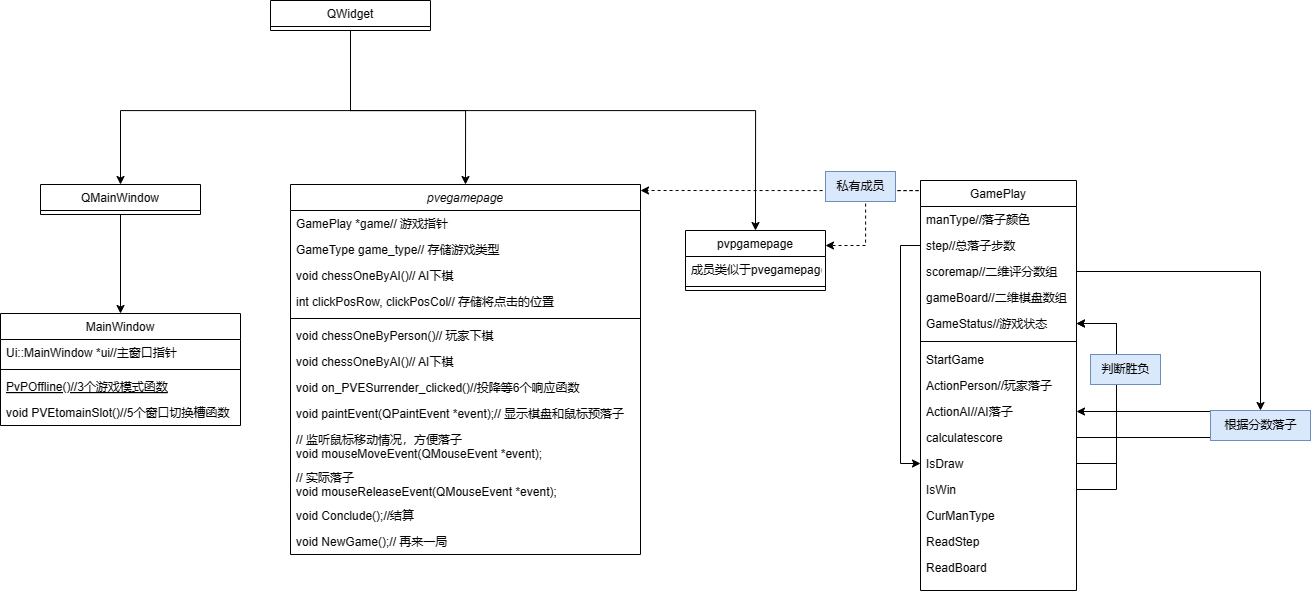
**一、设计与实现**

**1、设计思想**

1. 五子棋基本游戏逻辑的实现
   1. 将后端封装为一个GamePlay类，数据成员均为protected，外界通过接口函数访问。设计完善的public成员函数，方便前端开发人员和AI设计人员使用。
   2. 采用int型二维数组存储棋盘数据，符合其客观存在。
   3. 设计两套胜负判断方法：一套在二维数据上执行搜索，效率更高；另一套采用了线性数据结构，满足题目设计要求。
   4. 设计AI下棋算法，依据贪心算法的思想，根据目前棋局落子情况给所有未落子点赋分，取得分最高点落子
2. 前端设计
   1. 前后端完全分离，前端设计人员只需要按照头文件说明引用GamePlay类。
   2. 采用贴图+绘制的图形设计策略，加入部分动态效果，提高软件的美观和易用性。
   3. 对部分按钮和对局关键信息做突出化处理，突出用户友好的设计理念。

**2、类结构**



**3、主要数据结构**

1. 棋盘：二维数据，采用int型二维数组存储棋盘数据，将空设为0，白子设为1，黑子设为-1。
2. AI得分表：二维数据，与棋盘数据相对应，格子中储存当前位置的得分，AI取得分最高者落子。
3. 五子连珠判断
   1. 实装版本：依托于棋盘数据执行搜索
   2. 线性表版本（仅测试，未实装）：创建两个链表，分别储存黑子和白子的落子位置。将二位数组情况下，向周围搜索同色子改为对同色子链表中元素的查找。

**4、算法设计**

1）五子连珠判断

当前落子坐标减去棋盘边界的坐标，得到不同方向上的最大搜索距离；在搜索时遇到异色字则停止；统计x，y，y=x，y=-x四个方向上同色子数量。

对于二维数据版本，直接利用棋盘和下标索引进行搜索；对于链表版本，搜索变为了对落子链表中元素的搜索。

i）用链表记录落子位置：

typedef struct Node {

    int row;

    int col;

    struct Node\* next;

