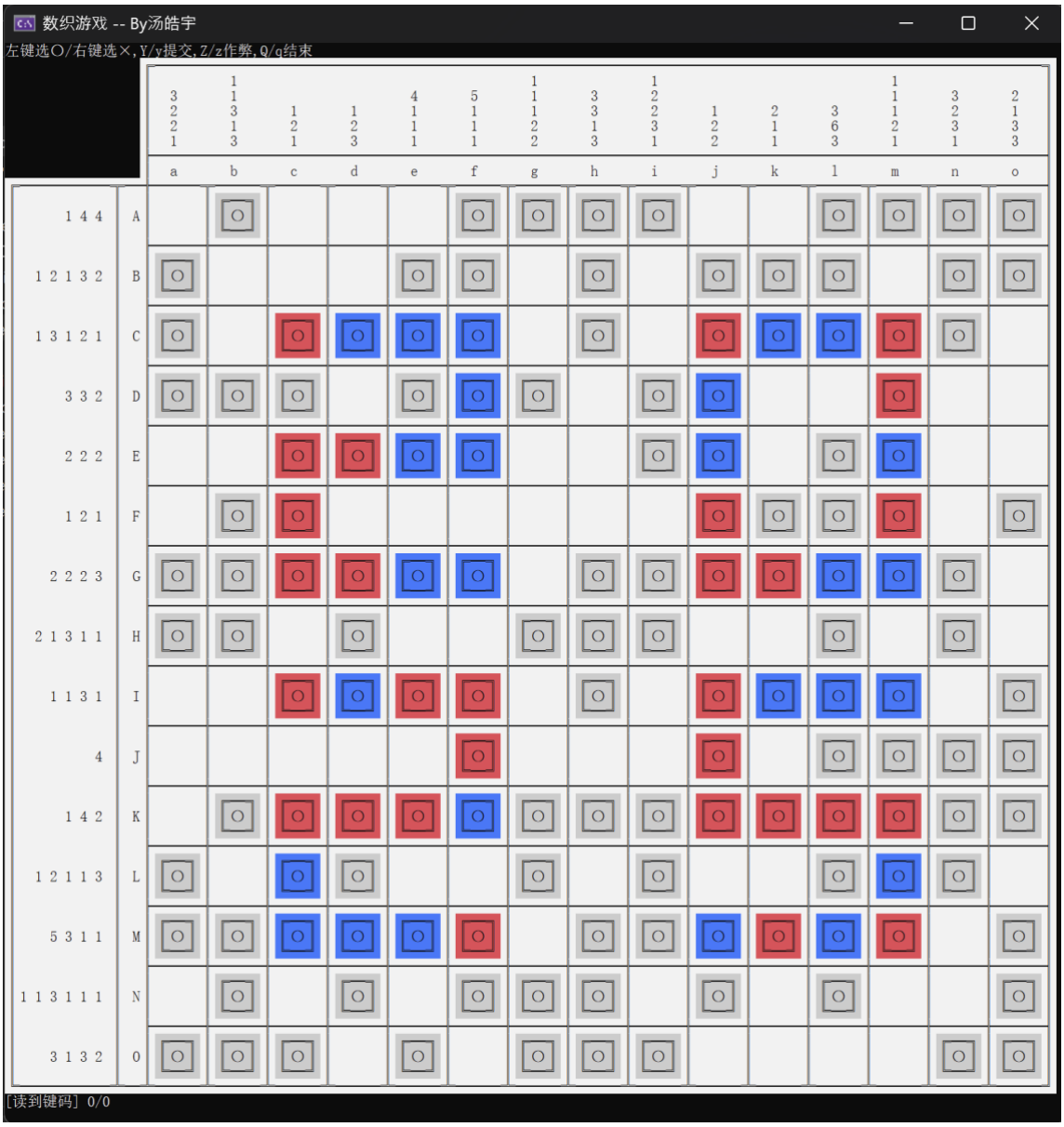
数织游戏实验报告



姓名：汤皓宇
班级：5000244001608
学号：2454307
完成日期：2025年5月15日

1. 题目

数织游戏是一种基于矩阵的逻辑推理类游戏，其游戏区域为一个矩形网格，最大支持在线操作 25×25 的大小，在 Demo 中为 15×15。在正式游戏中，系统会在该区域中随机生成一半（向上取整）数量的球，并根据球在行与列上的分布生成提示栏。提示栏位于游戏区的右侧（行提示）和下方（列提示），其中数字表示该行或列中从左到右或从上到下出现的连续球的个数。例如，“3 1”表示该行/列中存在连续 3 个球的一段与连续 1 个球的一段，中间由空格隔开。游戏的基本操作方式为：玩家通过鼠标左键标记某格存在球，右键标记不存在球，再次点击可取消已有标记。标记完成后点击“完成”按钮，系统将判断玩家推理的结果是否与实际布局一致，判断正确即通关，否则提示错误位置。

为完成该游戏的开发，需要采用伪图形界面方式构建游戏界面。开发过程中需参考附件中的控制台版本 Demo 程序 90-b2-demo.exe，要求在设置新版控制台时关闭快速编辑模式和插入模式，确保操作响应性和可视化效果。开发项目命名为 90-b2，其中包含 2 个头文件和 5 个源程序文件，最终编译生成 90-b2.exe。五个核心提交文件为：pullze.h、pullze_main.cpp、pullze_base.cpp、pullze_console.cpp 与 pullze_tools_.cpp，不得修改或提交 cmd_console_tools.cpp 与 cmd_console_tools.h。工具函数必须使用指定的伪图形库实现，项目中所有源文件需统一放在同一目录内。在界面展示上应充分利用控制台的宽高显示能力，在 1920×1080、100%缩放的分辨率下显示正常，并通过中文制表符画出边框。

