同济大学

实验报告

彩球游戏

姓名：雷翔

学号：2053932

完成日期：2022.11

1. 题目

1.1 题目简介

题目是 Windows 版的 Color linez 游戏的简化版本。要求可以设置游戏的行列值，7 个小问题层层递进，从在数组里实现在最终图形化界面。题目设置的具体游戏规则如下：初始状态有 5 个球，每移动一次球会随机产生 3 个不同颜色的球，球的颜色会提前进行预告，当同一颜色的球在横向、纵向达到 5 个及以上时，可以消除，同时得到相应的分数，假设消除 n 个球，此次得到的分数为(n-1) * (n-2)。

1.2 题目分析

首先要实现一个菜单函数，让用户通过菜单进行选择，从而执行相应的选项。选项 1 比较简单，选项 2 难点在于如何找到从起点到终点的路径，这里我采用的是深度优先搜索 DFS+回溯的方法，选项 3 是一个完整的实现方案。选项 4、5 比较简单，主要是绘制伪图形界面，选项 6 难点在于阅读老师提供的两个代码文件，读懂获取鼠标坐标和鼠标点击等功能的实现方法。之前所采用的寻路算法移动过程比较慢，在选项 7 中，我实现了另一种寻路方法，广度优先搜索 BFS，可以找到最优路径。

1.3 题目具体要求

1.3.1 选项 1、2 要求

首先输入行列，选项 1 是在数组中随机生成 5 个球，选项 2 是在数组中随机生成 60%个球，然后打印数组。除此之外，选项 2 要求输入起始和目的坐标，并判断是否合理，然后输出移动路径。

1.3.2 选项 3 要求

首先输入行列，要求通过输入起始和目的坐标，实现完整的游戏过程，遇到 5 个及以上的球需要消除，并且输出分数。每次输入起始和目的坐标后，需要判断是否存在路径，如果存在路径，移动并且更新数组以及接下来出现的 3 个球，否则给予相关提示。

1.3.3 选项 4、5 要求

首先输入行列，选项 4 是以无分割线方式画出伪图形界面，选项 5 是以有分割线方式画出伪图形界面。

1.3.4 选项 6、7 要求

首先输入行列，选项 6 是生成 60%个球，选项 7 是生成 5 个球，通过鼠标来选择要移动的球，要求鼠标在移动过程实时显示位置，移动过程要显示完整的移动轨迹。选项 6 是一次移动，选项 7 是重复移动，直至游戏结束。除此之外，选项 7 还要记录每次移动后的得分情况。

2. 整体设计思路

以选项 7 为例，需要完成的核心任务有通过鼠标选择正确的起始和目的坐标 MouseAction、找到从起始到目的坐标的移动路径 BFS、画出完整的移动路径 DrawMove、计算本次移动得分 IsDelete 等。

针对整个程序，首先需要输入行列数，这里需要对错误输入进行处理，然后进行在数组中随机生成 5 个球，接着随机生成 3 个球，并记录这三个球在数组中的位置和颜色，页面显示题目要求的信息，

接着就是通过鼠标来选择正确的起始和目标坐标 MouseAction，这是一个非常重要的函数，函数参数我特意设置成 pre_line, pre_col, cur_line, cur_col，而不是 start_line, start_col, end_line, end_col，这样做的好处是在循环执行 MouseAction 时，能比较清楚的知道此时变量值的含义。具体来说，首先调用 cct_read_keyboard_and_mouse，获取当前鼠标的坐标，接着判断是否点击鼠标右键，如果是，直接退出；如果不是，判断是否点击鼠标左键，如果不是，直接退出，如果是，需要判断选中了球还是选中空白位置，分情况具体处理。

