同济大学
TONGJI UNIVERSITY

《高级程序语言设计》

实验报告

实 验 名 称: 数织游戏的设计与实现

班　　级: 数据科学与大数据技术

学　　号: 2351495

姓　　名: 闫业豪

完 成 时 间: 2025 年 6 月 13 日

1. 题目

1.1.总体描述

本次实验要求通过伪图形界面方式完成数织游戏，通过菜单方式选择不同的功能。数织游戏是一种逻辑拼图游戏，玩家需要根据给定的行和列的提示数字，推理出每个格子是否有球。游戏的目标是通过逻辑推导，将所有球的位置标记出来。具体而言，有以下功能：

A: 初始化矩阵并打印。键盘输入行列值，用于确定游戏区域的大小，并对输入错误进行处理。显示初始数组，其中行号从 A 开始，列号从 a 开始，用“0”表示此位置有球，空格表示无球。为了便于查看，每 5 行/5 列输出 1 个分隔行/分隔列，注意交叉位置的符号不同。

B：生成行提示栏和列提示栏并打印。在 A 的基础上进行。行提示栏从左到右排列，数字之间有空格，整个行提示栏为右对齐，宽度要求动态调整。列提示栏从上到下排列，数字之间有空格，整个列提示栏为下对齐，高度要求动态调整。

C：无伪图形界面游戏模式。初始只显示提示区，数据区为空。键盘输入坐标（严格区分大小写），表示“标记该位置的球存在”，再次输入相同坐标则取消标记。不需要完成“标记为不存在”的操作。在作弊模式下，无颜色的“0”是有球存在但未标记过的，有颜色的“0”是有球存在且已标记过的，标记错误的显示为“X”。

D/H：在伪图形界面下画出初始状态。D 相当于 A 的伪图形化实现，数据区外边有边框，球之间没有分隔线,边框必须使用中文制表符绘制。H 相当于子题目 D 的有分隔线版本，分隔线也必须使用中文制表符绘制，每个球自身带一个小边框，同样需要用中文制表符绘制.

E/I：在伪图形界面下画出初始状态（含提示栏）。E 相当于 B 的伪图形化实现，球之间没有分隔线。行提示栏和行标、列提示栏和列标之间要有分隔线。希望和子题目 D 共用画框架的参数，通过参数决定是否打印状态栏。I 相当于子题目 E 的有分隔线版本，在子题目 E 的基础上增加分隔线的绘制

F/J：在伪图形界面下画出初始状态并支持鼠标移动。F 在 E 的基础上进行，鼠标操作的方法可参考 test_cct，无需额外查阅其他资料。只有在数据区才显示坐标，其他位置均为非法。将实时读取的坐标在数据区下方打印。在合法位置单击左键或右键后，打印读取的坐标并结束。J 相当于 F 的有分隔线版本，在子题目 F 的基础上考虑分隔线的影响。鼠标移动到数据区的分隔线上，也属于非法操作。

G/K：在伪图形界面下完成完整的游戏。G 在 F 的基础上进行。按 Y/y 提交，若正确则结束游戏；否则按从上到下、从左到右的顺序给出第一个错误位置并继续游戏。按 Q/q 键退出游戏，返回菜单。K 相当于 G 的有分隔线版本，整合子题目 I 和子题目 J 实现的功能，完成整个游戏的有分隔线版本开发。

2. 整体设计思路

2.1.程序的主体架构设计

本程序以G_function和K_function为最终目标，采用模块化分层设计，主体分为两个部分：基础支撑部分与核心实现部分：

基础支撑部分：

1. main主函数: 负责程序流程控制和功能分发，为G/K功能提供统一的调用入口。

2. 提供的cmd_console_tools图形库：提供G/K功能所需的底层技术支撑，包括精确坐标控制、颜色管理、鼠标键盘交互等。

核心实现部分主要可以分为3个部分：

1. 算法逻辑部分：数织游戏核心算法，包括generate_balls随机生成、calculate_all_hints提示计算、check_solution_by_hints解答验证等，为G/K提供游戏数学基础。

2. 界面绘制部分： draw_game_frame、draw_hints_area、display等函数实现G/K所需的复杂图形界面绘制和动态显示更新。

3. 交互控制部分： handle_game_logic函数实现G/K的完整交互体验，包括鼠标点击操作、键盘快捷键、状态切换等

在设计G/K功能时我考虑到这是程序中最复杂的交互功能，需要集成完整的游戏逻辑、图形界面、用户交互等多个要素，因此我采用了模块化分层设计：通过A-F等简单功能逐步验证，然后在G/K中进行集成和完善。这种设计方式保证了G/K功能的完整性和稳定性，最终实现了功能完备的数织游戏。

2.2.实验项目的文件组成

2.2.1. pullze.h

统一的头文件，包含所有必要的常量定义以及全部函数声明。为G/K功能提供统一的接口规范，确保各模块间的有序通信和数据类型一致性。定义了游戏网格的最大尺寸和提示数据的存储限制。

2.2.2. pullze_main.cpp

程序的入口，实现主循环和菜单系统。通过条件分支将用户选择分发到对应的功能模块，为G/K功能提供统一的调用环境和程序流程控制。

2.2.3. pullze_base.cpp

包含G/K功能的核心算法实现，主要有数织游戏的数学逻辑：球的随机分布生成、行列提示数据的自动计算等，为G/K提供完整的游戏逻辑支撑和数据处理能力。

2.2.4. pullze_console.cpp

负责G/K功能的图形界面实现，包含复杂的控制台绘制功能，实现游戏框架的动态构建、提示区域的精确显示、用户标记的实时更新等。同时处理鼠标键盘交互事件，为G/K提供完整的用户交互体验和视觉反馈系统，支持作弊模式和多种显示效果。

