五子棋人机博弈问题实验报告

装

订

线

目录

装 订 线

1.实验概述

1.1 实验目的

熟悉和掌握博弈树的启发式搜索过程、α－β剪枝算法和评价函数，并利用α－β剪枝算法开发一个五子棋人机博弈游戏。

1.2 实验内容

1）以五子棋人机博弈问题为例，实现α－β剪枝算法的求解程序（编程语言不限），要求设计适合五子棋博弈的评估函数。

2）要求初始界面显示15*15的空白棋盘，电脑执白棋，人执黑棋，界面置有重新开始、悔棋等操作。

3）设计五子棋程序的数据结构，具有评估棋势、选择落子、判断胜负等功能。

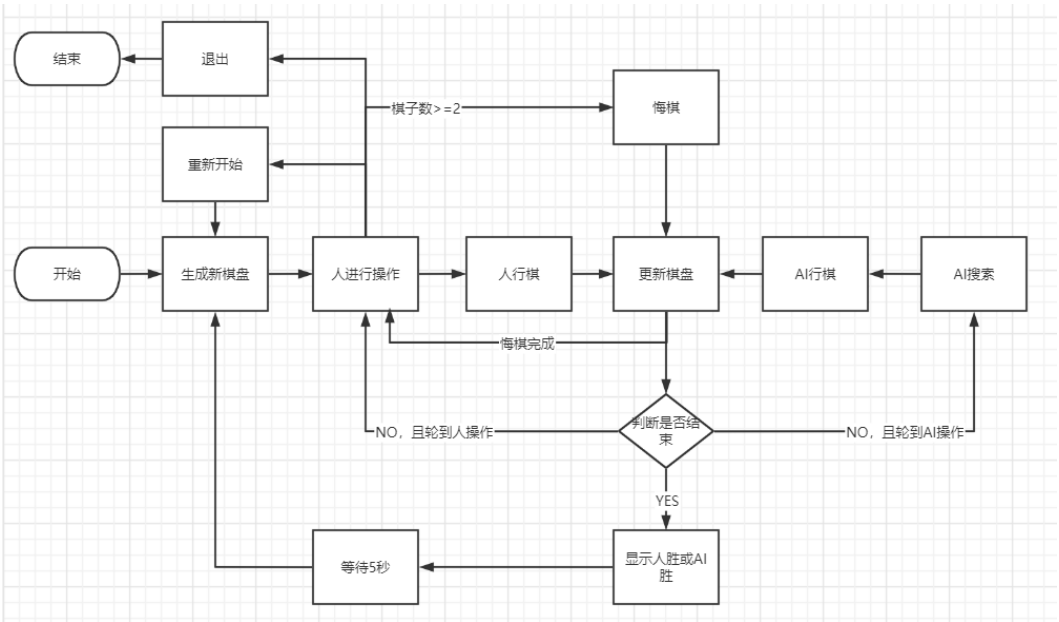
2.实验方案设计

2.1 总体设计思路与总体架构

实验首先生成新棋盘，然后人进行操作，操作一重新开始，重新生成新棋盘；操作二退出，游戏结束；操作三悔棋（在棋盘上棋子数>=2时），更新棋盘，继续人进行操作；操作四行棋，更新棋盘，判断是否结束；若结束，显示人胜，等待5秒，重新生成新棋盘；若未结束，则转入AI搜索，搜索完成后AI行棋，然后更新棋盘，判断是否结束；若结束显示AI胜，等待5秒，重新生成新棋盘；否则，转由人操作。

由上述思路建立总体架构，共六大模块，分别为初始化，游戏，搜索，更新棋盘，判断结束，悔棋。

流程图如下：



2.2 核心算法及基本原理

核心算法为极大极小搜索和α－β剪枝。

极大极小搜索，其思路为把五子棋每一步的走法展开，就是一颗巨大的博弈树。在这个树中，从根节点开始，奇数层表示玩家可能的走法，偶数层表示电脑可能的走法。有了对博弈树的基本认识，我们就可以用递归来遍历这一棵树。

那么我们如何才能知道哪一个分支的走法是最优的，我们就需要一个评估函数能评估当前整个局势，返回一个分数。我们规定对电脑越有利，分数越大，对玩家越有利，分数越小。电脑走棋的层我们称为 MAX 层，这一层电脑要保证自己利益最大化，那么就需要选分最高的节点。玩家走棋的层我们称为 MIN 层，这一层玩家要保证自己的利益最大化，那么就会选分最低的节点。

Alpha Beta 剪枝算法是一种安全的剪枝策略，也就是不会对棋力产生任何负面影响。它的基本依据是：棋手不会做出对自己不利的选择。依据这个前提，如果一个节点明显是不利于自己的节点，那么就可以直接剪掉这个节点。

在 MAX 层，假设当前层已经搜索到一个最大值 X， 如果发现下一个节点的下一层（也就是 MIN 层）会产生一个比 X 还小的值，那么就直接剪掉此节点。通俗点来讲就是，AI 发现这一步是对玩家更有利的，那么当然不会走这一步。在 MIN 层，假设当前层已经搜索到一个最小值 Y， 如果发现下一个节点的下一层（也就是 MAX 层）会产生一个比 Y 还大的值，那么就直接剪掉此节点。这个是一样的道理，如果玩家走了一步棋，发现其实对 AI 更有利，玩家必定不会走这一步。

2.3 模块设计

本实验共有六大模块分别为初始化，游戏，搜索，更新棋盘，判断结束，悔棋。

初始化模块初始化棋盘和全局变量、评估函数六元组（八个方向长度为六的棋子排列）等。

游戏模块进行对局游戏。

搜索模块进行计算杀棋搜索、迭代加深的极大极小搜索，实现 AI 决策。

更新棋盘模块在棋盘上把棋子展示出来。

判断结束模块判断对局是否结束。

悔棋模块实现悔棋。

2.4 其他创新内容或优化算法

1）采用局部搜索。只考虑那些能和棋子产生关系的空位置，而不用考虑所有空位置，这样能极大减小分支数。我对于棋盘上棋子周围深度为 3 的区域进行搜索，认为这些地方与现有棋子有联系。

2）采用静态评价启发。因为如果越早搜索到较优走法，剪枝就会越早发生。如果对可走节点的评估分数进行简单排序，就可以提高剪枝速度。我根据局部搜索得到的所有可走节点的分数放入优先队列，然后转入数组（针对 MAX 层），或者先转入栈，再转入数组（针对 MIN 层），用于之后的搜索。

