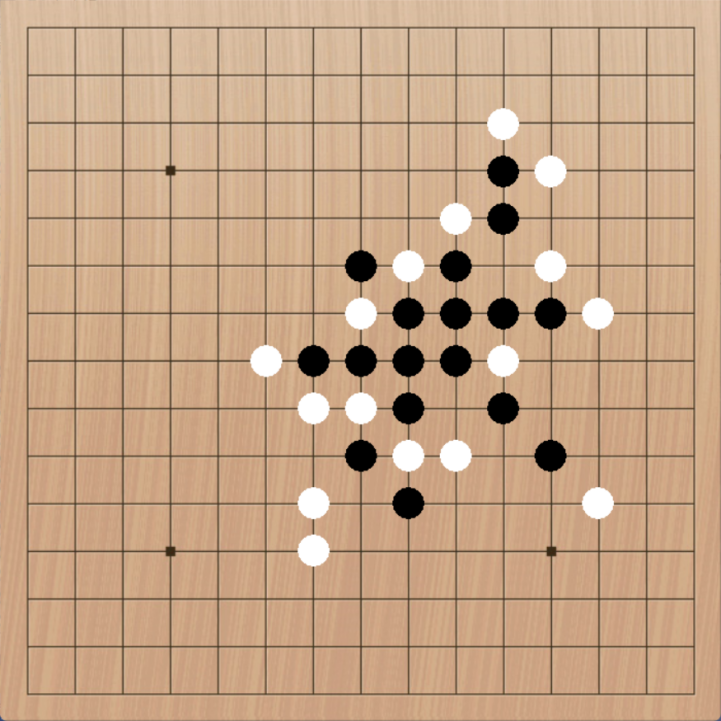


图 9 选择白棋的游戏过程截图

## 3.3 测试结论

在玩家棋力不是很高的情况下，是否进行深度搜索对于玩家来讲并不是很重要。只要评分表做的较为合理，在大部分情况下，大众就很难下赢机器。但是如果多玩几局就会发现这种算法的落子习惯。系统落子的表现会逐渐显得套路化，虽然依旧很难下赢，但是会很快失去乐趣。而使用深度搜索算法进行的机器方，套路化表现并不明显，落子布局更加长远，但是偶尔却会陷入“想的太多”的误区，使得落子布局虽然更加长远了却失去了对当下而言最有利的选择，从而出现一种舍近求远的现象。如果想要再进一步或许需要研究更加科学的棋局价值评估算法。

# 4 总结

1. 该程序的目的是实现在个人计算机上的能快速反应的可以人机对弈的五子棋游戏。
2. 为了节约时间，系统使用C/C++语言书写。
3. 该程序以是否进行深度搜索为区分，共设有两个模式，以便对比。
4. 系统使用五元组作为最基本的计算单位。
5. 系统的两个模式都可以实现目的，在面对广大的业余玩家时，都可以拥有不错的胜率。在运行速度上，两个模式相差不大，不进行深度搜索的话速度略高。
6. 在不进行深度搜索的模式中，如果多玩几局就很容易看出一些系统的落子规律，容易腻，而且如果设下陷阱，系统也不会规避。在进行深度搜索的情况下。
7. 在进行深度搜索的模式中，深度搜索会使得程序表现得更加“人性化”，会出现一些不落在最优解上的“错误”，但是相较于上个模式可以多规避一些“陷阱”。

**参考文献**

[1]陈树彬; 和昱旻; 原菊梅 五子棋落子算法的研究[J] 电脑与信息技术 2021-09-30

[2]Stanley B. Lippman . C++ Primer（第五版）[M]. 电子工业出版社 2015.

[3]周洋; 邓莉 一种五子棋博弈算法的分析 [J] 谢煜 现代计算机(专业版) 2017-04-05

[4]欧俊臣; 沙玲; 杨淞文 基于 MCTS 和卷积神经网络的五子棋策略研究[J] 软件 2020-04-15

[5]刘瑞 五子棋人工智能算法设计与实现[DB] 华南理工大学 2012-05-01

[6]牛恺泽, 邓鑫. 五子棋人工智能研究与实践[J]. 数字通信世界, 2019(01): 32-33.

[7]孙世文.五子棋人工智能算法实现研究[J].中国新通信,2018,20(23):143.

[8]宋万洋.基于α-β剪枝树算法的安卓五子棋程序设计与实现[J].现代信息科技,2019,3(11):92-93+97.

[9]许南山,丛磊,孙风平.并行实现有自学习能力的五子棋AI[J].计算机工程与应用,2006(30):45-47.

[10]王长飞,蔡强,李海生.智能五子棋算法的设计实现[J].系统仿真学报,2009,21(04):1051-1054。

[11]董慧颖,王杨.多种搜索算法的五子棋博弈算法研究[J].沈阳理工大学学报,2017,36(02):39-43+83。

[12]Silver D,Huang A,Maddison C J,et al.Mastering the game of Go with deep neural networks and tree search[J].Nature,2016,529(7587):484-489.

[13]Silver D,Schrittwieser J,Simonyan K,et al. Mastering the game of Go without human knowledge[J].Nature,2017,550(7676):354-359

[14]Silver D,Hubert T,Schrittwieser J,et al.Mastering chess and shogi by self-play with a general reinforcement learning algorithm[EB/OL].arXiv preprint arXiv:1712.01815,2017.

致谢

大学毕业并不代表学习的结束。这毕业设计让我明显的认识到“书到用时方恨少”。感谢老师在我迷茫的时候给我的指导，让我可以重新确立方向，继续前行。感谢舍友们的帮忙，帮我指出了系统编写中犯下的错误，让我少走弯路。还要感谢父母给我安慰，让我静下心来，专心致志。也要感谢这些天遇到的困难，你让我认识到了自己的浅薄，让我明白了终身学习的必要。人这一生啊，总会不停地遇到不会的，不了解的东西。这就需要我们不断地去学习，就像这篇论文一样，在学习中成长，在学习中进步。

在此衷心的感谢这些帮助我的人，正是你们的支持，才有了我的进步，谢谢！