

**课 程 实 验 报 告**

**课程名称： 人工智能导论**

**专业班级： CS2004**

**学 号： U202015396**

**姓 名： 沈子旭**

**指导教师： 金 燕**

**报告日期： 2021.12.31**

**计算机科学与技术学院**

井字棋剪枝算法实现

一、知识内容介绍。

1.井字棋介绍

“井字棋”游戏（又叫“三子棋”），是一款十分经典的益智小游戏，操作简单，娱乐性强。两个玩家，一个打圈(O)，一个打叉(X)，轮流在3乘3的格上打自己的符号，最先以横、直、斜连成一线则为胜。

如果双方都下得正确无误，将得和局。这种游戏实际上是由第一位玩家所控制，第一位玩家是攻，第二位玩家是守。这种游戏的变化简单，常成为博弈论和赛局树搜寻的教学例子。这个游戏只有765个可能局面，26830个棋局。如果将对称的棋局视作不同，则有255168个棋局。

将井字棋游戏设计为人机对弈系统，分别用X代替人（玩家）的棋子,用O代替机（电脑）的棋子。玩家可以选择自己或电脑先下（游戏实际上是由先下的一方所控制），电脑一方是由程序选择对自己最有利的棋局决定下一步，程序利用剪枝算法实现电脑的走步。结局只有三种情况：玩家赢，电脑赢或平局。

2.剪枝算法介绍

1、简介

在搜索算法中优化中，剪枝，就是通过某种判断，避免一些不必要的遍历过程，形象的说，就是剪去了搜索树中的某些“枝条”，故称剪枝。应用剪枝优化的核心问题是设计剪枝判断方法，即确定哪些枝条应当舍弃，哪些枝条应当保留的方法。

2、剪枝优化三原则: 正确、准确、高效.原则

搜索算法,绝大部分需要用到剪枝.然而,不是所有的枝条都可以剪掉,这就需要通过设计出合理的判断方法,以决定某一分支的取舍. 在设计判断方法的时候,需要遵循一定的原则.

剪枝的原则

1） 正确性

正如上文所述,枝条不是爱剪就能剪的. 如果随便剪枝,把带有最优解的那一分支也剪掉了的话,剪枝也就失去了意义. 所以,剪枝的前提是一定要保证不丢失正确的结果.

2）准确性

在保证了正确性的基础上,我们应该根据具体问题具体分析,采用合适的判断手段,使不包含最优解的枝条尽可能多的被剪去,以达到程序“最优化”的目的. 可以说,剪枝的准确性,是衡量一个优化算法好坏的标准.

3）高效性

设计优化程序的根本目的,是要减少搜索的次数,使程序运行的时间减少. 但为了使搜索次数尽可能的减少,我们又必须花工夫设计出一个准确性较高的优化算法,而当算法的准确性升高,其判断的次数必定增多,从而又导致耗时的增多,这便引出了矛盾. 因此,如何在优化与效率之间寻找一个平衡点,使得程序的时间复杂度尽可能降低,同样是非常重要的。

1. 分类
2. 可行性剪枝：该方法判断继续搜索能否得出答案，如果不能直接回溯。
3. 最优性剪枝：又称为上下界剪枝，是一种重要的搜索剪枝策略。它记录当前得到的最优值，如果当前结点已经无法产生比当前最优解更优的解时，可以提前回溯。
4. 算法原理和算法步骤
5. minmaxn(极大极小法)算法思想

1.设博弈双方中一方为maxn，另一方为min。然后为其中的一方(在此问题中为计算机)找一个最佳走法。当然在程序设计中，为了避免与C++自带的maxn与min函数重复导致错误，我们对maxn与min，命名时要进行进一步注释。

2.为了找到当前棋局的最优走法，需要对各个可能的走法所产生的后续棋局进行比较，同时也要考虑对方可能的走法，并对后续棋局赋予一定的权值。也就是说，以当前棋局为根节点生成一棵博弈树，N步后的棋局作为树的叶子节点。同时从树根开始轮流给每层结点赋予maxn和min的名字。

