

数织游戏的设计与实现

班级：10071706

学号：2452654

姓名：郭炫君

完成日期：2025. 6. 13

1. 题目

1.1. 基本要求

本次作业要求基于控制台环境下，逐步实现完整的数织游戏系统，作业共分为 A 到 K 共 11 个小题，其中 A~C 是文字版本，D~G 是伪图形无线框版本，H~K 是伪图形有线框版本，主要内容如下：

功能 A：初始化矩阵并用文字方式打印矩阵，不带行列提示，带行标列标，分别从 A 和 a 开始，每 5 行/5 列有一个分割线。

功能 B：再子题目 A 的基础上加上行列提示，要求行提示栏的宽度和列提示栏的高度能根据实际情况动态调整，行提示右对齐，列提示下对齐。

功能 C：文字打印的游戏版本，初始打印只有行列提示的空矩阵，根据键盘输入的坐标（坐标严格区分大小写）在矩阵内对应位置打印 O，如果此处已经标记，就取消标记。按 z 可以开启作弊模式，已经标记的位置高亮，标记错了打印 X，未标记但存在球的位置不高亮，打印 O。按 y 可以提交答案，若正确则游戏结束。按 x 结束返回主菜单。

功能 D：相当于子题目 A 的伪图形化实现，但是矩阵内部无分割线，边框要求是中文制表符。

功能 E：相当于功能 B 的伪图形化实现，同样内部没有分割线，尽量和功能 D 共用参数。

功能 F：在伪图形界面下画出初始状态，然后读取鼠标和键盘的操作，对于鼠标的移动而言，在显示球的区域外，一律为非法位置，不显示坐标，在合法位置才显示坐标，按下鼠标左键或右键需要反馈读到左/右键；对键盘而言，读到键盘操作输出对应键码，读到箭头要输出读到了上/下/左/右箭头，读到回车键结束当前功能。

功能 G：在伪图形界面下的游戏版本，打印图形逻辑稍有不同：在非作弊模式下共有三种图形，空白、蓝圈、红叉，左键为标记存在球，显示为蓝圈；右键为标记不存在球，显示为红叉；如果此处已经被标记有球，按下左键回到空白，按下右键直接变为红叉；标记为没有球的同理。在作弊模式下有 6 种图形，蓝圈：标记有球实际有球，红圈：标记有球实际无球，蓝叉：标记无球实际有球，红叉：标记无球实际无球，空白：无球且未标记，白圈：有球但未标记。按 q 返回主菜单。

功能 H~K：功能 D~G 加上线框版本，其他全部都一样，不仅每个格子之间有相框分割，每个球打印的时候也有线框包围，一个中文制表符占据两个字节。

1.2. 其他要求

打印线框的时候不能打表，使用参数判定加上循环的方式打印，尽量多个功能共用一个函数，减少冗余代码，伪图形打印不要使用作业给的之外的函数。

2. 整体设计思路

本程序采用“主菜单控制 + 模块化实现 + 参数统一切换”的总体架构。

主菜单位于 `pullze_tools.cpp`, 通过 `basis(char choose)` 函数根据用户选择调用不同的功能。整体上分为一下几个模块:

2.1. 参数设置

`matrix` 数组存储数织游戏球的位置信息, `mark` 数组存储游戏版本中是否有标记的信息, `picture` 数组用于存储游戏版本有没有画图的信息, `size` 存储矩阵的尺寸大小, `row_hints` 和 `col_hints` 数组分别存储行列提示的信息, `input` 数组存储游戏版本中坐标输入以及命令执行的内容, `cheat` 用于判断当前是否是作弊模式, 还有头文件里的全局常量 `max_size` 是矩阵的最大尺寸, `start_X` 和 `start_Y` 是打印伪图形矩阵的起始参考位置。

2.2. 数据模块

`get_size()`, `generate_matrix()`, `generate_hints()` 三个函数用于获取基本数据: 矩阵的尺寸大小、矩阵内部的数据数组、行列提示数组。

