北京大学 信息科学技术学院 2019 级 李思哲 1900013061

计算概论作业 亚马逊棋

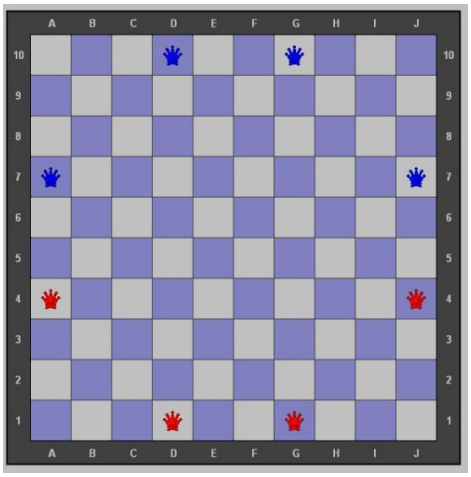
实验报告

北京大学 信息科学技术学院
2019 级 本科生
李思哲 1900013061
指导教师：刘譞哲
助教：杨程旭
2019.12

一、亚马逊棋简介

1. 概述

亚马逊棋（Game of the Amazons），由阿根廷人 WalteZamkauska 在 1988 年推出的两人棋类，是奥林匹亚电脑游戏程式竞赛的比赛指定棋类，由于局面过于复杂，仅第一步就有 2176 种走法，故该棋类多不用于人类之间比赛，而是用于计算机博弈相关方面的比赛与研究 。



2. 棋具简介

（1） 棋盘制式为 8*8 方格棋盘；

（2） 每方各 4 枚棋子，以黑白两色区分敌我；

（3） 另有 92 枚其他色棋子双方共用，又称为箭（或障碍），未被棋子和箭所占据的位置成为空地。

3. 规则

（1） 在 8*8 的棋盘上，每方有四个棋子；每个棋子都相当于国际象棋中的皇后，它们的行棋方法与皇后相同，可以在八个方向上任意行走，但不能穿过阻碍；

（2） 当轮到一方行棋时，此方只能而且必须移动四个棋子中的一个，并在移动完成后，由当前移动的棋子释放一个障碍，障碍的释放方法与棋子的移动方法相同，同样障碍的放置也是必须的；

（3） 当某方完成某次移动后，对方四个棋子均不能再移动时，对方将输掉比赛；

（4） 整个比赛中双方均不能吃掉对方或己方的棋子或障碍。

二、撰写日志

2019.11.24 开工了，阅读游戏说明，画好了棋盘，游戏规则基本实现，在 npy 的帮助下改了一个很蠢的 bug，现在可以在本地让两个人类玩家下棋

2019.11.25 写好了第一版 AI，无脑搜索，计算灵活度，目测只能打过随机数，连我自己都能打赢它

2019.11.26 第二版 AI 完成，评估函数中加入了 Queenmove 和 Kingmove 参数，系数还需要调整

2019.11.27 优化了参数，评估函数中的系数随棋局进行而变化

2019.11.28 调整了棋盘的一些参数，现在的棋盘没有那么那么丑了，本地版界面更加人性化

2019.12.02 函数中加入了 position 参数，但是现在很不稳定

2019.12.02 20:32 尝试搜了两层，用了简单的剪枝，从第 25 回合开始搜两层，可以保证大部分情况下不超时

2019.12.06 改了本地版关于判断落子是否合法的两个 bug，之前的判断方式若三点共线且最初点在中间，会被程序判断为非法输入

2019.12.09 自学了文件，写好了存盘读盘，但是存盘只能存一次，很不友好；还想再加个提示功能，bot 的强度还需要提高

2019.12.10 换了另一种方式存盘，可以实现随时手动存盘了，从记录落子方式变为记录当前棋盘状态；修改了非法落子死循环的 bug，即非法落子后清空输入流和缓冲区

2019.12.10 22:59 添加了菜单选项，可以随时存盘退出，加入了提示功能，不用手动调用菜单，声明了一个字符型变量，用户体验会好一些

2019.12.14 调参，调整参数之后 bot 的强度稍稍好了一点

2019.12.17 开始给本地版加功能，实现悔棋，悔棋可以一直到当前游戏模式的第一步，现在完全不

用手动下棋了,添加了背景音乐，不过有点蠢

2019.12.18 写好了 restart，其实只要再次调用 playgame 函数，再修改全局变量就行了，现在可以随时重新开始，直接回到开始界面，可以重新选择游戏模式

2019.12.18 13:52 加入了难度选择功能，放了之前的三版 AI 和一个随机版本，打不过终版 bot 也可以打随机玩一玩 2333

2019.12.18 16:10 托管功能实现，游戏功能变得愈发沙雕，可以手动设置托管局数，修改了继续游戏时回合数和当前玩家混乱的 bug

2019.12.19 写报告

三、算法介绍

1. 枚举与搜索算法

对于当前 AI 执子的颜色，枚举所有可能的移步方式，并存入数组。枚举完成后，对于每一种可行的走法，利用评估函数计算若使用该走法后我方的价值，寻找最大价值即为当前最优走法。

2. 贪心算法

对于当前 AI 执子的颜色，枚举所有可能的移步方式，存入数组。枚举完成后，对于每一种可行的走法，利用评估函数分别计算使用该走法后我方与对方价值的差值，利用博弈思想，寻找差的最大值即为当前的最优解。

3. 极大极小搜索与贪心算法（botzone 提交算法）

结合了极大极小搜索及贪心算法。1 与 2 两种算法只能实现对下一步的模拟，而利用极大极小搜索即可实现对下两步甚至更多步的模拟。

对于枚举出的我方的每一种落子方式，模拟落子后枚举当前状态下对方的所有可能落子方式，并假设对方会选择对我方最不利的落子方式。即对于我方每一步的决策，对方会有相应的下一步决策，两步决策之后双方价值差的最小值即为我方该种决策方式的价值；寻找所有最小值中的最大值对应我方的决策方式即为当前局部最优解。

此外，由于两层枚举的时间效率较低，应配合 α—β 剪枝和极大极小搜索。即若发现我方该种落子方式能够对应对方某个落子方式使两步后双方价值差小于当前的最大价值，则我方的这种落子方式一定不是最优，不用继续枚举。

四、评估函数

（一） 自变量：现阶段共五个

1. mobility 特征值

灵活度用来衡量棋子向相邻八个方向移动的灵活性的大小，计算过程如下：

①步骤 1 计算棋盘中所有空格的灵活度值。

②步骤 2 记录棋子 a 采用 Queen 走法时一步之内能到达的空格。

③步骤 3 对步骤 2 记录的每个空格，计算空格的灵活度值除以棋子 a 采用 King 走法到达该空格的步数的值，然后将这些值累加，即得到当前棋子 a 的灵活度。

但是实际实现过程中，由于该过程需要对棋盘进行多次扫描，耗时太多，故采取简化方法，即统计当前棋局双方棋子可行着法数目，将一方的可行着法数目作为该方的灵活度。

此外，游戏过程中还可能存在一种情况，即开具阶段某方的灵活度比对方高得多，但其一个棋子

过早被堵死。为解决这个问题，应在计算灵活度值的同时还计算一个最小灵活度值，并将该值乘以相应的系数加到灵活度评估特征中，这个最小灵活度值是指一方四个棋子中的可行着法数目最小的那个值，这样可以有效缓解一个棋子过早被堵死的问题。

2. territory 特征值

territory 特征值用来衡量双方基于 Queen 以及 King 走法时对空格的控制权的评估值。

对 territory 特征值的计算涉及两种走棋方式，一个是 Queen 走法，另外一个是 King 走法。Queen 走法是指移动的八个方向上只要没有跨越障碍就可以移动，每次可以移动任意步长；而 King 走法是指一次移动距离是 1，也就是只能走步到相邻的空格。Queenmove 和 Kingmove 值代表对于一个空格，某方的四个棋子通过 Queen 和 King 的走法走到此空格最少需要移动多少步。

如上所述，计算每一空格的 QueenMove 和 KingMove 值，大体可以看出棋局的优劣。

territory 计算公式如下：

$$t_i = \sum_a \Delta(D_i^1(A), D_i^2(A)) \quad （i = 1 或 2）$$

$$\Delta(D_i^1(A), D_i^2(A)) = \begin{cases} 0 & (D_i^1(A) = D_i^2(A) = \infty) \\ k & (D_i^1(A) = D_i^2(A) \neq \infty) \\ 2 & (D_i^1(A) < D_i^2(A)) \\ -2 & (D_i^1(A) > D_i^2(A)) \end{cases}$$

