# Quoridor AI

**目录**

1. 主要算法
2. 主要函数及其功能
3. 算法增强
4. 特殊情况处理
5. **主要算法**

贪心算法，采用评估函数分别对筛选出的最佳挡板、最佳下一步位置进行评分，选择评分高的操作。

1. **主要函数及其功能**

**Location seek\_path(const QuoridorUtils::Location myLoc, int& min\_step,int target,Location exclude);//寻找最短获胜路径，返回下一步最佳位置**

采用广度优先算法（BFS)

**Step1：**假设存在一个空的搜索队列Queue,首先将当前位置A添加进队列Queue

**Step2：**判断队列第一个节点是否属于目标底线，若不是，则将第一个节点的直接子节点（即可以移向的下一位置）添加进队列，并将第一个节点涂黑（以后不再搜索该节点）

**Step3：**重复判断，直到第一个节点为目标位置，或者队列为空（即代表路径被堵死）

数据结构：

struct node {

int x , y ;//坐标

node \* next ;//头部插入，自动反向，便于查找第一个位置

};

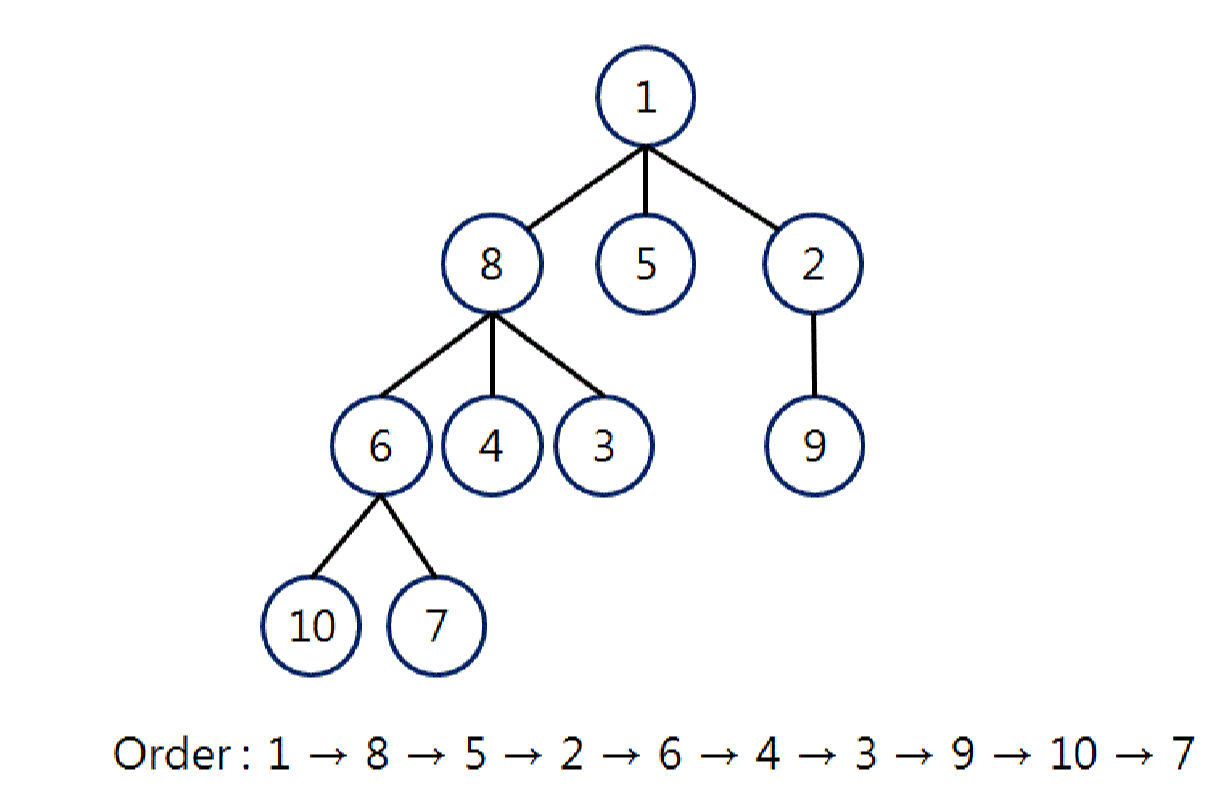
node queue[1000];//结构数组queue存储node

BFS算法每计算一次，都将经过的点颜色涂黑加入队列，使用头部插入法加入链表。

找到最短路径后，读取链表，找到倒数第二个节点对应的坐标，即最短路径上下一步的位置。

增加功能：输入exclude位置后，搜索时跳过该坐标

算法图例：