2.3. 画图模块

`set_size()` 用于调整控制台的字体大小和窗口、缓冲区宽度高度。

`draw_matrix()` 负责画没有行列提示的矩阵。

`draw_with_hints_noborder()` 和 `draw_with_hints_border()` 分别用于画出功能 E~G 和功能 I~K 中带行列提示栏的空白矩阵。

`draw_inside_picture()` 用于打印内部图形, 可以调用不同的参数打印不同的图形, 有无边框都能够使用。

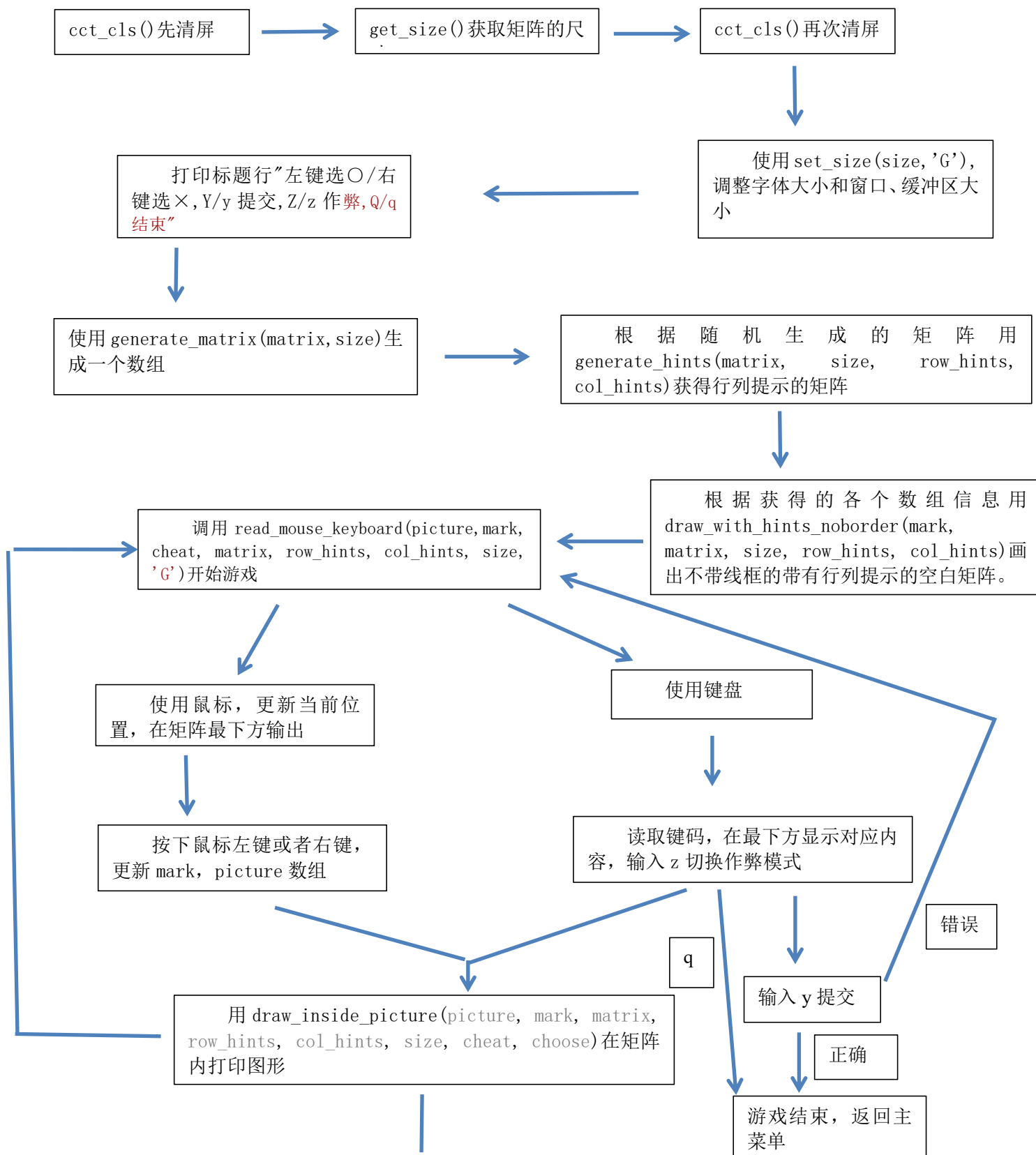
`draw_frame()` 用于打印矩阵时画一些很长的线条, 防止画空白矩阵的函数内部看起来过于混乱, `kind1~4` 是无线框矩阵中的线条, `kind5~8` 是有线框矩阵中的线条。