3）迭代加深搜索。首先搜索 2 层，然后逐步增加搜索深度，直到找到胜利走法或者达到深

度限制为止。比如我们搜索 4 层深度，那么我们先尝试 2 层，如果没有找到能赢的走法，再尝试 4 层。我们只尝试偶数层。因为在奇数层，其实是电脑比玩家多走了一步，忽视了玩家的防守，并不会额外找到更好的解法。

4）计算杀棋。杀棋就是指一方通过连续的活三和冲四进行进攻，一直到赢的一种走法。一般，我们优先进行成五和冲四，没有找到结果的时候再进行活三。很显然，同样的深度，计算杀棋要比前面讲的搜索效率高很多。因为计算杀棋的情况下，每个节点只计算活三和冲四和成五的子节点。搜索只能进行 4 层，而计算杀棋很多时候可以进行到 12 层以上。这其实也是一种极大极小值搜索，具体的策略是这样的：

MAX 层，只搜索己方的活三和冲四和成五节点，选择最优的行棋。

MIN 层，选择最优落子点即可。

3.实验过程

3.1 环境说明

操作系统：Windows 10

开发语言：C++

开发环境：Visual Studio 2019

核心使用库：C++ STL 库、EasyX 库

3.2 源代码文件清单，主要函数清单

代码清单：

main.cpp：主函数所在源文件，用于定义一系列全局变量,调用初始化函数和游戏函数。

init.cpp：初始化源文件，用于初始化棋盘和全局变量、评估函数六元组等。

game.cpp：游戏源文件，用于进行对局，包括人和 AI 的各种操作，如落子、悔棋、判断结束、退出、更新棋盘等。

core.cpp：核心算法源文件，用于计算杀棋搜索、迭代加深的极大极小搜索。

all.h：头文件，包含程序所需头文件、宏定义、函数声明、全局变量声明、结构体声明。

主要函数清单：

main 函数：调用初始化函数和游戏函数

init_tuple6type 函数：初始化评估函数六元组

init 函数：初始化棋盘和全局变量等

isend 函数：判断对局结束

retract 函数：悔棋

evaluate 函数：评估函数

seekPoints 函数：局部搜索

maxmin 函数：极大极小搜索

seek_kill_points 函数：算杀点搜索

analyse_kill 函数：算杀搜索

game 函数：对局函数

装 订 线

3.3 源代码

all.h：

```
#pragma once
#include <iostream>
#include <ctime>
#include <stdlib.h>
#include <graphics.h>
#include<queue>
#include<stack>
#include <map>
using namespace std;
#define MAX 100000000//极大极小算法的初始化
#define board_length 15//棋盘长度
#define search_deep 3//局部搜索的深度
#define kill_deep 16//算杀搜索的深度
#define maxmin_deep 4//极大极小搜索的深度
#define C_NONE 0//棋子：黑子,白子,无子
#define C_BLACK 1
#define C_WHITE 2
//棋型代号 下标 权重
#define OTHER 0//0,其他棋型不考虑
#define WIN 1//10000000,白赢
#define LOSE 2//-10000000
#define FLEX4 3//10000,白活4
#define flex4 4//-100000
#define BLOCK4 5//400
#define block4 6//-100000
#define FLEX3 7//400
#define flex3 8//-10000
#define BLOCK3 9//20
#define block3 10//-40
#define FLEX2 11//20
#define flex2 12//-40
#define BLOCK2 13//1
#define block2 14//-2
#define FLEX1 15//1
#define flex1 16//-2
//八个方向
extern int dx[8];
extern int dy[8];
extern int whowin;//0为人，1为AI
extern int whoplay;//0为人，1为AI
extern int allstep;//总步数
extern int tuple6type[4][4][4][4][4][4];//棋型辨识数组,0无子,1黑子,2白子,3边界
//评估
struct EVALUATION
```

```
{
    int score;
    int result;//算杀判断输赢
    int STAT[8];//储存部分棋形的个数(杀棋棋型),下标WIN=1为白连5,LOSE=2为黑连5,FLEX4=3为白活4,BLOCK4=5为白冲4,FLEX3=7为白活3
};
//落子
struct pos
{
    int x;
    int y;
    int worth;//评估值
    bool operator<(const pos& b)const
    {
        return worth < b.worth;
    }
};
extern priority_queue<pos> que, kill;//优先队列用于算杀和极大极小搜索
extern pos s[255];
//棋盘
struct BOX
{
    int x;
    int y;
    int value;//0为人，1为AI
    int isnew;//是否滑到新框
    void draw();//画出棋子
};
extern BOX box[board_length][board_length];

void init_tuple6type();//初始化评估六元组
void init();//初始化棋盘和全局变量等
bool isend();//判断对局结束
void retract(int i, int j);//悔棋
EVALUATION evaluate(int board[board_length][board_length]);//评估函数
pos seekPoints(int board[board_length][board_length], int who, int deep, bool choice);//局部搜索
int maxmin(int board[board_length][board_length], int deep, int alpha, int beta, pos& ans);//极大极小搜索
void seek_kill_points(int board[board_length][board_length]);//算杀点搜索
bool analyse_kill(int board[board_length][board_length], int deep, int who, pos& ans);//算杀搜索
void game();//对局
```

main.cpp：

```
#include"all.h"
int dx[8] = { 1,0,1,1,-1,0,-1,-1 };
int dy[8] = { 0,1,1,-1,0,-1,-1,1 };
int whowin = -1;//0为人，1为AI
int whoplay = -1;//0为人，1为AI
int allstep = 0;//总步数
int tuple6type[4][4][4][4][4][4];//棋型辨识数组,0无子,1黑子,2白子,3边界
priority_queue<pos> que, kill;//优先队列用于算杀和极大极小搜索
```

```
pos s[255] = { 0 };//棋盘落子全记录
BOX box[board_length][board_length];
int main()
{
    init_tuple6type();//初始化评估六元组
    while (1)//游戏
    {
        init();//初始化棋盘和全局变量等
        game();//对局
    }
    Sleep(5000);//暂停5秒
    closegraph();//关闭绘图窗口
    return 0;
}
```