注释：

其中 A 代表当前棋局中未被占领的空格，i 取 1 表示基于 Queen 走法，取 2 表示基于 King 走法，$D_1^i(A)$、$D_2^i(A)$ 的值分别表示白方、黑方到空格 A 的 Queenmove 及 Kingmove 值。在第二种情形下，$D_i^1(A) = D_i^2(A) \neq \infty$ 时，对该空格 A 的控制权由先走棋那一方决定，k 代表的是先行方的优势大小，本 AI 中取 1。计算结果 t_i 表示分别采用 Queenmove 或 Kingmove 时，白方和黑方对所有空格占有情况的综合评估得分。

3. Position 特征值

Position 分别表示双方基于 Queen 以及 King 走法相对空格的位置优劣的评估值。

从 t_i 计算公式可以得知基于 territory 的评估仅仅反映了双方对空格的控制权归属，但未能反映对空格的控制权差值大小，以及控制权差值大小所反映的位置特征。实际上双方对一空格控制权的差值能反映出一方是否具有地理位置的优势。距离空格较近一方控制权大，而距离空格较远的一方控制权小，处于地理位置上的劣势。同样地，这种地理位置优劣评估即对于 position 特征值的计算也涉及 Queen 和 King 两种走步方式，可以利用 territory 特征评估中使用的 QueenMove 和 KingMove。

position 特征值 p_i 计算公式为：

$$p_1 = 2\sum_A (2^{-D_1^1(A)} - 2^{-D_1^2(A)})$$

$$p_2 = \sum_A \min(1, \max(-1, (D_2^2(A) - D_2^1(A))/6))$$

（二） 参数

以上t_1、t_2、p_1、p_2、m这五个评估因子都是针对棋局某一方面特征的评估，而亚马逊棋棋局特点在不同比赛阶段差异很大，所以五个评估因子的权重的取值不能固定不变而应该是随着比赛阶段而改变。

具体实现，设置前 12 步为开局阶段，12~41 为中局阶段、42 步以后为终局阶段。通过在 botzone 上的多次实验得到的目前最佳的各阶段评估因子的权重取值如表 1 所示。

表 1 评估因子权重系数取值

	评估特征	Territory		Position		Mobility
	评估因子	t_1	t_2	p_1	p_2	m
	权重系数	a	b	c	d	e
阶段	开局	18	40	78	78	20
	中局	7	6	24	24	2
	残局	9	2	30	30	1

五、模块划分

本地版算法中主要将代码分为四个模块，即**规则实现、菜单选项、AI 决策、游戏模式**，下作详细介绍。

（一）规则实现

1. 简介

可以实现最基本的游戏规则，包括棋盘的状态与游戏基本界面、落子、判断输赢等等，是各种游戏模式的基础。

2. 函数介绍

（1）`void DrawTheBoard();`

画棋盘函数，输出棋盘，每次运用 system（“cls”）进行清屏，同时按照根据输出菜单提示和回合数。

（2）`bool inMap(int, int);`

判断该棋子是否在棋盘内，需要在判断是否符合规则中调用。

（3）`bool check(int);`

检查某种颜色是否有棋可走，是游戏是否结束的依据。只需搜索该方四个棋子周围的八个格子是否有空格，只要能移动棋子就能放置障碍。

（4）`void PutChess(int, int, int, int, int, int, int);`

落子函数。

(5) void turn_normal(int, int, int, int);

将有特殊标记的棋子和障碍变回正常。

(6) bool legal(int, int, int, int, int, int, int);

判断落子是否合法，即要移动的是否是己方的棋子，移动到的位置和防止障碍的位置是否是空格且不相同。

(7) bool obey_the_rule(int, int, int, int, int, int, int);

判断落子是否符合 queenmove 规则，运用枚举法逐个搜索。

3. 重点函数

(1) void turn_normal(int, int, int, int);

由于棋盘输出棋子时采用了宏定义，而对于转态不同的同一方的棋子定义不同。若不将特殊标记的棋子改变回来，会影响对落子合法性及游戏是否已结束的判断，也会导致下一回合中不能输出正常棋子。

(2) bool obey_the_rule(int, int, int, int, int, int, int);

用于判断落子是否符合游戏规则，枚举找到棋子和障碍是在哪一条直线上行进。同时，写的过程中出现了两个问题，首先，判断障碍放置是否合理时需要将已经被移动的棋子移走，即搜索至该棋子所在位置时需要 continue；这样的做法会造成若出现障碍放置和选定的落子目的地一致时障碍会覆盖棋子而不会被判为非法棋步，解决方式是添加一个单独判断。

（二）菜单选项

1. 简介

仅在人机对局时起作用，可选择游戏模式或直接落子，比起按键进入菜单用户体验更好。

2. 函数介绍

(1) void human_decision();

人类决策，类似菜单选项，可直接调用功能或输入移动的坐标。若检测到的第一个字符是规定的字母，则进入该功能；若检测到第一个字符是合法的数字字符，则进入落子功能；否则，提示重新输入。

(2) void save();

存盘函数，记录当前的棋盘状态、回合数、我方执子颜色以及选择的难度。

(3) void regret();

悔棋函数，人类和 AI 的每一步决策都存进全局变量的二维数组中，每次悔棋退回两步，回合数减 2，依次清除落子即可。

(4) void AI_play(int);

托管函数，可以观看两只 AI 对局，托管函数需调用的 continue_game 函数，将在游戏模式部分中具体介绍。

3. 重点函数

void human_decision();

供人类玩家选择下一步想要落子还是调用菜单中的某些功能。为了优化游戏体验，将第一个字符声明为字符型，可省去调用菜单的步骤。若接收到的第一个字符是规定的字母，即可直接调用对应函数实现相应功能；若检测到的第一个字符是合法的数字，则自动进入落子状态；否则，程序提醒非法输入，用 cin.clear 以及 cin.ignore 函数清空输入流和缓冲区，解决鲁棒性问题，防止进入死循环。

（三）AI 决策

1. 简介

AI 利用评估函数决策下一步落子的位置坐标。

2. 函数介绍

（1）int mobility(int);

计算某种颜色的棋子能够到达的空格的数量，返回值为灵活度的值。

（2）void ClearChess(int, int, int, int, int, int, int);

清除模拟落子函数，AI 决策搜索过程中需模拟落子，相当于回溯，注意回溯顺序即可。

（3）ter_posi king_move(int);

计算 Kingmove 移动方式下的 territory 和 position 值，返回值为结构体。

（4）double square(int, int);

计算幂次方，queenmove 的 position 计算时需要。

（5）ter_posi queen_move(int);

计算 Queenmove 移动方式下的 territory 和 position 值，返回值为结构体，具体实现与 king_move 类似，不再多做说明。

（6）void determine_4(int);

AI 难度为 4，运用 mobility，territory，position 作为参数，25 回合后剪枝搜两层。

（7）void determine_3(int);

AI 难度为 3，运用 mobility 和 territory 为参数，搜索一步。

（8）void determine_2(int);

AI 难度为 2，运用 mobility 为参数，搜索一步。

（9）void determine_1(int);

AI 难度为 1，随机落子算法。

（10）void AI_decision(int);

AI 决策函数，根据难度选择调用相应的 determine 函数。

3. 重点函数

（1）int mobility(int);

具体做法：扫描棋盘，找到这种颜色四个棋子的位置，向八个方向进行搜索，一旦遇到障碍便 break，记录能到达的格子数已经四个棋子中灵活度最小的棋子能到达的格子数。

（2）ter_posi king_move(int);

具体做法：用 for 循环扫描棋盘的方式替代宽度优先搜索。即对于某一方，先找到这一方的所有棋子，之后从这些格子出发，找到走一步能到达的格子并标记为 1，依次类推，直到所有能到达之处均被标记。利用公式分别计算 territory 和 position，返回结构体以便调用。

（3）void determine_4(int);

具体做法——

25 回合前：

枚举我方所有可能的移步和放置障碍的方式，存入数组，模拟落子后利用评估函数对双方分别进行评估，计算双方价值的差值。枚举结束后寻找价值差值最大的方案即可。

25 回合后：

枚举我方所有可能的移步和放置障碍的方式，存入数组；对于我方每一种决策，再枚举对方所有可能决策，并继续模拟对方落子。计算双方各进行一步操作后双方价值的差值，寻找价值差值最大的方案。