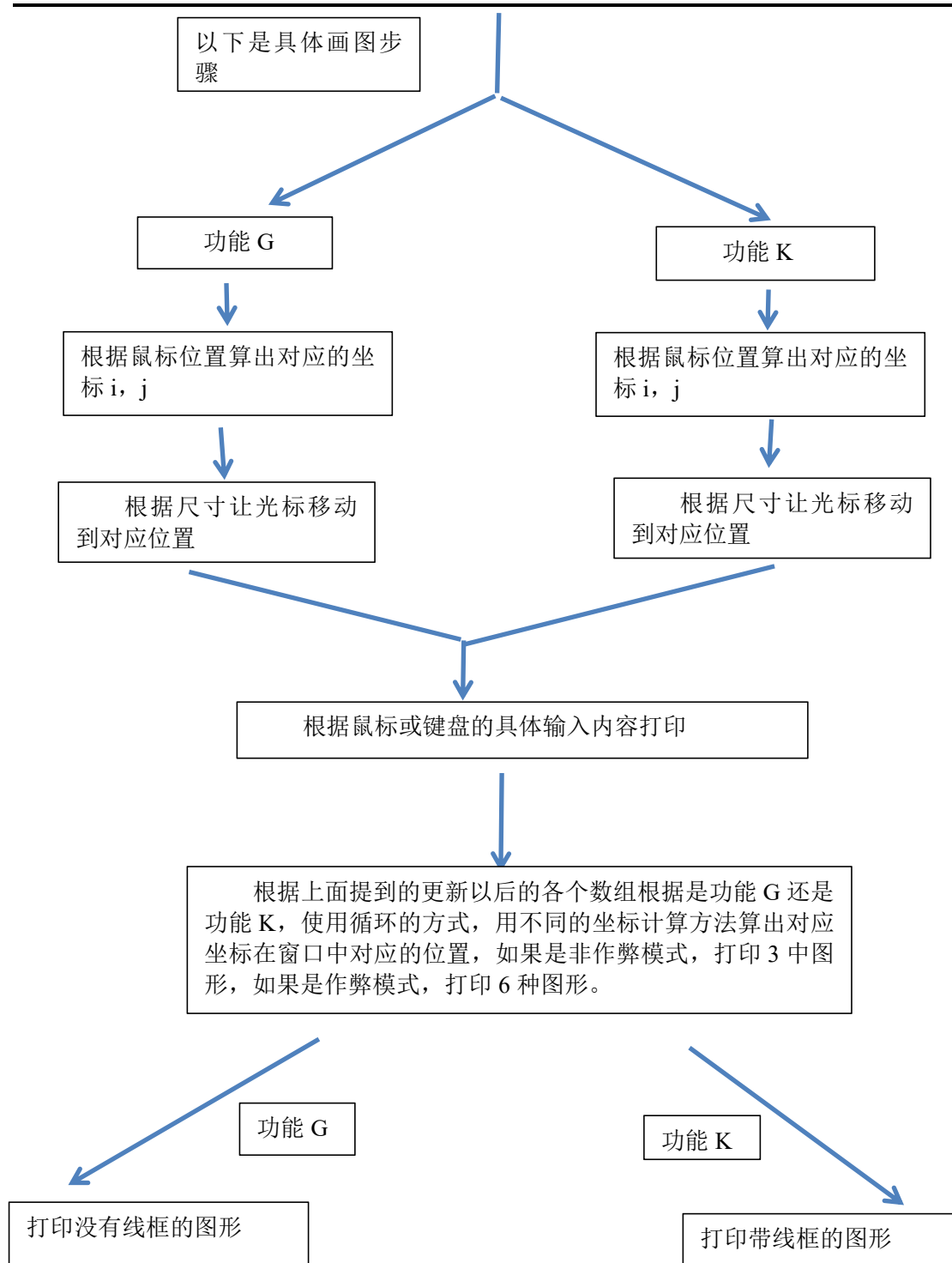
`Draw_picture_border()` 专门用于打印带边框的图形, 用于功能 H~K。