init.cpp：

```
#include"all.h"
void init_tuple6type()//初始化评估六元组
{
    memset(tuple6type, 0, sizeof(tuple6type));//全部设为0
    //白连5,ai赢
    tuple6type[2][2][2][2][2][2] = WIN;
    tuple6type[2][2][2][2][2][0] = WIN;
    tuple6type[0][2][2][2][2][2] = WIN;
    tuple6type[2][2][2][2][2][1] = WIN;
    tuple6type[1][2][2][2][2][2] = WIN;
    tuple6type[3][2][2][2][2][2] = WIN;//边界考虑
    tuple6type[2][2][2][2][2][3] = WIN;
    //黑连5,ai输
    tuple6type[1][1][1][1][1][1] = LOSE;
    tuple6type[1][1][1][1][1][0] = LOSE;
    tuple6type[0][1][1][1][1][1] = LOSE;
    tuple6type[1][1][1][1][1][2] = LOSE;
    tuple6type[2][1][1][1][1][1] = LOSE;
    tuple6type[3][1][1][1][1][1] = LOSE;
    tuple6type[1][1][1][1][1][3] = LOSE;
    //白活4
    tuple6type[0][2][2][2][2][0] = FLEX4;
    //黑活4
    tuple6type[0][1][1][1][1][0] = flex4;
    //白活3
    tuple6type[0][2][2][2][0][0] = FLEX3;
    tuple6type[0][0][2][2][2][0] = FLEX3;
    tuple6type[0][2][0][2][2][0] = FLEX3;
    tuple6type[0][2][2][0][2][0] = FLEX3;
    //黑活3
    tuple6type[0][1][1][1][0][0] = flex3;
    tuple6type[0][0][1][1][1][0] = flex3;
    tuple6type[0][1][0][1][1][0] = flex3;
    tuple6type[0][1][1][0][1][0] = flex3;
    //白活2
    tuple6type[0][2][2][0][0][0] = FLEX2;
```

```
    tuple6type[0][2][0][2][0][0] = FLEX2;
    tuple6type[0][2][0][0][2][0] = FLEX2;
    tuple6type[0][0][2][2][0][0] = FLEX2;
    tuple6type[0][0][2][0][2][0] = FLEX2;
    tuple6type[0][0][0][2][2][0] = FLEX2;
    //黑活2
    tuple6type[0][1][1][0][0][0] = flex2;
    tuple6type[0][1][0][1][0][0] = flex2;
    tuple6type[0][1][0][0][1][0] = flex2;
    tuple6type[0][0][1][1][0][0] = flex2;
    tuple6type[0][0][1][0][1][0] = flex2;
    tuple6type[0][0][0][1][1][0] = flex2;
    //白活1
    tuple6type[0][2][0][0][0][0] = FLEX1;
    tuple6type[0][0][2][0][0][0] = FLEX1;
    tuple6type[0][0][0][2][0][0] = FLEX1;
    tuple6type[0][0][0][0][2][0] = FLEX1;
    //黑活1
    tuple6type[0][1][0][0][0][0] = flex1;
    tuple6type[0][0][1][0][0][0] = flex1;
    tuple6type[0][0][0][1][0][0] = flex1;
    tuple6type[0][0][0][0][1][0] = flex1;

    int p1, p2, p3, p4, p5, p6, x, y, ix, iy;//x:左5中黑个数,y:左5中白个数,ix:右5中黑个数,iy:右5中白个数
    for (p1 = 0; p1 < 4; ++p1) {
        for (p2 = 0; p2 < 3; ++p2) {
            for (p3 = 0; p3 < 3; ++p3) {
                for (p4 = 0; p4 < 3; ++p4) {
                    for (p5 = 0; p5 < 3; ++p5) {
                        for (p6 = 0; p6 < 4; ++p6) {
                            x = y = ix = iy = 0;

                            if (p1 == 1)x++;
                            else if (p1 == 2)y++;

                            if (p2 == 1) { x++; ix++; }
                            else if (p2 == 2) { y++; iy++; }

                            if (p3 == 1) { x++; ix++; }
                            else if (p3 == 2) { y++; iy++; }

                            if (p4 == 1) { x++; ix++; }
                            else if (p4 == 2) { y++; iy++; }

                            if (p5 == 1) { x++; ix++; }
                            else if (p5 == 2) { y++; iy++; }

                            if (p6 == 1)ix++;
                            else if (p6 == 2)iy++;

                            if (p1 == 3 || p6 == 3) {//有边界
                                if (p1 == 3 && p6 != 3) {//左边界
```

```
                    //白冲4
                    if (ix == 0 && iy == 4) {//若右边有空位是活4也没关系，因为活4权重远大于冲4，再加上冲4权重变化可以不计
                        if (tuple6type[p1][p2][p3][p4][p5][p6] == 0)
                            tuple6type[p1][p2][p3][p4][p5][p6] = BLOCK4;
                    }
                    //黑冲4
                    if (ix == 4 && iy == 0) {
                        if (tuple6type[p1][p2][p3][p4][p5][p6] == 0)
                            tuple6type[p1][p2][p3][p4][p5][p6] = block4;
                    }
                    //白眠3
                    if (ix == 0 && iy == 3) {
                        if (tuple6type[p1][p2][p3][p4][p5][p6] == 0)
                            tuple6type[p1][p2][p3][p4][p5][p6] = BLOCK3;
                    }
                    //黑眠3
                    if (ix == 3 && iy == 0) {
                        if (tuple6type[p1][p2][p3][p4][p5][p6] == 0)
                            tuple6type[p1][p2][p3][p4][p5][p6] = block3;
                    }
                    //白眠2
                    if (ix == 0 && iy == 2) {
                        if (tuple6type[p1][p2][p3][p4][p5][p6] == 0)
                            tuple6type[p1][p2][p3][p4][p5][p6] = BLOCK2;
                    }
                    //黑眠2
                    if (ix == 2 && iy == 0) {
                        if (tuple6type[p1][p2][p3][p4][p5][p6] == 0)
                            tuple6type[p1][p2][p3][p4][p5][p6] = block2;
                    }
                }