} Node;

// 定义链表头指针 : GamePlay类中有两个protected对象，whiteHead和blackHead

// 添加节点到链表末尾

void GamePlay::appendNode(int type, int row, int col) {

if(type == -1){

head = blackhead;

}else

Head = whitehead;

    Node\* newNode = (Node\*)malloc(sizeof(Node));

    if (!newNode) {

        perror("Memory allocation failed");

        exit(EXIT\_FAILURE);

    }

    newNode->row = row;

    newNode->col = col;

    newNode->next = NULL;

    if (head == NULL) {

        head = newNode;

    } else {

        Node\* current = head;

        while (current->next != NULL) {

            current = current->next;

        }

        current->next = newNode;

    }

}

// 查找节点，如果找到返回节点指针，否则返回NULL

bool GamePlay::findNode(int type, int row, int col) {

if(type == -1){

head = blackhead;

}else

Head = whitehead;

    Node\* current = head;

    while (current != NULL) {

        if (current->row == row && current->col == col) {

            return true;

        }

        current = current->next;

    }

    return false;

}

// 释放链表内存

void GamePlay::freeList() {

    Node\* current = blackHead;

    Node\* temp;

    while (current != NULL) {

        temp = current;

        current = current->next;

        free(temp);

    }

blackHead = NULL;

Node\* current = whiteHead;

    Node\* temp;

    while (current != NULL) {

        temp = current;

        current = current->next;

        free(temp);

    }

whiteHead = NULL;

}

2）AI算法

遍历棋盘所有未落子点，对每个点周围对方与己方落子情况分别进行判断，如杀三，杀四等。针对对方和己方落子情况分别给该点加分，通过调节参数可以调整不同情况该点加分值，从而使AI更加注重防守或者进攻。最后取得分最高点落子。若多个点分数相同，通过当前时间引入随机数，随机选取一个点落子

i）AI计算双方落子和空位情况

for (int row = 0; row < BOARDSIZE; row++)

        for (int col = 0; col < BOARDSIZE; col++)

        {

            // 空白点就算

            if (row > 0 && col > 0 &&

                gameBoard[row][col] == 0)

            {

                // 遍历周围八个方向

                for (int y = -1; y <= 1; y++)

                    for (int x = -1; x <= 1; x++)

                    {

                        // 重置

                        personNum = 0;

                        botNum = 0;

                        emptyNum = 0;

                        // 原坐标不算

                        if (!(y == 0 && x == 0))

                        {

                            // 每个方向延伸4个子

                            // 对玩家白子评分（正反两个方向）

                            for (int i = 1; i <= 4; i++)

                            {

                                if (row + i \* y > 0 && row + i \* y < BOARDSIZE &&

                                    col + i \* x > 0 && col + i \* x < BOARDSIZE &&

                                    gameBoard[row + i \* y][col + i \* x] == -manType) // 玩家的子

                                {

                                    personNum++;

                                }

                                else if (row + i \* y > 0 && row + i \* y < BOARDSIZE &&

                                         col + i \* x > 0 && col + i \* x < BOARDSIZE &&

                                         gameBoard[row + i \* y][col + i \* x] == 0) // 空白位

                                {

                                    emptyNum++;

                                    break;

                                }

                                else            // 出边界

                                    break;

                            }

ii）AI对潜在落子点打分

if (personNum == 1)                      // 杀二

    scoremap[row][col] += 10;

else if (personNum == 2)                 // 杀三

{

    if (emptyNum == 1)

    scoremap[row][col] += 30;

    else if (emptyNum == 2)

        scoremap[row][col] += 40;

}

else if (personNum == 3)                 // 杀四

{

                                // 量变空位不一样，优先级不一样

    if (emptyNum == 1)

        scoremap[row][col] += 60;

else if (emptyNum == 2)

        scoremap[row][col] += 110;

