同济大学计算机系

人工智能课程五子棋博弈问题

姓名学号 李辉 1652286、刘兵 1752397、孙文丽 1752844

专　　业 计算机科学

授课老师 武妍

一、 实验概述

(一)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。

(二)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。

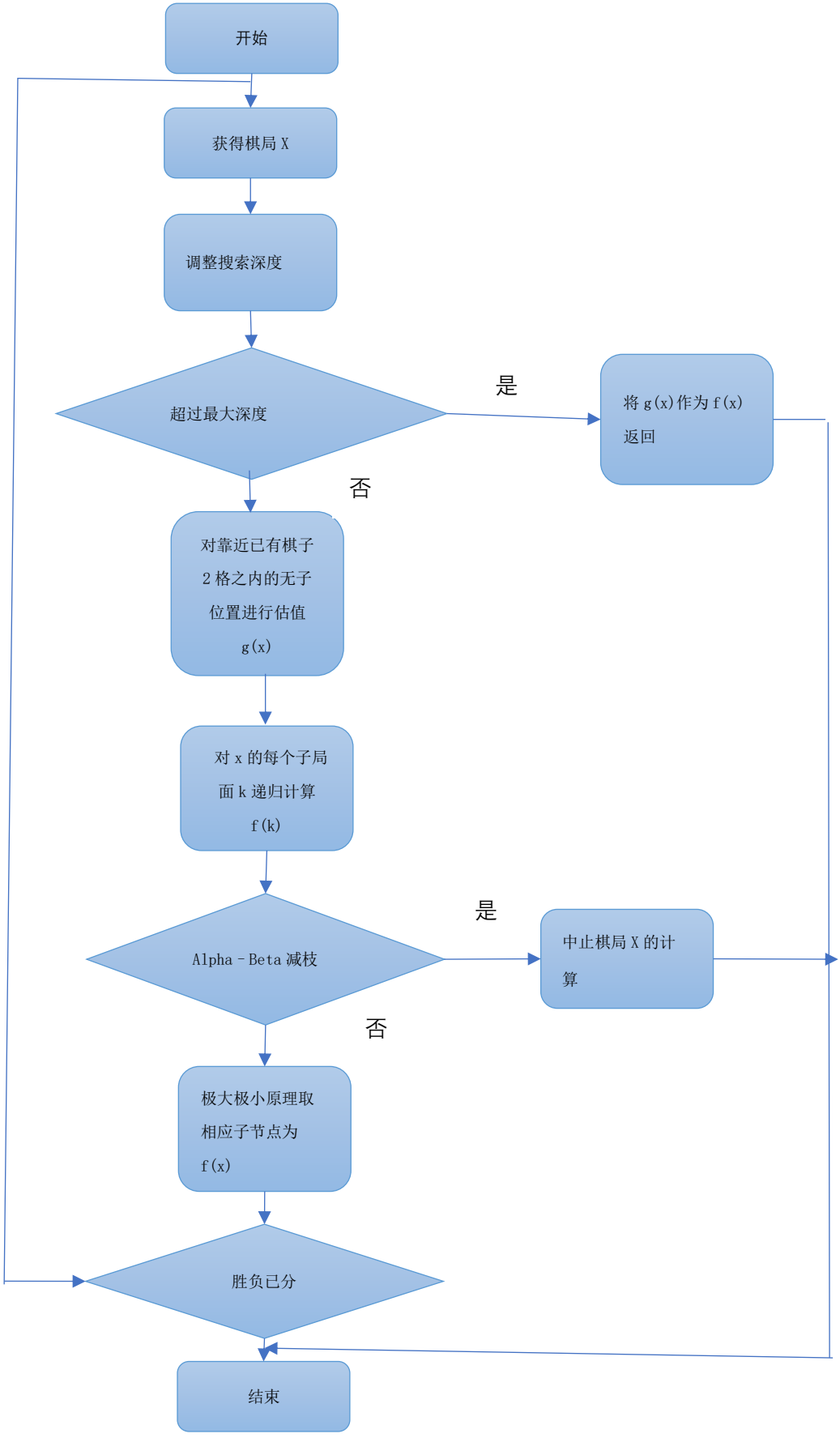
2.要求初始界面显示15*15的空白棋盘，电脑执白棋，人执黑棋，界面置有重新开始、悔棋等操作。