3.用一个打分函数来分析计算各个后续棋局（即叶子节点）的权值，估算出来的分数为静态估值。要注意将某方获胜的状态节点的评估函数值设为程序中设计的最大数（无穷大）或最小数（无穷小）以表明在该状态下有一方获胜。

4.当端节点的估值计算出来后，再推算出父节点的得分。推算的方法是：对于处于maxn层的节点，选其子节点中一个最大的得分作为父节点的得分，这是为了使自己在可供选择的方案中选一个对自己最有利的方案；对处于min层的节点，选其子节点中一个最小的得分作为父节点的得分，这是为了立足于最坏的情况，这样计算出的父节点的得分为倒推值。

如此反推至根节点下的第一层孩子，如果其中某个孩子能获得最大的倒推值，则它就是当前棋局最好的走法。

不过在极大极小法中，必须求出所有端点的评估值，当预先考虑的棋步比较多时，计算量会大大增加，从而加长计算机得出最后结果的时间。因此，在端点情况较多时，α-β剪枝方法是一种效率比较高的方法。在α-β剪枝方法中，采用了两个变量α和β，它们是最终可以获得的对最大评估值和最小评估值的估计值。

（2）α－β剪枝的算法思想

1.maxn节点的α值为当前子节点的最大倒推值；

2.min节点的β值为当前子节点的最小倒推值；

3.α－β剪枝的规则如下：

1.任何maxn节点n的α值大于或等于它祖辈节点的β值，则n以下的分枝可停止搜索，并令节点n的倒推值为α。这种剪枝称为β剪枝。

2.任何min节点n的β值小于或等于它祖辈节点的α值，则n 以下的分支可停止搜索，并令节点n的倒推值为β。这种剪枝称为α剪枝。

（3）伪代码如下

#include<stdio.h>

#include<string.h>

#include <iostream>

#define true 1

#define false 0

int chesselem[3][3];

int X, Y, count;

int WIN()//判断是否胜利

{//返回-1表示胜利，返回1表示失败

int i;

for (i = 0; i < 3; i++)

{

if (电脑列满足三个一样) return 1;

if (玩家列满足三个一样) return -1;

}

//列比较

for (i = 0; i < 3; i++)

{

if (电脑行满足三个一样) return 1;

if (玩家行满足三个一样) return -1;

}

//行比较

if (电脑斜着满足三个一样) return 1;

if (玩家斜着满足三个一样) return -1;

}

int EVA()//评估函数，主要思想是计算每一行、每一列、斜线中连成3个棋子的有多少个。

{

int i, j, t[3][3];

X = Y = 0;

for (i = 0; i < 3; i++)

{

for (j = 0; j < 3; j++)

{

if (没有被填写) t[i][j] = 1;

else t[i][j] = chesselem[i][j];

//这步主要目的是创建一个新的棋盘来保存状态现在的棋盘状态，有助于后续统计出每行每列有多少个棋子，为判定结束提供条件

}

}

for (i = 0; i < 3; i++) X += (t[i][0] + t[i][1] + t[i][2]) / 3;//列循环

for (i = 0; i < 3; i++) X += (t[0][i] + t[1][i] + t[2][i]) / 3;行循环

X += (t[0][0] + 2\*t[1][1] + t[2][2] + t[2][0] + t[0][2]) / 3;

Y is the same

return X + Y;//用于判断总步骤数

}

int AlphaBeta(int& value, int d, bool maxn)//剪枝函数

{//函数具体实现

bool trim = false;

int i, j, flag, t;

if (d == 3 || d + count == 9)

{

return EVA();

}

if (WIN() == 1)

{

value = 10000;

return 0;

}

if (maxn) flag = 10000;

else flag = -10000;

for (i = 0; i < 3 && !trim; i++)

{

for (j = 0; j < 3 && !trim; j++)

{

if (chesselem[i][j] == 0)

{

if (maxn)

{

chesselem[i][j] = -1;

if (WIN() == -1) t = -10000;

else t = AlphaBeta(flag, d + 1, !maxn);

if (t < flag) flag = t;

if (flag <= value) trim = true;