**bool set\_block(QuoridorUtils::BlockBar new\_blo);//判断在new\_blo放置挡板是否合法**

**检查：**

是否超出边界？：bool isNan()判断

什么方向放置？：bool isV(), isH()判断

是否与其他挡板冲突：将欲添加的挡板分为两段“半挡板”，遍历存储全部挡板信息的vector，检查是否有一段或两段“半挡板”被已存在的挡板覆盖

示例：检查水平方向的挡板是否合法

是否堵死路径：放置后查找敌我方能否找到获胜最短路径

**void evaluate(const ChessboardChange& newChange, Location& next\_loc,**

**QuoridorUtils::BlockBar& next\_bar, double &mark\_bar, double &mark\_step);//评估函数**

总体思路：

**Step1：**根据最短路径，找到下一步行走时的目的地next\_loc

**Step2：**遍历棋盘，依次在合法的位置放置挡板，并计算放置后自己与敌方获胜的最短路径变化；找出最佳的挡板位置next\_bar

**Step3：**分别对行走、放置挡板两个操作进行评分，选择评分高的操作

**评估标准：**

**参考因素：**己方所剩木板数、敌方所剩木板数、己方与敌方获胜最短路径步数、放置挡板后敌我双方获胜最短路径差

**评估方式：**

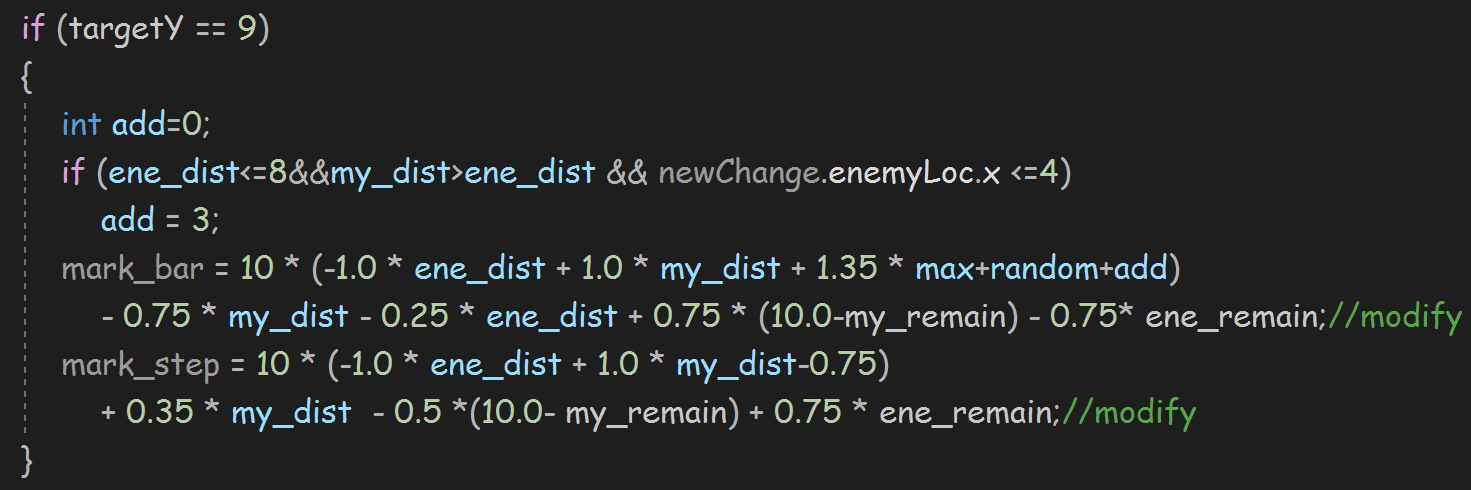
己方与敌方获胜最短路径的差权重最大，双方剩余木板数目权重最小；计算出两种操作的分数后，选择分数更高的操作。

