**ai五子棋实验报告**

---报告人：陈钰江

**目录：**

1. 主要数据结构；
2. 主要算法；
3. 核心算法实现；
4. 评估函数；
5. 测试、分析；
6. 探索、尝试；
7. **主要数据结构：**

**博弈树：存储棋盘关系；**

**二维数组：存储棋盘信息；**

//当前棋盘的子节点

typedef struct children{

vector<chess\_map> vec;

}children;

**阐述：**

**Chess\_map ：二维数组，存储棋盘信息；**

**Children ： 为当前棋盘节点的子节点；**

////当前棋盘状态，模拟的子棋盘状态

map<chess\_map,children> mp;

**阐述：**

**使用c++map类模板，将节点与子节点关联起来；**

////存储的是相应节点的父节点，用于反向传播

map<chess\_map,chess\_map> father;

**阐述：**

**创建map类变量father,用于将节点与父节点链接起来；**

1. **主要算法：**
2. **极大极小搜素；**
3. **Alpha\_beta剪枝算法；**
4. **启发式搜索；**
5. **神经网络模拟；**
6. **算法核心实现：**
7. 极大极小搜索and alpha\_beta剪枝算法；

if(player==1) {

for (; it != mp[ans].vec.end(); it++) {

score= ab\_search(depth - 1, \*it, -player, a, b);

if(score>best\_score){

best\_score=score;

father[ans]=\*it;

}

a=max(a,score);

if (a>b) { break;}

}

return best\_score;

}

else {

for (; it != mp[ans].vec.end(); it++) {

score = ab\_search(depth - 1, \*it, -player, a, b);

if(score<best\_score){

best\_score=score;

father[ans]=\*it;

}

b=min(b,score);

if (a>b) { break; }

}

return best\_score;

}

**阐述：**

**迭代器it遍历子节点，直至搜索深度降为0；**

**返回子节点中，“最佳分值”，作为下一次剪枝搜索的上界或下界，用于判断是否剪枝；**

**将选择的最佳节点，用map类father和父节点关联，用于后期模拟神经网络**

1. **启发式搜索：**

if(own\_center.first){return own\_center.second;}

////防

else if(enemy\_center.first){return enemy\_center.second;}

pair<int,int> center;

int cnt=0,p1=0,p2=0;

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

{

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

{

if(x.map[i][j]==player){

cnt++;

p1+=i;

p2+=j;

}

}

}

////控制初始首棋于中心位置

if(!cnt){return make\_pair(WIDTH/2,LENTH/2);}

p1=max(0,p1/cnt);

p2=max(0,p2/cnt);

center.first=p1;

center.second=p2;

return center;

**阐述：**

**原理：参考棋盘上的棋形、棋子分布确定搜索中心，以搜素中心进行一定范围搜索，减少计算；**

**Own\_center和enemy\_center 获取敌我的必胜棋形的棋子中心；**

**遍历获取棋盘上各子的横、纵坐标，取均值，得到棋子中心；**

1. **神经网络模拟 注：非完全神经网络，权重和估值是提前设定的**

it=mp[x].vec.begin();

for (; it != mp[x].vec.end(); it++)

{

//当前层进行权重估分

score\_1=calculate\_score(\*it, player)/2;

int i=\_depth;

\_\_it=\*it;

while(i--){

score\_1+=calculate\_score(father[\_\_it],player\*=-1)/3;

\_\_it=father[\_\_it];

}

if(score\_total<score\_cur) {score\_total=score\_cur;\_it=it;}

}

**阐述:mp[x].vec中存储的是当前棋盘的所有子节点，**

**father[\_\_it]获取的是每层剪枝的“最佳棋盘”，**

**对不同层棋盘进行加权取值，选择均值最高的棋盘路线作为走法选择；**

1. **评估函数：**

**评估因素：**

1. **棋形： 如活四、冲四、活三、眠三等，统计棋盘上形成的棋形个数及类别，分别进行赋分**
2. **棋子连通性： 己方棋子相连的越多，往往优势更大，所以统计棋盘上的棋子相连情况，进行赋分；**
3. **棋子的“目数”：由围棋引入而来，相连棋子的四周空格越多，棋子的效用更大，进行赋分；**

**注：目数评分<<连通性评分<<棋形平分；**

**代码：**

**连通性评分**

int link\_evaluate(chess\_map x,int player){

int board[WIDTH][LENTH]={0};

int count=0;

#define grade 1

for(int i=0;i<WIDTH;i++){

for(int j=0;j<LENTH;j++){

if(x.map[i][j]){

int n=i,m=j;

board[i][j]=1;

while(m<LENTH&&x.map[n][m++]==player){count++;board[n][m]+=1;}

**遍历棋盘，从四个方向进行统计连接性，进行赋分；**

**目数估分：**

**int \*\*board=(int\*\*)malloc(sizeof(int)\*LENTH\*WIDTH);**

**int count=0;**

**for(int i=0;i<WIDTH;i++){**

**for(int j=0;j<LENTH;j++){**

**if(x.map[i][j]==player&&board[i][j]==0){**

**count+=\_\_centerlise(LENTH, board, x, player, make\_pair(i, j));**

**}**

**}**

**}**

**free(board)**

**return count;**

**阐述：遍历棋盘，统计空格数，进行目数估分；**

**棋形估分：**

vector<vector<int>> chess\_record={//冲4

{0,player,player,player,player,0},//1

//眠四

{-player,player,player,player,player,0},{0,player,player,player,0,player,0},{0,player,player,0,player,player,0},//4

//活三

{0,player,player,player,0},{0,player,player,0,player,0},//6

//眠三

**阐述：chess\_record用于存储棋形**

for ( it=chess\_record.begin(); it != chess\_record.end(); it++) {

for(int m=max(0,i-4);m<=min(WIDTH-1,i+4);m++) {

flag1=0,ind=0;

if (ans.map[m][j] == (\*it)[ind]) {

int k = m;

int flag=1;

if(flag) {

for (ind = 1; ind < (\*it).size()&& m<=min(WIDTH-1,i+4)&&(board[m][j]!=1||it1!=it);m++,ind++) {

if (ans.map[m][j] != (\*it)[ind])

{m=k;flag1+=1;break;}

}

if(ind==(\*it).size())flag=0;

}

**阐述：对遍历到的己方棋子，从四个方向进行棋形核对，每个方向进行一次棋形颠倒核对，即可实现八个方向的核对；**

1. **测试、分析；**

**1.经过多次优化，ai的棋力相对之前有了较好的改观**

**2.但每一步棋所花的时间较长；**

**3.评估函数，提升很大，仍存在不合理之处，部分棋局显得呆板**

**4.神经网络过于呆滞，不能动态加权估分；**

1. **探索、尝试；**

**1.评估函数优化：**

仅有棋形一个评估因素，发现ai表现较为死板，

1.引入了棋子的连通性；

2. 又从围棋中引入目数；

**2.启发式搜索优化：**

原本仅考虑棋子在棋盘上的分布，来确定落子中心，导致关键局部位置无法遍历到；

1. 在原基础上，引入关键棋形来确定落子中心。

**3.alpha\_beta剪枝与神经网络模拟的融合应用**

由于alpha\_beta剪枝的“默认最佳”，导致只能在有限个最终棋盘中选择“最佳的”，不能真正表示智能；

所以引入神经网络，剪枝选取路线的对各层棋盘进行评分加权求和，从而扩大了选择范围，也更贴近现实。