**利用α-β搜索过程的博弈树搜索算法编写一字棋游戏**

1. 实验目的

1.初步了解了博弈树算法的基本过程，知道了博弈树算法的原理与步骤，能简单的运用它来设计棋牌游戏。

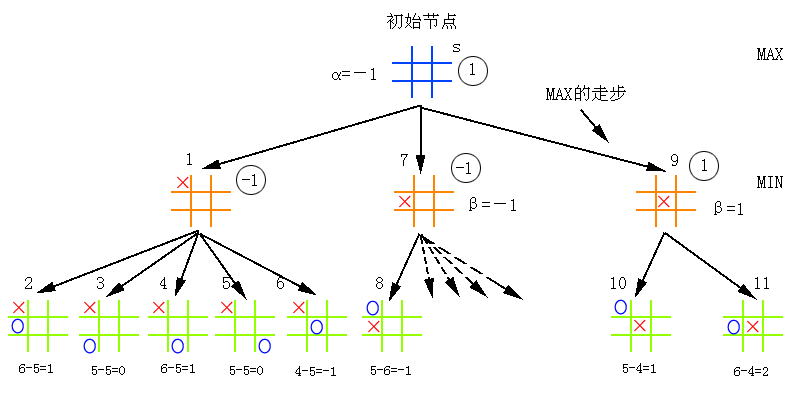
2.了解极大极小算法的原理和使用方法，并学会用α-β剪枝来提高算法的效率。对一些简单的棋牌游戏能大概上看懂它们的运行过程。

二、实验原理

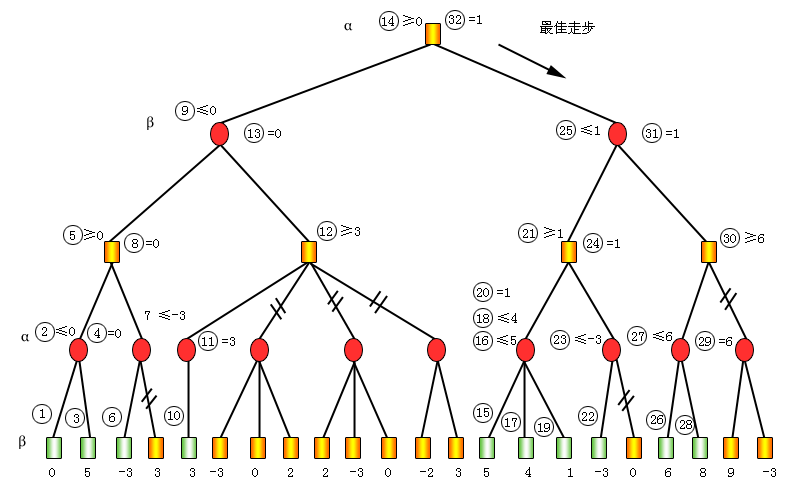
游戏由两个人轮流来下，分别用“X”和“O”来代替自身的棋子。棋盘分9个格，双方可以在轮到自己下的时候，可以用棋子占领其中一个空的格子。如果双方中有一方的棋子可以连成一条直线，则这一方判胜，对方判负。当所有的格子都被占领，但双方都无法使棋子连成一条直线的话，则判和棋。下棋的过程中可以中途按下“0”退出。当用户与计算机分出了胜负后，机器会显示出比赛的结果，并按任意键退出。如果用户在下棋的过程中，输入的是非法字符，机器不会做出反应。