但是注意到，由于第二步决策是由对方做出的，很显然对方并不会挑选对你最有利的方式，所以，两步后的最大价值差有极大概率无法达到，因此需要引入极大极小搜索。即对于我方的每一种落子方法，假定对方会选择对我方最不利的决策。则应计算在我方的特定决策下，再经过一回合后双方价值差的最小值为我方该种决定方式的价值。取所有最小值中的最大值即为当前我方的局部最优解。

做到这一步，又有一个明显的问题出现——决策超时，此时引入 α－β 剪枝算法。即对于我方的一种决策，枚举对方决策时，若存在对方某种决策使两回合后价值差小于当前的最大值，那么很显然我方该种决策不是最优解，应进行剪枝，没有必要再去枚举对方的剩余决策方式了。

（四）游戏模式

1. 简介

程序一开始时需要人类玩家选择游戏模式，为了方便 restart 的递归，把所有的玩游戏的语句都写进了 play_game 函数，减少 main 函数的语句。

2. 函数介绍

（1）`void play_with_yourself();`

自己和自己下棋，用于两个无聊的人类玩家 。

（2）`void new_game();`

新游戏函数，和 AI 下一盘棋，包括难度选择、颜色选择等功能。

（3）`void continue_game();`

继续游戏函数，在上一局函数和托管函数中调用，是指从当前回合继续下棋。

（4）`void last_game();`

上一局函数，读盘复盘后调用 continue_game 函数。

（5）`void play_game();`

真正的玩游戏函数，可选择游戏模式，在 restart 功能中也需调用。

六、独特性

1. **更加人性化的菜单选项。**

下棋过程中无需特地调用菜单，只需按照要求输入指令即可完成相应操作。

如右图，玩家只需直接输入坐标或按照指示输入相应字母，即可完成对应目的。

2. **悔棋功能**

悔棋可以从当前状态一直到进入当前游戏时的状态（即若选择的模式是“last game”，且存盘时回合数为 7，现在回合数为 13，那么可以一直悔棋直到第 7 回合）；另外，托管功能不支持悔棋。

3. **提示功能**

由 AI 帮助你进行下一步的决策，算法为随机算法

4. 托管功能

可以设置需要托管的回合数，完成相应回合数后自动继续游戏；若未完成相应回合数就已决胜负，则直接退出游戏。托管全过程界面有提示。

为了增强用户的观看体验，托管过程中使用了 sleep 函数，每完成一步移动后停留 0.5 秒，便于用户更清楚地观看现在的棋局。

5. 难度选择功能

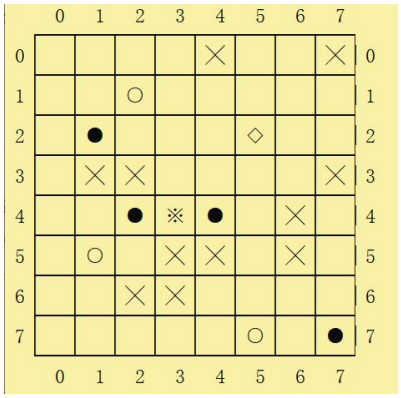
考虑到用户水平的差异，在人机对局的开始设置了难度选择功能，用户可根据需求选择不同算法的 AI 进行对弈。

