# 数织调试报告

## ——重生之我要成为小函数分解的神之中道崩殂版

**2452545 刘晴语 计算机**

**课号：5000244001608**

**2025.6.11**

## 1. 题目

数织游戏规则描述：

2、矩形初始为空，然后在矩形中生成一半的球(向上取整，即5*5为13个球)，再生成行提示栏及列提示栏，提示栏中为多个数字，表示从左到右连续出现的球的个数.

3、用鼠标对数据区进行操作，左键标记为该位置球存在，右键标记为不存在，再次点击则消除标记

4、将所有球位置标记出来后，点击“完成”按钮，系统自动告知游戏是否成功

整个题目分为三大板块，分别为不同呈现方式的相同功能实现。

### 1.1 内部数组版

- 子题目A:初始化矩阵并打印

  键盘输入行列(要处理输入错误，下同)。显示初始数组，行号从A开始，列号从a开始，输出0表示此位置有球，空格表示无球。每5行/5列输出1个分隔行/分隔列

- 子题目B:生成行提示栏和列提示栏并打印

  行提示栏从左到右，整个行提示栏为右对齐，数字之间有空格，宽度要求动态调整列提示栏从上到下，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

- 子题目C:初始矩阵及行/列提示栏生成后，可以开始游戏

  键盘输入坐标(严格区分大小写)，表示“标记该位置的球存在”(等价于左键)，再次输入相同坐标则取消标记选择作弊模式，无颜色的“0”是有球存在但未标记过的，有颜色“0”是有球存在且标记过的，标记错误的显示为“X”

### 1.2 n*n的框架（无分隔线）

共4小题，需要实现的功能与1.1相似，在此基础上将键盘输入坐标改为读取鼠标光标的坐标值并且根据点击的按键（左右）改变内部状态。

### 1.3 n*n的框架（有分隔线）

共4小题，需要实现的功能与1.2相似，在此基础上将整个棋盘扩大，并且改变球的形状。

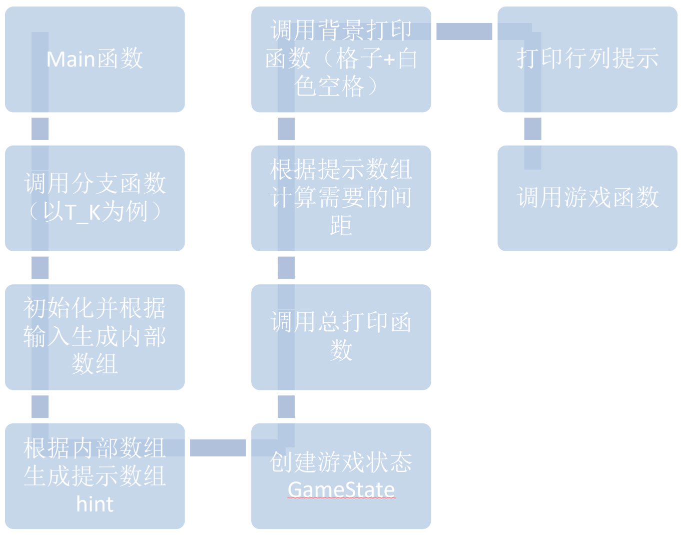
## 2. 整体设计思路

对于main函数，需要考虑输入错误，并且需要可以重复进行游戏，因此直接将整个main函数设为永真循环（按Q/q退出）。引入menu函数，返回值为输入的选项，main函数根据返回值确定调用的另一个cpp中的分支函数。

对于ABC功能，在写整个程序前，为了整合相同功能至一个函数中以减少代码行数与工作量，我尽力进行了统筹规划，大概确定了需要的函数有生成内部数组、读取并生成提示数组，打印背景+数组函数。游戏功能需要额外的判断提交是否正确的对照函数。