2.4. 交互模块

交互功能也就是一切鼠标键盘的操作统一在函数 `read_mouse_keyboard()` 内实现, 通过调用不同的参数达到对应的效果, 比如要在矩阵最下方打印鼠标或键盘当前的活动, 位置不同, 就需要让光标到达不同的位置

3. 主要功能的实现





4. 调试过程碰到的问题

4.1. 列提示需要下对齐，行提示需要右对齐问题

如果直接按照获得的行列提示数组的顺序打印下来，无法实现对齐的要求，因此引入了最大宽度 `row_hint_width` 和最大高度 `col_hint_height`，还加入了 `count` 变量，用于计算每一组行列提示不是 0 的个数，如果循环中 `i < count` 就不打印。

4.2. 在游戏版本中打了红叉的位置再按右键不会消失

这是由于没有标记为无球和标记了无球时在 `mark` 数组里都是 `false`，这样就无法正常区分不同的情况，因此需要再引入一个数组，也就是 `picture`，用于记录这个位置有没有被标记过，这样各种不同的情况都能够区分开了。

4.3. 如何让有边框和无边框两种情况共用一个函数

有边框和无边框每一格的长度不同，在无边框的情况下可以连着打印，而在有边框的情况下中间需要打印线框，如果把打印内部图形和打印矩阵的功能都放在一起，会变得很复杂。因此我用一个函数专门打印空白的矩阵，一个函数负责打印内部的图形，具体在窗口的哪个位置就能够根据选择的功能进行计算，也不会很复杂。

4.4. 如何更方便地打印伪图形矩阵

比如在打印横线时，如果使用 `cct_showstr()`，在打印伪图形矩阵需要一直计算各个位置的坐标，但是如果不使用，光标只会移动一格而打印的制表符占 2 格，这时可以多打印一个横线，把原来的位置覆盖掉，打印效果不变，可以连着往下打印，不用每打印一个制表符都需要计算位置。

4.5. 打印带边框的图形

在完成功能 H^K 时要求打印的图形带边框，如果全部放在一起打印会很麻烦，要计算很多坐标位置因此我把打印带边框图形的功能单拎出来写一个函数 `draw_picture_border()`，有效地减少了打印矩阵内部图形函数的复杂程度。

5. 心得体会

5.1. 本次作业心得体会与经验教训

5.1.1

渐进式开发的重要性：从功能 A 到功能 K，每一步的实现都为后续功能提供基础，尤其是矩阵生成与提示数组的构建逻辑，是整个系统的基础。通过从字符输出过渡到伪图形显示，我逐步建立了层层封装的开发模式。

5.1.2

图形调试能力的提升：项目中不断出现的坐标偏移、图形覆盖等问题，让我更加熟练地掌握了字符图形绘制技巧与调试方法，也培养了我调试时分步输出与可视验证的能力。

5.1.3

模块化设计的必要性：由于题目结构复杂，若不采用统一的结构和参数，极易造成混乱。将功能封装为函数、划分清晰的模块，让我在实现后续功能时更加轻松。

5.1.4

用户交互逻辑的构建：功能 G/K 中需要支持鼠标与键盘的混合输入，要求对事件驱动逻辑有清晰理解。通过设计状态刷新函数 `draw_inside_picture()`，我实现了统一的交互响应机制，能够在不重复函数内容的情况下实现多个交互功能共用多个函数。