为了实现完整的游戏功能，开发任务被划分为多个子题目，由浅入深逐步实现游戏的各个部分。子题目 A 要求初始化矩阵，支持从用户输入中识别合法行列标（行号从 A 开始，列号从 a 开始），并以“0”代表有球、空格代表无球的方式打印矩阵内容，每隔 5 行/5 列加入分隔线。子题目 B 在此基础上添加提示栏，数字对齐显示，宽度与高度可变。子题目 C 实现玩家输入坐标进行标记的交互机制，并添加作弊模式用于调试。子题目 D 采用伪图形绘制无分隔线的边框界面；子题目 E 引入提示栏与区域间分隔线；子题目 F 进一步加入鼠标移动支持，仅限数据区内实时读取并打印坐标；子题目 G 实现完整的游戏流程，支持按回车键提交答案并验证，若正确则结束游戏，错误则指出首个错误点，支持 Q/q 退出。后续子题目 H 至 K 为有分隔线版本的 D 至 G，增强可视化效果并严格控制鼠标坐标有效区域，使游戏在结构和交互上更完整、用户体验更清晰。

2. 整体设计思路

该数织游戏采用模块化的整体设计思路，将游戏逻辑、界面显示与用户交互进行分层处理，实现了功能清晰、结构分明的代码体系。游戏主体分为三个主要模块：Game 类负责游戏的核心逻辑，如球的位置存储、行列提示的计算、正确性验证与作弊模式控制，保证了整个游戏运行的逻辑性与独立性；Interface 类负责将这些数据转换为用户可见的界面，支持文字与图形两种风格，并根据玩家选

择的网格大小灵活调整界面布局；pullze_tools 函数组则承担用户输入、模式选择与流程控制的任务，通过菜单、提示行和输入响应等方式完成与玩家的交互。三者之间通过传递指针和参数联动，构成一个“输入-逻辑-显示”的完整闭环。

在功能配合上，各模块各司其职又紧密协作。Game 类专注于数据与规则管理，比如生成球的位置并计算行列中的连续段数，为玩家提供解谜依据，同时支持验证玩家标记是否正确。而 Interface 类通过指向 Game 的引用来获取当前状态并完成界面渲染，在文本模式下使用字符画出网格，在图形模式下则用颜色和图形符号增加可视化效果，并负责将鼠标点击等界面行为转化为坐标输入。工具函数则贯穿整个游戏流程，从最初的模式与尺寸选择，到后续的操作处理与提示展示，无论是在命令行操作中解析坐标输入，还是在图形模式下响应鼠标点击，都负责协调 Game 与 Interface 的联动，并提供玩家操作的实时反馈。