                else if (p6 == 3 && p1 != 3) {//右边界
                    //白冲4
                    if (x == 0 && y == 4) {
                        if (tuple6type[p1][p2][p3][p4][p5][p6] == 0)
                            tuple6type[p1][p2][p3][p4][p5][p6] = BLOCK4;
                    }
                    //黑冲4
                    if (x == 4 && y == 0) {
                        if (tuple6type[p1][p2][p3][p4][p5][p6] == 0)
                            tuple6type[p1][p2][p3][p4][p5][p6] = block4;
                    }
                    //黑眠3
                    if (x == 3 && y == 0) {
                        if (tuple6type[p1][p2][p3][p4][p5][p6] == 0)
                            tuple6type[p1][p2][p3][p4][p5][p6] = BLOCK3;
                    }
                    //白眠3
                    if (x == 0 && y == 3) {
                        if (tuple6type[p1][p2][p3][p4][p5][p6] == 0)
                            tuple6type[p1][p2][p3][p4][p5][p6] = block3;
                    }
```

```
                                    //黑眠2
                                    if (x == 2 && y == 0) {
                                        if (tuple6type[p1][p2][p3][p4][p5][p6] == 0)
                                            tuple6type[p1][p2][p3][p4][p5][p6] = BLOCK2;
                                    }
                                    //白眠2
                                    if (x == 0 && y == 2) {
                                        if (tuple6type[p1][p2][p3][p4][p5][p6] == 0)
                                            tuple6type[p1][p2][p3][p4][p5][p6] = block2;
                                    }
                                }
                            }
                            else {//无边界
                                //白冲4
                                if ((x == 0 && y == 4) || (ix == 0 && iy == 4)) {
                                    if (tuple6type[p1][p2][p3][p4][p5][p6] == 0)
                                        tuple6type[p1][p2][p3][p4][p5][p6] = BLOCK4;
                                }
                                //黑冲4
                                if ((x == 4 && y == 0) || (ix == 4 && iy == 0)) {
                                    if (tuple6type[p1][p2][p3][p4][p5][p6] == 0)
                                        tuple6type[p1][p2][p3][p4][p5][p6] = block4;
                                }
                                //白眠3
                                if ((x == 0 && y == 3) || (ix == 0 && iy == 3)) {
                                    if (tuple6type[p1][p2][p3][p4][p5][p6] == 0)
                                        tuple6type[p1][p2][p3][p4][p5][p6] = BLOCK3;
                                }
                                //黑眠3
                                if ((x == 3 && y == 0) || (ix == 3 && iy == 0)) {
                                    if (tuple6type[p1][p2][p3][p4][p5][p6] == 0)
                                        tuple6type[p1][p2][p3][p4][p5][p6] = block3;
                                }
                                //白眠2
                                if ((x == 0 && y == 2) || (ix == 0 && iy == 2)) {
                                    if (tuple6type[p1][p2][p3][p4][p5][p6] == 0)
                                        tuple6type[p1][p2][p3][p4][p5][p6] = BLOCK2;
                                }
                                //黑眠2
                                if ((x == 2 && y == 0) || (ix == 2 && iy == 0)) {
                                    if (tuple6type[p1][p2][p3][p4][p5][p6] == 0)
                                        tuple6type[p1][p2][p3][p4][p5][p6] = block2;
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
void init()//初始化棋盘和全局变量等
{
```

```
    initgraph(450, 500);//初始化绘图环境
    setbkcolor(BLACK);
    cleardevice();
    setbkmode(TRANSPARENT);//设置透明文字输出背景
    IMAGE img;
    loadimage(&img, _T("2.jpg"));//载入棋盘图片
    putimage(0, 0, &img);
    LOGFONT f;
    gettextstyle(&f);
    f.lfHeight = 30;
    wcscpy_s(f.lfFaceName, L"微软雅黑 Light");
    f.lfQuality = ANTIALIASED_QUALITY;
    settextstyle(&f);
    settextcolor(WHITE);
    setfillcolor(RGB(100, 100, 100));
    solidrectangle(25, 460, 125, 490);//绘制边框
    solidrectangle(175, 460, 275, 490);
    solidrectangle(325, 460, 425, 490);
    outtextxy(27, 460, _T("重新开始"));
    outtextxy(200, 460, _T("退出"));
    outtextxy(350, 460, _T("悔棋"));
    //变量初始化
    whowin = -1;
    whoplay = 0;
    allstep = 0;
    memset(s, 0, sizeof(s));
    while (!que.empty())
        que.pop();
    while (!kill.empty())
        kill.pop();
    for (int i = 0; i < board_length; i++)
        for (int j = 0; j < board_length; j++)
        {
            box[i][j].value = -1;
            box[i][j].x = j * 30;
            box[i][j].y = i * 30;
        }

}
```