对于后续EFGHIJKL功能，除了按照之前的还需要考虑到标记错误的功能。因此引入结构体标记每个球的原状态和标记状态。同时根据给出的工具函数（读取光标等）构建游戏函数。

## 3. 主要功能的实现



Main函数
调用分支函数（以T_K为例）
初始化并根据输入生成内部数组
根据内部数组生成提示数组hint
创建游戏状态GameState
调用总打印函数
根据提示数组计算需要的间距
调用背景打印函数（格子+白色空格）
打印行列提示
调用游戏函数

## 4. 调试过程碰到的问题

1. 无论什么情况下提交都显示正确
   最后调试了半天发现是两个函数句子写反了，未生成就直接输入导致全是0，核对下来也都确实正确
2. 背景经常变成各种颜色
   每次进行输出前都先提前设置一次颜色（缺省也要设置）。

## 5. 心得体会

一开始没考虑到后面的四个功能在打印上具体如何实现，只想到了逻辑相同，所以没有引入scale（打印背景的放大倍数），导致后续改变打印格式时需要重新计算并调试所有参数，也不得已引入了逻辑完全相同只是参数处理非常不同的print2函数。应该更全面地思考一个完整、多功能的工具函数需要具备哪些功能和变通的地方。

在运算时最好不要写纯数字而是定义一个const变量并加注释，否则第二天再读会完全忘记为什么要这么算。

## 6. 附件：源程序

```
#include<iostream>
#include<conio.h>
#include <iomanip>
#include "cmd_console_tools.h"
#include "pullze.h"

#define MAX_HINTS 8
using namespace std;




//伪图形界面的相关函数


struct puzzle {
    int size;
    int data[15][15]; // 按最大尺寸分配
};

struct Hints {
    int row_hints[15][MAX_HINTS] = { 0 }; // 行提
示 [行号][提示序列]
    int col_hints[15][MAX_HINTS] = { 0 }; // 列提
示 [列号][提示序列]
    int row_counts[15] = { 0 }; // 每行的提示数
    int col_counts[15] = { 0 }; // 每列的提示数
};
//先填充内部数组
puzzle input(int* SIZE)
{
    puzzle p;

    while (1) {

        cout << "请输入区域大小(5/10/15) : ";
        cin >> *SIZE;
        if (*SIZE == 10 || *SIZE == 15 || *SIZE
== 5)
            break;
    }
    //输入完毕
    cout << endl;
    cout << "初始数组：" << endl;

    p.size = *SIZE;


    for (int i = 0; i < p.size; i++) {
        for (int j = 0; j < p.size; j++) {
            p.data[i][j] = rand() % 2;
        }
    }
    return p;

}



//根据内部数组填充提示数组
void input_hint(const puzzle& p, Hints& h)
{
    for (int i = 0; i < p.size; i++) {
        int count = 0;
        int hint_idx = 0;
        for (int j = 0; j < p.size; j++) {
            if (p.data[i][j] == 1) {
                count++;
            }
            else if (count > 0) {
                h.row_hints[i][hint_idx++] =
count;
                count = 0;
            }
        }
        if (count > 0) {
            h.row_hints[i][hint_idx++] = count;
        }
        h.row_counts[i] = hint_idx;
    }

    //每行
    for (int j = 0; j < p.size; j++) {
        int count = 0;
        int hint_idx = 0;
        for (int i = 0; i < p.size; i++) {
            if (p.data[i][j] == 1) {
                count++;
            }
            else if (count > 0) {
                h.col_hints[j][hint_idx++] =
count;
                count = 0;
            }
        }
        if (count > 0) {
            h.col_hints[j][hint_idx++] = count;
        }
        h.col_counts[j] = hint_idx;
    }

    //每列



}



//获取行提示的最大数量
int get_max_row_hints(const Hints& h, int size) {
    int max_hints = 0;
    for (int i = 0; i < size; i++) {
        if (h.row_counts[i] > max_hints) {
            max_hints = h.row_counts[i];
        }
    }
    return max_hints;
}

// 获取列提示的最大数量
int get_max_col_hints(const Hints& h, int size) {
    int max_hints = 0;
    for (int j = 0; j < size; j++) {
        if (h.col_counts[j] > max_hints) {
            max_hints = h.col_counts[j];
        }
    }
    return max_hints;
}



//内部数组处理完毕

//输出abcd行，换行
```

```