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。

二、 实验方案设计

(一)总体设计思路与总体架构

用一个二维数组存储棋盘状态，空白、黑子、白子分别用0、1、2表示， 通过Alpha－Beta 搜索算法alphaBeta得出下一步落子位置，运用评估函数evaluate对玩家当前局面进行打分，分数越高对玩家越有利，对电脑每步的操作时间进行计时，并且每次落子进行胜负判断。


开始
获得棋局 X
调整搜索深度
超过最大深度
是
将 g(x)作为 f(x)
返回
否
对靠近已有棋子
2 格之内的无子
位置进行估值
g(x)
对 x 的每个子局
面 k 递归计算
f(k)
Alpha－Beta 减枝
是
中止棋局 X 的计
算
否
极大极小原理取
相应子节点为
f(x)
胜负已分
结束

说明：若棋局 k 是棋局 x 的某个子节点，则 g(k)表示对当前旗面 k 应用估值函数得到的估值，f(x)表示对 x 的所有子节点应用最大最小原理后得到的值。

(二)核心算法及基本原理

Alpha-Beta 搜索算法：在博弈问题中，每一个格局可供选择的行动方案都有很多，因此会生成十分庞大的博弈树。一般地只生成一定深度的博弈树，然后进行极小极大搜索。

极大极小搜索是指:在一棵博弈树中,当轮到甲走时,甲定会选择子节点值最大的走法;而轮到乙走时，乙则会选择子节点值最小的走法。 使用估值函数对博弈树的每一个局面进行估值后，就可以通过极大极小搜索在博弈树中寻找最佳的合法走法。

在极大极小搜索的过程中，存在着一定程度的数据冗余。如下图左半部所示的一棵极大极小树的片断。其中节点下方数字为该节点的值，方形框节点代表计算机走，圆形框节点代表人走。A 节点表示计算机走，由于 A 是极大值点，根据极小极大搜索原理它要从 B 和 C 当中选最大的值。假设目前已经通过估值得出 B 为 18，当搜索 C 节点时，因为 C 该人走，所以根据极小极大搜索原理要从 D、E、F 中 选取最小的值。此时如果估出 D 为 16，那么 C 的值必小于或等于 16。又因为已经得出 B 的值为 18，说明节点 A 的值为 Max（B，C）=18，也就是说无须求出节点 C 的其他子节点如 E、F 的值就可以得出父节点 A 的值。这种将节点 D 的后继兄弟节点剪去的方法称为 Alpha 剪枝。同理，在下图右半部一棵极大极小树的片段中，将节点 D 的后继兄弟节点剪去称为 Beta 剪枝。将 Alpha 剪枝和 Beta 剪枝加入极大极小搜索，就得到 Alpha-Beta 搜索算法。

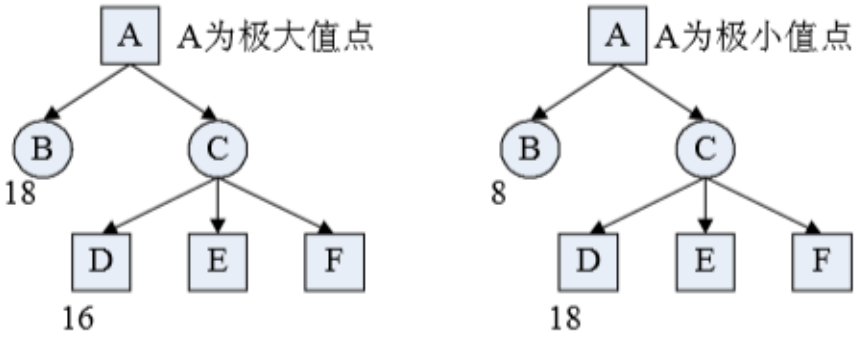


图 1　Alpha-Beta 剪枝

(三)模块设计

①数据结构设计

以数组形式保存当前棋盘的盘面情况：

```
int chessBoard[GRID_NUM][GRID_NUM]//当前棋盘，数组的每一个元素对应棋盘上的
```