5.1.5

时间管理与任务分配的教训：初期对题目复杂度估计不足，导致中后期有部分返工和重复劳动。之后通过合理分阶段、每日功能目标规划，逐渐回到正轨，这让我认识到提前设计规划和分阶段测试的重要性，如果全部一起完成会出现很多问题，进度会很慢。

5.2 函数复用与模块整合的反思

1. 提示生成函数的复用：`generate_matrix()` 与 `generate_hints()` 被几乎所有子题使用，形成稳定模块。

2. 绘制函数按结构分层复用：字符版与图形版函数对应一致，便于通过 `choose` 参数控制版本切换。

3. 将打印矩阵和在矩阵内部打印图形 2 个功能分开来写，能够让这些函数在其他功能中被服用

4. 每一个函数最好只实现一个功能，提高独立性，这样函数的复用率才会更高。

6. 源代码

```

int basis(char choose)
{
    bool matrix[max_size][max_size]; // 正确数组
    bool mark[max_size][max_size] = {false}; // 游戏
    版数组
    bool picture[max_size][max_size] = { false }; //
    判断这一格是否有图形
    int size; // 数组大小
    int row_hints[max_size][max_size / 2 + 1];
    int col_hints[max_size][max_size / 2 + 1];
    char input[10]; // 输入判断
    bool cheat = false; // 判断是否需要使用作弊模式
    switch (choose) {
        case 'G':
            cct_cls();
            size = get_size();
            cct_cls();
            set_size('G', size);
            cout << "左键选O/右键选X,Y/y 提交,Z/z 作
            弊,Q/q 结束";
            generate_matrix(matrix, size);
            generate_hints(matrix, size, row_hints,
            col_hints);
            draw_with_hints_noborder(mark, matrix,
            size, row_hints, col_hints);
            read_mouse_keyboard(picture, mark, cheat,
            matrix, row_hints, col_hints, size, 'G');
            cout << endl;
            finish();
            break;

        case 'K':
            draw_with_hints_border(mark, matrix, size,
            row_hints, col_hints);
            } // 就这一句和 G 功能不一样

// 输入矩阵的尺寸, 含输入错误处理
int get_size()
{
    int size;
    while (1) {
        cout << "请输入区域大小(5/10/15) : ";
        cin >> size;
        if (cin.fail()) {
            cin.clear();
            cin.ignore(1024, '\n');
            continue;
        }
        if (size == 5 || size == 10 || size == 15) {
            cin.ignore(1024, '\n');
            break;
        }
    }
}

return size;
}

Void draw_with_hints_noborder(bool
mark[max_size][max_size], const bool
matrix[max_size][max_size], int size,
const int row_hints[max_size][max_size / 2 + 1],
const int col_hints[max_size][max_size / 2 + 1])
{
    int col_hint_height = 0;
    for (int j = 0; j < size; ++j) {
        int h = 0;
        while (h < max_size / 2 + 1 &&
col_hints[j][h] != 0)
            ++h;
        if (h > col_hint_height)
            col_hint_height = h;
    }

    int row_hint_width = 0;
    for (int i = 0; i < size; ++i) {
        int count = 0;
        while (count < max_size / 2 + 1 &&
row_hints[i][count] != 0)
            ++count;
        if (count > row_hint_width)
            row_hint_width = count;
    }

    // 打印列提示栏
    draw_frame(1, start_X + (1 + row_hint_width) * 2,
start_Y, size, row_hint_width);

    for (int i = 0; i < col_hint_height; i++) {
        cct_showstr(start_X + (1 + row_hint_width)
* 2, start_Y + 1 + i, " | ", COLOR_WHITE, COLOR_BLACK);
        for (int j = 0; j < size; j++) {
            int count = 0;
            while (count < max_size / 2 + 1 &&
col_hints[j][count] != 0)
                ++count;
            if (i >= col_hint_height - count) {
                cout << " " << col_hints[j][i -
(col_hint_height - count)];
            }
            else {
                cout << " ";
            }
        }
        cct_showstr(start_X + 2 + 2 * size + (1 +
row_hint_width) * 2, start_Y + 1 + i, " | ", COLOR_WHITE,
COLOR_BLACK);
    }

    draw_frame(2, start_X + (1 + row_hint_width) * 2,