void print_row_abcd(int size, int max_row, int
max_col,bool printhint)
{
    //设置字体大小

    cct_setfontsize( "仿宋",32);







    cct_setcolor();
    cct_gotoxy(max_row, max_col+1);
    cout << "    ";
    if (printhint) {
        cct_setcolor(COLOR_HWHITE, COLOR_BLACK);
        cct_showstr(max_row + 2, max_col + 1, "║
", COLOR_HWHITE, COLOR_BLACK);
    }
    for (int i = 0;i < size;i++) {
        cout << ' '<< char('a' + i) ;


    }
    cout << endl;
}



//根据尺寸，缓冲区，再打印白色背景
void print_bg(int size, bool print, int max_row,
int max_col)
{
    // 计算需要的最大间距（根据提示数字的最大位
数）


    cct_setconsoleborder(max_row + 4 * size, 80);

    cout << endl;


    // 打印行提示（如果需要）
    if (print) {

        cct_showstr(0, 2 + max_col, "╔",
COLOR_HWHITE, COLOR_BLACK); //2是字母+框的大小
        cct_showstr(1, 2 + max_col, "═",
COLOR_HWHITE, COLOR_BLACK, max_row - 1);


        //第一行

        for (int i = 0;i < size;i++) {
            cct_showstr(0, i + 1 + 2 + max_col, "
║", COLOR_HWHITE, COLOR_BLACK);
```

```
            for (int j = 0;j < max_row;j++) {
                cct_showstr(j + 1, i + 1 + 2 +
max_col, " ", COLOR_HWHITE, COLOR_BLACK);

            }
            // cct_showstr(2 * size + 4 +
max_row, 1 + i, "‖", COLOR_HWHITE, COLOR_BLACK);
        }

        //中间

        cct_showstr(max_row - 2, 2 + max_col , "
╦", COLOR_HWHITE, COLOR_BLACK);
        cct_showstr(max_row - 2+1, 2 + max_col, "
═", COLOR_HWHITE, COLOR_BLACK);
        for (int i = 0;i < size ;i++) {
            cct_showstr(max_row - 2, 1+2 +
max_col + i, "‖", COLOR_HWHITE, COLOR_BLACK);
        }
       // cct_showstr(max_row - 2, 1 +
max_col+size+2, "╩", COLOR_HWHITE, COLOR_BLACK);



        // ABCD 和数字间的分隔线



        cct_showstr(0, max_col + size + 3, "╚",
COLOR_HWHITE, COLOR_BLACK);
        cct_showstr(1, max_col + size + 3, "═",
COLOR_HWHITE, COLOR_BLACK, max_row - 1);
        cct_showstr(max_row - 2, 1 + max_col +
size + 2, "╩", COLOR_HWHITE, COLOR_BLACK);
        cct_showstr(max_row - 2+1, 1 + max_col +
size + 2, "═", COLOR_HWHITE, COLOR_BLACK);
        //最后一行打印完毕


    }




    // 打印列提示（如果需要）
    if (print) {

        cct_showstr(2 + max_row, 0, "╔",
COLOR_HWHITE, COLOR_BLACK);
        cct_showstr(3 + max_row, 0, "═",
COLOR_HWHITE, COLOR_BLACK, size + 1);
        cct_showstr(2 * size + 4 + max_row, 0, "
╗", COLOR_HWHITE, COLOR_BLACK);

        //第一行

        for (int i = 0;i < max_col + 2;i++) {
            cct_showstr(2 + max_row, 1 + i, "‖",
COLOR_HWHITE, COLOR_BLACK);
            for (int j = 0;j < 2 * size + 2;j++)
{
                cct_showstr(j + 3 + max_row, 1 +
i, " ", COLOR_HWHITE, COLOR_BLACK);

            }
            cct_showstr(2 * size + 4 + max_row, 1
+ i, "‖", COLOR_HWHITE, COLOR_BLACK);
        }

        //中间


        cct_showstr(2 + max_row, max_col, "╠",
COLOR_HWHITE, COLOR_BLACK);
        cct_showstr(3 + max_row, max_col, "═",
COLOR_HWHITE, COLOR_BLACK, size + 1);
        cct_showstr(2 * size + 4 + max_row,
max_col, "╣", COLOR_HWHITE, COLOR_BLACK);
        //最后一行打印完毕




    }



    //铺完两侧背景，开始打印中间方格



    // 打印 abcd 行（调整起始位置）