整套代码设计兼顾功能完整性与可扩展性，在显示方式上支持多种模式组合（如是否显示段数提示、是否有分隔线），在界面适配上通过动态调整控制台字体与边界确保美观清晰，同时还提供提示栏与状态栏帮助玩家理解当前状态。在结构层面，每个模块通过接口进行数据交流，常量与设置集中定义，函数设计尽量保持独立，便于后期维护或扩展，如新增一种界面风格或玩法时，只需扩展对应类而不影响整体结构。总的来看，这种设计方式不仅确保了游戏玩法的准确性与丰富性，也提升了交互体验，使得该游戏代码具有较高的清晰度、可维护性与可玩性。

3. 主要功能的实现

3.1. 游戏本体的构建

Game 类是数织游戏的核心逻辑模块，负责管理整个游戏的状态、规则验证和相关数据的存储。它包含多个关键的数据成员来支持游戏运行：int size 记录网格边长，决定整体规模；int numBalls 表示球的总数，其计算公式为网格面积的一半向上取整。bool map[MAX_SIZE][MAX_SIZE] 是二维布尔数组，用于标记每个网格位置是否有球。此外，还有 row_maxnumseg 与 col_maxnumseg 记录每行或每列中连续球段的最大数量，用于界面提示布局；row_numseg[] 与 col_numseg[] 则记录实际的段数，配合 row_seg[][] 与 col_seg[][] 这类数组用于记录每段的球数。玩家的操作状态通过 markings[][] 枚举数组记录，其值为 NONE、MARK_O、或 MARK_X。为了支持作弊模式与错误提示，还有如 correct、wrong_row、wrong_col 和 hideBalls 等状态标志。

该类的成员函数涵盖了从初始化、交互、作弊到验证的完整功能流程。构造函数 Game(int size) 初始化网格，调用 placeBalls() 随机生成球的位置，并依次调用 row_cntSegments() 与 col_cntSegments() 计算行列的段数提示。玩家可以调用 markO() 或 markX() 标记某个格子为有球或无球，再次点击可取消标记；作弊模式下可以用 markAllBalls() 自动标出所有球的位置。函数

get_rowSeg() 和 get_colSeg() 用于提供界面提示所需的段数信息。其他函数如 getSize()、isBall()、isOuted() 等则是基础的访问或判断接口，而 toggleCheatMode() 和 isCheating() 用于管理和查询作弊状态。

游戏核心逻辑集中体现在验证环节，即玩家完成标记后通过 submmit() 函数提交答案，系统将遍历所有行列并重新计算每行每列的球段数，与初始化阶段生成的段数提示进行比对。如果全部一致，则说明解谜成功；否则，会记录出错的行列（wrong_row、wrong_col），以便提示玩家错误位置。这个流程使得 Game 类不仅是游戏逻辑的中枢，也兼顾了提示机制、交互反馈和教学辅助功能，构成了一个完整而自洽的数织游戏框架。

3.2. 游戏界面的构建

Interface 类是数织游戏中连接用户与逻辑核心的关键模块，负责将 Game 类的游戏数据渲染为控制台界面，并支持用户输入的坐标解析与交互反馈。该类通过 Game* game 成员变量与游戏逻辑进行绑定，同时维护了界面渲染所需的布局参数，如 size 表示网格大小，xsegN 和 ysegN 用于记录行列最大段数，POSX_LU 与 POSY_LU 用于界面左上角坐标定位，保证不同规模网格在显示上的对齐与美观。此外，openGUI、showNumBar、showDividingLine 等布尔变量用于控制当前显示模式、是否显示提示段数以及是否启用分隔线，增强了界面自定义的灵活性。