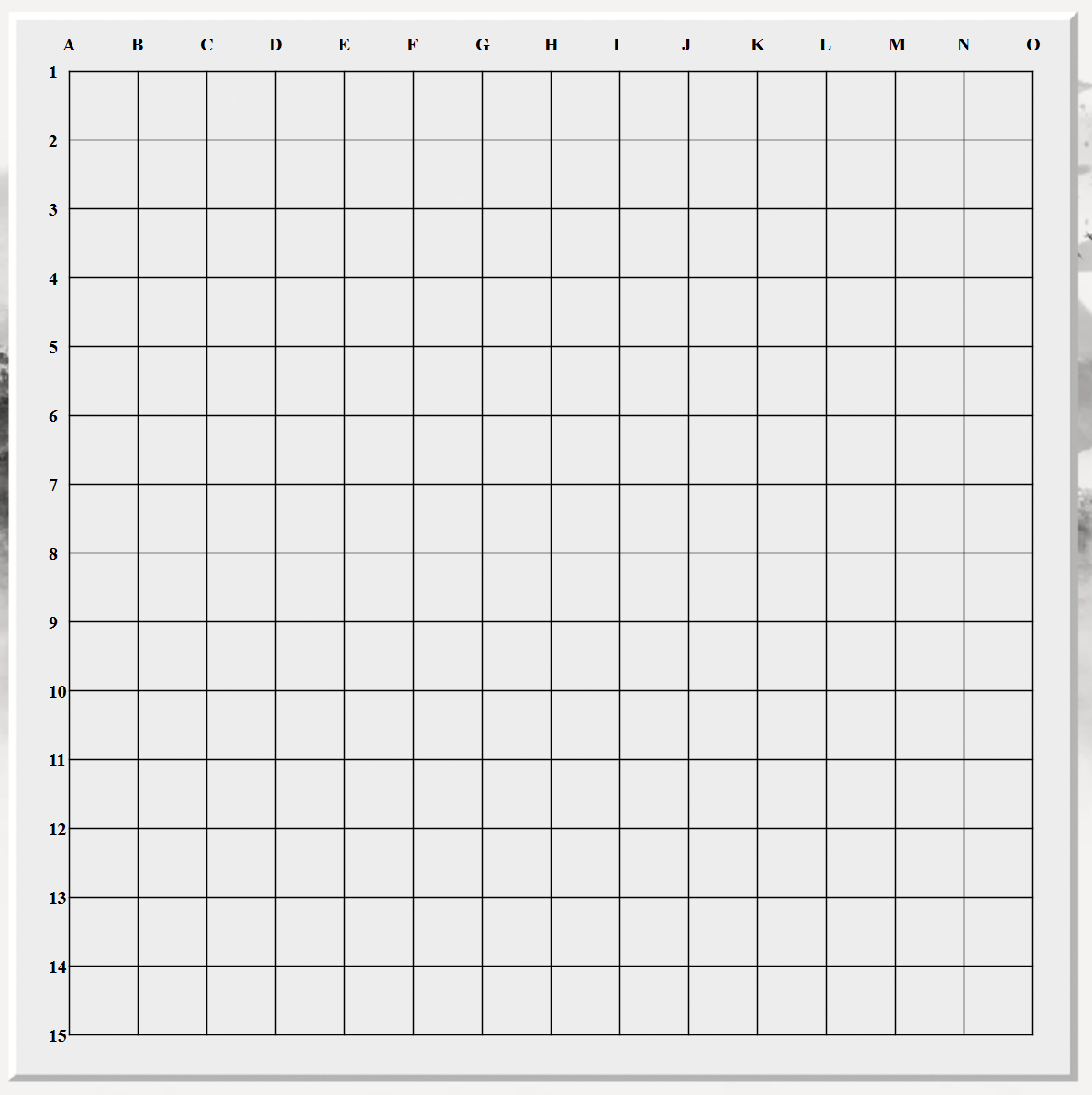
}

    else if (personNum == 4)            // 杀五

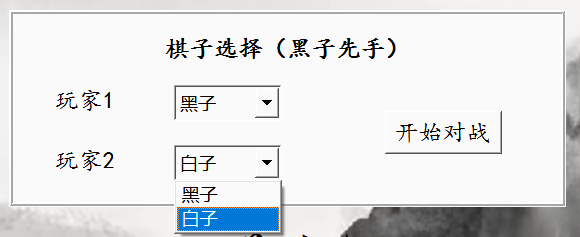
                                scoremap[row][col] += 10100;

**5、对题目要求的实现情况**

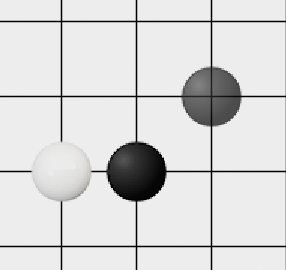
1. 创建、绘制棋盘
   1. 用图形界面或字符界面都可以。如果用字符界面，可以用字符集里面的制表符来表示棋盘。
   2. 棋盘上的格点坐标从下到上用阿拉伯数字1—15顺序标记，从左到右用英文字母A—O顺序标记。