}

else

{

chesselem[i][j] = 1;

if (WIN() == 1) t = 10000;

else t = AlphaBeta(flag, d + 1, !maxn);

if (t > flag) flag = t;

if (flag >= value) trim = true;

}

chesselem[i][j] = 0;

}

}

}

if (maxn) if (flag > value) value = flag; //判断状态

else if (flag < value) value = flag;

return flag;

}

void CInput()//玩家输入函数

{

int x, y;//行和列

while (true)

{

输出"\n请输入你想填入的坐标(X Y):";

输入x;

输入y;

if (输入坐标合法)

{

将输入坐标值设置成-1；

退出循环；

}

else 输出"输入错误!";

}

}

void ChessTable()//将棋盘打印出来

{

int i, j;

for (i = 0; i < 3; i++)

{

for (j = 0; j < 3; j++)

{

if (电脑下的)输出"○";//对应1

if (没有棋子) 输出"□";//对应0

if (玩家下的) 输出"×";//对应1

}

}

}

int TChess()//井字棋运行的主要程序

{

int x, y, t;

int m = -10000, value = -10000, d = 1;

count = 0;

将棋盘初始化；

输出"棋盘:";//人机交互

ChessTable();//再将棋盘现在的状态打印一遍

while (1)

{

输出"\n你想让谁执行第一步:\n1)玩家. 2)电脑.\n";//人家交互

输入t;

if (t的输入有问题)继续循环; //直到输入正确才会停止

else 退出循环;

}

if (玩家先开始)//玩家输入情况

{

while (true)

{

CInput();

ChessTable();

count++;

EVA();

//判断输赢

if (平局)

{

输出"平局!";//人机交互

return 0;

}

if (玩家胜利)

{

输出"你赢了!";

return 0;

}

for (int i = 0; i < 3; i++)

{

for (int j = 0; j < 3; j++)

{

if (chesselem[i][j] == 0)

{

chesselem[i][j] = 1;

AlphaBeta(value, d, 1);//运行剪枝算法

//通过二层循环遍历棋盘的每一个部分进行剪枝运算，达到化简的目的

if (电脑赢了)

{

输出"电脑将棋下在了:";//输出位置

ChessTable();

输出"电脑赢了!";

return 0;

}

if (没有判断出谁赢的状态)

{

m = value;

x = i; y = j;

}//重新设置行和列进行循环进行剪枝运算

value = -10000;

chesselem[i][j] = 0;

}

}

}

if (玩家胜利)

{

输出"你赢了!";

return 0;

}

chesselem[x][y] = 1;

value = -10000; m = -10000; d = 1；//重新设置值

输出"电脑将棋下在了:";//输出电脑下的位置

ChessTable();

count++;

EVA();//和上面那个模块原理相同

if (平局)

{

输出"平局!";

return 0;

}

}

}

else//电脑输入的条件

{

while (true)

{

for (int i = 0; i < 3; i++)

{

for (int j = 0; j < 3; j++)

{

if (棋盘这个位置没有下棋)

{

chesselem[i][j] = 1;

AlphaBeta(value, d, 1);

if (电脑赢了)

{

输出"电脑将棋下在了：”;

ChessTable();

输出"电脑赢了!";

return 0;

}

//用状态的极大和极小判断哪一方胜利

if (无法判断输赢情况)

{

m = value;

x = i; y = j;

}

value = -10000;

chesselem[i][j] = 0;

}

}

}

chesselem[x][y] = 1;

value = -10000; m = -10000; d = 1;

输出"电脑将棋下在了:";

ChessTable();//打印棋盘

count++;

EVA();

if (平局情况)

{

输出"平局!";

return 0;

}

CInput();//玩家输入

ChessTable();

count++;

EVA();

if (平局)

{

输出"平局!";

return 0;

}

if (玩家胜利)

{

输出"你赢了!";

return 0;

}

}

}

return 0;

}

int main()

{

int p;

while (true)

{

TChess();

输出"\n再来一局?\n1)再来一局！.\t2)算了\n";