```

```

start_Y + col_hint_height + 1, size, row_hint_width);
    cct_showstr(start_X + (1 + row_hint_width) * 2,
start_Y + col_hint_height + 2, " | ", COLOR_WHITE,
COLOR_BLACK);
    for (int i = 0; i < size; i++) {
        cout << " " << char('a' + i);
    }
    cct_showstr(start_X + 2 + 2 * size + (1 +
row_hint_width) * 2, start_Y + col_hint_height + 2, "
| ", COLOR_WHITE, COLOR_BLACK);
    draw_frame(3, start_X - 4, start_Y +
col_hint_height + 3, size, row_hint_width);

//打印每一行：行提示 + 行标 + 空白矩阵
for (int i = 0; i < size; i++) {
    cct_showstr(start_X - 4, start_Y +
col_hint_height + 4 + i, " | ", COLOR_WHITE,
COLOR_BLACK);
    int hint_count = 0;
    while (hint_count < max_size / 2 + 1 &&
row_hints[i][hint_count] != 0)
        ++hint_count;
    for (int j = 0; j < row_hint_width -
hint_count; ++j)
        cout << " ";
    for (int j = 0; j < hint_count; ++j)
        cout << row_hints[i][j] << " ";
    cct_showstr(start_X - 2 + 2 * row_hint_width,
start_Y + col_hint_height + 4 + i, " | ", COLOR_WHITE,
COLOR_BLACK);
    cout << " " << char(i + 'A');
    cct_showstr(start_X + 2 + 2 * row_hint_width,
start_Y + col_hint_height + 4 + i, " | ", COLOR_WHITE,
COLOR_BLACK);

    for (int j = 0; j < size; j++) {
        cout << " ";
    }
    cct_showstr(start_X + 4 + 2 *
(row_hint_width + size), start_Y + col_hint_height + 4
+ i, " | ", COLOR_WHITE, COLOR_BLACK);
}
draw_frame(4, start_X - 4, start_Y +
col_hint_height + size + 4, size, row_hint_width);

cct_setcolor(COLOR_BLACK, COLOR_WHITE);
}

//打印伪图形版本含行列提示，含边框空白矩阵，部分仿照
数组版本
void draw_with_hints_border(bool
mark[max_size][max_size], const bool
matrix[max_size][max_size],
int size, const int row_hints[max_size][max_size
/ 2 + 1], const int col_hints[max_size][max_size / 2
+ 1])
{

```

```

// 计算列提示最大高度
int col_hint_height = 0;
// 计算最大行提示宽度
int row_hint_width = 0;

//打印列提示栏
draw_frame(5, start_X + (1 + row_hint_width) * 2,
start_Y, size, row_hint_width);

for (int i = 0; i < col_hint_height; i++) {
    cct_showstr(start_X + (1 + row_hint_width)
* 2, start_Y + 1 + i, " | ", COLOR_WHITE, COLOR_BLACK);
    for (int j = 0; j < size; j++) {
        int count = 0;
        while (count < max_size / 2 + 1 &&
col_hints[j][count] != 0)
            ++count;
        if (j < size - 1) {
            if (i >= col_hint_height - count)
{
                cout << " " <<
col_hints[j][i - (col_hint_height - count)] << "
";
            }
            else {
                cout << " ";
            }
        }
        else
            if (i >= col_hint_height - count)
{
                cout << " " <<
col_hints[j][i - (col_hint_height - count)] << "
";
            }
            else {
                cout << " ";
            }
        }
        cout << " | ";
    }
    draw_frame(6, start_X + (1 + row_hint_width) * 2,
start_Y + col_hint_height + 1, size, row_hint_width);

    cct_showstr(start_X + (1 + row_hint_width) * 2,
start_Y + col_hint_height + 2, " | ", COLOR_WHITE,
COLOR_BLACK);
    for (int i = 0; i < size; i++) {
        if (i < size - 1) {
            cout << " " << char('a' + i) << " ";
        }
        else {
            cout << " " << char('a' + i) << " ";
        }
    }
    cout << " | ";

    draw_frame(7, start_X - 4, start_Y +