一个交叉点，用“0”，“1”，“2”分别表示空白，黑子，白子。

使用元素类型为 pair 的栈存储每一步的落子位置，pair 的两个元素分别表示横纵坐标：

```
stack<pair<int, int>> steps;当需要悔棋时，栈顶的元素即为上次落子的位置。
```

②功能说明

函数名	作用域	功能

bool makeMove(int x, int y, int player)	全局函数	执行落子操作
bool unMakeMove(int x, int y)	全局函数	撤销落子操作
bool regretMove()	全局函数	撤销落子
void move()	全局函数	读取指令，移动光标位置，进行悔棋或者重新开始操作
void print()	全局函数	棋盘打印
pair<int,int>searchMove(int player)	全局函数	搜索函数主体
int alphaBeta(int player, int alpha, int beta, int depth, int turn, int val)	全局函数	Alpha-Beta 搜索算法
bool gameover(int x, int y, int player)	全局函数	判断游戏是否结束
bool checkHorizontal(int, int, int);	全局函数	检查水平方向是否有 5 子相连
bool checkVertical(int, int, int);	全局函数	检查垂直方向是否有 5 子相连
bool checkDiagonal(int, int, int);	全局函数	检查斜线方向是否有 5 子相连
int evaluate_one(int x, int y, int me, int plyer)	全局函数	估值算法
bool inBound(int x, int y)	全局函数	判断与棋盘上现有点是否有联系
Int createMoves(int* moves_x, int* moves_y, int* values, int player)	全局函数	生成全部合法走法集

(四)创新内容

创新内容或优化点：

1、将各个功能模块（剪枝搜索模块，落子模块，评估函数模块等）利用不同的头文件和 cpp 文件区分开，便于区分管理，理清思路，有利于项目进展

2、在 define.cpp 中限定待评估的空格的范围在当前棋盘所有棋子向外延伸距离为 2 的范围内，因为此范围之外的落子对局面分数不影响，从而缩小搜索范围。

3、先将所有评估了的空格的评估结果进行排序取前面一部分分值较大的进行剪枝搜索，提升了搜索效率

4、evaluate.cpp 评估函数模块中，一个重要的问题是方向问题，由于着眼对棋型的判断，

所以首先要解决方向问题，考虑到常见棋型都是线性排列的，所以利用相对位置将二维坐标转换为一维坐标，类似极坐标的概念，一位坐标里只需要存储方向和相对位置即可，减小了难度。

引用的参考网址：

www.cnblogs.com/maxuewei2/x,y/4825520.html

5、孙文丽利用 easyx 将李辉的控制窗口的棋盘效果进行了完善，用蓝色显示 AI 的最后一步棋。在正常的下棋过程中，这个功能不难实现。但是悔棋功能要显示上一回合 AI 的最后一步棋稍微复杂一些，需要利用堆栈实现。

三、 实验过程

(一) 环境说明

操作系统：Windows；

语言：C++;