输入p;

if (p != 1 并且 p != 2) ;

if (p == 2) break;//用于判断玩家是否还玩

}

return 0;

}

1. 实验结果及分析

输出符合预期，人机交互良好，算法无误，程序执行正常。

如下为程序部分运行结果

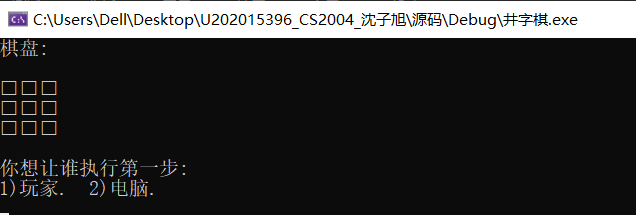


图1.初始界面

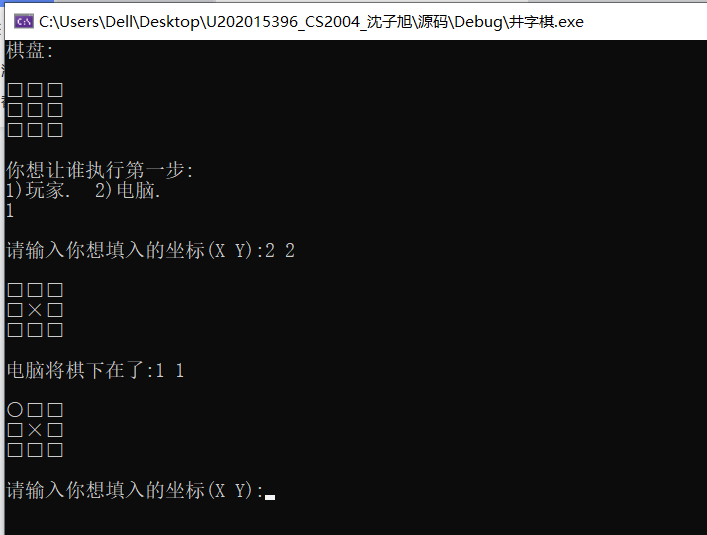


图2.操作界面

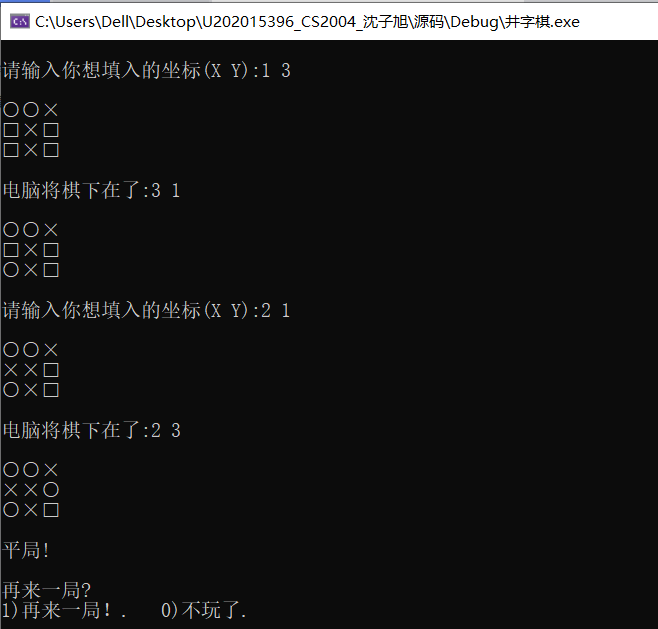


图3.是否继续的询问

分析：由于每一步都选择最优解，电脑先手保证不会输。

剪枝算法的优缺点：优点，大大优化了程序，使得程序减少了很多没有必要的循环运算操作，提高了代码的运算效率。

缺点：节点划分出现大量重复，使得分支变多。

1. 心得体会

本项目要求将各个函数细分功能，每个函数完成对应的任务即可。同时还要注意刷新功能，设计不慎可能会使得程序崩溃。本项目采用最大最小思想和剪枝算法，让问题的解决方案得到了很大的优化。在设计函数时，参考了经典的井字棋解决案例和maxmin算法以及α-β剪枝算法，有效地提高了编程水平。