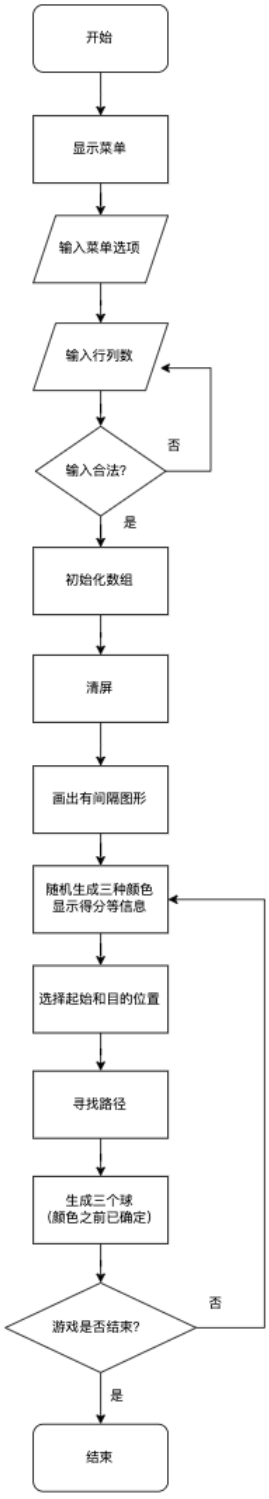
如果 pre_line, pre_col 是球的坐标，cur_line, cur_col 是空白的坐标，此时需要执行 DrawMove，DrawMove 会首先执行 BFS 来寻找路径，如果没有路径，会有提示信息；如果存在路径，会画出完整的移动路径，移动完成后会计算本次移动得分。

接着会将之前记录的三个球花在原图形上，因为之前保存了位置和颜色信息，因此其实并不需要覆盖原来的图形界面。

所有上述步骤完成之后就完成了一次移动，之后需要判断游戏是否结束，具体来说：如果下一次移动数组中球的数量达到 lines*cols，游戏失败；如果数组中球的数量等于 0，游戏成功，当然这种情况是几乎不可能发生的。

3. 主要功能的实现

3.1 功能实现流程图：



3.2 寻路函数 DFS 和 BFS

DFS 的算法思路：

1. 访问起点；
2. 判断是否是终点；
3. 如果是，直接返回；否则执行下列操作；
4. 将起点插入路径；
5. 顺时针依次递归访问未被访问过的周围的四个点；
6. 取消访问；
7. 从路径中删除起点。

BFS 的算法思路：

1. 首先创建一个辅助二维数组，记录到某点的上一点的位置；
2. 维护一个队列，将起点添加到队列；
3. 设置起点的上一点位置为(-1, -1)；
4. 依次周围未被访问过的四个点，入队，并设置上一点的位置；
5. 重复步骤 4 直至队列为空；
6. 通过递归函数 InsertPath 得到从起点到终点的最短路径，存到 Path 数组中。

针对 DFS 和 BFS 函数，由于我进行了特殊的处理，让这两个函数的输入和输出保持一致，因此这两个函数在程序中可以直接调换，区别在于 DFS 能够比较快的找到从起点到终点的一条路径，并将其存到 Path 数组中，但是这条路径并不是最短路径，甚至有的时候要走很长；而 BFS 可以将从起点到终点的最短路径存到 Path 数组中。我还对 DFS 算法进行了优化，传统的 DFS+回溯一般是用来找所有路径，因此体现不出来 DFS 的快，经过处理之后的 DFS 算法在找到第一条路径后就直接退出回溯，大大提高了寻找速度。

尽管 DFS 搜索的速度很快，但是 Path 数组中存储的路径并非最优，因此我又写了一个 BFS 算法，在向四周进行探索时，通过一个二维数组保存了到某点的前一点的信息，由于一个点的信息有行列两个数字，因此我定义了一个结构体 Point，进而创建一个结构体二维数组来存储(line, col)(1 <= line <= lines, 1 <= col <= cols)前一点的信息。