6. 提示 AI 的移动方式

每完成一次决策，移动的棋子和新放置的障碍用特殊标记标出（●○变为◆◇，╳变为※），便于观察 AI 的移动方式，且下一步落子完成后自动恢复。

该功能可用于正常游戏、悔棋、托管等场景。

悔棋时可自动标记再上一步的移动方式，托管可显示每一步的走法。



7. 背景音乐功能

不同的游戏模式对应不同背景音乐。

七、Bug 修复

1. 关于输入的鲁棒性的问题

问题描述 1：输入格式错误，导致程序无法继续进行

解决方式 1：改用循环进行输入，若输入合法则跳出循环，否则继续提示继续输入，直到合法为止。

问题描述 2：若输入变量类型错误，系统不断提示重新输入，进入死循环，程序崩溃

解决方式 2：利用 cin.clear() 以及 cin.ignore()函数清空输入流和缓冲区，再重新输入

2. 关于继续游戏时判断当前执子玩家的问题

问题描述：继续游戏时，不应该落子的一方再次落子

解决方式：利用回合数的奇偶性判断当前的玩家

附：源代码

```
#include<iostream>
#include<fstream>
#include<Windows.h>
#include<mmsystem.h>
#include<iomanip>
#include<cstdlib>
#include<cstdio>
#include<ctime>
#pragma comment(lib,"winmm.lib")

#define blackone "●"
#define whiteone "○"
#define obstacleone "╳"
#define newblackone "◆"
#define newwhiteone "◇"
#define newobstacleone "※"
#define GRIDSIZE 8
#define black 1
#define white -1
#define new_black 11
#define new_white 9
#define new_obstacle 12
#define OBSTACLE 2

using namespace std;
int board[8][8] = { 0 };
int dx[] = { -1,-1,-1,0,1,1,1,0 };
int dy[] = { 1,0,-1,-1,-1,0,1,1 };
int cycle = 0, me, mode, now_step, difficulty, watch = 0;//me:玩家的颜色，mode:模式，now_step:记录落子的次数便于悔棋，difficulty:难度选择，watch:是否正在观看
int record[60][7] = { 0 };
struct ter_posi {
     int ter;
     double posi;
};

int maxx(int, int); //两个无脑函数
int minn(int, int); //两个无脑函数
void DrawTheBoard(); //画棋盘函数
bool inMap(int, int); //判断该棋子是否在棋盘内
bool check(int); //检查某种颜色是否有棋可走
int mobility(int);//计算某种颜色的棋子能够到达的空格的数量
void PutChess(int, int, int, int, int, int, int); //落子函数
void turn_normal(int, int, int, int); //将有特殊标记的棋子和障碍变回正常
void ClearChess(int, int, int, int, int, int, int); //清除模拟落子函数
bool legal(int, int, int, int, int, int, int); //判断落子是否合法
bool obey_the_rule(int, int, int, int, int, int, int); //判断落子是否符合规则
void human_decision(); //人类决策，类似菜单选项，为了方便写成了函数
ter_posi king_move(int); //计算 Kingmove 移动方式下的 territory 值
double square(int, int); //计算幂次方
ter_posi queen_move(int); //计算 Queenmove 移动方式下的 territory 值
void determine_4(int); //AI 难度为 4
void determine_3(int); //AI 难度为 3
```

```
void determine_2(int); //AI 难度为 2
void determine_1(int); //AI 难度为 1
void AI_decision(int); //AI 决策函数
void save(); // 存盘函数
void regret(); //悔棋函数
void AI_play(int); //托管函数
void play_with_yourself(); //自己和自己下棋
void new_game(); //新游戏函数，和 AI 下一盘棋
void continue_game(); // 继续游戏函数，在上一局函数中调用
void last_game(); // 上一局函数
void play_game(); // 真正的玩游戏函数

int maxx(int x, int y)
{
	if (x > y) return x;
	return y;
}

int minn(int x, int y)
{
	if (x > y) return y;
	return x;
}

void DrawTheBoard()//画棋盘函数，直接调用即可
{
	system("cls");
	cout << endl << endl << "      THE AMAZONS" << endl << endl;
	cout << "    0   1   2   3   4   5   6   7" << endl;
	cout << "  ┌───┬───┬───┬───┬───┬───┬───┬───┐" << endl;
	for (int i = 0; i < 7; i++)
	{
		cout << " " << i;
		for (int j = 0; j < 8; j++)
		{
			if (board[i][j] == 1)
				cout << "│ " << setw(2) << blackone;
			else if (board[i][j] == -1)
				cout << "│ " << setw(2) << whiteone;
			else if (board[i][j] == 2)
				cout << "│ " << setw(2) << obstacleone;
			else if (board[i][j] == 11)
				cout << "│ " << setw(2) << newblackone;
			else if (board[i][j] == 9)
				cout << "│ " << setw(2) << newwhiteone;
			else if (board[i][j] == 12)
				cout << "│ " << setw(2) << newobstacleone;
			else
				cout << "│ " << setw(2) << ' ';
		}
		cout << "│ " << i << " ";
		cout << endl;
		cout << "  ├───┼───┼───┼───┼───┼───┼───┼───┤" << endl;
	}
	cout << " 7";
	for (int j = 0; j < 8; j++)
```

```
    {
        if (board[7][j] == 1)
            cout << "| " << setw(2) << blackone;
        else if (board[7][j] == -1)
            cout << "| " << setw(2) << whiteone;
        else if (board[7][j] == 2)
            cout << "| " << setw(2) << obstacleone;
        else if (board[7][j] == 11)
            cout << "| " << setw(2) << newblackone;
        else if (board[7][j] == 9)
            cout << "| " << setw(2) << newwhiteone;
        else if (board[7][j] == 12)
            cout << "| " << setw(2) << newobstacleone;
        else
            cout << "| " << setw(2) << ' ';
    }
    cout << "| " << 7 << "  ";
    cout << endl;
    cout << "  └────┴────┴────┴────┴────┴────┴────┴────┘" << endl;
    cout << "    0   1   2   3   4   5   6   7" << endl;
    cout << endl;

    if (cycle != 0)
    {
        cout << "       回合数: " << cycle << endl;
        if (mode == 1 || mode ==2)
        {
            if (me == 1)
                cout << "您是黑方" << endl;
            else if(me == -1)
                cout << "您是白方" << endl;
            if (watch)
            {
                cout << "       托管中";
                for (int i = 0; i <= cycle % 4; i++)
                    cout << ".";
                cout << endl;
            }
            else
            {
                cout << "Press E (退出)   Press H (提示) Press R (悔棋)" << endl;
                cout << "Press T (重新开始) Press B (托管)" << endl;
                cout << "或输入坐标" << endl;
            }
        }
    }
    cycle++; //每画一次棋盘，回合数 +1
}//画棋盘函数，直接调用即可

bool inMap(int x, int y)//判断该棋子是否在棋盘内
{
    if (x < 0 || x > 7 || y < 0 || y > 7)
        return false;
    return true;
}//判断该棋子是否在棋盘内
```

```
bool check(int color)//检查某种颜色是否有棋可走
{
	for (int i = 0; i < GRIDSIZE; i++)
	{
		for (int j = 0; j < GRIDSIZE; j++)
		{
			if (board[i][j] != color)
				continue;
			for (int k = 0; k < 8; k++) //找到一个棋子后向八个方向搜索，只要能走就能放障碍
			{
				int xx = i + dx[k], yy = j + dy[k];
				if (!inMap(xx, yy))
					continue;
				if (board[xx][yy] == 0)
					return true;
			}
		}
	}
	return false;
}//检查某种颜色是否有棋可走

int mobility(int color)//计算某种颜色的棋子能够到达的空格的数量，作为灵活度的结果
{
	int mark[GRIDSIZE][GRIDSIZE] = { 0 };
	int total = 0;
	int totalmin = 70, total_one = 0;

	for (int i = 0; i < GRIDSIZE; i++)
	{
		for (int j = 0; j < GRIDSIZE; j++)
		{
			if (board[i][j] != color)
				continue;
			total_one = 0;
			for (int k = 0; k < 8; k++)
			{
				for (int l = 1; l < GRIDSIZE; l++)
				{
					int xx = i + dx[k] * l;
					int yy = j + dy[k] * l;
					if (!inMap(xx, yy) || board[xx][yy] != 0)
						break;
					if (mark[xx][yy] == 0)
					{
						total++;
						total_one++;
						mark[xx][yy] = 1;
					}
				}
			}
			if (total_one < totalmin)
				totalmin = total_one;//计算最小灵活度，防止一开始的时候某个棋子先被堵死
		}
	}
	return total + totalmin;
}//计算某种颜色的棋子能够到达的空格的数量
```

```
void PutChess(int x0, int y0, int x1, int y1, int x2, int y2, int color)//落子函数，落子的过程用全局变量记录，方便悔棋
{
	record[cycle][1] = x0; record[cycle][2] = y0;
	record[cycle][3] = x1; record[cycle][4] = y1;
	record[cycle][5] = x2; record[cycle][6] = y2;
	board[x0][y0] = 0;
	board[x1][y1] = color + 10;
	board[x2][y2] = new_obstacle;
}//落子函数，落子的过程用全局变量记录，方便悔棋

void turn_normal(int x1, int y1, int x2, int y2)//将特殊标记的棋子和障碍变回正常
{
	board[x1][y1] -= 10;
	board[x2][y2] = OBSTACLE;
}// 特殊标记的棋子和障碍变回正常

void ClearChess(int x0, int y0, int x1, int y1, int x2, int y2, int color)//清除模拟落子函数
{
	board[x2][y2] = 0;
	board[x1][y1] = 0;
	board[x0][y0] = color;//注意回溯的顺序，很重要!
}//清除模拟落子函数

bool legal(int x0, int y0, int x1, int y1, int x2, int y2, int color)//判断落子是否合法
{
	if (!inMap(x0, y0) || !inMap(x1, y1) || !inMap(x2, y2))
		return false;
	if (board[x0][y0] != color || board[x1][y1] != 0)
		return false;
	if (board[x2][y2] != 0 && !(x2 == x0 && y2 == y0))
		return false;
	if (x1 == x2 && y1 == y2)
		return false;
	return true;
}//判断落子是否合法

bool obey_the_rule(int x0, int y0, int x1, int y1, int x2, int y2)//判断落子是否符合规则
{
	if (x0 != x1 && y0 != y1 && x0 + y0 != x1 + y1 && x0 - y0 != x1 - y1)//棋子是否在一条直线上移动
		return false;
	if (x1 != x2 && y1 != y2 && x1 + y1 != x2 + y2 && x1 - y1 != x2 - y2)
		return false;
	if (x0 == x1)
		for (int i = minn(y0, y1) + 1; i < maxx(y0, y1); i++)
			if (board[x0][i] != 0)
				return false;
	if (y0 == y1)
		for (int i = minn(x0, x1) + 1; i < maxx(x0, x1); i++)
			if (board[i][y0] != 0)
				return false;
	if (x0 + y0 == x1 + y1)
		for (int i = minn(x0, x1) + 1; i < maxx(x0, x1); i++)
			if (board[i][x0 + y0 - i] != 0)
				return false;
```

```
    if (x0 - y0 == x1 - y1)
        for (int i = minn(x0, x1) + 1; i < maxx(x0, x1); i++)
            if (board[i][i + y0 - x0] != 0)
                return false;

    if (x2 == x1)
        for (int i = minn(y2, y1) + 1; i < maxx(y2, y1); i++)
        {
            if (i == y0 && x2 == x0) continue;//如果越过移动前的位置，直接 continue
            if (board[x2][i] != 0)
                return false;
        }
    if (y2 == y1)
        for (int i = minn(x2, x1) + 1; i < maxx(x2, x1); i++)
        {
            if (i == x0 && y2 == y0) continue;
            if (board[i][y2] != 0)
                return false;
        }
    if (x2 + y2 == x1 + y1)
        for (int i = minn(x2, x1) + 1; i < maxx(x2, x1); i++)
        {
            if (i == x0 && x2 + y2 - i == y0) continue;
            if (board[i][x2 + y2 - i] != 0)
                return false;
        }
    if (x2 - y2 == x1 - y1)
        for (int i = minn(x2, x1) + 1; i < maxx(x2, x1); i++)
        {
            if (i == x0 && i + y2 - x2 == y0) continue;
            if (board[i][i + y2 - x2] != 0)
                return false;
        }
    return true;
}//判断落子是否符合规则

void human_decision()//人类决策函数，选择模式或落子