附录：源码：

head.h

#include<stdio.h>

#include<string.h>

#include <iostream>

#define true 1

#define false 0

main.cpp

#include"head.h"

int chesselem[3][3];

//预设3\*3的棋盘

int X, Y, count;

int WIN()//返回-1表示胜利，返回1表示失败

{

int i;

for (i = 0; i < 3; i++)

{

if (chesselem[i][0] == 1 && chesselem[i][1] == 1 && chesselem[i][2] == 1) return 1;

if (chesselem[i][0] == -1 && chesselem[i][1] == -1 && chesselem[i][2] == -1) return -1;

}

//列比较

for (i = 0; i < 3; i++)

{

if (chesselem[0][i] == 1 && chesselem[1][i] == 1 && chesselem[2][i] == 1) return 1;

if (chesselem[0][i] == -1 && chesselem[1][i] == -1 && chesselem[2][i] == -1) return -1;

}

//行比较

if ((chesselem[0][0] == 1 && chesselem[1][1] == 1 && chesselem[2][2] == 1) || (chesselem[2][0] == 1 && chesselem[1][1] == 1 && chesselem[0][2] == 1)) return 1;

if ((chesselem[0][0] == -1 && chesselem[1][1] == -1 && chesselem[2][2] == -1) || (chesselem[2][0] == -1 && chesselem[1][1] == -1 && chesselem[0][2] == -1)) return -1;

return 0;

}

int EVA()//打分函数，计算每一行、每一列、斜线中相连的棋子有多少个。

{

int i, j, t[3][3];

X = Y = 0;

for (i = 0; i < 3; i++)

{

for (j = 0; j < 3; j++)

{

if (chesselem[i][j] == 0) t[i][j] = 1;

else t[i][j] = chesselem[i][j];

}

}

for (i = 0; i < 3; i++) X += (t[i][0] + t[i][1] + t[i][2]) / 3;

for (i = 0; i < 3; i++) X += (t[0][i] + t[1][i] + t[2][i]) / 3;

X += (t[0][0] + 2\*t[1][1] + t[2][2] + t[2][0] + t[0][2]) / 3;

for (i = 0; i < 3; i++)

{

for (j = 0; j < 3; j++)

{

if (chesselem[i][j] == 0) t[i][j] = -1;

else t[i][j] = chesselem[i][j];

}

}

for (i = 0; i < 3; i++) Y += (t[i][0] + t[i][1] + t[i][2]) / 3;

for (i = 0; i < 3; i++) Y += (t[0][i] + t[1][i] + t[2][i]) / 3;

Y += (t[0][0] + 2\*t[1][1] + t[2][2] + t[2][0] + t[0][2]) / 3;

return X + Y;

}

int AlphaBeta(int& value, int d, bool maxn)

//进行α-β剪枝，获取接下来的最优路径

{

bool trim = false;

int i, j, flag, t;

if (d == 3 || d + count == 9) return EVA();

if (WIN() == 1)

{

value = 10086;

return 0;

}

if (maxn) flag = 10086;

else flag = -10086;

for (i = 0; i < 3 && !trim; i++)

{

for (j = 0; j < 3 && !trim; j++)

{

if (chesselem[i][j] == 0)

{

if (maxn)

{

chesselem[i][j] = -1;

if (WIN() == -1) t = -10086;

else t = AlphaBeta(flag, d + 1, !maxn);

if (t < flag) flag = t;

if (flag <= value) trim = true;

}

else

{

chesselem[i][j] = 1;

if (WIN() == 1) t = 10086;

else t = AlphaBeta(flag, d + 1, !maxn);

if (t > flag) flag = t;

if (flag >= value) trim = true;

}

chesselem[i][j] = 0;

}

}

}

if (maxn) if (flag > value) value = flag; //判断状态

else if (flag < value) value = flag;

return flag;

}

void Cinput()//坐标输入函数

{

int x, y;

while (true)