    print_row_abcd(size, max_row, max_col,
print);




    if (print)
        cct_showstr(2 + max_row, 2 + max_col, "╬
", COLOR_HWHITE, COLOR_BLACK);
    else
        cct_showstr(2 + max_row, 2 + max_col, "╔
", COLOR_HWHITE, COLOR_BLACK);

    cct_showstr(3 + max_row, 2 + max_col, "═",
COLOR_HWHITE, COLOR_BLACK, size + 1);

    if(print)
        cct_showstr(2 * size + 4 + max_row, 2 +
max_col, "╣", COLOR_HWHITE, COLOR_BLACK);
    else
        cct_showstr(2 * size + 4 + max_row, 2 +
max_col, "╗", COLOR_HWHITE, COLOR_BLACK);
```

```
    //第一行打印完毕



    for (int i = 0;i < size;i++) {
        if (print)
        {
            cct_showch(0 + max_row, i + 3 +
max_col, char(i + 'A'), COLOR_HWHITE,
COLOR_BLACK);

cct_setcolor(COLOR_HWHITE,COLOR_BLACK);
            cout << ' ';
            cct_setcolor();
        }
        else
            cct_showch(0 + max_row, i + 3 +
max_col, char(i + 'A')); //前面的 ABCD
        cct_showstr(2 + max_row, 3 + i + max_col,
"‖", COLOR_HWHITE, COLOR_BLACK);
        for (int j = 0;j < 2 * size + 2;j++) {
            cct_showstr(j + 3 + max_row, 3 + i +
max_col, " ", COLOR_HWHITE, COLOR_BLACK);

        }
        cct_showstr(2 * size + 4 + max_row, 3 + i
+ max_col, "‖", COLOR_HWHITE, COLOR_BLACK);
    }




    cct_showstr(2 + max_row, size + 3 + max_col,
"╚", COLOR_HWHITE, COLOR_BLACK);
    cct_showstr(3 + max_row, size + 3 + max_col,
"═", COLOR_HWHITE, COLOR_BLACK, size + 1);
    cct_showstr(2 * size + 4 + max_row, size + 3
+ max_col, "╝", COLOR_HWHITE, COLOR_BLACK);
    //最后一行打印完毕


}

void print_square(int X, int Y,int COLOR) {
    const int width = 3; // 调整为更合适的尺寸

    X += 3;

    // 绘制上边框
    cct_showstr(X - 1 - 1, Y - 1, "╔", COLOR,
COLOR_BLACK);
    cct_showstr(X - 1, Y - 1, "═", COLOR,
COLOR_BLACK, 2 * width - 4);
    cct_showstr(X + 2 * width - 2 - 2, Y - 1, "╗
", COLOR, COLOR_BLACK);

    // 绘制左右边框
    for (int i = 0; i < width - 2; i++) {
        cct_showstr(X - 1 - 1, Y + i, "‖",
COLOR, COLOR_BLACK);
        cct_showstr(X + 2 * width - 2 - 2, Y + i,
"‖", COLOR, COLOR_BLACK);
    }

    // 绘制下边框
    cct_showstr(X - 1 - 1, Y + 2 - 1, "╚",
COLOR, COLOR_BLACK);
    cct_showstr(X - 1, Y + 2 - 1, "═", COLOR,
COLOR_BLACK, 2 * width - 4);
    cct_showstr(X + 2 * width - 2 - 2, Y + 2 - 1,
"╝", COLOR, COLOR_BLACK);

    // 在中心绘制圆
    cct_showstr(X + (width - 2) / 2, Y + (width -
2) / 2, "○", COLOR, COLOR_BLACK);

    cct_setcolor(); // 恢复原始颜色
}

void print_false_square(int X, int Y,int COLOR) {
    const int width = 3; // 调整为更合适的尺寸

    X += 3;

    // 绘制上边框
    cct_showstr(X - 1 - 1, Y - 1, "╔", COLOR,
COLOR_BLACK);
    cct_showstr(X - 1, Y - 1, "═", COLOR,
COLOR_BLACK, 2 * width - 4);
    cct_showstr(X + 2 * width - 2 - 2, Y - 1, "╗
", COLOR, COLOR_BLACK);

    // 绘制左右边框
    for (int i = 0; i < width - 2; i++) {
        cct_showstr(X - 1 - 1, Y + i, "‖",
COLOR, COLOR_BLACK);
        cct_showstr(X + 2 * width - 2 - 2, Y + i,
"‖", COLOR, COLOR_BLACK);
    }

    // 绘制下边框
    cct_showstr(X - 1 - 1, Y + 2 - 1, "╚",
COLOR, COLOR_BLACK);
    cct_showstr(X - 1, Y + 2 - 1, "═", COLOR,