在显示功能上，Interface 提供了多种辅助函数用于渲染游戏界面。构造函数 Interface(Game* game) 负责初始化界面参数并绑定游戏数据，adjustConsole() 会根据 size 与是否开启 GUI 模式来设置控制台字体与边框大小，使界面适配不同的显示需求。函数 printNstr() 和 printnum() 提供基础文本输出与数字格式化显示，而 printMarking() 则会根据格子的标记状态与正确性判断，输出带颜色的字符，如正确标记“O”为蓝色，错误标记“X”为红色，配合 showDividingLine 可显示边框线条。在网格渲染方面，printMap() 使用 ASCII 字符绘制纯文本模式的网格，并根据 showNumBar 显示提示数字；而 GUI_showMap() 则通过图形字符与颜色块在 GUI 模式下绘制更精致的界面，结合 POSX_LU 与 POSY_LU 实现对齐美观的界面排布，display() 会根据 openGUI 状态选择调用哪种渲染方式。

在交互机制方面，Interface 支持对用户输入的坐标进行解析和界面更新，使标记与显示状态实时同步。函数 GUI_updatePoint(row, col) 用于更新单个格子在 GUI 模式下的显示效果，而 GUI_update() 则用于遍历并刷新整个网格界面，确保所有状态可视化一致。为支持鼠标操作，GUI_get_relativePoint() 提供将鼠标绝对坐标转换为网格相对位置的能力，自动过滤非法区域（如边框、分隔线），并根据像素比例进行位置修正，如将列坐标除以 2 处理字符宽度差异。整体而言，Interface 类将抽象的数据状态与动态的用户交互通过渲染和映射紧密结合，构建了一个直观、清晰、响应及时的游戏界面，是游戏逻辑与用户体验之间不可或缺的桥梁。

3.3. 辅助功能模块的构建

pullze_tools 函数组是数织游戏的辅助功能模块，负责游戏运行过程中的用户交互、模式选择、状态提示与流程控制。在游戏启动阶段，Menu() 函数会首先调用，显示不同游戏模式（如文本模式、GUI 模式、带分隔线版本等）供玩家选择，利用 cct_setconsoleborder 和 cct_setfontsize 调整控制台布局，并通过 _getch() 读取并验证玩家输入。随后，SizeSelector() 进入执行，引导玩家选择网格大小（5x5、10x10、15x15），采用 cin 输入并结合输入流清理机制处理非法值，确保游戏尺寸合法。以上两个函数共同完成了游戏初始化配置，是进入正式游戏前的关键准备步骤。

在具体游戏过程中，pullze_tools 提供了两个主要的交互处理函数，分别服务于文本模式与图形界面模式。CommandLine(Game& game, Interface& interface) 用于命令行模式（如模式 C），引导用户通过键盘指令进行标记（如 Aa 表示第 0 行第 0 列），并支持提交（Y/y）、开启作弊模式（Z/z）或退出（X/x）。每次操作后界面通过 interface.display() 实时刷新，形成完整的人机交互流程。另一方面，EventResponder(Game &game, Interface &interface, bool testmode = false) 专为 GUI 模式设计，监听鼠标与键盘事件，在鼠标点击时使用 interface.GUI_get_relativePoint() 将像素坐标转换为网格坐标，处理左键标记 O、右键标记 X，并通过方向键、Z/Y/Q 控制作弊、提交或退出逻辑。若启用 testmode，函数将只显示事件不实际执行逻辑，便于调试或教学使用。这两种输入机制有效覆盖了不同玩家偏好的操作方式。

为增强用户体验，pullze_tools 还提供了一系列辅助提示与流程控制函数。StatusLine(const char content[]) 用于控制台底部显示状态信息，如“提交成功”“坐标无效”等，利用 cct_gotoxy 定位并清除原行内容，再写入新信息；TipLine(const char content[]) 则在顶部展示当前操作提示，如“左键选〇/右键选×”，起到操作引导作用；而 Endle() 函数用于每一小题结束时暂停流程，要求用户输入 “End” 才能继续，用于实现有节奏的模式切换与阶段推进。以上功能通过提示与流程暂停引导玩家操作与理解规则，提升游戏的可用性与趣味性。整体而言，pullze_tools 函数组与 Game 类和 Interface 类紧密协作，组成了数织游戏中完整、流畅且多样化的用户交互体系。