2.2.5. pullze_tools.cpp

提供G/K功能所需的辅助支持，包括用户界面的菜单展示、游戏尺寸的选择处理、各类输入的验证检查、程序流程的等待控制等。同时包含解答检查和错误定位相关算法，为G/K的完整游戏体验提供必要的工具支持和用户引导功能。

3. 主要功能的实现

3.1.主要功能的函数实现

3.1.1. init_2d_array(int arr[][MAX_SIZE], int size)函数

该函数用来初始化二维数组，将指定尺寸的数组中所有元素设置为0。输入参数为arr（二维数组指针）、size（数组尺寸）。此函数在每次开始新的游戏操作前都需要调用，用于设置初始状态，确保数据的清洁性和一致性。函数通过双重循环遍历数组的每个位置，将所有元素初始化为0值。

3.1.2. generate_balls(int game_data[][MAX_SIZE], int size)函数

该函数用来确定游戏数据和随机生成球的分布位置。输入参数为game_data（游戏数据数组）、size（游戏尺寸）。函数内部包含完整的随机数生成逻辑，使用时间种子确保每次运行的随机性。生成的球数量约为总格子数的一半，通过while循环不断随机选择位置，直到放置足够数量的球，避免重复放置在同一位置。

3.1.3. set_console(int game_data[][MAX_SIZE], int size, bool with_separator, bool with_hints, bool game_mode)函数

该函数用来动态配置控制台环境，根据游戏参数自动调整窗口大小和字体设置。输入参数为game_data（游戏数据）、size（游戏尺寸）、with_separator（是否带分隔符）、with_hints（是否显示提示）、game_mode（游戏模式标志）。函数首先计算游戏界面所需的空间尺寸，包括提示区域的宽度和高度，然后根据不同的显示模式选择合适的字体大小，最后设置控制台窗口的总体尺寸。这确保了G/K功能在不同配置下都能获得最佳的视觉效果和用户体验。

3.1.4. draw_game_frame(int game_data[][MAX_SIZE], int size, bool with_separator, bool with_hints)函数

该函数用来绘制游戏的基础框架结构，包括网格边框、行列标签和分隔线等元素。输入参数为game_data（游戏数据）、size（游戏尺寸）、with_separator（是否带分隔符）、with_hints（是否显示提示）。函数首先清屏并设置光标不可见，然后根据是否显示提示计算起始坐标，接着绘制列标签（a、b、c 等）和行标签（A、B、C 等），最后构建完整的网格框架。

3.1.5. calculate_all_hints(int size, int game_data[][MAX_SIZE], int row_hints[][MAX_HINTS], int col_hints[][MAX_SIZE])函数

该函数用来计算游戏所有行列的提示数字，是Nonogram游戏的核心算法实现。输入参数为size（游戏尺寸）、game_data（游戏标准答案数据）、row_hints（行提示输出数组）、col_hints（列提示输出数组）。函数首先初始化所有提示数组为0，然后通过analyze_line_hints辅助函数逐行逐列分析连续球块的分布情况。对于每一行，函数统计连续的球块长度并存储到行提示数组中；对于每一列，函数执行相同的操作并存储到列提示数组中。这个函数是整个Nonogram游戏逻辑的数学基础，确保生成的提示数据完全符合游戏规则。

3.1.6. draw_hints_area(int game_data[][MAX_SIZE], int size, bool with_separator)函数

该函数用来绘制提示数字区域，显示每行每列的连续球块提示信息。输入参数为 game_data（游戏数据）、size（游戏尺寸）、with_separator（是否带分隔符）。函数首先调用 calculate_all_hints 计算所有提示数据，然后确定提示区域的布局尺寸，接着分别绘制行提示区域和列提示区域的框架结构。最后将计算得到的数字提示精确定位到相应位置显示，确保提示数字与游戏网格完美对齐。这个函数是 G/K 功能中至关重要的组成部分，为用户提供了解题所需的全部数字线索。

3.1.7. display(int size, int game_data[][MAX_SIZE], int user_marks[][MAX_SIZE], bool with_separator, bool with_hints, bool cheat_mode)函数

该函数用来实时显示游戏当前状态，支持多种显示格式和作弊模式。输入参数为 size（游戏尺寸）、game_data（标准答案）、user_marks（用户标记）、with_separator（是否带分隔符）、with_hints（格式类型）、cheat_mode（作弊模式状态）。函数根据参数控制在不同位置显示游戏信息，支持普通模式、分隔符模式和作弊模式的显示需求。在作弊模式下通过颜色编码区分正确答案、用户标记和错误标记，为用户提供直观的指导。

3.1.8. handle_game_logic(int size, bool with_separator, bool with_hints, int game_data[][MAX_SIZE], int user_marks[][MAX_SIZE], bool* cheat_mode)函数

该函数用来实现 G/K 功能的完整交互操作，包括鼠标键盘事件处理和各种显示更新。输入参数为 size（游戏尺寸）、with_separator（是否带分隔符）、with_hints（是否显示提示）、game_data（标准答案）、user_marks（用户标记）、cheat_mode（作弊模式指针）。函数通过事件循环机制实现用户操作的实时响应，处理鼠标点击进行标记切换，支持键盘快捷操作如 Y 键验证、Z 键作弊、Q 键退出，并根据参数控制不同的显示方式。

3.1.9. check_line_hints(int user_marks[][MAX_SIZE], int line_index, int size, int hints[], int hint_count, bool is_row, int ball_mark_value)函数

该函数用来验证用户在某一行或某一列的标记是否与标准提示数据相符合。输入参数为user_marks（用户标记数据）、line_index（行或列的索引）、size（游戏尺寸）、hints（标准提示数组）、hint_count（提示数量）、is_row（是否为行验证）、ball_mark_value（球标记值）。函数首先分析用户在指定行或列的标记情况，统计连续的球标记长度生成用户提示数组，然后将用户生成的提示与标准提示进行逐一对比。只有当提示数量和每个提示值都完全匹配时才返回true。这个函数是解答验证系统的核心组件，确保用户的每一步操作都能得到准确的反馈，为G/K功能提供可靠的正确性判断机制。