COLOR_BLACK, 2 * width - 4);
    cct_showstr(X + 2 * width - 2 - 2, Y + 2 - 1,
"╝", COLOR, COLOR_BLACK);

    // 在中心绘制圆
    cct_showstr(X + (width - 2) / 2, Y + (width -
2) / 2, "×", COLOR, COLOR_BLACK);

    cct_setcolor(); // 恢复原始颜色
}
```

```
void print_restore_square(int X, int Y) {
    const int width = 3; // 调整为更合适的尺寸

    X += 3;

    // 绘制上边框
    cct_showstr(X - 1 - 1, Y - 1, "  ",
COLOR_HWHITE, COLOR_BLACK);
    cct_showstr(X - 1, Y - 1, "  ", COLOR_HWHITE,
COLOR_BLACK, 2 * width - 4);
    cct_showstr(X + 2 * width - 2 - 2, Y - 1, "
", COLOR_HWHITE, COLOR_BLACK);

    // 绘制左右边框
    for (int i = 0; i < width - 2; i++) {
        cct_showstr(X - 1 - 1, Y + i, "  ",
COLOR_HWHITE, COLOR_BLACK);
        cct_showstr(X + 2 * width - 2 - 2, Y + i,
"  ", COLOR_HWHITE, COLOR_BLACK);
    }

    // 绘制下边框
    cct_showstr(X - 1 - 1, Y + 2 - 1, "  ",
COLOR_HWHITE, COLOR_BLACK);
    cct_showstr(X - 1, Y + 2 - 1, "  ",
COLOR_HWHITE, COLOR_BLACK, 2 * width - 4);
    cct_showstr(X + 2 * width - 2 - 2, Y + 2 - 1,
"  ", COLOR_HWHITE, COLOR_BLACK);

    // 在中心绘制圆
    cct_showstr(X + (width - 2) / 2, Y + (width -
2) / 2, "  ", COLOR_HWHITE, COLOR_BLACK);

    cct_setcolor(); // 恢复原始颜色
}

void print_puzzlewithhints(bool play,const
puzzle& p, const Hints& h)
{
    int size = p.size;
    int max_row_hints = get_max_row_hints(h,
size);
    int max_col_hints = get_max_col_hints(h,
size);

    // 计算需要的间距
    int row_hint_space = max_row_hints * 2 + 6;
// 每个提示数字占 2 字符宽
    int col_hint_space = max_col_hints + 1;

    print_bg(p.size, true, row_hint_space,
col_hint_space);

    // 打印行提示（左侧）
    for (int i = 0; i < size; i++) {
        for (int k = 0; k < h.row_counts[i]; k++)
{
            cct_showint(row_hint_space -
(h.row_counts[i] - k) * 2-2,
                3 + col_hint_space + i,
                h.row_hints[i][k],
COLOR_HWHITE, COLOR_BLACK);

        }
    }

    // 打印列提示（上方）
    for (int j = 0; j < size; j++) {
        for (int k = 0; k < h.col_counts[j]; k++)
{
            cct_showch(4 + row_hint_space + j *
2,
                col_hint_space - h.col_counts[j]
+ k,
                ' ', COLOR_HWHITE, COLOR_BLACK);
            cct_showint(4 + row_hint_space + j *
2+1,
                col_hint_space - h.col_counts[j]
+ k,
                h.col_hints[j][k], COLOR_HWHITE,
COLOR_BLACK);
        }
    }

    if(!play)
    {
        // 打印数据内容
        for (int i = 0; i < size; i++) {
            for (int j = 0; j < size; j++) {
                if (p.data[i][j]) {
                    int x = 4 + row_hint_space +
j * 2;
                    int y = 3 + col_hint_space +
i;
                    print_circle(x, y);
                }
            }
        }
    }
}


//游戏函数

struct GameState {
    puzzle puzzle;
    Hints hints;
    int markState[15][15] = { 0 }; // 0=未标记,
1=圆圈, 2=叉号
    bool cheatMode = false;
};

// 检查完成函数
bool check_complete(const GameState& game) {
    for (int i = 0; i < game.puzzle.size; i++) {
        for (int j = 0; j < game.puzzle.size;
j++) {
            if (game.puzzle.data[i][j] == 1 &&
game.markState[i][j] != 1) {