game.cpp：

```
#include"all.h"
void BOX::draw()//画出棋子
{
    COLORREF thefillcolor = getfillcolor();//备份填充颜色
    setlinestyle(PS_SOLID, 2);//线样式设置
    setwritemode(R2_XORPEN);//设置XOR绘图模式
    setlinecolor(LIGHTGRAY);
    if (whoplay == 0)
    {
        line(x + 1, y + 2, x + 8, y + 2);
        line(x + 2, y + 1, x + 2, y + 8);
```

```
        line(x + 29, y + 2, x + 22, y + 2);
        line(x + 29, y + 1, x + 29, y + 8);
        line(x + 2, y + 29, x + 8, y + 29);
        line(x + 2, y + 22, x + 2, y + 29);
        line(x + 29, y + 29, x + 22, y + 29);
        line(x + 29, y + 22, x + 29, y + 29);
    }
    setwritemode(R2_COPYPEN);
    if (value == 0)//黑棋
    {
        setfillcolor(BLACK);
        fillcircle(x + 15, y + 15, 13);
    }
    else if (value == 1)//白棋
    {
        setfillcolor(WHITE);
        fillcircle(x + 15, y + 15, 13);
    }
    setfillcolor(thefillcolor); //还原填充色
}
bool isend()//是否结束
{
    for (int i = 0; i < board_length; i++)//遍历每个可能的位置
    {
        for (int j = 0; j < board_length; j++)
        {
            if (box[i][j].value == whoplay)
            {
                int length[8] = { 0,0,0,0,0,0,0,0 };//八个方向的长度
                for (int k = 0; k < 8; k++)
                {
                    int nowi = i;
                    int nowj = j;
                    while (nowi <= 14 && nowj <= 14 && nowi >= 0 && nowj >= 0 &&
box[nowi][nowj].value == whoplay)
                    {
                        length[k]++;
                        nowj += dx[k];
                        nowi += dy[k];
                        if (length[k] >= 5)
                        {
                            return true;
                        }
                    }
                }
            }
        }
    }
    return false;
}
void retract(int i, int j)//悔棋
{
    if (box[i][j].value == 0)//黑棋
```

```
    {
        setfillcolor(BLACK);
        fillcircle(box[i][j].x + 15, box[i][j].y + 15, 13);
    }
    else if (box[i][j].value == 1)//白棋
    {
        setfillcolor(WHITE);
        fillcircle(box[i][j].x + 15, box[i][j].y + 15, 13);
    }
}
void game()//对局
{
    int oldi = -1, oldj = -1;//上一个鼠标停的坐标
    //box[oldi][oldj].draw();
    while (1)
    {
        bool retraction = false;//还没有悔棋
        if (whoplay == 0)// 玩家下棋
        {
            MOUSEMSG mouse = GetMouseMsg();//获取鼠标信息
            for (int i = 0; i < board_length; i++)
            {
                for (int j = 0; j < board_length; j++)
                {
                    //重新开始
                    if (mouse.x < 150 && mouse.x>0 && mouse.y < 500 && mouse.y> 450)
                    {
                        setfillcolor(RGB(150, 150, 150));
                        solidrectangle(25, 460, 125, 490);
                        outtextxy(27, 460, _T("重新开始"));
                        if (mouse.uMsg == WM_LBUTTONUP)
                        {
                            mouse.uMsg = WM_MOUSEMOVE;
                            return;
                        }
                    }
                    //退出
                    if (mouse.x < 300 && mouse.x>150 && mouse.y < 500 && mouse.y> 450)
                    {
                        setfillcolor(RGB(150, 150, 150));
                        solidrectangle(175, 460, 275, 490);
                        outtextxy(200, 460, _T("退出"));
                        if (mouse.uMsg == WM_LBUTTONUP)
                        {
                            mouse.uMsg = WM_MOUSEMOVE;
                            exit(0);
                        }
                    }
                    //悔棋
                    if (mouse.x < 450 && mouse.x>300 && mouse.y < 500 && mouse.y> 450 &&
allstep > 2)
                    {
                        setfillcolor(RGB(150, 150, 150));
```

```
                    solidrectangle(325, 460, 425, 490);
                    outtextxy(350, 460, _T("悔棋"));
                    if (mouse.uMsg == WM_LBUTTONUP)
                    {
                        mouse.uMsg = WM_MOUSEMOVE;
                        IMAGE img;//重新把之前的棋盘情况生成一遍
                        loadimage(&img, _T("2.jpg"));
                        putimage(0, 0, &img);
                        allstep--;
                        box[s[allstep].y][s[allstep].x].value = -1;
                        allstep--;
                        box[s[allstep].y][s[allstep].x].value = -1;
                        for (int i = 0; i < allstep; i++)
                        {
                            retract(s[i].y, s[i].x);
                        }
                        retraction = true;
                        oldi = -1;
                        oldj = -1;
                        break;
                    }
                }
                //判断位置
                if (mouse.x > box[i][j].x && mouse.x<box[i][j].x + 30 &&
mouse.y>box[i][j].y && mouse.y < box[i][j].y + 30 && box[i][j].value == -1)
                {
                    if (oldi >= 0 && oldj >= 0)
                    {
                        box[oldi][oldj].isnew = false;//如果停在某一个空位置上面,更新
选择框
                        box[oldi][oldj].draw();
                    }
                    box[i][j].isnew = true;
                    box[i][j].draw();
                    if (mouse.mkLButton)// 如果按下了
                    {
                        pos my;//放入棋盘数组
                        my.x = j;
                        my.y = i;
                        s[allstep] = my;
                        box[i][j].value = 0; //下棋
                        box[i][j].draw();//画出棋子
                        if (isend())//判断结束
                        {
                            outtextxy(225, 250, _T("玩家胜"));
                            Sleep(5000);
                            return;
                        }
                        else if (allstep == 255)//判断平局
                        {
                            outtextxy(225, 250, _T("平局"));
                            Sleep(5000);
                            return;
```

```
                        }
                        oldi = -1;
                        oldj = -1;
                        allstep++;//总步数+1
                        whoplay = 1;//转到AI
                        break;
                    }
                    oldi = i;//旧位置更新
                    oldj = j;
                }
            }
            if (whoplay == 1 || retraction == true)//跳出循环
                break;
        }
    }
    else//AI下棋
    {
        pos best = { 0,0,0 };
        int board[board_length][board_length];
        for (int i = 0; i < board_length; i++)
        {
            for (int j = 0; j < board_length; j++)
            {
                board[i][j] = box[i][j].value;
            }
        }
        while (!kill.empty())//清空算杀队列
            kill.pop();
        if (!analyse_kill(board, kill_deep, 1, best))//算杀搜索并判断
        {
            for (int i = 0; i < board_length; i++)
            {
                for (int j = 0; j < board_length; j++)
                {
                    board[i][j] = box[i][j].value;
                }
            }
            maxmin(board, maxmin_deep, -MAX, MAX, best);//极大极小搜索
        }
        box[best.y][best.x].value = 1;//下在最佳位置
        box[best.y][best.x].isnew = true;
        s[allstep] = best;
        box[best.y][best.x].draw();
        if (isend())
        {
            outtextxy(225, 250, _T("AI胜"));
            Sleep(5000);
            return;
        }
        allstep++;
        whoplay = 0;
    }
}
```

```
}
```