开发环境：Dev-C++

核心使用库：ctime，stdio.h,string.h,math.h,conio.h,iostream,algorithm,stack，graphics.h

(二) 源代码文件清单

1)头文件

createmoves.h,

define.h,

evaluate.h,

makemove.h,

searchmove.h,

2）源文件

createmoves.cpp,

define.cpp,

evaluate.cpp,

gameover.cpp,

main.cpp,

makemove.cpp,

printchessboard.cpp,

searchmove.cpp

3）可执行文件

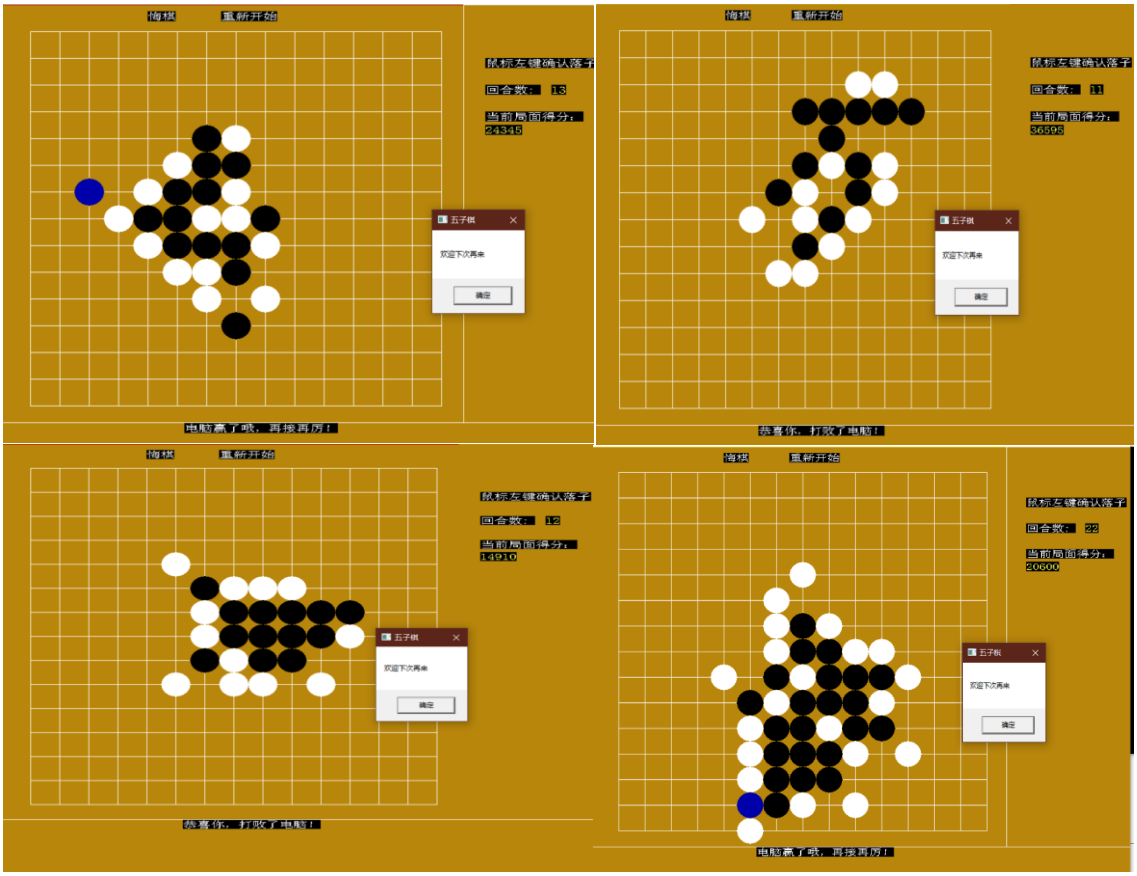
gobang.exe:未优化前的五子棋图形界面实现

五子棋.exe:优化后的五子棋图形界面实现

(三) 实验结果展示

说明：玩家为黑子，电脑白子，蓝子为电脑最后一次落子的位置

从左至右，从上到下依次为电脑先手且电脑胜，电脑先手且电脑负，玩家先手玩家胜，玩家先手玩家负



四、 总结

（一）实验中存在的问题及解决方案

问题 1： 主要难点在局面分数评估以及基于极大极小搜索的阿尔法贝塔剪枝算法的设计，这是与电脑棋力以及电脑落子速度紧密关联的地方，评估函数的打分直接影响到电脑落子的偏好，剪枝节点的顺序以及层数影响到剪枝效率，这些都是需要考虑的地方。

解决方案：通过查找文献资料深入了解 alpha-beta 搜索算法后得以解决

问题 2：图形化时，获取鼠标动作的时间用一个标志位控制；由于画布的是矩形，坐标 x 存在一定程度的缩放，如果按照鼠标获取的坐标画棋子，会造成显示与记录的棋子位置不一致，导致棋面上获胜而游戏继续进行，以及明明是空位却无法下子的问题。

解决方案：仔细分析检查程序

（二） 心得体会

通过这次实验，我深刻地理解到 Alpha-Beta 搜索算法在博弈问题上的重大作用，

如果只是使用极大极小搜索，则不可避免地会出现两种明显的冗余现象，分别是极大值冗余和极小值冗余，这将导致更大的空间和时间浪费。在小组合作方面，我们彼此分工明确，相处融洽，积极参与讨论，指出彼此的不足，互帮互助，使得问题得到了比较妥善的解决，彼此都一定程度上提升了一些。