三、实验分析

为了使生成和估值过程紧密结合，采用有界深度优先策略进行搜索，这样当生成达到规定深度的节点时，就立即计算其静态估值函数，而一旦某个非端节点有条件确定其倒推值时就立即计算赋值。从图1中标记的节点生成顺序号（也表示节点编号）看出，生成并计算完第6个节点后，第1个节点倒推值完全确定，可立即赋给倒推值－1。这时对初始节点来说，虽然其他子节点尚未生成，但由于s属极大值层，可以推断其倒推值不会小于－1，我们称极大值层的这个下界值为α，即可以确定s的α＝－1。这说明s实际的倒推值决不会比－1更小，还取决于其他后继节点的倒推值，因此继续生成搜索树。当第8个节点生成出来并计算得静态估值为－1后，就可以断定第7个节点的倒推值不可能大于－1，我们称极小值层的这个上界值为β，即可确定节点７的β＝－1。有了极小值层的β值，很容易发现若α≥β时，节点7的其他子节点不必再生成，这不影响高一层极大值的选取，因s的极大值不可能比这个β值还小，再生成无疑是多余的，因此可以进行剪枝。这样一来，只要在搜索过程记住倒推值的上下界并进行比较，就可以实现修剪操作，称这种操作为α剪枝。类似的还有β剪枝，统称为α-β剪枝技术。在实际修剪过程中，α、β还可以随时修正，但极大值层的倒推值下界α永不下降，实际的倒推值取其后继节点最终确定的倒推值中最大的一个倒推值。而极小值层的倒推值上界β永不上升，其倒推值则取后继节点最终确定的倒推值中最小的一个倒推值。

  
3.1 在进行α-β剪枝时，应注意以下几个问题：  
（1）比较都是在极小节点和极大节点间进行的，极大节点和极大节点的比较，或者极小节点和极小节点间的比较是无意义的。  
（2）在比较时注意是与"先辈层"节点比较，不只是与父辈节点比较。当然，这里的"先辈层"节点，指的是那些已经有了值的节点。  
（3）当只有一个节点的"固定"以后，其值才能够向其父节点传递。  
（4）α-β剪枝方法搜索得到的最佳走步与极小极大方法得到的结果是一致的，α-β剪枝并没有因为提高效率，而降低得到最佳走步的可能性。  
（5）在实际搜索时，并不是先生成指定深度的搜索图，再在搜索图上进行剪枝。如果这样，就失去了α-β剪枝方法的意义。在实际程序实现时，首先规定一个搜索深度，然后按照类似于深度优先搜索的方式，生成节点。在节点的生成过程中，如果在某一个节点处发生了剪枝，则该节点其余未生成的节点就不再生成了。

3.2 α-β剪枝搜索过程(如图2)  
在搜索过程中，假定节点的生成次序是从上到下，从左到右进行的。图中带圈的数字，表示节点的计算次序，在叙述时，为了表达上的方便，该序号也同时表示节点。当一个节点有两个以上的序号时，不同的序号，表示的是同一个节点在不同次序下计算的结果。



四、实验代码

#include<iostream>

using namespace std;

int num=0; //记录棋盘上棋子的个数

int p,q;

int tmpQP[3][3]; //表示棋盘数据的临时数组，其中的元素0表示该格为空，

int cur[3][3]; //存储当前棋盘的状态

const int depth=3; //搜索树的最大深度

void Init() //初始化棋盘状态

{

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

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

{

cur[i][j]=0;

}

}

void PrintQP() //打印棋盘当前状态

{

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

{

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

cout<<cur[i][j]<<'\t';

cout<<endl;

}

}

void UserInput()//用户通过此函数来输入落子的位置，比如：用户输入31，则表示用户在第3行第1列落子。

{

int pos,x,y;

L1: cout<<"Please input your qizi (xy): ";

cin>>pos;

x=pos/10,y=pos%10;

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

{

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

}

else

{

cout<<"Input Error!";

goto L1;

}

}

int CheckWin() //检查是否有一方赢棋（返回 0：没有任何一方赢；1：计算机赢；-1：人赢）

{ //该方法没有判断平局

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

{

if(cur[i][0]==1&&cur[i][1]==1&&cur[i][2]==1)

{

return 1;

}

if(cur[i][0]==-1&&cur[i][1]==-1&&cur[i][2]==-1)

{

return -1;

}

}

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

{

if(cur[0][i]==1&&cur[1][i]==1&&cur[2][i]==1)

{

return 1;

}

if(cur[0][i]==-1&&cur[1][i]==-1&&cur[2][i]==-1)