                return false;
            }
        }
    }
    return true;
}


//由数据计算并输出坐标
void getlocation(int x, int y, const int max_row,
const int max_col, const int size, bool play, int
scale)
{
    cct_setcolor();

    // 计算格子坐标
    int row = (x - max_row * 2 - 10) / (2 *
scale);
    int col = (y - max_col - 4) / scale;

    // 默认认为在有效区域
    bool is_valid = (row >= 0 && row < size &&
col >= 0 && col < size);

    // 只有当 scale>1 时才检查中心区域
    if (scale > 1 && is_valid) {
        // 计算鼠标在格子内的相对位置
        int relative_x = (x - max_row * 2 - 10) %
(2 * scale);
        int relative_y = (y - max_col - 4) %
scale;

        // 定义 3×3 中心区域范围
        int center_size = 3;  // 3×3 的中心区域
        int x_center_start = scale - center_size
/ 2 - 1;
        int x_center_end = scale + center_size /
2 + 1;
        int y_center_start = scale / 2 -
center_size / 2 - 1;
        int y_center_end = scale / 2 +
center_size / 2 + 1;

        // 检查是否在 3×3 中心区域内
        is_valid = (relative_x >= x_center_start
&& relative_x < x_center_end) && (relative_y >=
y_center_start && relative_y < y_center_end);
    }

    cct_gotoxy(10, max_col + size * scale + 5);
    if (is_valid) {
        cout << "[当前光标]" << " " << char('A' +
col) << "行" << " " << char('a' + row) << "列";
    }
    else {
        cout << "[当前光标]" << "位置非法";
    }
}




void getxy_puzzle(bool play, const int max_row,
const int max_col, int size, GameState& game, int
scale = 1) {
    cct_enable_mouse();
    cct_setcolor();

    if (play) {
        cct_gotoxy(10, (scale == 1 ? max_col +
size : max_col +scale * ( size)) + 7);
        cout << "左键选○，右键选×，Y/y 提交，Z/z
作弊，Q/q 结束";
    }

    while (1) {
        int x, y, mouseAction, keycode1,
keycode2;
        int event =
cct_read_keyboard_and_mouse(x, y, mouseAction,
keycode1, keycode2);

        // 鼠标移动事件-显示当前位置
        if (event == CCT_MOUSE_EVENT &&
mouseAction == MOUSE_ONLY_MOVED) {
            getlocation(x, y, max_row, max_col,
size, play, scale);
        }

        // 键盘事件处理
        if (event == CCT_KEYBOARD_EVENT) {
            cct_setcolor();
            if (keycode1 == 'Y' || keycode1 ==
'y') { // 检查完成
                if (play) {
                    bool correct = true;
                    for (int i = 0; i < size &&
correct; i++) {
                        for (int j = 0; j < size;
j++) {
                            if
((game.puzzle.data[i][j] == 1 &&
game.markState[i][j] != 1) ||

(game.puzzle.data[i][j] == 0 &&
game.markState[i][j] == 1)) {
                                correct = false;
                                break;
                    }
                }
            }

            cct_gotoxy(10, (scale == 1 ?
max_col + size : scale * (max_col + size)) + 6);
            if (correct) {
                cout << "拼图完成!
";
            }
            else {
                cout << "提交错误，可用作
弊模式查看          ";
            }
        }
    }
    else if (play && (keycode1 == 'Z' ||
keycode1 == 'z')) { // 作弊模式切换
        game.cheatMode = !game.cheatMode;
        cct_gotoxy(10, (scale == 1 ?