在找到目标位置后，在新建的二维数组从后往前依次找到该点的前一位置，这样就可以的到最优的路径。

3.3 移动函数 DrawMove

针对 DrawMove 函数，我将移动过程分成了四个子函数，分别是 MoveUp、MoveRight、MoveDown 和 MoveLeft，对应画出向上、右、下、左的动画过程。

每个子函数通过多次调用 showstr 来实现移动效果的绘制，通过增加延迟 Sleep，让移动过程更加舒畅。

3.4 画带分隔线图形

针对 DrawArr2()函数，主要使用来画带分隔线的图形，具体来说，主要将图形界面分成三个部分，依次画出这三块，其中第二块绘制时，让循环变量和数组范围保持一致，这样绘制整体的逻辑比较清楚。

3.5 删除出连续球函数 IsDelete

针对 IsDelete()函数，判断得分的规则主要由这个函数确定，在本题中，当同一颜色的球在横向、纵向达到 5 个及以上时，可以消除，同时得到相应的分数，假设消除 n 个球，此次得到的分数为(n-1) * (n-2)。具体执行方法是，从目的位置出发，依次向上、右、下和左这四个方向试探，分别统计上下、左右的数量，如果上下或左右连续同颜色的球超过 5 个(包括 5 个)，记录删除的球的颜色和数量，并从数组中删除这些球，最后返回删除得分。

4. 调试过程碰到的问题

4.1 DFS 寻找路径，一直找不到正确路径

看到这道题，我首先就是想到回溯，回溯一般是用来找从起点到终点的全部路径，而在此处我想找到一条路径就返回，从而加快寻找速度。由于不能设置一个全局变量来保存找到结果的标志，就想从回溯算法的返回值入手，经过不断试错，当时头都大了，递归本身也比较难排查错误，最后睡了一觉，思路清晰以后在删除 DFS 的全部代码，重新写，结果正确！

4.2 每次消掉 5 个及以上连续球时再次选择空白地方会产生一个黑球

这个跟 MouseAction 函数有关，消除球后，pre_line, pre_col 还储存着上一个位置的信息，把这个信息删除就可以(置-1)，至于为什么储存了上一个位置的信息就会产生黑球，这个跟我写的 MoveAction 的判断逻辑有关。

4.3 设置终端大小跟我想的不一样

有的时候结束一个选项后返回菜单界面，尽管我已经设置了窗口大小，但并没有按照我设置的改变。老师给的 cmd_console_tools.cpp 里面说到必须先设置缓冲区，再设置窗口大小，否则若窗口大小大于当前缓冲区（未设置前）则设置失败，但没找到例子。经过网上查阅资料，明白了缓冲区的意思：在一行跟一屏能够显示的字符数量。然后在 to_be_continued 函数中重复设置一次窗口大小，解决了这一问题。

5. 完成本次作业心得体会

5.1 完成本次作业得到的一些心得体会、经验教训

1. 要注意变量的命名规则，这点主要体现在 MouseAction 函数，pre、cur 和 start、end 的区别；
2. 多用 const 常量定义，最好在一开始就先整体分析一遍题目，看有哪些地方可能会需要，避免后面懒得改的情况；
3. 虽然我有一个函数只写了一行，但我觉得这个函数非常有用，这个函数是 RecoverMark，让我不在需要判断在 cmd 上画图位置和数组位置的关系；
4. 在可以新建一个文件测试函数的正确性，比如我新建了一个文件专门测试 DFS 的正确性；
5. 尽量少用或不同全局变量，本题中要求不准用全局变量；
6. 写好函数的注视，包括参数的含义，函数的函数值，以及函数的作用等，好让自己和别人能够快速理解或使用这个函数；
7. 多写函数，控制一个函数中的代码长度，多通过函数嵌套来实现比较复杂的功能；
8. 程序出错，有可能不是编译器报错的地方出错，而是代码前面的代码有问题；
9. 代码写久了，遇到算法方面的问题，睡一觉吧，也许第二天问题可以轻松解决；
10. 对于递归函数，要从更高层面或者说直接思考函数的作用，而不是自己在脑中模拟递归函数的执行过程。