{

return -1;

}

}

if((cur[0][0]==1&&cur[1][1]==1&&cur[2][2]==1)||(cur[2][0]==1&&cur[1][1]==1&&cur[0][2]==1))

{

return 1;

}

if((cur[0][0]==-1&&cur[1][1]==-1&&cur[2][2]==-1)||(cur[2][0]==-1&&cur[1][1]==-1&&cur[0][2]==-1))

{

return -1;

}

return 0;

}

int value()//评估当前棋盘状态的值（同时可以用p或q判断是否平局）

{

p=0;

q=0;

for(int i=0;i<3;i++) //计算机一方

{ //将棋盘中的空格填满自己的棋子，既将棋盘数组中的0变为1

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

{

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

{

tmpQP[i][j]=1;

}

else

{

tmpQP[i][j]=cur[i][j];

}

}

}

for(i=0;i<3;i++) //计算共有多少连成3个1的行

{

p+=(tmpQP[i][0]+tmpQP[i][1]+tmpQP[i][2])/3;

}

for(i=0;i<3;i++) //计算共有多少连成3个1的列

{

p+=(tmpQP[0][i]+tmpQP[1][i]+tmpQP[2][i])/3;

}

p+=(tmpQP[0][0]+tmpQP[1][1]+tmpQP[2][2])/3;//计算共有多少连成3个1的对角线

p+=(tmpQP[2][0]+tmpQP[1][1]+tmpQP[0][2])/3;

for(i=0;i<3;i++) //人一方

{ //将棋盘中的空格填满自己的棋子，既将棋盘数组中的0变为-1

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

{

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

{

tmpQP[i][j]=-1;

}

else

{

tmpQP[i][j]=cur[i][j];

}

}

}

for(i=0;i<3;i++) //计算共有多少连成3个-1的行

{

q+=(tmpQP[i][0]+tmpQP[i][1]+tmpQP[i][2])/3;

}

for(i=0;i<3;i++) //计算共有多少连成3个1的列

{

q+=(tmpQP[0][i]+tmpQP[1][i]+tmpQP[2][i])/3;

}

q+=(tmpQP[0][0]+tmpQP[1][1]+tmpQP[2][2])/3;//计算共有多少连成3个1的对角线

q+=(tmpQP[2][0]+tmpQP[1][1]+tmpQP[0][2])/3;

return p+q; //返回评估出的棋盘状态的值

}

int cut(int &val,int dep,bool max)//主算法部分，实现a-B剪枝的算法，val为上一层的估计值，dep为搜索深度，max记录上一层是否为极大层

{

if(dep==depth||dep+num==9) //如果搜索深度达到最大深度，或者深度加上当前棋子数已经达到9，就直接调用估计函数

{

return value();

}

int i,j,flag,temp; //flag记录本层的极值，temp记录下层求得的估计值

bool out=false; //out记录是否剪枝，初始为false

/\*if(CheckWin()==1) //如果计算机赢了，就置上一层的估计值为无穷（用很大的值代表无穷）

{

val=10000;

return 0;

}\*/

if(max) //如果上一层是极大层，本层则需要是极小层，记录flag为无穷大；反之，则为记录为负无穷大

{

flag=10000; //flag记录本层节点的极值

}

else

{

flag=-10000;

}

for(i=0;i<3 && !out;i++) //双重循环，遍历棋盘所有位置

{

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

{

if(cur[i][j]==0) //如果该位置上没有棋子

{

if(max) //并且上一层为极大层，即本层为极小层,轮到用户玩家走了。

{

cur[i][j]=-1; //该位置填上用户玩家棋子

if(CheckWin()==-1) //如果用户玩家赢了

{

temp=-10000; //置棋盘估计值为负无穷

}

else

{

temp=cut(flag,dep+1,!max); //否则继续调用a-B剪枝函数

}

if(temp<flag) //如果下一步棋盘的估计值小于本层节点的极值，则置本层极值为更小者

{

flag=temp;

}

if(flag<=val) //如果本层极值已小于上一层的估计值，则不需搜索下去，剪枝

{

out=true;