max_col + size : scale * (max_col + size)) + 8);
        cout << "作弊模式 " <<
(game.cheatMode ? "开启" : "关闭") << "
";

        // 计算行和列提示空间
        int row_hint_space =
get_max_row_hints(game.hints, size) * 2 + 6;
        int col_hint_space =
get_max_col_hints(game.hints, size) + 1;


        for (int i = 0; i < size; i++) {
            for (int j = 0; j < size;
j++) {
                if (scale == 1) {
                    int x = 4 +
row_hint_space + j * 2;
                    int y = 3 +
col_hint_space + i;

                    if (game.cheatMode) {
                        if
(game.puzzle.data[i][j]) {

cct_showstr(x, y, "〇", COLOR_WHITE,
COLOR_BLACK);
                        }
                        // 覆盖标记
                        if
(game.markState[i][j] == 1) {
                            int color =
(game.puzzle.data[i][j] == 1) ? COLOR_HGREEN :
COLOR_RED;

cct_showstr(x, y, "〇", color, COLOR_BLACK);
                        }
                        else if
(game.markState[i][j] == 2) {
                            int color =
(game.puzzle.data[i][j] == 0) ? COLOR_HGREEN :
COLOR_RED;

cct_showstr(x, y, "×", color, COLOR_BLACK);
                        }
                    }
                    else {
                        if
(game.markState[i][j] == 1) {

print_circle(x, y);
                        }
                        else if
(game.markState[i][j] == 2) {

print_false(x, y);
                        }
                        else {

print_restore(x, y);
                        }
                    }
                }
                else { // scale > 1
                    int x = max_row * 2 +
11 + 2 * scale * j;
                    int y = max_col + 6 +
scale * i;

                    if (game.cheatMode) {
                        if
(game.puzzle.data[i][j]) {

print_square(x, y, COLOR_WHITE);
                        }
                        if
(game.markState[i][j] == 1) {
                            int color =
(game.puzzle.data[i][j] == 1) ? COLOR_HGREEN :
COLOR_RED;

print_square(x, y, color);
                        }
                        else if
(game.markState[i][j] == 2) {
                            int color =
(game.puzzle.data[i][j] == 0) ? COLOR_HGREEN :
COLOR_RED;

print_false_square(x, y, color);
                        }
                    }
                    else {
                        if
```

```
(game.markState[i][j] == 1) {

print_square(x, y, COLOR_HBLUE);
                                }
                                else if
(game.markState[i][j] == 2) {

print_false_square(x, y, COLOR_RED);
                                }
                                else {

print_restore_square(x, y);
                                }
                            }
                        }
                    }
                }
            }
            else if (keycode1 == 'Q' || keycode1
== 'q') { // 退出
                break;
            }
        }

        // 鼠标点击处理
        if (play && event == CCT_MOUSE_EVENT &&
            (mouseAction ==
MOUSE_LEFT_BUTTON_CLICK || mouseAction ==