3.1.10. check_solution_by_hints(int game_data[][MAX_SIZE], int user_marks[][MAX_SIZE], int size, int ball_mark_value)函数

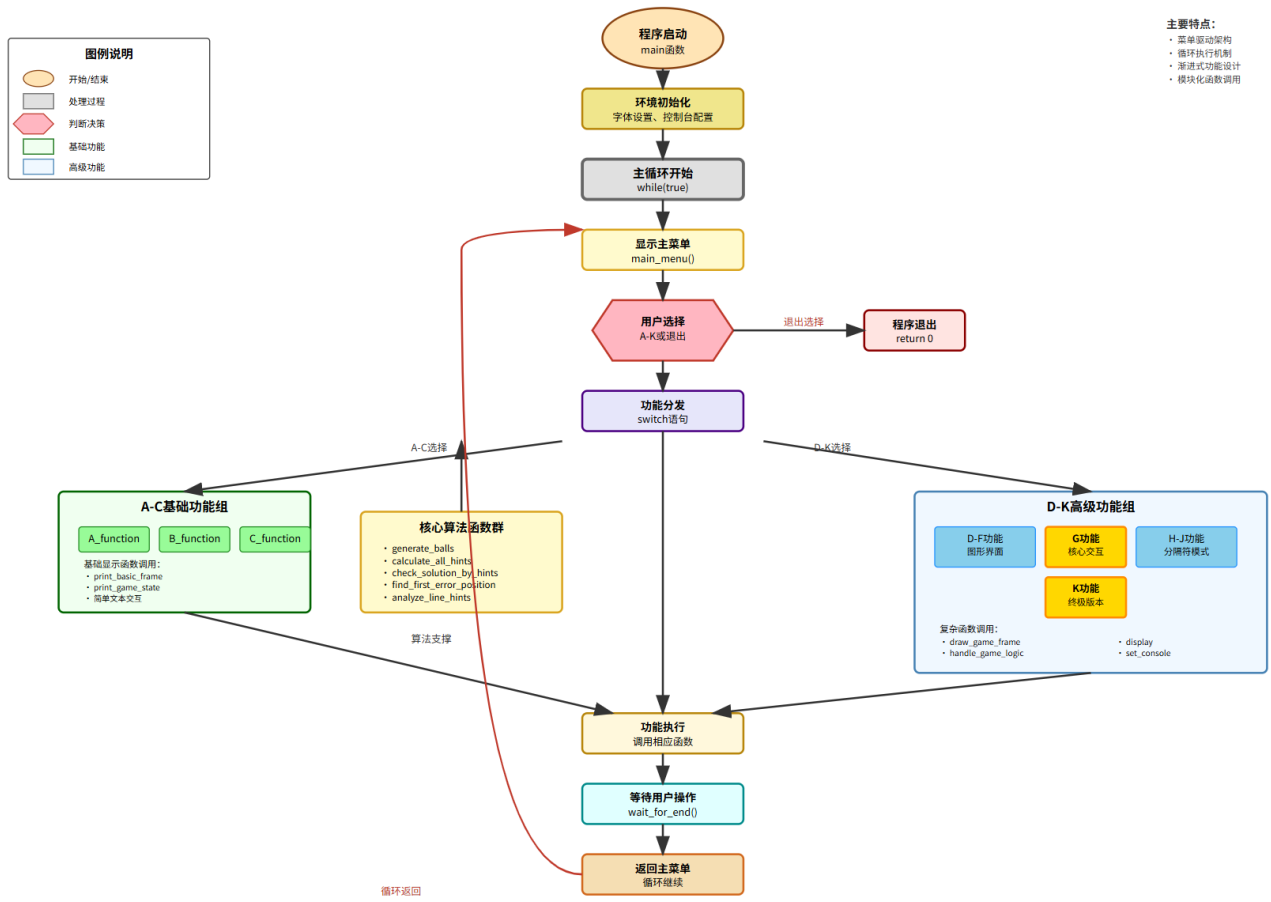
该函数用来检查用户解答的正确性验证，判断当前标记是否符合游戏规则。输入参数为 game_data（标准答案数据）、user_marks（用户标记数据）、size（游戏尺寸）、ball_mark_value（球标记值）。函数通过重新计算标准答案的提示数据，然后使用 check_line_hints 辅助函数逐行逐列验证用户标记生成的提示是否与标准提示一致。只有当所有行列的提示都完全匹配时才返回 true，确定解答的准确性，为 G/K 功能提供完整的验证机制和成功判定支持。

3.1.11. 其它辅助函数与解函数

除上述核心函数外，程序还包含多个辅助函数，如 calculate_hint_dimensions (计算提示栏长宽)、find_first_error_position (找到第一个错误位置)、wait_for_enter(等待回车键)、sizechoice(获取 size 大小)、to_be_continued(按回车键继续)等，以及 A 到 K 十一个菜单解函数，用来完成每一个菜单选项的功能。每个解函数都按照具体要求将上述核心函数组合调用，实现对应的功能需求。

3.2.主要功能的流程框架

该程序的主要流程为：通过main函数中的循环不断调用main_menu()函数进行选择解决函数，不同的解决函数通过调用不同的任务函数来完成该项解法的要求。程序采用菜单驱动的架构模式，在main函数中建立了一个持续的while循环，每次循环都会调用main_menu()函数显示功能选择界面，用户可以通过输入字符A-K来选择相应的功能模块。当用户选择具体功能后，程序通过switch语句进行功能分发，调用对应的解决函数来执行相应的任务。其中A-C项属于基础功能层次，主要需要用到基础显示函数，这些功能专注于数据的静态展示和简单的文本交互。D-K项则属于伪图形功能层次，需要通过复杂的图形界面函数和交互处理函数来完成完整游戏体验，包括draw_game_frame、handle_game_logic、display等函数的协同工作，实现了鼠标键盘交互、实时界面更新、状态管理等复杂功能。整个程序设计体现了从简单到复杂的渐进式功能实现思路，每个功能层都为后续更复杂的功能积累技术基础，最终在G/K功能中实现完整的游戏。