{
    //if (cycle == 2)
    //{
        //cout << "          回合数：1" << endl << "您是黑方" << endl;
        //cout << "Press E (退出) Press H (提示) Press R (悔棋)" << endl;
        //cout << "Press T (重新开始) Press B (托管)" << endl;
        //cout << "或输入坐标" << endl;
    //}
    char condition;
    int x0, y0, x1, y1, x2, y2;
    cin >> condition;//读入的第一个操作声明为字符型，用户体验比较好
    while (true)
    {
        if (condition == 'H' || condition == 'h')//提示
        {
            now_step++;
            determine_1(me);
            return;
        }
```

```
        else if (condition == 'E' || condition == 'e')//存盘退出
        {
            save();
            exit(0);
        }
        else if ((condition == 'R' || condition == 'r') && cycle >= 4 && now_step > 0)//悔棋
        {
            regret();
            return;
        }
        else if (condition == 'T' || condition == 't')//restart 重新开始，重新选择游戏模式
        {
            save();
            for (int i = 0; i < GRIDSIZE; i++)
                for (int j = 0; j < GRIDSIZE; j++)
                    board[i][j] = 0;
            play_game();
        }
        else if (condition == 'B' || condition == 'b')//托管
        {
            int your_cycle;
            cout << "请输入需要托管的回合数" << endl;
            cin >> your_cycle;
            AI_play(2 * your_cycle);
            return;
        }
        else if (condition >= '0' && condition <= '7')//落子
        {
            x0 = condition - '0';
            cin >> y0 >> x1 >> y1 >> x2 >> y2;

            while (!legal(x0, y0, x1, y1, x2, y2, me) || !obey_the_rule(x0, y0, x1, y1, x2, y2))
            {
                cout << "illegal put,please put again" << endl;
                cin.clear();
                cin.ignore();
                cin >> x0;
                cin >> y0 >> x1 >> y1 >> x2 >> y2;
            }
            now_step++;//落子的回合数+1
            PutChess(x0, y0, x1, y1, x2, y2, me);
            DrawTheBoard();
            turn_normal(x1, y1, x2, y2);
            break;
        }
        else
        {
            cout << "illegal put, please put again" << endl;
            cin.clear();
            cin.ignore();
            cin >> condition;
        }
    }
}//人类决策函数，选择模式或落子

ter_posi king_move(int color)//Kingmove 走法的 territory 和 position 值，返回类型为结构体
```

```
{
    int step[8][8][4] = { 0 };//0 表示白色，1 表示黑色，2 表示 territory 值 ，3 表示 position 值
    ter_posi king_result;
    king_result.ter = 0;
    king_result.posi = 0;

    for (int i = 0; i < 8; i++)
        for (int j = 0; j < 8; j++)
            for (int k = 0; k < 2; k++)
                step[i][j][k] = 100;//先全部初始化为 100

    for (int nowcolor = 1; nowcolor >= -1; nowcolor -= 2)
    {
        int which_one = nowcolor;
        if (nowcolor == -1) which_one = 0;
        int other_one = 1 - which_one;
        int steps;

        for (int i = 0; i < 8; i++)
        {
            for (int j = 0; j < 8; j++)
            {
                if (board[i][j] == -nowcolor)
                    step[i][j][which_one] = -1;//对方占有的格子，我方永远无法到达
                if (board[i][j] == nowcolor)
                {
                    step[i][j][which_one] = 0;//我方占有的格子，需要的步数为 0
                    steps = 1;
                    for (int k = 0; k < 8; k++)
                    {
                        int xx = i + dx[k];
                        int yy = j + dy[k];
                        if (!inMap(xx, yy) || board[xx][yy] != 0)
                            continue;
                        step[xx][yy][which_one] = steps;//寻找需要 1 步可以到达的格子
                    }
                }
            }
        }
        int mark;//标记是否还有其他可以到达的格子
        do
        {
            mark = 0;
            steps++;
            for (int i = 0; i < 8; i++)
            {
                for (int j = 0; j < 8; j++)
                {
                    if (step[i][j][which_one] != steps - 1)
                        continue;
                    for (int k = 0; k < 8; k++)//从当前步数可到达的格子出发寻找下一步可到达的格子
                    {
                        int xx = i + dx[k];
                        int yy = j + dy[k];
                        if (!inMap(xx, yy) || board[xx][yy] != 0)
                            continue;
```

```
                                if (step[xx][yy][which_one] > steps)
                                {
                                    step[xx][yy][which_one] = steps;
                                    mark = 1;
                                }
                            }
                        }
                    }
                } while (mark == 1);
            }

            int this_one = color;
            if (color == -1)     this_one = 0;
            int other_one = 1 - this_one;

            for (int i = 0; i < 8; i++)//遍历每一个格子，按照公式计算 territory 和 position 值
            {
                for (int j = 0; j < 8; j++)
                {
                    step[i][j][3] = minn(1, maxx(-1, (step[i][j][other_one] - step[i][j][this_one]) / 6));
                    king_result.posi += step[i][j][3];
                    if (step[i][j][0] == 100 && step[i][j][1] == 100)
                        step[i][j][2] = 0;
                    else if (step[i][j][0] == step[i][j][1] && step[i][j][0] < 100)
                    {
                        step[i][j][2] = color;
                        king_result.ter += color;
                    }
                    else if (step[i][j][0] < step[i][j][1])
                    {
                        step[i][j][2] = -color * 2;
                        king_result.ter += -color * 2;
                    }
                    else if (step[i][j][0] > step[i][j][1])
                    {
                        step[i][j][2] = color * 2;
                        king_result.ter += color * 2;
                    }
                }
            }
            return king_result;
        }//Kingmove 走法的 territory 和 position 值，返回类型为结构体

        double square(int x, int y)//计算 x 的 y 次幂
        {
            double result = 1;
            if (y == 0)
                return 1;
            if (y > 0)
                for (int i = 1; i <= y; i++)
                    result = result * x;
            else
                for (int i = 1; i <= -y; i++)
                    result = result / x;
            return result;
        } // 计算 x 的 y 次幂
```

```
ter_posi queen_move(int color)//标记棋盘每个格子用 queen 走法所需的步数，具体方法与 king_move 函数相同
{
    double step[8][8][4] = { 0 };//0 表示白色，1 表示黑色，2 表示 territory 值，3 表示 position 值
    ter_posi queen_result;
    queen_result.ter = 0;
    queen_result.posi = 0.0;

    for (int i = 0; i < 8; i++)
        for (int j = 0; j < 8; j++)
            for (int k = 0; k < 2; k++)
                step[i][j][k] = 100;

    for (int nowcolor = 1; nowcolor >= -1; nowcolor -= 2)
    {
        int which_one = nowcolor;
        if (nowcolor == -1) which_one = 0;
        int other_one = 1 - which_one;
        int steps;

        for (int i = 0; i < 8; i++)
        {
            for (int j = 0; j < 8; j++)
            {
                if (board[i][j] == -nowcolor)
                    step[i][j][which_one] = -1;
                if (board[i][j] == nowcolor)
                {
                    step[i][j][which_one] = 0;
                    steps = 1;
                    for (int k = 0; k < 8; k++)
                    {
                        for (int delta = 1; delta < GRIDSIZE; delta++)
                        {
                            int xx = i + dx[k] * delta;
                            int yy = j + dy[k] * delta;
                            if (!inMap(xx, yy) || board[xx][yy] != 0)
                                continue;
                            step[xx][yy][which_one] = steps;
                        }
                    }
                }
            }
        }
        int mark;
        do
        {
            mark = 0;
            steps++;
            for (int i = 0; i < 8; i++)
            {
                for (int j = 0; j < 8; j++)
                {
                    if (step[i][j][which_one] != steps - 1)
                        continue;
                    for (int k = 0; k < 8; k++)
```

```
                    {
                        for (int delta = 1; delta < GRIDSIZE; delta++)
                        {
                            int xx = i + dx[k] * delta;
                            int yy = j + dy[k] * delta;
                            if (!inMap(xx, yy) || board[xx][yy] != 0)
                                continue;
                            if (step[xx][yy][which_one] > steps)
                            {
                                step[xx][yy][which_one] = steps;
                                mark = 1;
                            }
                        }
                    }
                }
            }
        } while (mark == 1);
    }

    int this_one = color;
    if (color == -1)    this_one = 0;
    int other_one = 1 - this_one;

    for (int i = 0; i < 8; i++)
    {
        for (int j = 0; j < 8; j++)
        {
            step[i][j][3] = 2 * (square(2, -step[i][j][this_one]) - square(2, -step[i][j][other_one]));