```

```
col_hint_height + 3, size, row_hint_width);
cout << endl;

//打印每一行：行提示 + 行标 + 空白矩阵
for (int i = 0; i < size; i++) {
    int hint_count = 0;
    while (hint_count < max_size / 2 + 1 &&
row_hints[i][hint_count] != 0)
        ++hint_count;
    for (int k = 0; k < 3; k++) {
        if (k == 1) {
            cout << " | ";
            for (int j = 0; j < row_hint_width
- hint_count; ++j)
                cout << " ";
            for (int j = 0; j < hint_count; ++j)
                cout << row_hints[i][j] << "
";
            cout << " | " << " " << char(i + 'A')
<< " | ";
        }
        else {
            cout << " | ";
            for (int j = 0; j < row_hint_width;
j++) {
                cout << " ";
            }
            cout << " | ";
        }
        for (int j = 0; j < size; j++) {
            cout << " | ";
        }
        cout << " " << endl;
    }
    cout << " | ";
    for (int j = 0; j < row_hint_width; j++) {
        cout << " ";
    }
    cout << " | ";
    cout << " |—————";
    for (int n = 0; n < size - 1; n++) {
        cout << " |—————";
    }
    cout << " | ";
    cout << endl;
}

draw_frame(8, start_X-4, start_Y +
col_hint_height + size*4 + 3, size, row_hint_width);
cct_setcolor(COLOR_BLACK, COLOR_WHITE);
}

//不同功能下打印矩阵内部的不同图形
void draw_inside_picture(bool
picture[max_size][max_size], bool
mark[max_size][max_size], const bool
matrix[max_size][max_size],
const int row_hints[max_size][max_size / 2 + 1],
```

```
const int col_hints[max_size][max_size / 2 + 1], int
size, bool cheat, char choose)
{
    // 计算列提示最大高度
    int col_hint_height = 0;
    // 计算最大行提示宽度
    int row_hint_width = 0;

    for (int i = 0; i < size; ++i) {
        int count = 0;
        while (count < max_size / 2 + 1 &&
row_hints[i][count] != 0)
            ++count;
        if (count > row_hint_width)
            row_hint_width = count;
    }

    int start_x = start_X + 2*row_hint_width+3;
    int start_y = start_Y + col_hint_height+2;
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            if (choose == 'G') {
                if (!cheat) { //三种图形
                    if (mark[i][j]) {
                        cct_showstr(start_x + 1
+ j * 2, start_y + 2 + i, " O ", COLOR_HBLUE,
COLOR_BLACK);}
                    else if(!picture[i][j]) {
                        cct_showch(start_x + 1 +
j * 2, start_y + 2 + i, ' ', COLOR_WHITE, COLOR_WHITE,
2);}
                    else {
                        cct_showstr(start_x + 1
+ j * 2, start_y + 2 + i, " X ", COLOR_HRED,
COLOR_BLACK);}
                }
            }
            else { //六种图形
                if (matrix[i][j] &&
mark[i][j]) {
                    cct_showstr(start_x + 1
+ j * 2, start_y + 2 + i, " O ", COLOR_HBLUE,
COLOR_BLACK);}
                    else if (!matrix[i][j] &&
mark[i][j]) {
                        cct_showstr(start_x + 1 + j *
2, start_y + 2 + i, " O ", COLOR_HRED, COLOR_BLACK);}
                    else if (matrix[i][j]
&& !mark[i][j] && !picture[i][j]) {
                        cct_showstr(start_x + 1 + j *
2, start_y + 2 + i, " O ", COLOR_HWHITE, COLOR_BLACK);}
                    else if (matrix[i][j]
&& !mark[i][j] && picture[i][j]) {
                        cct_showstr(start_x + 1
+ j * 2, start_y + 2 + i, " X ", COLOR_HBLUE,
COLOR_BLACK);}
                    else if (!matrix[i][j]
&& !mark[i][j] && picture[i][j]) {
```

```

        cct_showstr(start_x + 1
+ j * 2, start_y + 2 + i, " × ", COLOR_HRED,
COLOR_BLACK);}
        else {
            cct_showch(start_x + 1 +
j * 2, start_y + 2 + i, ' ', COLOR_WHITE, COLOR_WHITE,
2);}
    }
}

else if (choose == 'K') { //和G的分类
一样, 坐标参数不一样
}
}
}
cct_setcolor(COLOR_BLACK, COLOR_WHITE);
}

