关于Quoridor的实验报告

汇报人：张桓 CS 191220156

时间：2020年3月

整体设计

1. 向对方的底线移动 ➡ 寻找最短最合理的路线

2. 合理放置隔板 ➡ 尽可能延长对手的路线
并保证不能把对方或者自己堵死

*对于路线：不论是对手路线还是自己路线，通过BFS出最短的路线，得到最短路线的第一步(即下一步该走到哪里)，以及路线的长度

*对于隔板：主要考虑放置挡板是否合法的问题，放置隔板要先判断是否越界，再遍历vector里的挡板来判断是否重叠

整体设计

首先每一步轮到自己都有两种选择：**放置挡板** 或 **移动棋子**

对于每一个棋局，一开始的想法是先通过搜索双方的最短路径，得到双方最短路径的长度和按照最短路径的下一步目标位置坐标

后来进一步的尝试改进，通过搜索其实可以得到更多的可以利用的信息，比如最短路径的每一步是如何移动的，这样就改动了我之前的数据结构node，节点里变成了一个位置数组

```
struct node
{
    Location nextstep[100];
    int allsteps;
};
```

整体设计

初始想法：

*行棋条件：如果对方的最短路径比我的最短路径要长或者相等，就选择移动棋子，棋子的下一步位置由搜索出的最短路径得到

*放板条件：如果对方的最短路径比我的最短路径要短，就按照敌人的最短路径放置挡板

整体设计

怎么放板：

开始的想法比较简单，就是得到对手的最短路径后，判断他的下一步方向，直接在对手的下一步位置放置挡板挡住，这是因为之前的搜索函数只返回最短路径的第一步，

但是考虑到更多的信息，搜索函数换成可以返回最短路径的全部位置，放板就不仅仅只考虑第一步了，而是遍历尝试对手路径上可能放置的所有板子位置，并且逐个打分，返回对我最有利的板子位置

挡板得分=我的最短路径/对手的最短路径

```
Location enemystep[100];//记录敌人的路径
for (int i = 0; i < 100; i++)
    enemystep[i] = move(enemyLoc, enemytargetY).nextstep[i];
int score=0,minscore=10000;//每一步得到的优秀板子的分数以及最好板子的分数
BlockBar resultbar,bestbar;//每一步的板子以及最后最好的板子位置
for(int i=1;i<=move(enemyLoc,enemytargetY).allsteps;i++)
{
    bool flag1 = 0, flag2 = 0;
    double score1 = 0, score2 = 0;
    BlockBar bar1, bar2;//记录每一步的可能的两种板子的可行性分数以及位置
    Location lastLocation;//记录当前步的上一步位置
    if (enemystep[i + 1].x == -1 && enemystep[i + 1].y == -1)
        lastLocation = enemyLoc;
    else
```

整体设计

但是进一步思考，如果对手是先手的话，这样一来对手的路径长度是一开始就比我有优势，此时按照之前的策略我会选择一直放置板子。

考虑到对手也可能是搜索比较双方路径放置来判断是否放置板子，放置板子后对手也很可能挡住我的最短路径。而我本身就有后手劣势，路径长，最终板子用完我会直接毙命，因此考虑不应该我的路径一比对方长就选择放置板子，按我的理解是不应该立马激怒对方而断了自己的前程。

而且如果这样我的后手确实输给baseline了

整体设计

考虑到这些，我的想法是搜索得到对方的路径长度，如果长度还是比较长的话就先不去放挡板挡他，而是选择移动棋子，虽然这个长度的并没有很明确的判断标准。

但是和baseline的多次实验对战后最后得到了8步这个标准，如果对手的路径短到8步以内，对我的威胁很大了，我再去放置挡板拦截，这样我确实可以战胜baseline了

```
if (mylen <= enemylen||enemylen>=8)                    //如果我的路径更
{
    int tem = move(newChange.myLoc, mytargetY).allsteps;
    Location moveloc = move(newChange.myLoc, mytargetY).nextstep[tem];
    //这种情况也有最短路径存在但是被对手堵住
    step.myNewLoc = helpmove(newChange.myLoc, moveloc, newChange.enemyLoc);
    if (step.myNewLoc.x == -1 &&step.myNewLoc.y==-1 && blockcounts > 0)//敌
    {
        BlockBar tem = putblock(newChange.myLoc, newChange.enemyLoc);
            step.myNewBlockBar = tem;
            step.myNewBlockBar.normalization();
            blocks.push_back(step.myNewBlockBar);
            blockcounts--;
    }
```

实现框架

数据设计：

```
vector<BlockBar> blocks//存放所有挡板
int mytargetY//我的目标纵坐标
int enemytargetY//敌人的目标纵坐标
int blockcounts//我的剩余挡板数
struct node//move函数返回数据
{
    Location nextstep[100];//最短路的下一步位置
    int allsteps;//最短路的长度
}
```

实现中途

实现时困扰比较大的问题

node move(const Location& myLoc, int mytarget)

按照选手的当前位置坐标和该选手的目标纵坐标，通过广度优先搜索搜索出选手的最短路径的下一步位置以及最短路径的长度，考虑所有挡板但不考虑对手的位置，并通过node型数据返回

这其中包含了对于下一步位置的合法性判断，越界或被挡住

图为判断下一步是否合法，只考虑挡板和越界

```
bool directions[4];
directions[0] = last.y < SIZE;          // 是否可以向上走
directions[1] = last.y > 1;                          // 是否可以向下走
directions[2] = last.x > 1;                          // 是否可以向左走
directions[3] = last.x < SIZE;          // 是否可以向右走
for (auto block : blocks) {                        // 遍历挡板列表，找到被阻挡的方向
    if (block.isH()) {                               // 水平方向挡板，start 与 stop 的 y 相等
        if (block.start.x == last.x - 1 ||
            block.start.x == last.x - 2) {           // 可能挡路的挡板
            if (block.start.y == last.y) {
                directions[0] = false;
            }
            else if (block.start.y == last.y - 1) {
                directions[1] = false;
            }
        }
    }
    if (block.isV()) {                               // 竖直方向挡板，start 与 stop 的 x 相等
        if (block.start.y == last.y - 1 ||
            block.start.y == last.y - 2) {           // 可能挡路的挡板
            if (block.start.x == last.x) {
                directions[3] = false;
            }
            else if (block.start.x == last.x - 1) {
                directions[2] = false;
            }
        }
    }
```

实现中途

在广搜时判断某一点是否已经走过时发现总出现内存访问冲突的情况，后来发现是数组开小了，表示棋盘9*9掐头去尾开了11*11的数组就可以了

```
node tem;//要返回的结构体，包括第一步位置以及最短路径的长度
tem.allsteps = 0;
bool havegone[11][11] = { 0 };//记录点是否走过
Location before[11][11] = {0};//记录上一个节点的坐标
havegone[myLoc.x][myLoc.y] = 1;
```

演 示 完 毕　感 谢 观 看

汇 报 人： 张 桓