4. 调试过程碰到的问题

4.1. 问题1：有关作弊模式颜色显示的状态管理问题

在实现G和K功能的作弊模式时，遇到了复杂的颜色状态管理问题。作弊模式需要通过不同的颜色来区分用户的正确标记、错误标记和未标记状态，同时还要显示标准答案的提示信息。问题主要出现在display函数中，当用户频繁切换作弊模式时，不同状态下的颜色设置产生了冲突。在实际实现中发现，颜色状态在模式切换时没有得到正确的重置和更新。经过深入分析发现，问题的根源在于cheat_mode变量的状态传递和颜色设置的逻辑分支存在复杂的嵌套关系。最终通过重新整理display函数的逻辑结构，建立了清晰的颜色状态映射机制，确保每种游戏状态都有唯一对应的颜色表示，并且在模式切换时强制刷新所有显示区域，解决了颜色显示混乱的问题。

4.2. 问题2：有关鼠标坐标精确映射的算法问题

在实现handle_game_logic函数的鼠标交互功能时，遇到了坐标映射精度的复杂问题。游戏需要支持两种显示模式：无分隔符模式和分隔符模式，每种模式下的网格布局、单元格大小、间距都不相同。问题主要表现在用户点击位置与实际游戏逻辑坐标的转换上，特别是在分隔符模式下，由于每个单元格占

用8个字符宽度和4个字符高度，而且还要考虑分隔线的位置，坐标计算变得极其复杂。初期的实现使用了简单的除法运算来计算行列索引，但这种方法没有考虑到分隔符的特殊边界情况。当用户点击在分隔线上时，计算出的坐标可能指向错误的单元格。经过反复调试发现，需要在坐标转换过程中增加边界检查逻辑，特别是要排除点击在分隔符区域的情况。最终通过建立精确的坐标映射公式，结合模运算来检测点击位置是否在有效的游戏区域内，成功解决了坐标映射的精度问题。

5. 心得体会

5.1.完成本次作业得到的一些心得体会、经验教训

通过完成这次数织游戏项目，我最大的感受是编程能力得到了显著提升。从最初只会简单的算法实现，到最后能够构建一个功能完整的交互式游戏，这个过程让我对程序设计有了更深入的理解。在算法实现方面，数织游戏的核心提示计算算法让我深刻体会到了数学逻辑在编程中的重要作用，通过分析连续球块的分布规律来生成行列提示，这个过程需要考虑各种边界情况和特殊状态。在界面设计方面，从简单的文本输出到复杂的图形显示，学会了如何通过精确的坐标控制和颜色管理来创建用户友好的界面。调试过程中遇到的各种问题也给了我宝贵的经验，比如坐标映射错误、状态同步问题、边界条件处理等，通过反复测试和修正，我学会了如何系统性地分析和解决复杂的编程问题。项目的渐进式设计让我体会到了循序渐进的重要性，从A功能的基础实现开始，每增加一个功能都是对前一个功能的扩展和完善，这种方式不仅降低了编程难度，还帮助我更好地理解了程序的整体架构。最重要的收获是学会了如何平衡功能实现。

5.2.是否考虑了前后小题的关联关系

在完成各个功能时，我注重前后题目之间的代码复用和关联性。在设计A功能时就考虑到了后续功能的需求，将核心的游戏算法如随机球生成、提示计算、解答验证等函数设计得足够通用，使得这些函数在后续的所有功能中都能直接使用而无需修改。在数据结构设计上，统一使用game_data存储标准答案，user_marks记录用户操作，这种一致的数据表示让不同功能之间的数据传递变得简单可靠。在界面实现方面，通过参数化设计让同一套绘制函数能够适应不同的显示需求，比如通过布尔参数控制是否显示提示区域、是否使用分隔符等，避免了为不同模式重复编写相似的代码。功能实现采用了层层递进的策略，A、B、C功能建立了算法基础和基本显示框架，D、E、F功能在此基础上增加了图形界面和简单交互，G功能整合了完整的鼠标键盘交互逻辑，H、I、J功能验证了分隔符模式的可行性，最终K功能只需要调整参数配置就能实现最复杂的显示效果。要更好地重用代码，关键是要有完整的设计思维，并且要对项目足够的了解与熟悉，在编写每个函数时都要考虑它可能在哪些场景下被使用，然后通过合理的参数设计和接口规范来提高函数的通用性。同时要保持函数职责的单一性和清晰性，这样既便于调试和维护，也便于在不同的功能中灵活组合使用。

6. 附件：源程序

pullze.h

```
#include <iostream>
#include <iomanip>
#include <conio.h>
#include <ctime>
#include<Windows.h>
#include "cmd_console_tools.h"
using namespace std;
const int MAX_SIZE = 20;
const int MAX_HINTS =8;
char main_menu();
int sizechoice();
void to_be_continued(const char* prompt =
nullptr, const int X = 0, const int Y = 22);
void wait_for_end();
void init_2d_array(int arr[][MAX_SIZE], int
size);
void generate_balls(int game_data[][MAX_SIZE],
int size);
void calculate_hint_dimensions(int size, int
row_hints[][MAX_HINTS], int
col_hints[][MAX_SIZE], int* width, int* height);
void calculate_all_hints(int size, int
game_data[][MAX_SIZE], int
row_hints[][MAX_HINTS], int
col_hints[][MAX_SIZE]);
bool check_line_hints(int user_marks[][MAX_SIZE],
int line_index, int size, int hints[], int
hint_count, bool is_row, int ball_mark_value);
bool check_solution_by_hints(int
game_data[][MAX_SIZE], int
user_marks[][MAX_SIZE], int size, int
ball_mark_value);
bool find_first_error_position(int
game_data[][MAX_SIZE], int
user_marks[][MAX_SIZE], int size, int
ball_mark_value, int* error_row, int* error_col);
void A_function();
```