```

//读取键盘和鼠标操作

```

void read_mouse_keyboard(bool
picture[max_size][max_size], bool
mark[max_size][max_size], bool cheat,
const bool matrix[max_size][max_size], const int
row_hints[max_size][max_size / 2 + 1], const int
col_hints[max_size][max_size / 2 + 1], int size, char
choose)
{
    // 计算列提示最大高度
    int col_hint_height = 0;
    // 计算最大行提示宽度
    int row_hint_width = 0;
    int start_x = start_X + 2 * row_hint_width + 4;
    int start_y = start_Y + col_hint_height + 4;
    int X = 0, Y = 0;
    int ret, maction;
    int keycode1, keycode2;
    int loop = 1;

    cct_enable_mouse(); //使用鼠标
    cct_setcursor(CURSОР_INVISIBLE); //关闭光标

    while (loop) {
        draw_inside_picture(picture, mark, matrix,
row_hints, col_hints, size, cheat, choose);
        ret = cct_read_keyboard_and_mouse(X, Y,
maction, keycode1, keycode2);
        if (choose == 'F' || choose == 'G')
            cct_gotoxy(start_x - 4,
start_y + size + 1); //无边框
        else
            cct_gotoxy(start_x - 4, start_y + 4 *
size + 1); //有边框
        for (int i = 0; i < 30; i++) {
            cout << " ";
        }
        //移动光标, 和上面判断方法一样
    }
}

```

```

if (ret == CCT_MOUSE_EVENT) {
    int i, j;
    bool valid = false;
    if (choose == 'F' || choose == 'G') {
        if (X >= start_x && X < start_x +
size * 2 && Y >= start_y && Y < start_y + size) {
            valid = true;
            if ((X - start_x) % 2 != 0) //
一个球占两格
                X -= 1;
            i = Y - start_y;
            j = (X - start_x) / 2;
        }
    }
    else if (choose == 'J' || choose == 'K') {
        if (X >= start_x && X < start_x +
size * 8 - 2 && (X - start_x) % 8 < 6 && Y >= start_y
&& Y < start_y + 4 * size && (Y - start_y) % 4 < 3) {
            valid = true;
            X -= (X - start_x) % 8;
            Y -= (Y - start_y) % 4;
            i = (Y - start_y) / 4;
            j = (X - start_x) / 8;
        }
    }
    if (valid) {
        switch (maction) {
            case MOUSE_ONLY_MOVED:
                cout << "[当前光标] ";
                break;
            case
MOUSE_LEFT_BUTTON_CLICK:
                mark[i][j]
= !mark[i][j];
                picture[i][j] =
mark[i][j];
                cout << "[读到左键] ";
                break;
            case
MOUSE_RIGHT_BUTTON_CLICK:
                if (mark[i][j]
|| !picture[i][j]) {
                    picture[i][j] =
true;
                    mark[i][j] = false;
                }
                else {
                    picture[i][j] =
false;
                }
                cout << "[读到右键] ";
                break;
            default:
                cout << "[当前光标] ";
                break;
        }
        cout << char('A' + i) << "行" <<

```