4. 调试过程碰到的问题

4.1. 图形错位问题

在编写数织游戏代码过程中，为解决控制台图形错位问题，我深入分析了坐标映射偏差的根源，并通过优化坐标计算逻辑和界面更新机制实现了精准映射。首先，我统一了界面定位基准，在 Interface 类中设置了左上角与右下角坐标（POSX_LU 等），确保所有渲染操作有共同参考原点。其次，针对分隔线存在与否引起的格子尺寸差异，通过 showDividingLine 标志分别采用 4×4 与 1×1 的缩放逻辑精确计算物理位置，并在 GUI_get_relativePoint 中剔除分隔线区域点击，避免逻辑坐标

错误。为了适应不同网格尺寸和显示模式，我在 adjustConsole 中根据实际情况动态设置控制台边界和字体大小，防止字体溢出造成显示偏移。最后，在 GUI_updatePoint 中对单格更新位置进行精准计算，确保局部刷新时定位准确。通过标准化坐标、差异化处理模式、动态适配尺寸和坐标验证机制等一整套流程，我成功解决了图形错位问题，实现了控制台界面与逻辑网格的高一致性，为后续开发提供了稳定可靠的交互基础。

4.2. 函数参数冗长问题

在开发数织游戏过程中，为解决函数间参数传递冗长的问题，我采用了面向对象的封装思想，将相关数据和行为聚合到类中，从根本上简化了参数管理。通过将游戏核心数据（如网格状态、标记信息、提示分段等）封装进 Game 类，将界面控制参数（如显示模式、字体设置、坐标基准等）封装进 Interface 类，避免了在函数之间频繁传递大量分散的变量。各函数只需传递类的实例指针或引用，就能访问所有所需数据与操作方法，从而显著精简函数签名，提升代码的可读性与模块化程度。同时，主程序中通过类实例组织游戏状态和界面操作，使控制流程更加清晰；在工具函数中，统一使用类引用访问内部数据，进一步提升了数据一致性与可维护性。总体而言，通过类封装有效地实现了参数简化、职责分离与结构优化，提升了整个项目的开发效率与代码质量。

5. 心得体会

完成这次数织游戏大作业的过程中，我对模块化设计与面向对象编程有了更深入的理解。整个项目采用类似 MVC 的结构，将游戏逻辑、界面渲染和用户交互明确分层，实现“高内聚、低耦合”的结构优化。Game 类作为模型层，集中管理数据与规则，通过 map、row_seg、col_seg 以及 markings 数组完成球位置管理与段数提示计算，并提供如 mark0()、submmit() 等接口供外部调用。Interface 类负责视图展现，既能以字符界面渲染网格，也支持控制台模拟的 GUI 模式，adjustConsole() 方法根据网格尺寸调整字体和边界，提升显示适应性。控制层由 pullze_tools 提供，封装了命令输入、鼠标键盘事件监听等功能，让主函数只需管理流程与模式选择，整体结构清晰明了。

在代码结构设计时，我深刻体会到封装带来的灵活性与安全性。Game 类隐藏了内部实现细节，如段数计算与验证，仅通过对外方法操作状态，既保障了逻辑安全，又方便调用。Interface 类同样将显示逻辑与内部数据隔离，利用常量和枚举管理颜色、标记类型等配置，使得全局风格修改更为集中和便捷。此外，一些功能的扩展也体现了良好的设计，例如作弊模式的实现仅通过布尔开关 hideBalls 控制是否调用 markAllBalls()，没有破坏原有逻辑。这种封闭开放并存的结构设计，让程序具备了良好的扩展性和可维护性。

游戏界面与交互部分的实现也让我意识到，良好的用户体验需要在结构设计之初就加以考虑。无论是 G 模式的无分隔线简约风格，还是 K 模式的细化分隔线展现，二者都通过统一接口实现，体现

了代码的高复用性。尤其是 GUI_get_relativePoint() 方法的设计，它不仅完成鼠标坐标到网格位置的转换，还自动过滤分隔线区域，从而使界面布局与交互响应相互独立。再如 StatusLine 与 TipLine 的实时提示机制，在提交失败、切换模式等操作后都能及时反馈，增强了用户对系统状态的感知，同时也为调试提供了便利。