**放挡板评分：**先选出放置挡板的最佳位置，即（敌方获胜最短步数 - 己方获胜最短步数） 增加量最大处；（己方获胜最短步数 – 敌方获胜最短步数）增加量权重最大。

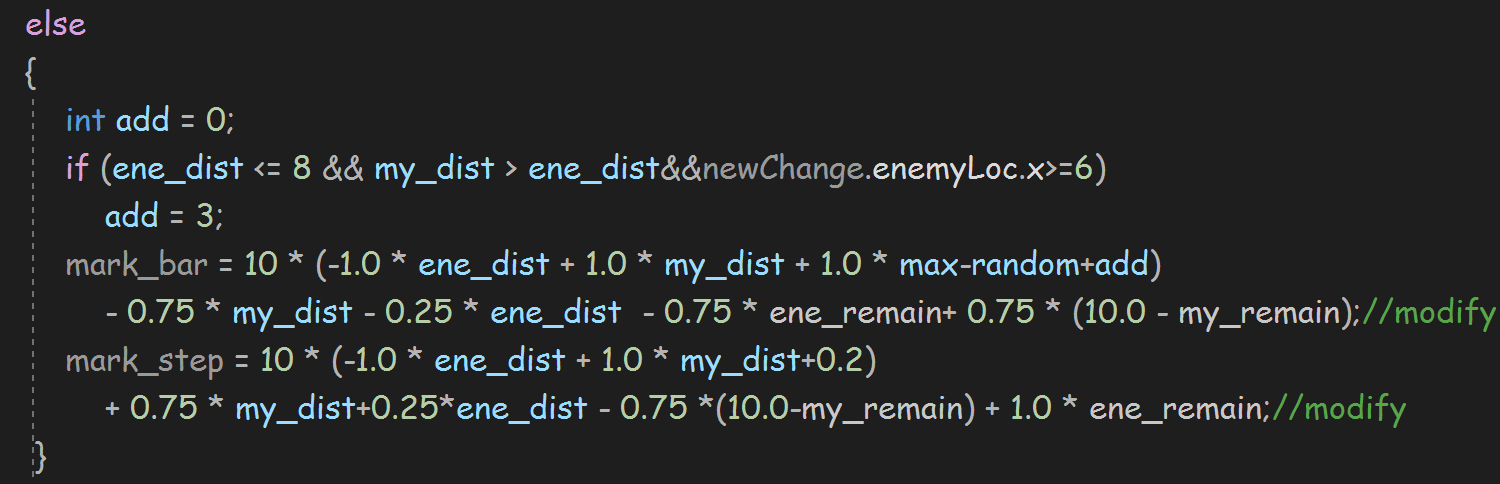
**行走评分：**先找出朝获胜最短路径的方向行走的下一步位置，再进行评分。评分时（己方获胜最短步数 – 敌方获胜最短步数）权重最大

**先手评分函数：**

自己先手时，更倾向于先走棋子，后放板子。所以在给板子评分时，给放了板子后，敌我双方获胜距离增加量最大的值赋予更高的权重。同时，在自己离获胜较远时，优先放棋子。自己所剩余板子越少，敌方剩余板子越少，越倾向于放板子。当敌方十分接近获胜时，自己处于劣势的情况下，人为通过add给放板子加分，尽量放板子。当首局先手失败后，第三局时人为通过random给放板子加分，策略改为倾向于先放板子，后走棋子。