（三） 后续改进方向

目前棋力不强，搜索深度大于 4 的话搜索时间很长，下一步可以考虑加入算杀模块，加入启发式搜索以及棋类对抗常用的 zobrist 算法进行优化。另一方面，也可以考虑使用置换表，在 Alpha-Beta 搜索过程中，为了避免重复搜索已经搜索过的节点，加快搜索速度，可以使用一张表格记录每一节点的搜索结果，对任意节点向下搜索之前先察看记录在表中的这些结果。如果将要搜索的某个节点在表中已有记录，就直接利用记录下来的结果。也可以减小搜索范围，目前的搜索把所有未下子的空白位置都搜索了一遍，事实上并没必要，一般只需要考虑据棋子 2 格左右的位置就可以了。

参考文献

https://www.ctolib.com/yfismine-GoBang-AI.html

https://blog.csdn.net/livingsu/article/details/104544562?utm_medium=distribute.pc_relevant.none-task-blog-BlogCommendFromBaidu-4&depth_1-utm_source=distribute.pc_relevant.none-task-blog-BlogCommendFromBaidu-4

https://blog.csdn.net/livingsu/article/details/104655537?utm_medium=distribute.pc_relevant.none-task-blog-OPENSEARCH-6&depth_1-utm_source=distribute.pc_relevant.none-task-blog-OPENSEARCH-6

https://blog.csdn.net/lihongxun945/article/details/50730231

https://www.cnblogs.com/maxuewei2/x,y/4825520.html

成员分工与自评

成员	分工
李辉	评估函数以及搜索过程的实现，编写代码
刘兵	总结实验，撰写实验报告，制作 PPT
孙文丽	五子棋游戏的图形化界面实现

1.李辉

此次五子棋试验我负责程序编写这一块儿，主要难点在局面分数评估以及基于极大极小搜索的阿尔法贝塔剪枝，这是与电脑棋力以及电脑落子速度紧密关联的地方，评估函数的打分直接影响到电脑落子的偏好，剪枝节点的顺序以及层数影响到剪枝效率，这些都是需要考

虑的地方。此次试验相对于之前工程量大了很多，通过这次实验我对对抗搜索以及阿尔法贝塔剪枝有了更进一步的了解，编码能力有了一定的提高。同时由于比较仓促，还有很多地方需要改进，例如阿尔法贝塔剪枝的结果是电脑主要进行防守而不进行攻击，搜索层数设置的大一些搜索速度变得非常缓慢，没有考虑其他复杂的棋型情况，例如连接的双活三、双活四这些局面的评估。如果要进一步改进，可以考虑进一步完善局面评估，增加算杀算法（即只考虑冲四活三这类必须防守的棋型），能在局部增加搜索深度而不减慢时间，同时使得电脑富有攻击性，还可以增加启发式搜索，进一步减小搜索范围。另外，即使有了上述的改进，下棋高手还是很容易判断电脑的落子偏好，轻易取胜，所以还可以考虑引入棋类对抗常用的zobrist 算法等，随机改变落子风格。人工智能是一片汪洋大海，我们所涉猎的还只是一片浅滩。

2.刘兵

我负责总结实验，撰写实验报告和制作 PPT,我参与了五子棋博弈问题实验的整个过程，通过对实验成果的整合，我加深了对 Alpha-Beta 搜索算法的理解，在编程和有关博弈问题的一些算法方面也增长了见识。

3.孙文丽

本次实验我主要负责实现五子棋的图形界面。由于主要源文件是用 C++实现的，所以我选择 Easyx 图形库绘图。首先我需要知道程序流程以及相关的各种变量，参数，接口，所以必须浏览项目的大部分源文件。

我想用鼠标控制落子，这个过程不是很顺利，所以对一些 bug 印象深刻：获取鼠标动作的时间用一个标志位控制；由于画布的是矩形，坐标 x 存在一定程度的缩放，如果按照鼠标获取的坐标画棋子,会造成显示与记录的棋子位置不一致,导致棋面上获胜而游戏继续进行，以及明明是空位却无法下子的问题。

创新点是用蓝色显示 AI 的最后一步棋。在正常的下棋过程中，这个功能不难实现。但是悔棋功能要显示上一回合 AI 的最后一步棋稍微复杂一些，需要利用堆栈实现。

除此之外，这次实验让我深刻理解了 α－β 剪枝的过程和功能，以及评估函数对攻守偏向的影响。