//其余类似

pullze_tools.cpp关键代码

// 检查整个游戏是否正确（基于提示而非原始位置）

```
bool check_solution_by_hints(int
game_data[][MAX_SIZE], int
user_marks[][MAX_SIZE], int size, int
ball_mark_value)
{
    int row_hints[MAX_SIZE][MAX_HINTS] = { 0 };
    int col_hints[MAX_HINTS][MAX_SIZE] = { 0 };
    calculate_all_hints(size, game_data,
row_hints, col_hints);
    for (int i = 0; i < size; i++)
    {
        int hints[MAX_HINTS];
        int hint_count = 0;
        for (int j = 0; j < MAX_HINTS &&
row_hints[i][j] != 0; j++)
        {
            hints[hint_count++] =
row_hints[i][j];
        }
        if (hint_count == 0)
        {
            hints[0] = 0;
            hint_count = 1;
        }
        if (!check_line_hints(user_marks, i,
size, hints, hint_count, true, ball_mark_value))
            return false;
    }
    for (int j = 0; j < size; j++)
    {
        int hints[MAX_HINTS];
        int hint_count = 0;
        for (int i = 0; i < MAX_HINTS &&
col_hints[i][j] != 0; i++)
        {
            hints[hint_count++] =
col_hints[i][j];
        }
        if (hint_count == 0)
        {
            hints[0] = 0;
            hint_count = 1;
        }
        if (!check_line_hints(user_marks, j,
size, hints, hint_count, false, ball_mark_value))
            return false;
    }
    return true;
}
```