core.cpp：

```
#include"all.h"
EVALUATION evaluate(int board[board_length][board_length])//评估函数
{
    //各棋型权重
    int weight[17] =
{ 0,10000000,-10000000,50000,-100000,400,-100000,400,-8000,20,-50,20,-50,1,-3,1,-3 };
    int i, j, type;
    int stat[4][17];//统计4个方向上每种棋型的个数
    memset(stat, 0, sizeof(stat));
    int STAT[17];//存在这种棋型的方向的个数
    memset(STAT, 0, sizeof(STAT));
    int A[17][17];//包括边界的虚拟大棋盘,board[i][j]=A[i-1][j-1],3表示边界
    for (int i = 0; i < 17; ++i)A[i][0] = 3;
    for (int i = 0; i < 17; ++i)A[i][16] = 3;
    for (int j = 0; j < 17; ++j)A[0][j] = 3;
    for (int j = 0; j < 17; ++j)A[16][j] = 3;
    for (int i = 0; i < 15; ++i)
        for (int j = 0; j < 15; ++j)
            A[i + 1][j + 1] = board[i][j] + 1;

    //判断横向棋型
    for (i = 1; i <= 15; ++i) {
        for (j = 0; j < 12; ++j) {
            type = tuple6type[A[i][j]][A[i][j + 1]][A[i][j + 2]][A[i][j + 3]][A[i][j +
4]][A[i][j + 5]];
            stat[0][type]++;
        }
    }
    //判断竖向棋型
    for (j = 1; j <= 15; ++j) {
        for (i = 0; i < 12; ++i) {
            type = tuple6type[A[i][j]][A[i + 1][j]][A[i + 2][j]][A[i + 3][j]][A[i + 4][j]][A[i
+ 5][j]];
            stat[1][type]++;
        }
    }
    //判断左上至右下棋型
    for (i = 0; i < 12; ++i) {
        for (j = 0; j < 12; ++j) {
            type = tuple6type[A[i][j]][A[i + 1][j + 1]][A[i + 2][j + 2]][A[i + 3][j + 3]][A[i
+ 4][j + 4]][A[i + 5][j + 5]];
            stat[2][type]++;
        }
    }
    //判断右上至左下棋型
    for (i = 0; i < 12; ++i) {
        for (j = 5; j < 17; ++j) {
            type = tuple6type[A[i][j]][A[i + 1][j - 1]][A[i + 2][j - 2]][A[i + 3][j - 3]][A[i
```

```
+ 4][j - 4]][A[i + 5][j - 5]];
                stat[3][type]++;
            }
        }
        EVALUATION eval;
        memset(eval.STAT, 0, sizeof(eval.STAT));

        int score = 0;
        for (i = 1; i < 17; ++i) {
            score += (stat[0][i] + stat[1][i] + stat[2][i] + stat[3][i]) * weight[i];//初步计分

            int count = stat[0][i] + stat[1][i] + stat[2][i] + stat[3][i];//统计所有方向上部分棋
型的个数
            if (i == WIN)eval.STAT[WIN] = count;
            else if (i == LOSE)eval.STAT[LOSE] = count;
            else if (i == FLEX4)eval.STAT[FLEX4] = count;
            else if (i == BLOCK4)eval.STAT[BLOCK4] = count;
            else if (i == FLEX3)eval.STAT[FLEX3] = count;
        }
        eval.result = -1;
        //白赢或输
        if (eval.STAT[LOSE] > 0)
            eval.result = 0;
        else if (eval.STAT[WIN] > 0)
            eval.result = 1;
        eval.score = score;
        return eval;
    }
    pos seekPoints(int board[board_length][board_length], int who, int deep, bool choice)//局部搜
索
    {
        while (!que.empty())//清空队列
            que.pop();
        int B[board_length][board_length];//局部搜索标记数组
        memset(B, 0, sizeof(B));
        for (int i = 0; i < board_length; ++i)//每个非空点附近8个方向延伸3个深度,若不越界则标记为
可走
        {
            for (int j = 0; j < board_length; ++j)
            {
                if (board[i][j] != -1)
                {
                    for (int l = 0; l < 8; l++)
                    {
                        int nowi = i;
                        int nowj = j;
                        for (int k = 1; k <= search_deep; ++k)
                        {
                            nowi += dy[l];
                            nowj += dx[l];
                            if (nowi <= 14 && nowj <= 14 && nowi >= 0 && nowj >= 0)
                            {
                                B[nowi][nowj] = 1;
```

```
                        }
                    }
                }

            }
        }
    }
    for (int i = 0; i < board_length; ++i)//根据可走点走，并评估，并放入队列
    {
        for (int j = 0; j < board_length; ++j)
        {
            if (board[i][j] == -1 && B[i][j] == 1)
            {
                board[i][j] = who;
                pos a;
                a.x = j;
                a.y = i;
                EVALUATION t = evaluate(board);
                board[i][j] = -1;
                a.worth = t.score;
                int result = t.result;
                if (result == 1 && who == 1 && deep == 2 && choice == 1)//迭代加深2层时
                    return a;
                que.push(a);

            }
        }
    }
    return { 0,0,0 };
}
int maxmin(int board[board_length][board_length], int deep, int alpha, int beta, pos& ans)//
极大极小搜索
{
    pos result = seekPoints(board, (deep + 1) % 2, deep, 1);//局部搜索
    if (result.x + result.y + result.worth != 0)//迭代加深2层的情况
    {
        return result.worth;
    }
    int k = 0;
    pos t[20] = { 0 };
    if (deep % 2 == 0)//队列转数组
    {
        while (!que.empty() && k < 20)
        {
            t[k] = que.top();
            que.pop();
            k++;
        }
    }
    else//队列先转栈再转数组
    {
        stack<pos> q;
        while (!que.empty())
```

```
        {
            q.push(que.top());
            que.pop();
        }
        while (!q.empty() && k < 20)
        {
            t[k] = q.top();
            q.pop();
            k++;
        }
    }
    if (deep == 0)//深度归0
    {
        return t[0].worth;
    }
    else if (deep % 2 == 0)//max层,AI决策
    {
        for (int i = 0; i < 10; ++i)
        {
            int a = 0;
            board[t[i].y][t[i].x] = 1;//模拟AI落子
            a = maxmin(board, deep - 1, alpha, beta, ans);
            board[t[i].y][t[i].x] = -1;//还原落子
            if (a > alpha)
            {
                alpha = a;
                if (deep == maxmin_deep)//4是自己设立的深度(可以改为6,8,但必须为偶数),用来找最佳落子
                {
                    ans.y = t[i].y;
                    ans.x = t[i].x;
                    ans.worth = a;
                }
            }
            if (beta <= alpha)//剪枝
                break;
        }
        return alpha;
    }
    else //min层,人决策
    {
        for (int i = 0; i < 10; ++i)
        {
            int a = 0;
            board[t[i].y][t[i].x] = 0;//模拟人落子
            a = maxmin(board, deep - 1, alpha, beta, ans);
            board[t[i].y][t[i].x] = -1;//还原落子
            if (a < beta)
                beta = a;
            if (beta <= alpha)
                break;//剪枝
        }
        return beta;
```

```
    }
}
void seek_kill_points(int board[board_length][board_length])//算杀点搜索
{
    int newboard[board_length][board_length];//找白棋的连5,活4,冲4,活3的杀棋位置
    for (int i = 0; i < 15; ++i)
        for (int j = 0; j < 15; ++j)
            newboard[i][j] = board[i][j];
    seekPoints(board, whoplay, 0, 0);//一般来说,能冲4或者活3的必在评分前20的点内
    int k = 0;
    pos t[20] = { 0 };
    while (!que.empty() && k < 20)
    {
        t[k] = que.top();
        que.pop();
        k++;
    }
    for (int i = 0; i < 20; ++i)
    {
        newboard[t[i].y][t[i].x] = 1;//模拟落子
        if (evaluate(newboard).STAT[WIN] > 0)
        {//产生连5
            kill.push(t[i]);
        }
        else if (evaluate(newboard).STAT[FLEX4] > evaluate(board).STAT[FLEX4])
        {//产生新活4
            kill.push(t[i]);
        }
        else if (evaluate(newboard).STAT[BLOCK4] > evaluate(board).STAT[BLOCK4])
        {//产生新冲4
            kill.push(t[i]);
        }
        else if (evaluate(newboard).STAT[FLEX3] > evaluate(board).STAT[FLEX3])
        {//产生新活3
            kill.push(t[i]);
        }
        newboard[t[i].y][t[i].x] = -1;//还原落子
    }
}
bool analyse_kill(int board[board_length][board_length], int deep, int who, pos& ans)//算杀搜索
{
    seekPoints(board, who, deep, 0);//局部搜索
    int k = 0;
    pos t[20] = { 0 };
    if (deep % 2 == 0)//队列转数组
    {
        while (!que.empty() && k < 20)
        {
            t[k] = que.top();
            que.pop();
            k++;
        }
```