MOUSE_RIGHT_BUTTON_CLICK)) {

            int row = (x - max_row * 2 - 10) / (2
* scale);
            int col = (y - max_col - 4) / scale;

            if (row >= 0 && row < size && col >=
0 && col < size) {
                if (scale == 1) {
                    if (x % 2 != 0) x--; // 调整
奇数坐标

                    if (mouseAction ==
MOUSE_LEFT_BUTTON_CLICK) {
                        game.markState[col][row]
= (game.markState[col][row] == 1) ? 0 : 1;
                        if
(game.markState[col][row] == 1)
                            print_circle(x, y);
                        else
                            print_restore(x, y);
                    }
                    else if (mouseAction ==
MOUSE_RIGHT_BUTTON_CLICK) {
                        game.markState[col][row]
= (game.markState[col][row] == 2) ? 0 : 2;
                        if
(game.markState[col][row] == 2)
                            print_false(x, y);
                        else
                            print_restore(x, y);
                    }
                }
                else { // scale > 1
                    int display_x = max_row * 2 +
11 + 2 * scale * row;
                    int display_y = max_col + 6 +
scale * col;

                    if (mouseAction ==
MOUSE_LEFT_BUTTON_CLICK) {
                        game.markState[col][row]
= (game.markState[col][row] == 1) ? 0 : 1;
                        if
(game.markState[col][row] == 1)

print_square(display_x, display_y, COLOR_HBLUE);
                        else

print_restore_square(display_x, display_y);
                    }
                    else if (mouseAction ==
MOUSE_RIGHT_BUTTON_CLICK) {
                        game.markState[col][row]
= (game.markState[col][row] == 2) ? 0 : 2;
                        if
(game.markState[col][row] == 2)

print_false_square(display_x, display_y,
COLOR_RED);
                        else

print_restore_square(display_x, display_y);
                    }
                }
            }
        }
    }
    cct_disable_mouse();
}

void T_J() {
    cct_cls();
    srand(unsigned int(time(0)));

    int size;
    puzzle p = input(&size);  // 生成谜题
    Hints h;
    input_hint(p, h);         // 生成提示

    int max_row_hints = get_max_row_hints(h,
size);
    int max_col_hints = get_max_col_hints(h,
size);

    // 创建游戏状态并初始化
    GameState game;
    game.puzzle = p;
    game.hints = h;


    print_puzzlewithhints_2(0, p, h);

    int scale = 5;
    getxy_puzzle(0, max_row_hints, max_col_hints,
size,game,scale);


    my_exit();
}






void T_K() {
    cct_cls();
    srand(unsigned int(time(0)));

    int size;
    puzzle p = input(&size);  // 生成谜题
    Hints h;
    input_hint(p, h);         // 生成提示

    int max_row_hints = get_max_row_hints(h,
size);
    int max_col_hints = get_max_col_hints(h,
size);

    // 创建游戏状态并初始化
    GameState game;
    game.puzzle = p;
    game.hints = h;


    print_puzzlewithhints_2(1, p, h);

    int scale = 5;
    getxy_puzzle(1, max_row_hints, max_col_hints,
size, game, scale);


    my_exit();
}
```