**后手评分函数：**

自己后手时，更倾向于先放板子，后走棋子。所以在给板子评分时，只给放了板子后，敌我双方获胜距离增加量最大的值赋予普通的权重。同时，在自己离获胜较远时，优先放棋子。自己所剩余板子越少，敌方剩余板子越少，越倾向于放板子。当敌方十分接近获胜时，自己处于劣势的情况下，人为通过add给放板子加分，尽量放板子。当首局先手失败后，第三局时人为通过random给放板子减分，策略改为倾向于先走棋子，后放板子。

**void possible\_way(const QuoridorUtils::Location& myLoc, bool directions[]);//寻找该位置处能够行走的方向**

**是否超出棋盘范围？**

将当前坐标与棋盘边界坐标进行比较，判断下一步的合法走向

**是否穿越挡板？**

遍历存储全部挡板信息的vector，先根据坐标挑选出当前位置周围的挡板，再逐一判断遮挡哪一方向的路径

**是否穿越敌方？**

移动完成后，将当前位置与敌方位置进行比对

判断是否被水平方向挡板遮挡的示例：



**bool is\_dirty(const ChessboardChange& newChange, BlockBar& anti\_dirty\_bar);//检查是不是脏套路**

**针对套路：**敌方依次放下板子1-8，而通常的算法会在第8块板子放下来之前一直往上走，掉入陷阱



**检查方式：**

己方步数在5-7步时进行检查：遍历地方前方一排的挡板，在合法处添加挡板，若添加挡板后己方获胜路径增加量>8，则可确认为陷阱。

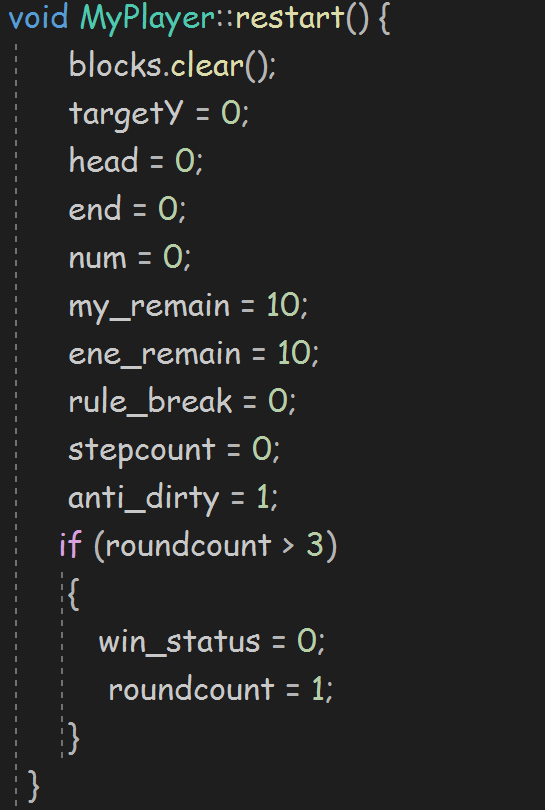
**解决方法：**

在右侧通道处放置挡板来阻断。

**void MyPlayer::restart()**

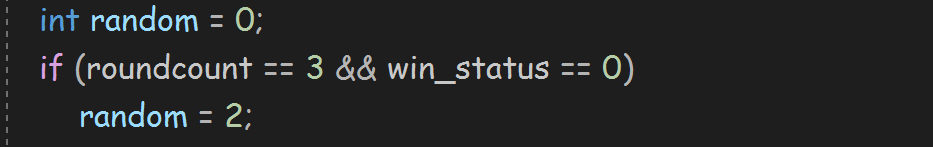
将自己定义的各种全局变量恢复成初始值

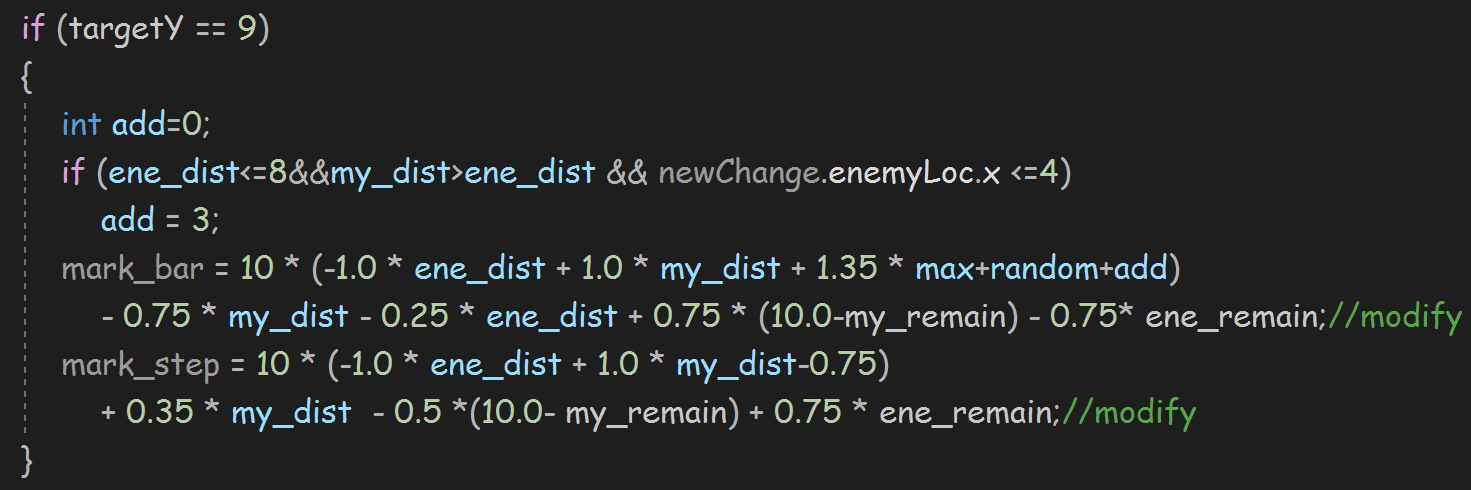
其中roundcount部分，当三个回合结束后，重新从第一个回合开始计算

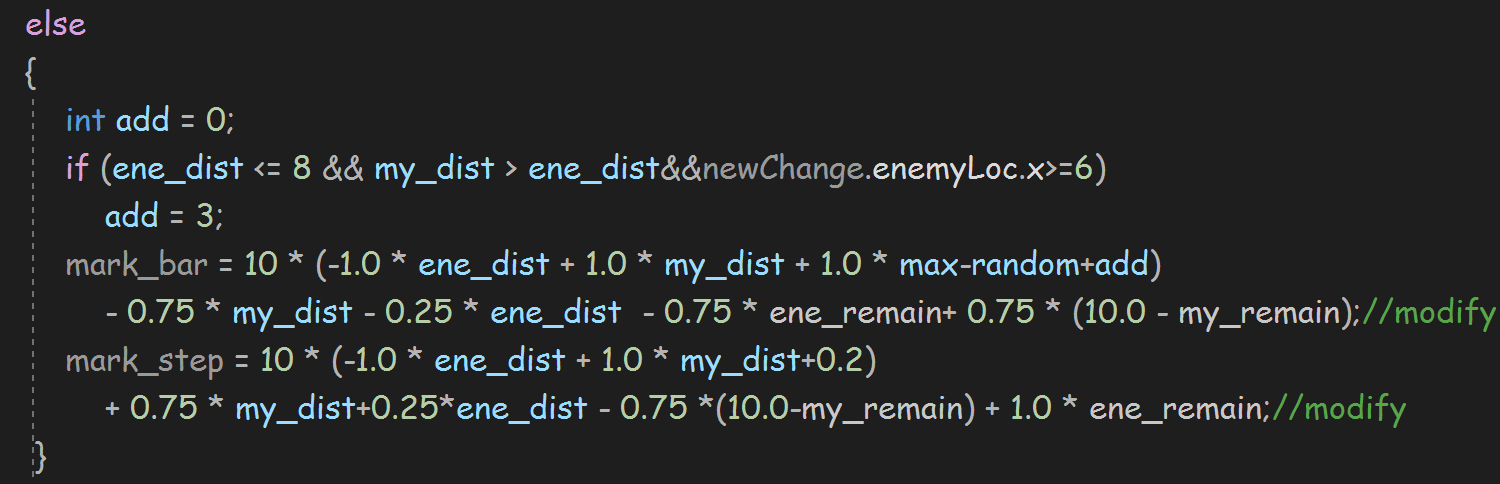