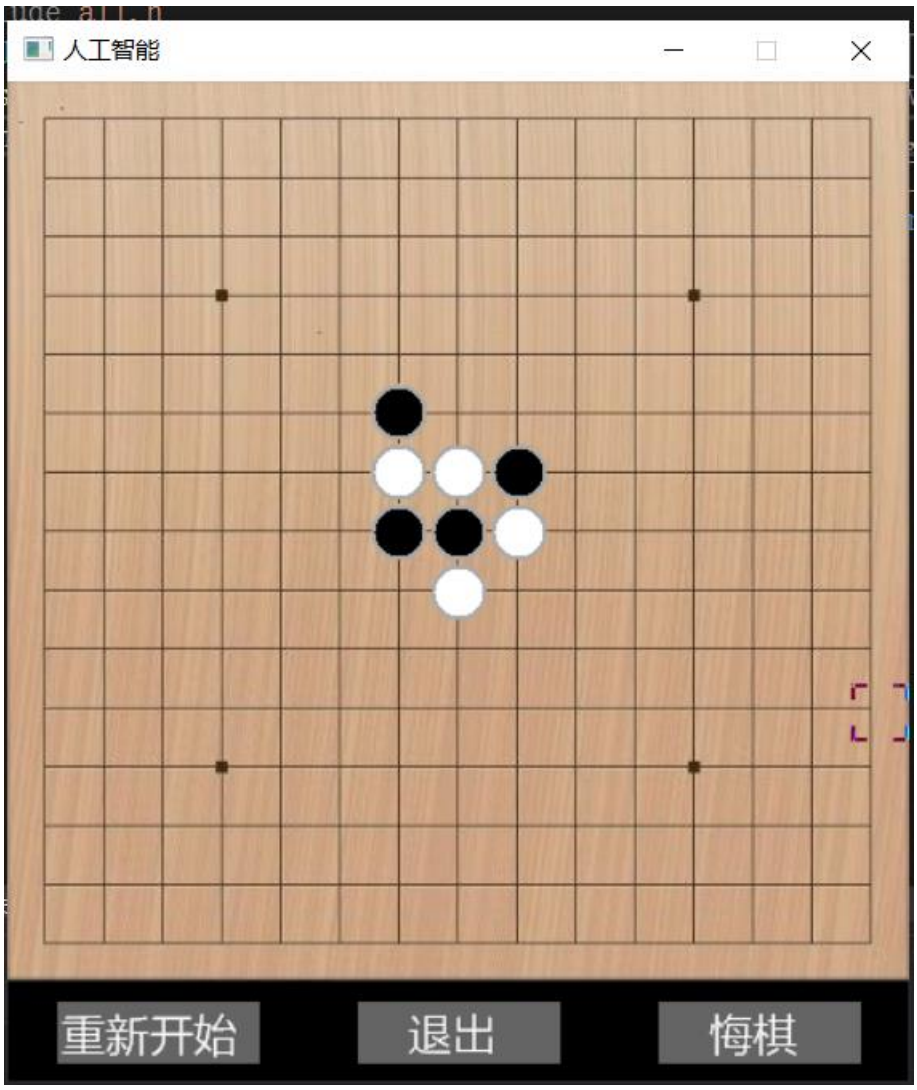
装 订 线

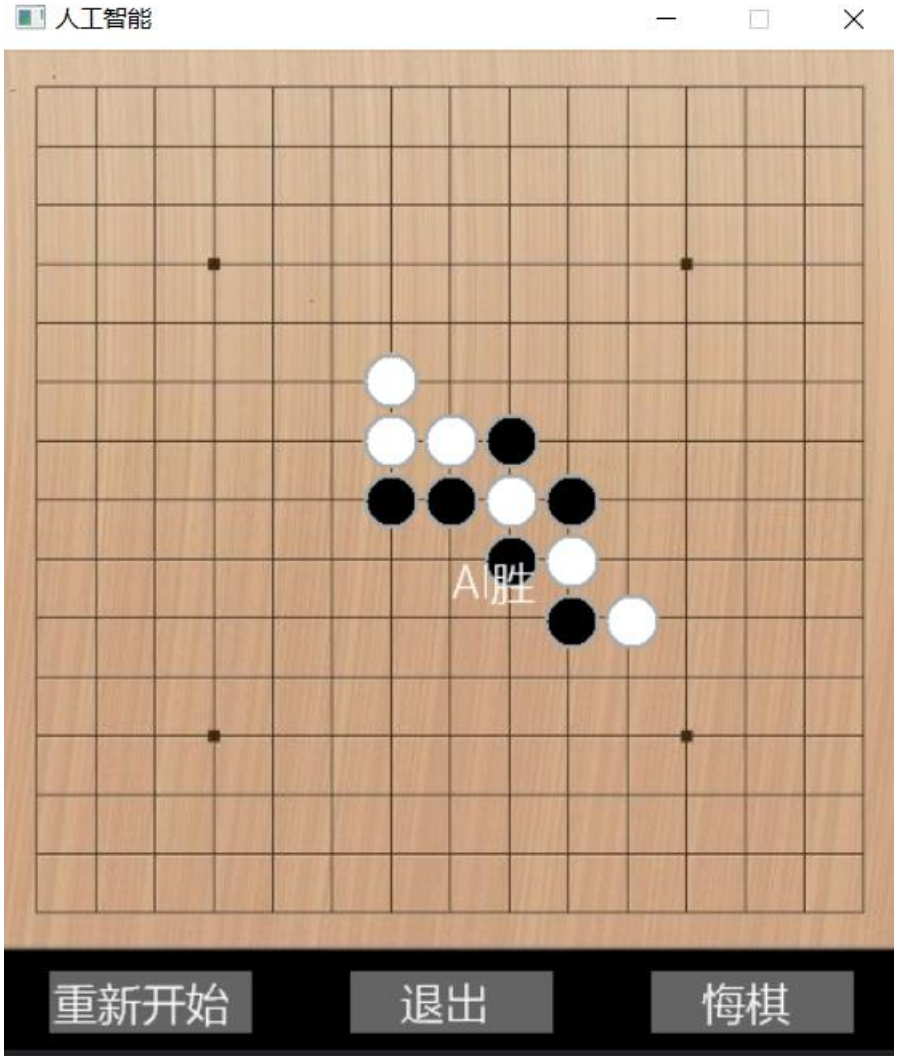
```
}
else//队列转栈再转数组
{
    stack<pos> q;
    while (!que.empty())
    {
        q.push(que.top());
        que.pop();
    }
    while (!q.empty() && k < 20)
    {
        t[k] = q.top();
        q.pop();
        k++;
    }
}
EVALUATION EVAL = evaluate(board);
if (deep == 0 || EVAL.result != -1)
{
    if (EVAL.result == 1)//找到白棋杀棋
        return true;
    else if (EVAL.result == 0)//白棋输
        return false;
    else if (deep == 0)//若抵达最深层,走一步对白棋的最好位置,若白棋还没赢则返回false
    {
        board[t[0].y][t[0].x] = 1;
        int result = evaluate(board).result;
        if (result == 1)
            return true;
        else
            return false;
    }
    else if (EVAL.result == 1)
        return true;//找到白棋杀棋
    else
        return false;//白棋输
}
else if (deep % 2 == 0)//max层,AI决策
{
    seek_kill_points(board);
    if (kill.empty())//没有杀棋点
        return false;
    while (!kill.empty())
    {
        pos a = kill.top();
        kill.pop();
        board[a.y][a.x] = 1;//模拟AI落子
        if (analyse_kill(board, deep - 1, 0, ans))
        {
            if (deep == kill_deep)
            {
                ans = a;
            }
```

```
                    return true;
                }
            }
            return false;
        }
        else //min层,人决策只下对自己最好的棋
        {
            board[t[0].y][t[0].x] = 0;//无需剪枝
            return analyse_kill(board, deep - 1, 1, ans);
        }

}
```