通过这次项目实践，我逐步建立起了面向工程化的开发思维。在主函数中，我学习如何用清晰的流程组织整体运行逻辑，通过菜单与尺寸选择模块初始化，再依据不同模式（如 G 或 K）进入各自的交互流程。在细节处理上，我也更加重视程序的健壮性，例如输入处理时加入容错控制、资源管理中启用与关闭鼠标功能等。此外，在代码规范方面，我保持注释清晰、命名直观，即使带有中文标记也不妨碍功能表达，保证了后续阅读与维护的可读性。这次项目让我认识到，写出高质量的代码不仅是功能的实现，更是一种结构化、抽象化思维的体现，对我未来的软件开发学习与实践具有深远的意义。

6. 附件：源程序

```
// pullze_base
class Game
{
private:
    int size;
    int numBalls;
    bool map[MAX_SIZE][MAX_SIZE];

    int row_maxnumseg = 0;
    int col_maxnumseg = 0;
    int row_numseg[MAX_SIZE];
    int col_numseg[MAX_SIZE];
    int row_seg[MAX_SIZE][MAX_NUMSEG];
    int col_seg[MAX_SIZE][MAX_NUMSEG];

    Mark markings[MAX_SIZE][MAX_SIZE];

    bool correct = false;
    int wrong_row;
    int wrong_col;

    bool hideBalls = true;
    // Display Options

    void placeBalls(int num);
    void row_cntSegments(int row);
    void col_cntSegments(int col);
public:
    Game(int size);

    int getSize();
    bool isOuted(int row, int col);
    bool isBall(int row, int col);

    int get_rowMaxNumSeg();
    int get_colMaxNumSeg();
    void get_rowSeg(int row, int arr[]);
    void get_colSeg(int col, int arr[]);

    void toggleCheatMode();
    bool isCheating();

    void markO(int row, int col);
    void markX(int row, int col);
    void markAllBalls();
    Mark getMarking(int row, int col);
    bool isCorrect(int row, int col);

    bool submmit();
    int get_rowWrong();
    int get_colWrong();

    ~Game();
};


// pullze_console
class Interface
{
private:
    Game* game;
    int size;
```

```
    int xsegN, ysegN;
    int POSX_LU, POSY_LU;
    int POSX_RD, POSY_RD;

    const int GUI_STARTLINE = 1;

    bool openGUI;
    bool showNumBar;
    bool showDividingLine;

    void adjustConsole();

    void printNstr(const char str[], int times =
1, bool onBoard = false,
        const int BCOLOR = DEFAULT_BCOLOR,
const int FCOLOR = DEFAULT_FCOLOR);
   void printnum(int num, bool onBoard = false);
    void printMarking(Mark MARKING, int X =
NOT_SET, int Y = NOT_SET, bool isCorrect = true);

    void printMap();
    void GUI_showMap();
public:
    Interface(Game* game);
    void display(bool openGUI, bool showNumBar =
false, bool showDividingLine = false);

    void GUI_updatePoint(int col, int row);
    void GUI_update();
    bool GUI_get_relativePoint(int& absRow, int&
absCol);
};

// pullze_tools
char Menu();
int SizeSelector();
void   CommandLine(Game&   game,   Interface&
interface);
void   EventResponder(Game   &game,   Interface
&interface, bool testmode = false);
void StatusLine(const char content[]);
void TipLine(const char content[]);
void Endle();

void Game::placeBalls(int num)
{
    while (num) {
        int row = rand() % size;
        int col = rand() % size;
        if (map[row][col])
            continue;
        map[row][col] = true;
        --num;
    }
}
void Game::row_cntSegments(int row)
{
    int cnt, col = 0;
    while (col < size) {
        cnt = 0;
        while (col < size && !map[row][col])
            ++col;
        while (col < size && map[row][col])
            cnt++, col++;
        if (cnt == 0)
            continue;
        row_seg[row][row_numseg[row]++] = cnt;
    }
    if (row_maxnumseg < row_numseg[row])
        row_maxnumseg = row_numseg[row];
}

void Game::col_cntSegments(int col)
{
    int cnt, row = 0;
    while (row < size) {
        cnt = 0;
        while (row < size && !map[row][col])
            ++row;
        while (row < size && map[row][col])
            cnt++, row++;
        if (cnt == 0)
            continue;
        col_seg[col][col_numseg[col]++] = cnt;
    }
    if (col_maxnumseg < col_numseg[col])
        col_maxnumseg = col_numseg[col];
}