5.2 通过综合题 1/2 中有关函数的分解与使用，总结你在完成过程中是否考虑了前后小题的关联关系，是否能尽可能做到后面小题有效利用前面小题已完成的代码，如何才能更好地重用代码？

前 6 道小题我都考虑到了前后小题的关联关系，尽可能在做后面小题时直接调用前面已完成的代码(函数)。通过对一个函数增加一个参数 choice 就可以很好的在前面代码的基础之上进行新的功能。如几个比较主要的函数，DoByChoice45 和 DoByChoice67 都是通过这种方式，当然对于选项 1 和选项 2 这种，我觉得没比较整合到一起。

但是在写完选项 6，写选项 7 的时候，我感觉我 MouseAction 这一函数写的不好，没有考虑到循环调用，同时还发现自己之间没找到用户选择一个球以后选择另一个球的情况，最后我选择先写选项 7，然后再选项 6，虽然直接写选项 7 需要很长时间，但整体逻辑会清楚一点，而且写完选项 7 之后再写选项 6 就非常简单，加几行代码就好了。

为了能更好的重用代码，我觉得需要缩小函数的功能，多写几个函数，在写函数的时候，写好函数输入参数的含义、输出结果的含义以及函数的作用，方便自己之后调用函数。

总结：拿到一个项目，要先完整跑一边项目程序，明白每一步应该做什么，具体到每一个选项，这样可以对项目有一个大概的把握，从而能够更加合理地对函数进行划分，达到重用代码的目的。

6. 源程序

此处给出选项 7 涉及到的主要代码。

```
1.  void DoByChoice67(int arr[11][11], int lines, int cols, int user_choice, int visite
    d[11][11], char path[100][2], int three_ball_color[3], int three_ball_position[3][2
    ], int delete_ball_num[8])
2.  {
3.      int pos_x, pos_y;
4.      if (user_choice == 6)
5.      {
6.          cout << endl;
7.          PrintArr(arr, lines, cols);
8.          cout << endl;
9.          cct_getxy(pos_x, pos_y);  // 获取当前光标所在位置
10.         to_be_continued(2, pos_x, pos_y);
11.     }
12.     cct_cls();
13.     cct_setcursor(CURSOR_INVISIBLE);  // 不显示光标
14.     cct_setconsoleborder(70, 60, 70, 60);
15.     cct_setfontsize("新宋体", 28); //Truetype 字体只需要给出一个参数高度即可
16.     cout << "屏幕: " << 23 << "行" << 70 << "列(右键退出)" << endl;
17.     DrawArr2(arr, lines, cols);  // 有间隔
18.
19.     int start_line = -1, start_col = -1;  // 起点坐标
20.     int end_line = -1, end_col = -1;      // 终点坐标
21.     int score = 0;  // 记录总得分
22.
23.
24.     while (1)
25.     {
26.         // 这里不需要判断是否有路径，因为之后有路径才会退出 MouseAction
27.         for (int i = 0; i < 3; i++)  // 为了提前预告球的颜色
28.         {
29.             int color = rand() % 7 + 1;
30.             three_ball_color[i] = color;  // 1 改成 color
31.         }
32.         if (user_choice == 7)
33.             ShowMessage(arr, score, three_ball_color, lines, cols, delete_ball_num)
    ;
34.         MouseAction(arr, lines, cols, start_line, start_col, end_line, end_col);
35.         if (end_line == -1 && end_col == -1)  // 说明按了右键
36.             break;
37.         if (user_choice == 6)
38.             break;
39.         int per_score = IsDelete(arr, end_line, end_col, lines, cols, delete_ball_n
    um);  // 如果有 5 个及以上的话，会更改 arr 的值
40.         if (per_score != 0)  // 说明本次移动完成了消球
41.         {
42.             start_line = -1;
43.             start_col = -1;
44.             end_line = -1;
45.             end_col = -1;
46.         }
47.         CreateThreeBallInArr(arr, lines, cols, three_ball_color, three_ball_positio
    n);
48.         score += per_score;
49.         // DrawArr2(arr, lines, cols);
```

```
        DrawThreeBall(three_ball_position, three_ball_color);

        int sum = GetBallNumInArr(arr, lines, cols);  // sum 为球的数量
        if (sum + 3 > lines * cols)
        {
            cct_gotoxy(BASIS_COL + 0, BASIS_LINE + 2 * lines + 2);
            cout << "失败···" << endl;
        }
        if (sum == 0)
        {
            cct_gotoxy(BASIS_COL + 0, BASIS_LINE + 2 * lines + 2);
            cout << "成功！！！" << endl;
            break;
        }
    }
}
```