3.4 实验结果展示





装 订 线

3.5 实验结论

五子棋 AI 设计要注重两个方面，一方面要选择好的行棋策略，另一方面要计算速度快。利用极大极小算法、α－β剪枝算法、计算杀棋算法、适合五子棋博弈的评估函数等，可以使得五子棋 AI 尽量选择最优行棋，而利用迭代加深、局部搜索、静态评价启发等可以让 AI 的反应速度变快。通过这些算法配合，该五子棋 AI 展现出了较为理想的状态。不会选择明显愚蠢的走法，同时较好的平衡了进攻和防守。

另外，依然有许多优化策略可以帮助五子棋 AI，例如置换表、开局库、局部刷新、多线程、强化学习等，AI 的棋力仍有提升余地。

4. 总结

4.1 实验中存在的问题及解决方案

1）只用极大极小算法和α－β剪枝，AI 剪枝效率不高，速度不够快。

利用局部搜索和静态评价启发优化。局部搜索只考虑那些能和棋子产生关系的空位置，静态评价启发的原理时如果越早搜索到较优走法，剪枝就会越早发生。

2）有时 AI 在自己已经胜券在握的情况下（有双三之类的棋可以走），竟然会走一些冲四之类的棋。这种走法出现的本质就是因为现在的 AI 只比较最终结果，并没有考虑到路径长短。导致了明明可以两步赢的棋，AI 非要走 3 步。

通过迭代加深来优先找到最短路径。从 2 层开始，逐步增加搜索深度，直到找到胜利走法或者达到深度限制为止。

3）一般的搜索还是效率太低，每增加一层，搜索节点呈指数级增长。

利用计算杀棋克服水平线效应，因为计算杀棋的情况下，每个节点只计算活三和冲四和成五的子节点。所以计算杀棋可以实现多层搜索，我设计的计算杀棋实现的是十六层搜索。

4.2 心得体会

回顾这次实验，感觉十分的充实，通过认真学习、亲自动手，使我进一步了解了极大极小算法、α－β剪枝算法、评估函数的基本知识和设计方法，为今后的学习奠定了良好的基础。

理论课让我们系统地学习了搜索的相关理论。而实验课给了我们一个很好的把理论应用到实践的平台，让我们能够很好的把书本知识转化到实际能力，提高了对于理论知识的理解，认识和掌握。

其次，实验很好地解决了我们实践能力不足且得不到很好锻炼机会的矛盾，通过实验，提高了自身的实践能力和思考能力，并且能够通过实验很好解决自己对于理论的学习中存在的一些知识盲点。

以此次实验为例，不仅完成了实验设计，充分理解了极大极小算法、α－β剪枝算法、评估函数，而且加入了迭代加深、局部搜索、静态评价启发、计算杀棋，使我更好地学习了五子棋 AI 的设计，扩展了自己的视野，提高了自己的实践能力。

4.3 后续改进方向

1）加入置换表，提高效率。对于大部分棋类来说，并不关心你是如何到达这个局面的，只要当前局面上的棋子一样，局势就是一样的。有了第一次的打分，完全可以缓存起来，遇到相同局面就不用打分直接使用缓存数据。以前的搜索结果也可以存下来，用在之后的搜索中。置换表就解决了这个问题。

2）加入局部刷新，提高效率。即如果一个棋子变动，比如棋盘上增加了一个棋子，那么只会在它的米子方向上的几个位置的分数会发生变化，其余的大部分棋子分数不会有任何变化。因此当我们更新分数的时候，只需要更新这这些点即可，这样能大幅降低计算量。

3）加入开局库，提高开局棋力。利用前人的棋谱，帮助 AI 开局。

4）其他各类优化算法例如位棋盘、主要变例搜索、多线程、强化学习等不再赘述。

5）重新规划 UI，例如加入棋谱保存功能，优化按钮设计等。

4.4 总结

搜索问题是一类常见的问题，应对搜索问题有各类搜索算法。对于五子棋人机博弈问题，核心算法还是极大极小搜索，如果加入α－β剪枝算法、计算杀棋、迭代加深、局部搜索、静态评价启发等将大大地减小问题规模，选择合适的评价函数也可以提高棋力、平衡攻防。另外在程序编写过程中，要充分利用 STL 库，例如优先队列等，可以减小程序规模，同时提高性能。

本次实验按照预期完成，五子棋 AI 设计成功，较为高效地解决五子棋人机博弈问题。

同时上述各类优化算法例如位棋盘、主要变例搜索、多线程、强化学习等仍可以做进一步的开发，另外 UI 设计仍有很大的进步空间。

5.参考文献

[1]Stuart Russell, Peter Norvig.人工智能:一种现代的方法[M].清华大学出版社:北京,2013:82-95.

装

订

线