void Game::get_rowSeg(int row, int arr[])
{
    int k = row_maxnumseg - row_numseg[row];
    int i = 0;
    while (i < k)
        arr[i++] = 0;
    while (i < row_maxnumseg)
        arr[i++] = row_seg[row][i - k];
}

void Game::get_colSeg(int col, int arr[])
{
    int k = col_maxnumseg - col_numseg[col];
    int i = 0;
    while (i < k)
```

```
        arr[i++] = 0;
    while (i < col_maxnumseg)
        arr[i++] = col_seg[col][i - k];
}

bool Game::submmit()
{
    bool numCorrect = true;
    for (int row = 0; row < size; row++) {
        int col = 0;
       for (int k = 0; k < row_numseg[row]; k++)
{
            int cnt = 0;
            while    (markings[row][col]    !=
MARK_O && col < size)
                col++;
            while    (markings[row][col]    ==
MARK_O && col < size)
                col++, cnt ++;
            if (cnt != row_seg[row][k])
                numCorrect = false;
        }
    }
    for (int col = 0; col < size; col++) {
        int row = 0;
       for (int k = 0; k < col_numseg[col]; k++)
{
            int cnt = 0;
            while    (markings[row][col]    !=
MARK_O && row < size)
                row++;
            while    (markings[row][col]    ==
MARK_O && row < size)
                row++, cnt++;
            if (cnt != col_seg[col][k])
                numCorrect = false;
        }
    }
    return numCorrect;
}
// pullze_base.cpp
void   EventResponder(Game   &game,   Interface
&interface, bool testmode)
{
    if (testmode)
        TipLine("测试键盘/鼠标左键/右键，按回车
退出");
    else
        TipLine("左键选O/右键选×,Y/y 提交,Z/z
作弊,Q/q 结束");
    int row, col;
    int maction, keycode1, keycode2;
    cct_enable_mouse();
    cct_setcursor(CURSOR_INVISIBLE);
    while (true) {
        int              EVENT              =
cct_read_keyboard_and_mouse(col,  row,  maction,
keycode1, keycode2);
        //cout << row << ' ' << col << ' ' <<
maction << ' ' << keycode1 << ' ' << keycode2 <<
endl;
        if (EVENT == CCT_KEYBOARD_EVENT) {
            if (!testmode) {
                if   (keycode1   ==   'Q'   ||
keycode1 == 'q') {
                    StatusLine("[读到 Q/q，游
戏结束]");
                    break;
                }
                if   (keycode1   ==   'Z'   ||
keycode1 == 'z') {
                    if (!game.isCheating()) {
                        StatusLine("[作弊模
式开]");

    game.toggleCheatMode();

    interface.GUI_update();
                        continue;
                    }
                    else {
                        StatusLine("[作弊模
式关]");

    game.toggleCheatMode();

    interface.GUI_update();
                        continue;
                    }
                }
                if   (keycode1   ==   'Y'   ||
keycode1 == 'y') {
                    StatusLine("[读到提交
键]");
                    if (game.submmit()) {
                        StatusLine("[提交成
功]");
                        break;
                    }
                    else {
                       StatusLine("提交错误，
可用作弊模式查看");
                        continue;
                    }
```

```
                }
            }
            if (keycode1 == 13) {
                if (testmode) {
                    StatusLine("读到回车键");
                     break;
                }
            }
            if (keycode1 == 224) {
                if (keycode2 == 72)
                    StatusLine("读到上箭头");
                if (keycode2 == 80)
                    StatusLine("读到下箭头");
                if (keycode2 == 75)
                    StatusLine("读到左箭头");
                if (keycode2 == 77)
                    StatusLine("读到右箭头");
            }
            else {
                StatusLine("[读到键码] ");