1. **算法增强**
2. 增加评估函数的动态调整

由于是三局两胜制，如果第一局自己输了，在双方均为固定套路下（普遍情况），第三局一定为负。所以可以记录第一局的胜负情况，如果第一局胜利，则第三局使用相同评估函数；如果第一局失败，则对第三局使用的评估函数进行调整。例如，第一句的评估函数更倾向于先放板子，则第三局更换为更倾向于先走步的评估函数。







2. 增加了主动犯规策略

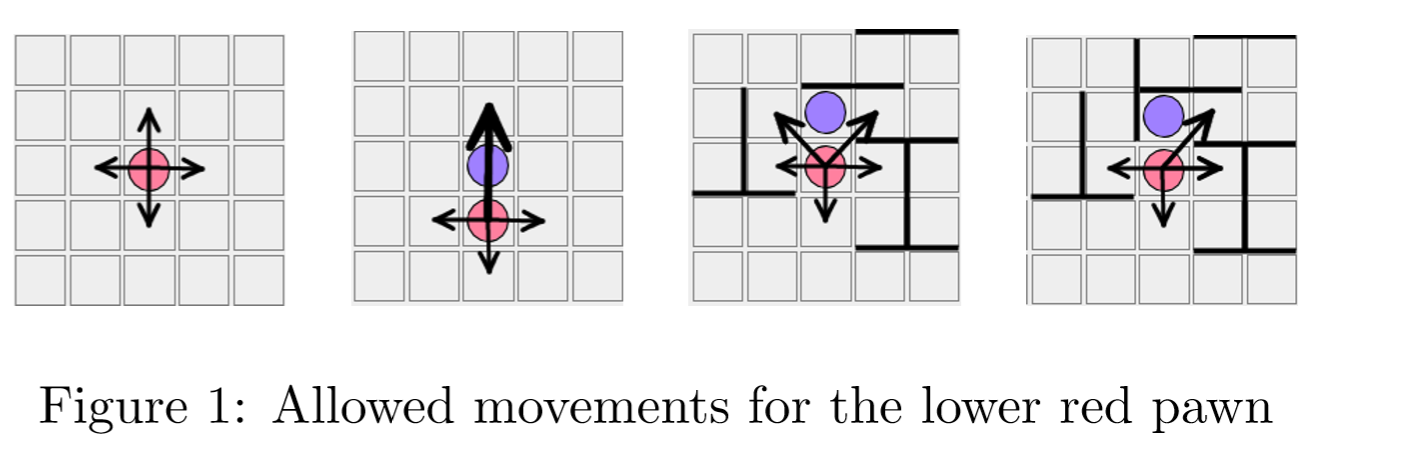
如果自己的最佳下一个位置是敌方所在位置而自己不能跳过去时，会优先选择放隔板。如果隔板已用完且犯规次数为0，选择在原地不动，主动犯规一次。否则选择后退或走向其他方向



4. 增加了对特殊套路的针对处理（详见第三部分）

**4. 特殊情况处理**

1. 若出现图中情况，不能斜向走，只能向左右或向下行走或放置挡板



（2）若自己放置挡板后，处在敌方获胜的唯一路径上，则不会放挡板，只会行走

（3）若自己只能后退时，尽量选择战术违规，留在原地