            queen_result.posi += step[i][j][3];
            if (step[i][j][0] == 100 && step[i][j][1] == 100)
                step[i][j][2] = 0;
            else if (step[i][j][0] == step[i][j][1] && step[i][j][0] < 100)
            {
                step[i][j][2] = color;
                queen_result.ter += color;
            }
            else if (step[i][j][0] < step[i][j][1])
            {
                step[i][j][2] = -color * 2;
                queen_result.ter += -color * 2;
            }
            else if (step[i][j][0] > step[i][j][1])
            {
                step[i][j][2] = color * 2;
                queen_result.ter += color * 2;
            }
        }
    }
    return queen_result;
}//标记棋盘每个格子用 queen 走法所需的步数

void determine_4(int color)// 选择难度系数为 4，运用 mobility，territory，position 作为参数，25 回合后剪枝搜两层
{
    int pos_begin[3000][2], pos_put[3000][2], pos_obs[3000][2];
    int pos_method1 = 0, choice;
```

```
double maxvalue = -300000;
for (int i = 0; i < GRIDSIZE; i++)
{
    for (int j = 0; j < GRIDSIZE; j++)
    {
        if (board[i][j] != color)
            continue;
        for (int k1 = 0; k1 < 8; k1++)//找到当前棋子所在位置，向八个方向搜索可行方案
        {
            for (int delta1 = 1; delta1 < GRIDSIZE; delta1++)
            {
                int xx = i + dx[k1] * delta1;
                int yy = j + dy[k1] * delta1;
                if (board[xx][yy] != 0 || !inMap(xx, yy))
                    break;
                for (int k2 = 0; k2 < 8; k2++)
                {
                    for (int delta2 = 1; delta2 < GRIDSIZE; delta2++)
                    {
                        int xxx = xx + dx[k2] * delta2;
                        int yyy = yy + dy[k2] * delta2;
                        if (!inMap(xxx, yyy))
                            break;
                        if (board[xxx][yyy] != 0 && !(i == xxx && j == yyy))
                            break;
                        if (legal(i, j, xx, yy, xxx, yyy, color))
                        {
                            PutChess(i, j, xx, yy, xxx, yyy, color);//模拟落子函数
                            turn_normal(xx, yy, xxx, yyy);
                            pos_begin[pos_method1][0] = i;
                            pos_begin[pos_method1][1] = j;
                            pos_put[pos_method1][0] = xx;
                            pos_put[pos_method1][1] = yy;
                            pos_obs[pos_method1][0] = xxx;
                            pos_obs[pos_method1][1] = yyy;//记录模拟的落子方案

                            if (cycle > 25)//极大极小搜索+剪枝
                            {
                                int pos_begin2[3000][2] = { 0 }, pos_put2[3000][2] = { 0 },
                                    pos_obs[3000][2] = { 0 };
                                double minvalue = 300000;
                                int flag = 1;
                                //搜索对方的可行方案
                                for (int ii = 0; ii < GRIDSIZE; ii++)
                                {
                                    for (int jj = 0; jj < GRIDSIZE; jj++)
                                    {
                                        if (board[ii][jj] != -color)
                                            continue;
                                        for (int k3 = 0; k3 < 8; k3++)
                                        {
                                            for (int delta3 = 1; delta3 < GRIDSIZE;
                                                                        delta3++)
                                            {
                                                int xx2 = ii + dx[k3] * delta3;
                                                int yy2 = jj + dy[k3] * delta3;
```

```
if (!inMap(xx2, yy2) ||
            board[xx2][yy2] != 0)
      break;

for (int k4 = 0; k4 < 8; k4++)
{
      for (int delta4 = 1; delta4 <
                  GRIDSIZE; delta4++)
      {
            int xxx2 = xx2 +
                  dx[k4] * delta4;
            int yyy2 = yy2 +
                  dy[k4] * delta4;
            if (!inMap(xxx2, yyy2))
                  break;
            if (board[xxx2][yyy2] != 0
      && !(ii == xxx2 && jj == yyy2))
                  break;
            if (legal(ii, jj, xx2, yy2,
                  xxx2, yyy2, -color))
            {
                  PutChess(ii, jj, xx2,
            yy2, xxx2, yyy2, -color);
                  turn_normal(xx2, yy2,
      xxx2, yyy2);//模拟对方落子
            }

            ter_posi now_king =
                  king_move(color);
            ter_posi now_queen =
                  queen_move(color);
            ter_posi other_king =
                  king_move(-color);
            ter_posi other_queen =
                  queen_move(-color);

            int now_mob =
mobility(color) - mobility(-color);
            int now_king_t =
now_king.ter - other_king.ter;
            int now_queen_t =
now_queen.ter - other_queen.ter;
            double now_queen_p =
now_queen.posi - other_queen.posi;
            double now_king_p =
(now_king.posi - other_king.posi) / 6;
            double now_p = now_king_p +
                  now_queen_p;

            int n1, n2, n3, n4;
            //参数按照回合数变化
            if (cycle <= 12)
            {
                  n1 = 18; n2 = 40;
                   n3 = 20; n4 = 78;
            }
```

```
                                                    else if (cycle < 42)
                                                    {
                                                        n1 = 7; n2 = 6;
                                                         n3 = 2; n4 = 24;
                                                    }
                                                    else { n1 = 9; n2 = 2;
                                                        n3 = 1; n4 = 30; }

                                                    double now_value = n1 *
now_queen_t + n2 * now_king_t + n3 * now_mob + n4 * now_p;//计算当前我方价值减去对方价值
                                                    ClearChess(ii, jj, xx2,
                                                yy2, xxx2, yyy2, -color);
                                                    if (now_value < minvalue)
                                                        minvalue =
                                        now_value;//计算我方每一种走法对应对方决策后我的最小价值

                                                    if (minvalue < maxvalue)
                                                                    //剪枝
                                                    {
                                                        flag = 0;
                                                        break;
                                                    }
                                                }
                                                if (flag == 0)
                                                    break;
                                            }
                                            if (flag == 0)
                                                break;
                                        }
                                        if (flag == 0)
                                            break;
                                    }
                                    if (flag == 0)
                                        break;
                                }
                                if (flag == 0)
                                    break;
                            }

                            if (minvalue > maxvalue)
                            //对方以对我最不利的方式决策后我的最大价值
                            {
                                maxvalue = minvalue;
                                choice = pos_method1;
                            }
                            pos_method1++;
                            ClearChess(i, j, xx, yy, xxx, yyy, color);
                        }
                        Else
                        //25 回合内由于可行走法太多，为追求时间效率，只搜一步，用贪心
                        {
                            ter_posi now_king = king_move(color);
                            ter_posi now_queen = queen_move(color);
                            ter_posi other_king = king_move(-color);
                            ter_posi other_queen = queen_move(-color);
```

```
                                        int now_mob = mobility(color) - mobility(-color);
                                        int now_king_t = now_king.ter - other_king.ter;
                                        int now_queen_t = now_queen.ter - other_queen.ter;
                                        double now_queen_p = now_queen.posi - other_queen.posi;
                                        double now_king_p = (now_king.posi - other_king.posi) / 6;
                                        double now_p = now_king_p + now_queen_p;

                                        int n1, n2, n3, n4;
                                        if (cycle <= 12)
                                        {