                cout << keycode1 << '/' <<
keycode2;
            }
        }
        if (EVENT == CCT_MOUSE_EVENT) {
            bool                          outed
= !interface.GUI_get_relativePoint(row, col);
            if (!testmode && !outed) {
                if        (maction        ==
MOUSE_LEFT_BUTTON_CLICK) {
                    game.mark0(row, col);

    interface.GUI_updatePoint(row, col);
                }
                if        (maction        ==
MOUSE_RIGHT_BUTTON_CLICK) {
                    game.markX(row, col);

    interface.GUI_updatePoint(row, col);
                }
            }
            switch (maction) {
            case MOUSE_LEFT_BUTTON_CLICK:
                StatusLine("[读到左键] ");
                break;
            case MOUSE_RIGHT_BUTTON_CLICK:
                StatusLine("[读到右键] ");
                break;
            default:
                StatusLine("[当前光标] ");
                break;
            }
            if (!outed) {
                cout << (char)(row + 'A') << "
行";
                cout << (char)(col + 'a') << "
列";
            }
            else
                cout << "位置非法";
        }
    }
    cct_setcursor(CURSOR_VISIBLE_FULL);
    cct_disable_mouse();
}


// pullze_console

void Interface::GUI_updatePoint(int row, int col)
{
    int X = POSX_LU + (showDividingLine ? col *
4 : col) * 2;
    int Y = POSY_LU + (showDividingLine ? row *
4 : row);
    if (game->isCheating())
        printMarking(game->getMarking(row,
col), X, Y, game->isCorrect(row, col));
    else
        printMarking(game->getMarking(row,
col), X, Y);
}

void Interface::GUI_update()
{
    for (int i = 0; i < game->getSize(); i++)
        for (int j = 0; j < game->getSize(); j++)
            GUI_updatePoint(i, j);
}

bool       Interface::GUI_get_relativePoint(int&
absRow, int& absCol)
{
    absRow -= POSY_LU;
    absCol -= POSX_LU;
    absCol /= 2;
    if (showDividingLine) {
        if (absRow % 4 == 3 || absCol % 4 == 3)
            return false;
        absRow /= 4;
        absCol /= 4;
    }
    return !game->isOuted(absRow, absCol);
}
```

```
void Interface::printMarking(Mark MARKING, int X, int Y, bool isCorrect)
{
    if (X == -1 || Y == -1)
        cct_getxy(X, Y);
    int BCOLOR = NOT_SET, FCOLOR = NOT_SET;
    if (MARKING == MARK_O) {
        BCOLOR = isCorrect ? MARKING_O_BCOLOR : WRONG_O_BCOLOR;
        FCOLOR = isCorrect ? MARKING_O_FCOLOR : WRONG_O_FCOLOR;
    }
    if (MARKING == MARK_X) {
        BCOLOR = isCorrect ? MARKING_X_BCOLOR : WRONG_X_BCOLOR;
        FCOLOR = isCorrect ? MARKING_X_FCOLOR : WRONG_X_FCOLOR;
    }
    if (MARKING == NONE) {
        BCOLOR = isCorrect ? BOARD_BCOLOR : MISSING_O_BCOLOR;
        FCOLOR = isCorrect ? BOARD_FCOLOR : MISSING_O_FCOLOR;
    }
    if (showDividingLine) {
        if (MARKING == NONE && isCorrect) {
            cct_showstr(X, Y, "      ", BCOLOR, FCOLOR);
            cct_showstr(X, Y + 1, "        ", BCOLOR, FCOLOR);
            cct_showstr(X, Y + 2, "        ", BCOLOR, FCOLOR);
        }
        else {
            cct_showstr(X, Y, "┌──┐", BCOLOR, FCOLOR);
            cct_showstr(X, Y + 1, "│    │", BCOLOR, FCOLOR);
            cct_showstr(X, Y + 2, "└──┘", BCOLOR, FCOLOR);
        }
        X += 1 * 2, Y += 1;
    }
    if (MARKING == MARK_O) {
        cct_showstr(X, Y, MARKING_O, BCOLOR, FCOLOR);
    }
    if (MARKING == MARK_X) {
        cct_showstr(X, Y, MARKING_X, BCOLOR, FCOLOR);
    }
    if (MARKING == NONE) {
        cct_showstr(X, Y, isCorrect ? "  " : MARKING_O, BCOLOR, FCOLOR);
    }
    cct_gotoxy(POSX_RD, POSY_RD);
    cct_setcolor(DEFAULT_BCOLOR, DEFAULT_FCOLOR);
}
```