1. 提供对弈功能
   1. 对弈开始前，两位棋手有机会选择黑棋或白旗；黑棋先行，交替落子；

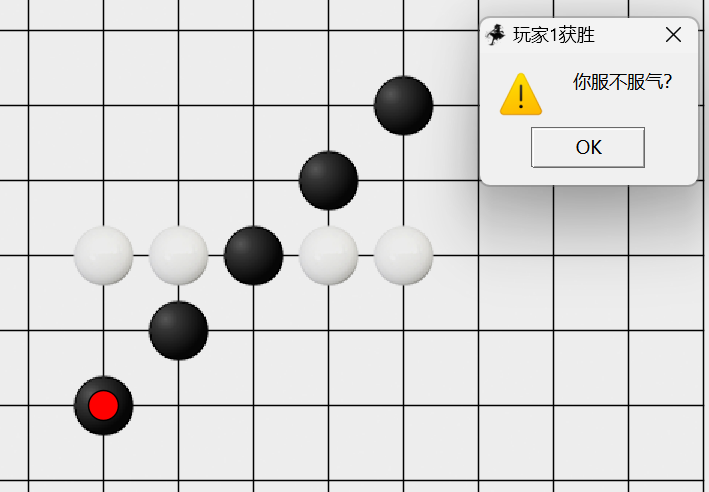


* 1. 根据所使用的界面，鼠标直接选点、坐标输入选点、方向键移动选点或其它方式都可以。选点后落子之前，需要确认（可设置为系统选项）。



鼠标悬停虚化处理，点击即为确认落子，棋子变为不透明。

* 1. 标记当前局面的最后一手棋，便于棋手参考；



最后一手棋中心标红处理。

* 1. 显示当前局面的总手数、下一手颜色等实时信息。

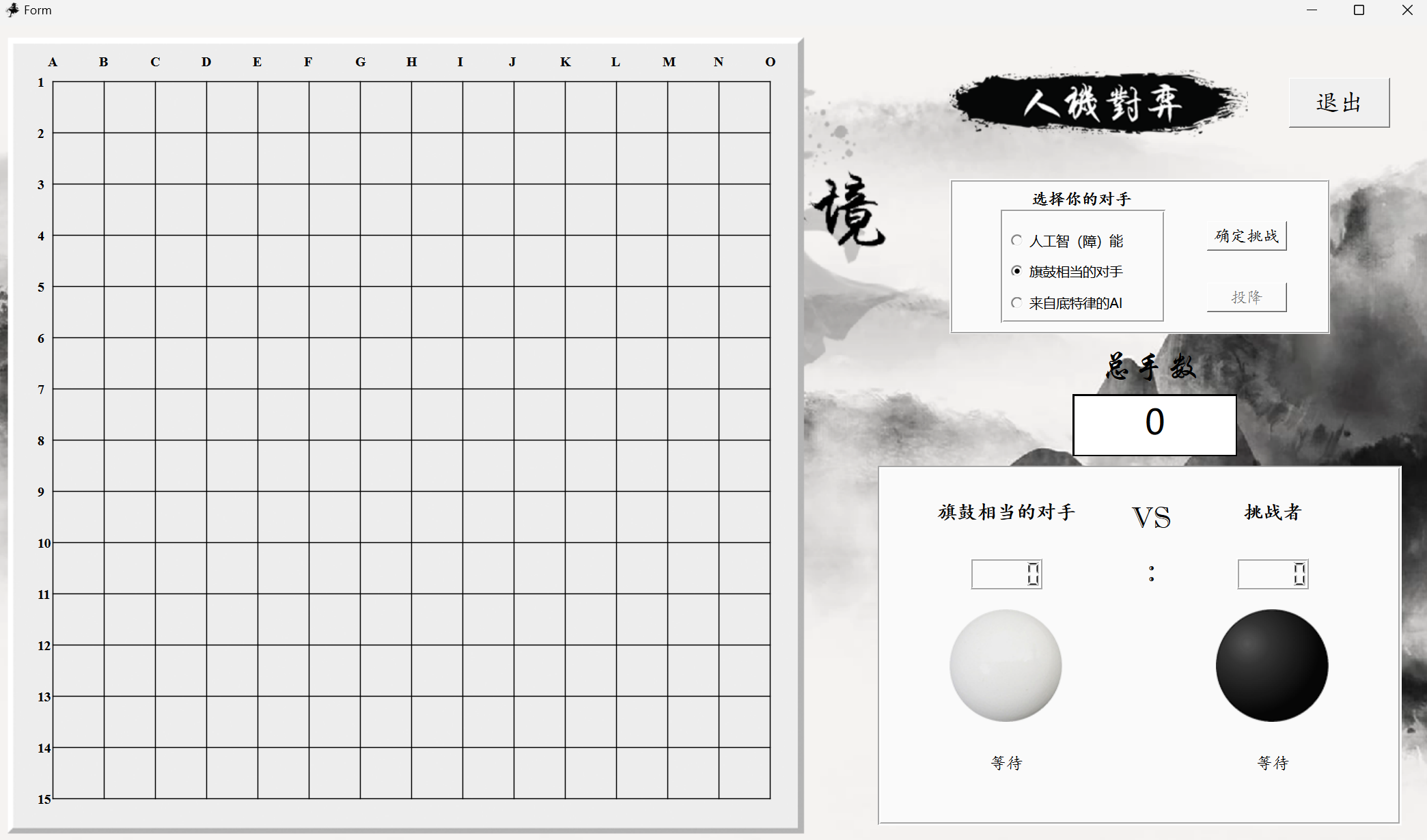


上方为总手数，本手棋子颜色以黄色的“行动”标志标识。

1. 提供胜负判别功能