pullze_console.cpp 关键代码

```
void set_console(int game_data[][MAX_SIZE], int
size, bool with_separator, bool with_hints,bool
game_mode = false)
{
    int start_x = 0, start_y = 0;
    int hint_width = 0, hint_height = 0;
    if (!with_hints)
    {
        start_x = 2;
        start_y = 1;
    }
    else
    {
        int row_hints[MAX_SIZE][MAX_HINTS] =
{ 0 };
```

```
        int col_hints[MAX_HINTS][MAX_SIZE] =
{ 0 };
        calculate_all_hints(size, game_data,
row_hints, col_hints);
        calculate_hint_dimensions(size,
row_hints, col_hints, &hint_width, &hint_height);
        start_x = hint_width * 2 + 6;
        start_y = hint_height + 3;
    }
    int data_x = start_x + 2;
    int data_y = start_y + 2;
    int cell_size = with_separator ? 8 : 2;
    int row_height = with_separator ? 4 : 1;
    int data_right = data_x + size * cell_size -
1;
    int data_bottom = data_y + size * row_height
- 1;
    int font_size;
    if (!with_separator)
    {
        font_size = 36;
    }
    else
    {
        if (size <= 5)
        {
            font_size = 24;
        }
        else if (size <= 10)
        {
            font_size = 16;
        }
        else
        {
            font_size = 12;
        }
    }
    cct_setfontsize("新宋体", font_size);
    int frame_right = start_x + 2 + size *
cell_size + 1;
    int frame_bottom = start_y + 2 + size *
row_height + 1;
    int info_area_height = 8;
    int total_width = 0;
    if(game_mode)
        total_width = frame_right + 20;
    else
        total_width = frame_right + 2;
    int total_height = frame_bottom +
info_area_height;

    cct_setconsoleborder(total_width,
total_height, total_width, total_height);
}
// 画游戏框架的统一函数
void draw_game_frame(int
game_data[][MAX_SIZE],int size,bool
with_separator, bool with_hints)
{
    cct_cls();
    cct_setcursor(CURSOR_INVISIBLE);
    int start_x=0, start_y=0;
    if (!with_hints)
    {
        start_x = 2;
        start_y = 1;
    }
    else
    {
        int row_hints[MAX_SIZE][MAX_HINTS] =
{ 0 };
        int col_hints[MAX_HINTS][MAX_SIZE] =
{ 0 };
        calculate_all_hints(size, game_data,
row_hints, col_hints);
        int hint_width = 0, hint_height = 0;
        calculate_hint_dimensions(size,
row_hints, col_hints, &hint_width, &hint_height);
        int col_sx = hint_width * 2 + 6, col_sy =
1;
        int row_sx = 0, row_sy = hint_height + 4;
        start_x = hint_width * 2 + 6;
        start_y = hint_height + 3;
    }
    cct_gotoxy(start_x, start_y);
    int st = (with_separator) ? 8 : 2;
    int label_sx = start_x + 2 *
(with_separator ? 2 : 1);
    cct_showstr(start_x, start_y, "  ",
COLOR_BLACK, COLOR_BLACK, 2 * (with_separator ?
2 : 1));
    cct_showstr(start_x, start_y+1, "╔",
COLOR_HWHITE, COLOR_BLACK, 1);
    if (!with_separator)
    {
        for (int i = 0; i < size; i++)
        {
            if (!with_hints)
            {
                cct_showch(label_sx + i * st,
start_y, (char)('a' + i), COLOR_BLACK,
COLOR_HWHITE, 1);
                cct_showch(label_sx + i * st + 1,
start_y, ' ', COLOR_BLACK, COLOR_HWHITE, st - 1);
            }
            else
            {
                cct_showch(label_sx + i * st,
start_y, ' ', COLOR_HWHITE, COLOR_HWHITE, 1);
                cct_showch(label_sx + i * st + 1,
start_y, (char)('a' + i), COLOR_HWHITE,
COLOR_BLACK, 1);
```

```
                cct_showch(label_sx + i * st + 2,
start_y, ' ', COLOR_HWHITE, COLOR_HWHITE, st -
1);
            }
        }
        cct_showstr(start_x+2, start_y + 1, "═",
COLOR_HWHITE, COLOR_BLACK, size);
        cct_showstr(start_x +2+ size * st,
start_y + 1, "┐", COLOR_HWHITE, COLOR_BLACK, 1);
        for (int i = 0; i < size; i++)
        {
            if (!with_hints)
                cct_showch(start_x-
2,start_y+2+i,(char)('A'+i),COLOR_BLACK,
COLOR_HWHITE, 1);
            else
            {
                cct_showstr(start_x-2, start_y +
2 + i, " ", COLOR_HWHITE, COLOR_HWHITE, 1);
                cct_showch(start_x - 1, start_y +
2 + i, (char)('A' + i), COLOR_HWHITE,
COLOR_BLACK, 1);
            }
            if (!with_hints)
                cct_showstr(start_x , start_y + 2
+ i, " ", COLOR_BLACK, COLOR_BLACK, 1);
            else
                cct_showstr(start_x , start_y + 2
+ i, " ", COLOR_HWHITE, COLOR_HWHITE, 1);
            cct_showstr(start_x , start_y + 2 +
i, "│", COLOR_HWHITE, COLOR_BLACK, 1);
            cct_showstr(start_x + 1, start_y + 2
+ i, " ", COLOR_HWHITE, COLOR_BLACK, 2 * size+1);
            cct_showstr(start_x+2*size+2, start_y
+ 2 + i, "│", COLOR_HWHITE, COLOR_BLACK, 1);
            my_sleep(with_hints);
        }
        cct_showstr(start_x, start_y + 2+size, "
└", COLOR_HWHITE, COLOR_BLACK, 1);
        cct_showstr(start_x + 2, start_y +
2+size, "═", COLOR_HWHITE, COLOR_BLACK, size);
        cct_showstr(start_x + 2 + size * st,
start_y + 2+size, "┘", COLOR_HWHITE,
COLOR_BLACK, 1);
    }
    //有分隔线同理
    // 画提示栏的统一函数
void draw_hints_area(int game_data[][MAX_SIZE],
int size,bool with_separator)
    {
        cct_setcursor(CURSOR_INVISIBLE);
        int row_hints[MAX_SIZE][MAX_HINTS] =
{ 0 };
        int col_hints[MAX_HINTS][MAX_SIZE] =
{ 0 };
        calculate_all_hints(size, game_data,
row_hints, col_hints);
        int hint_width = 0, hint_height = 0;
        calculate_hint_dimensions(size,
row_hints, col_hints, &hint_width, &hint_height);
        int col_sx = hint_width * 2 + 6, col_sy =
1;
        int row_sx = 0, row_sy = hint_height + 4;
        int st = (with_separator) ? 8 : 2;
        cct_showstr(col_sx, col_sy, "╔",
COLOR_HWHITE, COLOR_BLACK, 1);
        cct_showstr(row_sx, row_sy, "╔",
COLOR_HWHITE, COLOR_BLACK, 1);
        cct_showstr(row_sx + 2, row_sy, "═",
COLOR_HWHITE, COLOR_BLACK, hint_width);
        cct_showstr(row_sx + 2 + 2 * hint_width,
row_sy, "╤═══╪", COLOR_HWHITE, COLOR_BLACK, 1);
        if (!with_separator)
        {
            //列
            cct_showstr(col_sx + 2, col_sy, "═",
COLOR_HWHITE, COLOR_BLACK, size);
            cct_showstr(col_sx + size * st + 2,
col_sy, "┐", COLOR_HWHITE, COLOR_BLACK, 1);
            for (int i = 0; i < hint_height; i++)
            {
                cct_showstr(col_sx, col_sy + 1 +
i, "│", COLOR_HWHITE, COLOR_BLACK, 1);
                cct_showstr(col_sx + 2, col_sy +
1 + i, " ", COLOR_HWHITE, COLOR_HWHITE, size *
st);
                cct_showstr(col_sx + 2 + size *
st, col_sy + 1 + i, "│", COLOR_HWHITE,
COLOR_BLACK, 1);
            }
            for (int j = 0; j < size; j++)
            {
                int col_len = 0;