                                             n1 = 18; n2 = 40; n3 = 20; n4 = 78;
                                        }
                                        else if (cycle < 42)
                                        {
                                             n1 = 7; n2 = 6; n3 = 2; n4 = 24;
                                        }
                                        else { n1 = 9; n2 = 2; n3 = 1; n4 = 30; }

                                        double now_value = n1 * now_queen_t + n2 * now_king_t +
                                                                        n3 * now_mob + n4 * now_p;
                                        if (now_value > maxvalue)
                                        {
                                             maxvalue = now_value;
                                             choice = pos_method1;
                                        }
                                        pos_method1++;
                                        ClearChess(i, j, xx, yy, xxx, yyy, color);
                                   }
                              }
                         }
                    }
               }
          }
     }
     PutChess(pos_begin[choice][0], pos_begin[choice][1], pos_put[choice][0],
          pos_put[choice][1], pos_obs[choice][0], pos_obs[choice][1], color);
     DrawTheBoard();
     turn_normal(pos_put[choice][0],
          pos_put[choice][1], pos_obs[choice][0], pos_obs[choice][1]);
}// 选择难度系数为 4，运用 mobility，territory，position 作为参数，25 回合后剪枝搜两层

void determine_3(int color)//选择难度系数为 3，运用 mobility 和 territory 为参数，搜索一步
{
     int pos_begin[3000][2], pos_put[3000][2], pos_obs[3000][2];
     int pos_method1 = 0, choice;
     double maxvalue = -300000;
     for (int i = 0; i < GRIDSIZE; i++)
     {
          for (int j = 0; j < GRIDSIZE; j++)
          {
               if (board[i][j] != color)
                    continue;
               for (int k1 = 0; k1 < 8; k1++)
               {
                    for (int delta1 = 1; delta1 < GRIDSIZE; delta1++)
```

```
{
    int xx = i + dx[k1] * delta1;
    int yy = j + dy[k1] * delta1;
    if (board[xx][yy] != 0 || !inMap(xx, yy))
        break;
    for (int k2 = 0; k2 < 8; k2++)
    {
        for (int delta2 = 1; delta2 < GRIDSIZE; delta2++)
        {
            int xxx = xx + dx[k2] * delta2;
            int yyy = yy + dy[k2] * delta2;
            if (!inMap(xxx, yyy))
                break;
            if (board[xxx][yyy] != 0 && !(i == xxx && j == yyy))
                break;
            if (legal(i, j, xx, yy, xxx, yyy, color))
            {
                PutChess(i, j, xx, yy, xxx, yyy, color);
                turn_normal(xx, yy, xxx, yyy);
                pos_begin[pos_method1][0] = i;
                pos_begin[pos_method1][1] = j;
                pos_put[pos_method1][0] = xx;
                pos_put[pos_method1][1] = yy;
                pos_obs[pos_method1][0] = xxx;
                pos_obs[pos_method1][1] = yyy;

                ter_posi now_king = king_move(color);
                ter_posi now_queen = queen_move(color);
                ter_posi other_king = king_move(-color);
                ter_posi other_queen = queen_move(-color);

                int now_mob = mobility(color) - mobility(-color);
                int now_king_t = now_king.ter - other_king.ter;
                int now_queen_t = now_queen.ter - other_queen.ter;

                int n1, n2, n3;
                if (cycle <= 12)
                {
                    n1 = 14; n2 = 37; n3 = 40;
                }
                else if (cycle < 32)
                {
                    n1 = 6; n2 = 5; n3 = 2;
                }
                else { n1 = 8; n2 = 1; n3 = 0; }

                double now_value = n1 * now_queen_t + n2 * now_king_t + n3 * now_mob;
                    if (now_value > maxvalue)
                {
                    maxvalue = now_value;
                    choice = pos_method1;
                }
                pos_method1++;
                ClearChess(i, j, xx, yy, xxx, yyy, color);
            }
        }
```

```
                    }
                }
            }
        }
    }
    PutChess(pos_begin[choice][0], pos_begin[choice][1], pos_put[choice][0],
        pos_put[choice][1], pos_obs[choice][0], pos_obs[choice][1], color);
    DrawTheBoard();
    turn_normal(pos_put[choice][0],
        pos_put[choice][1], pos_obs[choice][0], pos_obs[choice][1]);
}//选择难度系数为 3，运用 mobility 和 territory 为参数，搜索一步

void determine_2(int color)//选择难度系数为 2，运用 mobility 为参数，搜索一步
{
    int pos_begin[3000][2], pos_put[3000][2], pos_obs[3000][2];
    int pos_method = 0, choice;
    int maxvalue = -300000;
    for (int i = 0; i < GRIDSIZE; i++)
    {
        for (int j = 0; j < GRIDSIZE; j++)
        {
            if (board[i][j] != color)
                continue;
            for (int k1 = 0; k1 < 8; k1++)
            {
                for (int delta1 = 1; delta1 < GRIDSIZE; delta1++)
                {
                    int xx = i + dx[k1] * delta1;
                    int yy = j + dy[k1] * delta1;
                    if (board[xx][yy] != 0 || !inMap(xx, yy))
                        break;
                    for (int k2 = 0; k2 < 8; k2++)
                    {
                        for (int delta2 = 1; delta2 < GRIDSIZE; delta2++)
                        {
                            int xxx = xx + dx[k2] * delta2;
                            int yyy = yy + dy[k2] * delta2;
                            if (!inMap(xxx, yyy))
                                break;
                            if (board[xxx][yyy] != 0 && !(i == xxx && j == yyy))
                                break;
                            if (legal(i, j, xx, yy, xxx, yyy, color))
                            {
                                PutChess(i, j, xx, yy, xxx, yyy, color);
                                turn_normal(xx, yy, xxx, yyy);
                                pos_begin[pos_method][0] = i;
                                pos_begin[pos_method][1] = j;
                                pos_put[pos_method][0] = xx;
                                pos_put[pos_method][1] = yy;
                                pos_obs[pos_method][0] = xxx;
                                pos_obs[pos_method][1] = yyy;

                                int nowvalue = mobility(color) - mobility(-color);
                                if (nowvalue > maxvalue)
                                {
                                    maxvalue = nowvalue;
```

```
                                        choice = pos_method;
                                    }
                                    pos_method++;
                                    ClearChess(i, j, xx, yy, xxx, yyy, color);
                                }
                            }
                        }
                    }
                }
            }
        }
        PutChess(pos_begin[choice][0], pos_begin[choice][1], pos_put[choice][0],
            pos_put[choice][1], pos_obs[choice][0], pos_obs[choice][1], color);
        DrawTheBoard();
        turn_normal(pos_put[choice][0],
            pos_put[choice][1], pos_obs[choice][0], pos_obs[choice][1]);
}//选择难度系数为 2，运用 mobility 为参数，搜索一步

void determine_1(int color)//选择难度系数为 2，随机落子，搜索一步
{
        int pos_begin[3000][2], pos_put[3000][2], pos_obs[3000][2];
        int pos_method = 0, choice;
        for (int i = 0; i < GRIDSIZE; ++i)
        {
            for (int j = 0; j < GRIDSIZE; ++j)
            {
                for (int k = 0; k < 8; ++k)
                {
                    for (int delta1 = 1; delta1 < GRIDSIZE; delta1++)
                    {
                        int xx = i + dx[k] * delta1;
                        int yy = j + dy[k] * delta1;
                        if (board[xx][yy] != 0 || !inMap(xx, yy))
                            break;
                        for (int l = 0; l < 8; ++l) {
                            for (int delta2 = 1; delta2 < GRIDSIZE; delta2++)
                            {
                                int xxx = xx + dx[l] * delta2;
                                int yyy = yy + dy[l] * delta2;
                                if (!inMap(xxx, yyy))
                                    break;
                                if (board[xxx][yyy] != 0 && !(i == xxx && j == yyy))
                                    break;
                                if (legal(i,j,xx,yy,xxx,yyy,color))
                                {
                                    pos_begin[pos_method][0] = i;
                                    pos_begin[pos_method][1] = j;
                                    pos_put[pos_method][0] = xx;
                                    pos_put[pos_method][1] = yy;
                                    pos_obs[pos_method][0] = xxx;
                                    pos_obs[pos_method][1] = yyy;
                                    pos_method++;
                                }
                            }

                        }
```

```
                    }