}

}

else //如果上一层为极小层，即本层为极大层,轮到计算机走了。

{

cur[i][j]=1; //该位置填上计算机棋子

if(CheckWin()==1) //如果计算机赢了

{

temp=10000; //置棋盘估计值为无穷

}

else

{

temp=cut(flag,dep+1,!max);//否则继续调用a-B剪枝函数

}

if(temp>flag)

{

flag=temp;

}

if(flag>=val)

{

out=true;

}

}

cur[i][j]=0; //把模拟下的一步棋还原，回溯

}

}

}

if(max) //根据上一层是否为极大层，用本层的极值修改上一层的估计值

{

if(flag>val)

{

val=flag;

}

}

else

{

if(flag<val)

{

val=flag;

}

}

return flag; //函数返回的是本层的极值

}

int main()//主程序

{

int m=-10000,val=-10000,dep=1; //m用来存放最大的val

int x\_pos,y\_pos; //记录最佳走步的坐标

Init();

cout<<"Qipan: "<<endl;

PrintQP();

char IsFirst;

cout<<"Do you want do first?(y/n)";

cin>>IsFirst;

while(IsFirst!='y'&&IsFirst!='n')

{

cout<<"ERROR!"<<"Do you want do first?(y/n)";

cin>>IsFirst;

}

if(IsFirst=='n')//---------------------------------计算机先走--------------------------------------

{

L5: for(int x=0;x<3;x++)

{

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

{

if(cur[x][y]==0)

{

cur[x][y]=1;

cut(val,dep,1);//计算机试探的走一步棋，棋盘状态改变了，在该状态下计算出深度为dep-1的棋盘状态估计值val

if(CheckWin()==1)

{

cout<<"The computer put the qizi at:"<<x+1<<y+1<<endl;

PrintQP();

cout<<"The computer WIN! GAME OVER."<<endl;

return 0;

}

if(val>m) //m要记录通过试探求得的棋盘状态的最大估计值

{

m=val;

x\_pos=x;y\_pos=y;

}

val=-10000;

cur[x][y]=0;

}

}

}

cur[x\_pos][y\_pos]=1;

val=-10000;

m=-10000;

dep=1;

cout<<"The computer put the qizi at:"<<x\_pos+1<<y\_pos+1<<endl;

PrintQP();

cout<<endl;

num++;

value();

if(p==0)

{

cout<<"DOWN GAME!"<<endl;

return 0;

}

UserInput(); //玩家走一步棋

PrintQP();

cout<<endl;

num++;

value();

if(p==0)

{

cout<<"DOWN GAME!"<<endl;

return 0;

}

if(CheckWin()==-1)

{

cout<<"Conguatulations! You Win! GAME OVER."<<endl;

return 0;

}

goto L5;

}

else //--------------------------------人先走-----------------------------------

{

L4: UserInput();

PrintQP();

cout<<endl;

num++;

value();

if(q==0)

{

cout<<"DOWN GAME!"<<endl;

return 0;

}

if (CheckWin()==-1)

{

cout<<"You Win! GAME OVER."<<endl;

return 0;

}

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

{

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

{

if(cur[x][y]==0)

{

cur[x][y]=1;

cut(val,dep,1);

if(CheckWin()==1)

{

cout<<"The computer put the qizi at:"<<x+1<<y+1<<endl;

PrintQP();

cout<<"The computer WIN! GAME OVER."<<endl;

return 0;

}

if(val>m)

{

m=val;

x\_pos=x;y\_pos=y;

}

val=-10000;

cur[x][y]=0;

}

}

}

cur[x\_pos][y\_pos]=1;

val=-10000;

m=-10000;

dep=1;

cout<<"The computer put the qizi at:"<<x\_pos+1<<y\_pos+1<<endl;

PrintQP();

cout<<endl;

num++;

value();

if(q==0)

{

cout<<"DOWN GAME!"<<endl;

return 0;

}

goto L4;

}

return 0;

}

五、实验结果