                for (int i = 0; i < MAX_HINTS &&
col_hints[i][j] != 0; i++)
                    col_len++;
                for (int i = 0; i < col_len; i++)
                    cct_showch(col_sx + 2 + j *
st + 1, col_sy + hint_height - col_len + i + 1,
'0' + col_hints[i][j], COLOR_HWHITE, COLOR_BLACK,
1);
            }
             cct_showstr(col_sx,
col_sy+hint_height+1, "╠", COLOR_HWHITE,
COLOR_BLACK, 1);
             cct_showstr(col_sx + 2, col_sy +
hint_height + 1, "═", COLOR_HWHITE, COLOR_BLACK,
size);
             cct_showstr(col_sx + size * st + 2,
col_sy + hint_height + 1, "╡", COLOR_HWHITE,
COLOR_BLACK, 1);
             cct_showstr(col_sx, col_sy +
hint_height + 2, "│", COLOR_HWHITE, COLOR_BLACK,
1);
```

```
            cct_showstr(col_sx + size * st + 2, col_sy + hint_height + 2, "┃", COLOR_HWHITE, COLOR_BLACK, 1);
            cct_showstr(col_sx + size * st + 2, col_sy + hint_height + 3, "┫", COLOR_HWHITE, COLOR_BLACK, 1);
            //行
            for (int i = 0; i < size; i++)
            {
                cct_showstr(row_sx, row_sy + i + 1, "┃", COLOR_HWHITE, COLOR_BLACK, 1);
                cct_showstr(row_sx + 2, row_sy + i + 1, " ", COLOR_HWHITE, COLOR_HWHITE, 2 * hint_width);
                cct_showstr(row_sx + 2 + 2 * hint_width, row_sy + i + 1, "┃", COLOR_HWHITE, COLOR_BLACK, 1);
            }
            for (int i = 0; i < size; i++)
            {
                int row_len = 0;
                for (int j = 0; j < MAX_HINTS && row_hints[i][j] != 0; j++)
                    row_len++;
                for (int j = 0; j < row_len; j++)
                    cct_showch(row_sx + 2 + 2 * hint_width - 2 - 2 * (row_len - 1 - j), row_sy + i + 1, '0' + row_hints[i][j], COLOR_HWHITE, COLOR_BLACK, 1);
            }
            cct_showstr(row_sx, row_sy + size + 1, "┗", COLOR_HWHITE, COLOR_BLACK, 1);
            cct_showstr(row_sx + 2, row_sy + size + 1, "═", COLOR_HWHITE, COLOR_BLACK, hint_width);
            cct_showstr(row_sx + 2 + 2 * hint_width, row_sy + size + 1, "┻┻", COLOR_HWHITE, COLOR_BLACK, 1);
        }
    //有分隔线同理
    // 游戏显示函数
    void display(int size, int game_data[][MAX_SIZE], int user_marks[][MAX_SIZE],bool with_separator, bool with_hints, bool cheat_mode)
    {
        int ball_startx = 0, ball_starty = 0;
        if (!with_hints)
        {
            ball_startx = 4;
            ball_starty = 3;
        }
        else
        {
            int row_hints[MAX_SIZE][MAX_HINTS] = { 0 };
            int col_hints[MAX_HINTS][MAX_SIZE] = { 0 };
            calculate_all_hints(size, game_data, row_hints, col_hints);
            int hint_width = 0, hint_height = 0;
            calculate_hint_dimensions(size, row_hints, col_hints, &hint_width, &hint_height);
            ball_startx = hint_width * 2 + 8;
            ball_starty = hint_height + 5;
        }
        if (!with_separator)
        {
            for (int i = 0; i < size; i++)
            {
                for (int j = 0; j < size; j++)
                {
                    int x = ball_startx + j * 2;
                    int y = ball_starty + i;
                    if (cheat_mode)
                    {
                        if (user_marks[i][j] == MARK_NONE)
                        {
                            if (game_data[i][j] == 1)
                                cct_showstr(x, y, "○", COLOR_WHITE, COLOR_BLACK, 1);
                            else
                                cct_showstr(x, y, "  ", COLOR_HWHITE, COLOR_BLACK, 1);
                        }
                        else if (user_marks[i][j] == MARK_LEFT)
                        {
                            if (game_data[i][j] == 1)
                                cct_showstr(x, y, "○", COLOR_HBLUE, COLOR_BLACK, 1);
                            else
                                cct_showstr(x, y, "○", COLOR_HRED, COLOR_BLACK, 1);
                        }
                        else if (user_marks[i][j] == MARK_RIGHT)
                        {
                            if (game_data[i][j] == 0)
                                cct_showstr(x, y, "×", COLOR_HRED, COLOR_BLACK, 1);
                            else
                                cct_showstr(x, y, "×", COLOR_HBLUE, COLOR_BLACK, 1);
                        }
                    }
                    else
                    {
```

```
                        if (user_marks[i][j] ==
MARK_LEFT)
                            cct_showstr(x, y, "○
", COLOR_HBLUE, COLOR_BLACK, 1);
                        else if (user_marks[i][j]
== MARK_RIGHT)
                            cct_showstr(x, y, "×
", COLOR_HRED, COLOR_BLACK, 1);
                        else
                            cct_showstr(x, y, "
", COLOR_HWHITE, COLOR_BLACK, 1);
                    }
                }
            }
        }
    //有分隔线同理
    // 游戏逻辑处理函数
    void handle_game_logic(int size, bool
with_separator, bool with_hints,int
game_data[][MAX_SIZE], int
user_marks[][MAX_SIZE],bool* cheat_mode)
    {
        int start_x = 0, start_y = 0;
        int hint_width = 0, hint_height = 0;
        if (!with_hints)
        {
            start_x = 2;
            start_y = 1;
        }
        else
        {
            int row_hints[MAX_SIZE][MAX_HINTS] =
{ 0 };
            int col_hints[MAX_HINTS][MAX_SIZE] =
{ 0 };
            calculate_all_hints(size, game_data,
row_hints, col_hints);
            calculate_hint_dimensions(size,
row_hints, col_hints, &hint_width, &hint_height);
            start_x = hint_width * 2 + 6;
            start_y = hint_height + 3;
        }
        int data_x = start_x + 2;
        int data_y = start_y + 2;
        int cell_size = with_separator ? 8 : 2;
        int row_height = with_separator ? 4 : 1;
        int data_right = data_x + size *
cell_size - 1;
        int data_bottom = data_y + size *
row_height - 1;
        int info_x = start_x;
        int info_y = data_bottom + 2;
        cct_showstr(0, 0, "左键选○/右键选×,Y/y
提交,Z/z 作弊,Q/q 结束", COLOR_BLACK,
COLOR_HWHITE, 1);
        cct_enable_mouse();
        cct_setcursor(CURSOR_INVISIBLE);
        int loop = 1;
        while (loop)
        {
            int X, Y, maction, keycode1,
keycode2;
            int ret =
cct_read_keyboard_and_mouse(X, Y, maction,
keycode1, keycode2);
            if (ret == CCT_MOUSE_EVENT)
            {
                cct_gotoxy(info_x, info_y);
                cout << "[当前光标] ";
                bool valid_position = false;
                int row = -1, col = -1;
                if (X >= data_x && X <=