有分割线的伪图形界面绘制。

```
void DrawArr2(int arr[11][11], int lines, int cols)
{
    // 表格第一行
    int x = 0;
    cct_showstr(BASIS_COL + x * 2, BASIS_LINE + 0, "╔", COLOR_HWHITE, COLOR_HBLACK)
;
    x++;
    while (x < 2 * cols - 1)
    {
        cct_showstr(BASIS_COL + x * 2, BASIS_LINE + 0, "═", COLOR_HWHITE, COLOR_HBL
ACK);
        x++;
        cct_showstr(BASIS_COL + x * 2, BASIS_LINE + 0, "╦", COLOR_HWHITE, COLOR_HBL
ACK);
        x++;
    }
    cct_showstr(BASIS_COL + x * 2, BASIS_LINE + 0, "═", COLOR_HWHITE, COLOR_HBLACK)
;
    x++;
    cct_showstr(BASIS_COL + x * 2, BASIS_LINE + 0, "╗", COLOR_HWHITE, COLOR_HBLACK)
;

    // 表格中间
    int i = 1;
    while (i <= lines)
    {
        cct_showstr(BASIS_COL + 0, BASIS_LINE + i * 2 -
 1, "║", COLOR_HWHITE, COLOR_HBLACK);
        int j = 1;
        while (j <= cols)
        {
            if (arr[i][j] == 0)
                cct_showstr(BASIS_COL + j * 4 - 2, BASIS_LINE + i * 2 -
 1, "  ", COLOR_HWHITE, COLOR_HBLACK);
            else
                cct_showstr(BASIS_COL + j * 4 - 2, BASIS_LINE + i * 2 - 1, "
", arr[i][j], COLOR_HWHITE);
            cct_showstr(BASIS_COL + j * 4, BASIS_LINE + i * 2 -
 1, "║", COLOR_HWHITE, COLOR_HBLACK);
            j++;
        }
        if (i == lines)  // 不画最后一次的下面部分
```

```
34.             break;
35.         cct_showstr(BASIS_COL + 0, BASIS_LINE + i * 2, "╟", COLOR_HWHITE, COLOR_HBL
    ACK);
36.         j = 1;
37.         while (j < cols)
38.         {
39.             cct_showstr(BASIS_COL + j * 4 -
     2, BASIS_LINE + i * 2, "=", COLOR_HWHITE, COLOR_HBLACK);
40.             cct_showstr(BASIS_COL + j * 4, BASIS_LINE + i * 2, "╫", COLOR_HWHITE, C
    OLOR_HBLACK);
41.             j++;
42.         }
43.         cct_showstr(BASIS_COL + j * 4 -
     2, BASIS_LINE + i * 2, "=", COLOR_HWHITE, COLOR_HBLACK);
44.         cct_showstr(BASIS_COL + j * 4, BASIS_LINE + i * 2, "╢", COLOR_HWHITE, COLOR
    _HBLACK);
45.         i++;
46.     }
47.     // 表格最后一行
48.     x = 0;
49.     cct_showstr(BASIS_COL + x * 2, BASIS_LINE + i * 2, "╙", COLOR_HWHITE, COLOR_HBL
    ACK);
50.     x++;
51.     while (x < 2 * cols - 1)
52.     {
53.         cct_showstr(BASIS_COL + x * 2, BASIS_LINE + i * 2, "=", COLOR_HWHITE, COLOR
    _HBLACK);
54.         x++;
55.         cct_showstr(BASIS_COL + x * 2, BASIS_LINE + i * 2, "╨", COLOR_HWHITE, COLOR
    _HBLACK);
56.         x++;
57.     }
58.     cct_showstr(BASIS_COL + x * 2, BASIS_LINE + i * 2, "=", COLOR_HWHITE, COLOR_HBL
    ACK);
59.     x++;
60.     cct_showstr(BASIS_COL + x * 2, BASIS_LINE + i * 2, "╜", COLOR_HWHITE, COLOR_HBL
    ACK);
61.
62.     cct_setcolor();  // 恢复默认颜色
63. }
```