{

printf("\n请输入你想填入的坐标(X Y):");

scanf\_s("%d",&x);

scanf\_s("%d",&y);

if (x > 0 && x < 4 && y > 0 && y < 4 && chesselem[x - 1][y - 1] == 0)

{

chesselem[x - 1][y - 1] = -1;

break;

}

else printf("输入错误!\n");

}

}

void ChessTable()//棋盘打印函数

{

int i, j;

printf("\n");

for (i = 0; i < 3; i++)

{

for (j = 0; j < 3; j++)

{

if (chesselem[i][j] == 1) printf("○");

if (chesselem[i][j] == 0) printf("□");

if (chesselem[i][j] == -1) printf("×");

}

printf("\n");

}

}

int Tchess()//井字棋运行主要程序

{

int x, y, t;

int m = -10086, value = -10086, d = 1;

count = 0;

memset(chesselem, 0, sizeof(chesselem));//初始化棋盘

printf("棋盘:\n");

ChessTable();//打印棋盘

while (true)

{

printf("\n你想让谁执行第一步:\n1)玩家. 2)电脑.\n");

scanf\_s("%d",&t);

if (t != 1 && t != 2)

{

getchar();//输入错误则继续读入

printf\_s("请输入1或2\n");

}

else break;

}

if (t == 1)//玩家先手

{

while (true)

{

Cinput();

ChessTable();

count++;

EVA();//判断输赢

if (Y == 0)

{

printf("\n平局!\n");

return 0;

}

if (WIN() == -1)

{

printf("\n你赢了!\n");

return 0;

}

for (int i = 0; i < 3; i++)

{

for (int j = 0; j < 3; j++)

{

if (chesselem[i][j] == 0)

{

chesselem[i][j] = 1;

AlphaBeta(value, d, 1); //运行α-β剪枝算法

if (WIN() == 1)

{

printf("\n电脑将棋下在了:%d %d\n", i + 1, j + 1);

ChessTable();

printf("\n电脑赢了!\n");

return 0;

}

if (value > m)

{

m = value;

x = i;

y = j;

}

value = -10086;

chesselem[i][j] = 0;

}

}

}

if (chesselem[x][y]) //判断玩家是否获胜

{

printf("\n你赢了!\n");

return 0;

}

chesselem[x][y] = 1;

value = -10086; m = -10086; d = 1; //设定极大极小值

printf("\n电脑将棋下在了:%d %d\n", x + 1, y + 1);

ChessTable();

count++;

EVA();//判断胜负

if (Y == 0)

{

printf("\n平局!\n");

return 0;

}

}

}

else//电脑输入的条件

{

while (true)

{

for (int i = 0; i < 3; i++)

{

for (int j = 0; j < 3; j++)

{

if (chesselem[i][j] == 0)

{

chesselem[i][j] = 1;

AlphaBeta(value, d, 1);

if (WIN() == 1)

{

printf("\n电脑将棋下在了:%d %d\n", i + 1, j + 1);

ChessTable();

printf("\n电脑赢了!\n");

return 0;

}

//用状态的极大和极小判断哪一方胜利

if (value > m)

{

m = value;

x = i;

y = j;

}

value = -10086;

chesselem[i][j] = 0;

}

}

}

chesselem[x][y] = 1;

//思想同上

value = -10086; m = -10086; d = 1;

printf("\n电脑将棋下在了:%d %d\n", x + 1, y + 1);

ChessTable();

count++;

EVA();

if (X == 0)

{

printf("\n平局!\n");

return 0;

}

Cinput();

ChessTable();

count++;

EVA();

if (X == 0)

{

printf("\n平局!\n");

return 0;

}

if (WIN() == -1)

{

printf("\n你赢了!.\n");

return 0;

}

}

}

return 0;

}

int main()

{

using namespace std;

int p;

while (true)

{

Tchess();

printf("\n再来一局?\n1)再来一局！.\t0)不玩了.\n");

scanf\_s("%d",&p);

if (p != 1 && p != 2) getchar();

if (p == 0) break;

}

return 0;

}