data_right && Y >= data_y && Y <= data_bottom)
                {
                    col = (X - data_x) /
cell_size;
                    row = (Y - data_y) /
row_height;
                    if (with_separator && (((X -
data_x - 6) % 8 == 0) || ((X - data_x - 7) % 8 ==
0) || ((Y - data_y - 3) % 4 == 0)))
                        valid_position = false;
                    else if (row >= 0 && row <
size && col >= 0 && col < size)
                        valid_position = true;
                }
                if (valid_position)
                {
                    cct_gotoxy(info_x + 11,
info_y);
                    cout << (char)('A' + row) <<
"行" << (char)('a' + col) << "列
";
                }
                else
                {
                    cct_gotoxy(info_x + 11,
info_y);
                    cout << "位置非法
";
                }
                switch (maction)
                {
                case MOUSE_ONLY_MOVED:
                    break;
                case MOUSE_LEFT_BUTTON_CLICK:
                    if (valid_position)
                    {
                        if (user_marks[row][col]
== MARK_LEFT)
                            user_marks[row][col]
= MARK_NONE;  // 如果已经是左键标记，清除
                        else
```

```
                        user_marks[row][col]
= MARK_LEFT;  // 否则设为左键标记
                        display(size, game_data,
user_marks, with_separator, with_hints,
*cheat_mode);
                        cct_gotoxy(info_x,
info_y);
                        cout << "[读到左键] " <<
(char)('A' + row) << "行" << (char)('a' + col) <<
"列
";
                    }
                    else
                    {
                        cct_gotoxy(info_x,
info_y);
                        cout << "[读到左键] 位置
非法
";
                    }
                    break;
                case MOUSE_RIGHT_BUTTON_CLICK:
                    if (valid_position)
                    {
                        if (user_marks[row][col]
== MARK_RIGHT)
                            user_marks[row][col]
= MARK_NONE;  // 如果已经是右键标记，清除
                        else
                            user_marks[row][col]
= MARK_RIGHT; // 否则设为右键标记
                        display(size, game_data,
user_marks, with_separator, with_hints,
*cheat_mode);
                        cct_gotoxy(info_x,
info_y);
                        cout << "[读到右键] " <<
(char)('A' + row) << "行" << (char)('a' + col) <<
"列
";
                    }
                    else
                    {
                        cct_gotoxy(info_x,
info_y);
                        cout << "[读到右键] 位置
非法
";
                    }
                    break;
                }
            }
            else if (ret == CCT_KEYBOARD_EVENT)
            {
                if (keycode1 == 'Y' || keycode1
== 'y')
                { // Y 键提交
                    // 使用基于提示的正确性检查
                    if
(check_solution_by_hints(game_data, user_marks,
size, MARK_LEFT))
                    {
                        cct_gotoxy(info_x,
info_y );
                        cout << "[提交成功]
";
                        cct_setcolor();
                        wait_for_end();
                        loop = 0;
                    }
                    else
                    {
                        int error_row, error_col;
                        if
(find_first_error_position(game_data, user_marks,
size, MARK_LEFT, &error_row, &error_col))
                        {
                            cct_gotoxy(info_x,
info_y);
                            cout << "提交失败，坐
标[" << (char)('A' + error_row) << "]行[" <<
(char)('a' + error_col) << "]列不符合要求";
                            cct_setcolor();
                        }
                        else
                        {
                            cct_gotoxy(info_x,
info_y );
                            cout << "提交错误，可
用作弊模式查看     ";
                        }
                        cct_setcolor();
                    }
                }
                else if (keycode1 == 'Q' ||
keycode1 == 'q') // 退出
                {
                    loop = 0;
                    cct_gotoxy(info_x, info_y);
                    cout << "读到 Q/q，游戏结束
";
                    wait_for_end();
                }
                else if (keycode1 == 'Z' ||
keycode1 == 'z')
                { // 切换作弊模式
                    *cheat_mode = !(*cheat_mode);
                    display(size, game_data,
user_marks, with_separator, with_hints,
*cheat_mode);
                    cct_gotoxy(info_x, info_y);
                    if (*cheat_mode)
                        cout << "[作弊模式开]
";
```

```
                    else
                        cout << "[作弊模式关]
";
                    cct_setcolor();
                }
                else
                {
                    cct_gotoxy(info_x, info_y);
                    cout << "[读到键码]
";
                    cct_gotoxy(info_x + 11,
info_y);
                    if (keycode2)
                        cout << keycode1 << '/'
<< keycode2 << "          ";
                    else
                        cout << keycode1 << "/0
";
                }
            }
        }
        cct_disable_mouse();
    }
    // G 功能
    void G_function()
    {
        int size = sizechoice();
        int game_data[MAX_SIZE][MAX_SIZE];
        int user_marks[MAX_SIZE][MAX_SIZE];
        init_2d_array(game_data, size);
        init_2d_array(user_marks, size);
        generate_balls(game_data, size);
        set_console(game_data, size, false, true,
true);
        bool cheat_mode = false;
        draw_game_frame(game_data, size, false,
true);
        draw_hints_area(game_data, size, false);
        display(size, game_data, user_marks,
false, true, cheat_mode);
        handle_game_logic(size, false, true,
game_data, user_marks, &cheat_mode);
    }
    // K 功能同理
```

pullze_base.cpp关键代码

```
    //初始化二维数组
    void init_2d_array(int arr[][MAX_SIZE], int
size)
    {
        for (int i = 0; i < size; i++)
            for (int j = 0; j < size; j++)
                arr[i][j] = 0;

    }
    // 生成随机球位置
    void generate_balls(int
game_data[][MAX_SIZE], int size)
    {
        srand((unsigned int)time(0));
        int total_cells = size * size;
        int ball_count = (total_cells + 1) / 2;
        int placed = 0;
        while (placed < ball_count)
        {
            int row = rand() % size;
            int col = rand() % size;
            if (game_data[row][col] == 0)
            {
                game_data[row][col] = 1;
                placed++;
            }
        }
    }
```