   1. 棋手可以主动认输，系统终止本次对弈；或者，当棋盘下满，无法落子时，系统判定平局，终止本次对弈。否则，系统监控棋盘局面，发现某一方的棋子形成横、纵或斜线任意方向的五子连珠情况，则提示该棋手获胜，终止本次对弈。

投降功能见“2）-d）”；获胜提示见“2）-c）”；五子连珠测试见“三-2用例测试”章节。

1. （体现OOP思想）设计系统时考虑留下人-机对弈的接口，即考虑怎样很好的进行代码复用，很容易地替换某一个棋手为AI算法。



人机对弈界面。

**6、核心代码展示**

复制核心代码片段（可多个片段，要体现核心价值），图片也可。

**二、测试与结论**

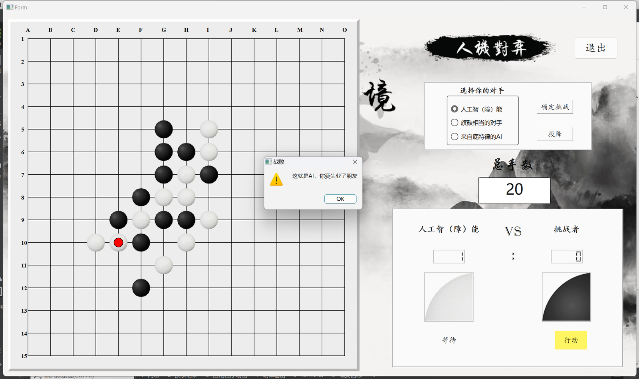
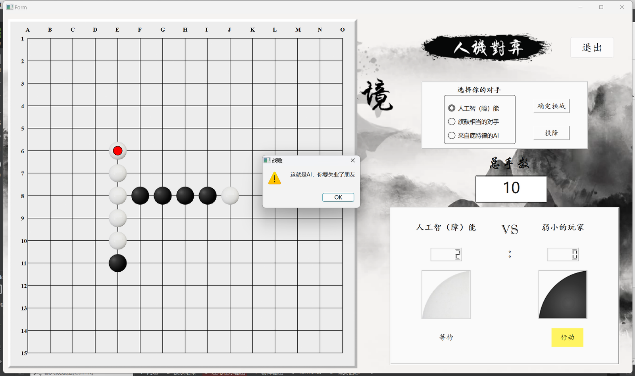
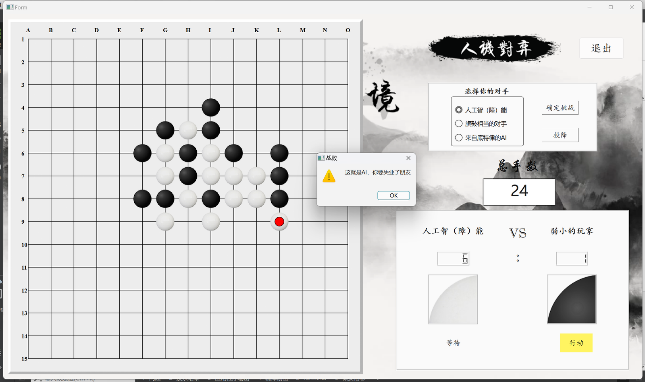
**1、测试环境与数据**