                }
            }
        }
        srand(time(0));
        choice = rand() % pos_method;//生成随机数
        PutChess(pos_begin[choice][0], pos_begin[choice][1], pos_put[choice][0],
            pos_put[choice][1], pos_obs[choice][0], pos_obs[choice][1], color);
        DrawTheBoard();
        turn_normal(pos_put[choice][0],
            pos_put[choice][1], pos_obs[choice][0], pos_obs[choice][1]);
}//选择难度系数为 2，随机落子，搜索一步

void AI_decision(int color)//AI 决策函数，AI 的强度与当前选择的难度有关
{
        if (difficulty == 4) determine_4(color);
        else if (difficulty == 3) determine_3(color);
        else if (difficulty == 2) determine_2(color);
        else if (difficulty == 1) determine_1(color);
}//AI 决策函数，AI 的强度与当前选择的难度有关

void save()//存盘函数
{
        ofstream fout("read_save.txt");
        fout << me << endl;//输出我方颜色
        for (int i = 0; i < GRIDSIZE; i++)//输出当前棋盘状态
        {
            for (int j = 0; j < GRIDSIZE; j++)
            {
                if(board[i][j] < 5)
                    fout << board[i][j] << " ";
                else fout << board[i][j] - 10 << " ";
            }
            fout << endl;
        }
        fout << cycle - 1 << endl;//输出当前回合数
        fout << difficulty;//输出选择的难度
        fout.close();
}//存盘函数

void regret()//悔棋函数
{
        now_step--;//还可以继续悔棋的步数减一
        cycle--;//每次悔棋撤回两回合
        ClearChess(record[cycle][1], record[cycle][2], record[cycle][3], record[cycle][4], record[cycle][5],
record[cycle][6], -me);
        cycle--;
        ClearChess(record[cycle][1], record[cycle][2], record[cycle][3], record[cycle][4], record[cycle][5],
record[cycle][6], me);
        cycle--;
        if (cycle >= 2 && now_step > 0)//把上一个回合中 AI 的决策体现出来，若无前一回合，则不体现
        {
            board[record[cycle][3]][record[cycle][4]] += 10;
            board[record[cycle][5]][record[cycle][6]] += 10;
            DrawTheBoard();
```

```
            turn_normal(record[cycle - 1][3], record[cycle - 1][4], record[cycle - 1][5], record[cycle - 1][6]);
      }
      else DrawTheBoard();
      human_decision();
}//悔棋函数

void AI_play(int your_cycle)//托管函数
{
      int player = me;
      watch = 1;//标记状态为托管中
      while (your_cycle--)
      {
            if (player == me)
                  determine_1(player);//托管当然要用最菜的 AI，谁让你懒呢
            if (player == -me)
                  AI_decision(player);

            player = player * (-1);
            if (!check(player))
            {
                  cout << "Game Over" << endl;
                  if (player == me)
                        cout << "输给了 AI 呢 TAT" << endl;
                  else cout << "恭喜你战胜了蠢萌的 AI 撒花" << endl;
                  cout << "goodbye" << endl;
                  exit(0);
            }
            Sleep(500);//每走一步停 0.5 秒，让你清楚的看到你是怎么死的
      }
      cycle--;
      watch = 0;
      continue_game();//托管结束后可继续游戏
}//托管函数

void play_with_yourself()//自己和自己下一盘棋
{
      PlaySound(TEXT("D:\\sweet bite.wav"), NULL, SND_FILENAME | SND_ASYNC);
      DrawTheBoard();
      int player = 1;
      int x0, y0, x1, y1, x2, y2;
      cin >> x0 >> y0 >> x1 >> y1 >> x2 >> y2;
      while (!legal(x0, y0, x1, y1, x2, y2, player) || !obey_the_rule(x0, y0, x1, y1, x2, y2))
      {
            cout << "illegal put,please put again" << endl;
            cin.clear();
            cin.ignore();
            cin >> x0 >> y0 >> x1 >> y1 >> x2 >> y2;
      }

      while (true)
      {
            PutChess(x0, y0, x1, y1, x2, y2, player);
            DrawTheBoard();
            turn_normal(x1, y1, x2, y2);
            player = player * (-1);
```

```
            if (!check(player))
            {
                cout << "Game Over" << endl;
                cout << "The Winner is PLAYER " << endl;
                if (player == 1)
                    cout << "1";
                else cout << "2";
                cout << "goodbye" << endl;
                break;
            }
            cin >> x0 >> y0 >> x1 >> y1 >> x2 >> y2;

            while (!legal(x0, y0, x1, y1, x2, y2, player) || !obey_the_rule(x0, y0, x1, y1, x2, y2))
            {
                cout << "illegal put,please put again" << endl;
                cin.clear();
                cin.ignore();
                cin >> x0 >> y0 >> x1 >> y1 >> x2 >> y2;
            }
        }
}//自己和自己下一盘棋

void new_game()//新游戏函数，和 AI 下一盘棋
{
    PlaySound(TEXT("D:\\Dusk.wav"), NULL, SND_FILENAME | SND_ASYNC);
    cycle = 0;
    DrawTheBoard();
    cout << "       请选择难度" << endl;
    cout << "     蠢萌：Press 1  休闲：Press 2" << endl;
    cout << "人类：Press 3    计算机科学家：Press 4" << endl;
    while (true)
    {
        cin >> difficulty;//选择难度
        if (difficulty == 1 || difficulty == 2 || difficulty == 3 || difficulty == 4)
            break;
        cin.clear();
        cin.ignore();
        cycle = 0;
        DrawTheBoard();
        cout << "          请重新选择难度" << endl;
        cout << "     蠢萌：Press 1  休闲：Press 2" << endl;
        cout << "人类：Press 3    计算机科学家：Press 4" << endl;
    }
    cycle = 0;
    DrawTheBoard();

    cout << " please choose your color" << endl;
    cout << "  black (press B) or white (press W)" << endl;
    char choose;
    cin >> choose;//选择我方执子颜色
    cycle = 0;
    DrawTheBoard();
    cycle++;

    while (true)
    {
```

```
        while (true)
        {
            if (choose == 'B' || choose == 'b')
            {
                cout << "        回合数：1" << endl << "您是黑方" << endl;
                cout << "Press E (退出)   Press H (提示) Press R (悔棋)" << endl;
                cout << "Press T (重新开始) Press B (托管)" << endl;
                cout << "或输入坐标" << endl;
                me = 1;
                break;
            }
            else if (choose == 'W' || choose == 'w')
            {
                me = -1;
                break;
            }
            else
            {
                cin.clear();
                cin.ignore();
                cout << "请重新选择您要执子的颜色" << endl;
                cout << "Black ( press B ) ; White ( press W )" << endl;
                cin >> choose;
            }
        }
        cin.clear();
        cin.ignore();
        int player = 1;

        while (true)
        {
            if (player == me)
                human_decision();
            else AI_decision(-me);

            player = player * (-1);
            if (!check(player))
            {
                cout << "Game Over" << endl;
                if (player == 1)
                    cout << "输给了 AI 呢 TAT" << endl;
                else cout << "恭喜你战胜了蠢萌的 AI 撒花" << endl;
                cout << "goodbye" << endl;
                exit(0);
            }
        }
    }
}//新游戏函数，和 AI 下一盘棋

void continue_game()//继续游戏函数
{
    int player = -1;
    int x0, y0, x1, y1, x2, y2;
    DrawTheBoard();
    if (cycle % 2 == 0)
        player = 1;//判断现在的玩家
```

```
	while (true)
	{
		if (player == me)
			human_decision();
		else AI_decision(-me);
		player = player * (-1);
		if (!check(player))
		{
			cout << "Game Over" << endl;
			if (player == me)
				cout << "输给了 AI 呢 TAT" << endl;
			else cout << "恭喜你战胜了蠢萌的 AI 撒花" << endl;
			cout << "goodbye" << endl;
			exit(0);
		}
	}
}//继续游戏函数

void last_game()//上一局游戏函数，读盘
{
	PlaySound(TEXT("D:\\tide.wav"), NULL, SND_FILENAME | SND_ASYNC);
	ifstream fin("read_save.txt");
	fin >> me;
	for (int i = 0; i < GRIDSIZE; i++)
		for (int j = 0; j < GRIDSIZE; j++)
			fin >> board[i][j];
	fin >> cycle;
	fin >> difficulty;
	fin.close();
	continue_game();
	return;
}//上一局游戏函数

void play_game()//真正的玩游戏函数
{
	printf("\a");
	now_step = 0;
	board[0][2] = 1; board[0][5] = 1; board[2][0] = 1; board[2][7] = 1;
	board[5][0] = -1; board[5][7] = -1; board[7][2] = -1; board[7][5] = -1;//初始化棋盘
	cycle = 0;
	DrawTheBoard();
	cout << "        Welcome to Amazon!" << endl;
	cout << " ● means black chess." << endl << "  ○ means white chess." << endl;
	cout << "  ↑ means obstacle." << endl << endl;
	cout << "     play with yourself (press M)" << endl;
	cout << "        new game (press N)" << endl;
	cout << "       last game  (press L)" << endl;

	char condition;
	while (true)
	{
		cin >> condition;
		printf("\a");
		if (condition == 'm' || condition == 'M')
		{
```

```
                mode = 0;
                play_with_yourself();
                break;
            }
            else if (condition == 'n' || condition == 'N')
            {
                mode = 1;
                new_game();
                break;
            }
            else if (condition == 'l' || condition == 'L')
            {
                mode = 2;
                last_game();
                break;
            }
            else
            {
                cycle = 0;
                DrawTheBoard();
                cout << "       请重新选择游戏模式" << endl;
                cout << "      play with yourself (press M)" << endl;
                cout << "         new game (press N)" << endl;
                cout << "       last game  (press L)" << endl;
            }
        }
}//玩游戏函数

int main()
{
    system("mode con cols=80 lines=35");//窗口宽度高度
    system("color 0E0");//背景颜色
    play_game();
    return 0;
}
```