本项目使用C++ Qt 6.7.0和Qt Creator 13.0.0 Community编写，用Qt 6.7.0 (MinGW 11.2.0 64-bit)构建。本软件有两个发行版本，exe + 动态链接库、单exe文件。前者在装在win11系统，分辨率大于的不同设备上均正常运行，后者在少数设备上出现运行失败的情况。

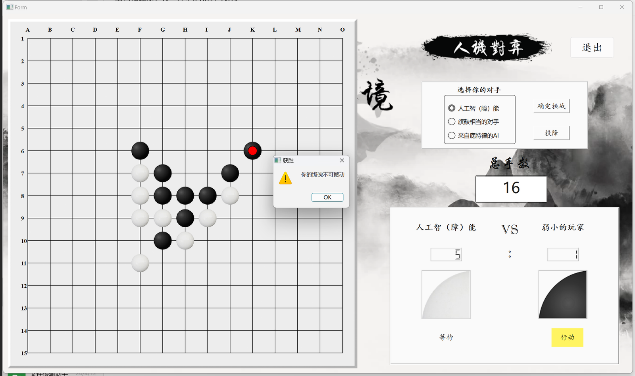
软件测试数据来自组内5名成员的大量人人、人机对局。

**2、测试用例**

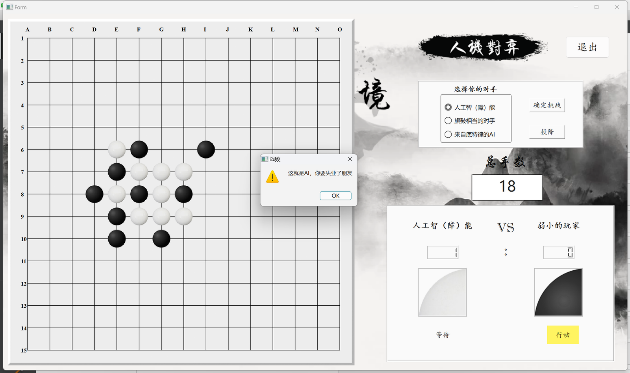
1. AI（白子）获胜。包含在多个方向上的测试



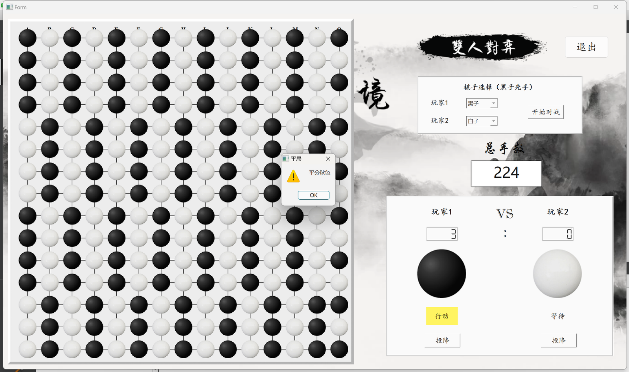
1. 玩家（黑子）获胜



1. 投降



1. 平局



**3、测试结论**

软件各部分运行正常，高质量完成了题目要求。

**三、总结与思考**

**1、题目难点要点**

1. 棋盘布局与交互设计：棋子的放置、移动，游戏规则的清晰传达以及用户界面的友好性。
2. 实时游戏状态更新：在五子棋游戏中，游戏状态需要实时更新，包括棋子的移动、吃子、胜负判定等。在前端实现这些实时更新需要考虑性能和用户体验，确保游戏状态同步和流畅。
3. 游戏规则的实现：合法落子的验证、各种棋局状态的管理（如判断胜负、平局等），确保游戏的公平性和可玩性。

**2、本组工作特点**

独创性：独特的胜负判断逻辑，接入AI算法，优秀的前端设计。

拓展性：拥有良好设计的API，前端开发和AI开发者均遵循OOP而非面向过程。

题目中最有价值的内容：OOP设计理念和项目开发能力。

**3、本组改进方向**

1. 游戏窗口大小固定，不能动态调整。考虑引入自适应布局。
2. 提供远程联机对战模式，支持玩家联机对战，并提供登录系统，保存用户信息。
3. 拓展AI的种类和下棋策略，增添人机对战的趣味性。
4. 对游戏所需的动态链接库进行枝剪，缩小游戏的大小。
5. 拓展游戏支持的设备类型，推出不同平台的版本（不同版本的Linux，MacOS，安卓等）。