使用广度优先算法 BFS 寻求最短路径，结果存储在 path 数组中。

```
1.  int BFS(int arr[11][11], int lines, int cols, int start_line, int start_col, int en
    d_line, int end_col, int visited[11][11], char path[100][2])
2.  {
3.
4.      int is_path = 0;
5.      Point pre[11][11];  // pre 用于存储该点的前一点（类似父节点）从这一点走过来
6.
7.      int to_line;
8.      int to_col;
9.
10.     // 使用 BFS 遍历时，为了获得最短路径，每次路径需要保存上一点位置
11.     int path_index_arr[100][2];
12.     for (int i = 0; i < 100; i++)
13.     {
14.         path_index_arr[i][0] = -1;
15.         path_index_arr[i][1] = -1;
16.     }
17.
```

```
18.     int front = -1, tail = -1;  // front 指向队首，tail 指向队尾 front 指向第一个元素的前一项，tail 指向最后一个元素
19.     tail++;
20.     path_index_arr[tail][0] = start_line;
21.     path_index_arr[tail][1] = start_col;
22.
23.     pre[start_line][start_col].line = -1;
24.     pre[start_line][start_col].col = -1;
25.     while (front != tail)
26.     {
27.         int line = path_index_arr[front + 1][0];  // (line, col)为队头点
28.         int col = path_index_arr[front + 1][1];
29.         front++;  // 队首元素出队，移动队首指针就行
30.         if (line == end_line && col == end_col)  // 找到终点
31.         {
32.             is_path = 1;  // 存在路径
33.             break;
34.         }
35.         visited[line][col] = 1;
36.         if (arr[line - 1][col] == 0 && line > 1 && visited[line - 1][col] == 0)  // 向上
37.         {
38.             to_line = line - 1;
39.             to_col = col;
40.             visited[to_line][to_col] = 1;
41.             tail++;
42.             path_index_arr[tail][0] = to_line;
43.             path_index_arr[tail][1] = to_col;
44.             pre[to_line][to_col].line = line;
45.             pre[to_line][to_col].col = col;
46.         }
47.         if (arr[line][col + 1] == 0 && col < cols && visited[line][col + 1] == 0) // 向右
48.         {
49.             to_line = line;
50.             to_col = col + 1;
51.             visited[to_line][to_col] = 1;
52.             tail++;
53.             path_index_arr[tail][0] = to_line;
54.             path_index_arr[tail][1] = to_col;
55.             pre[to_line][to_col].line = line;
56.             pre[to_line][to_col].col = col;
57.         }
58.         if (arr[line + 1][col] == 0 && line < lines && visited[line + 1][col] == 0)  // 向下
59.         {
60.             to_line = line + 1;
61.             to_col = col;
62.             visited[to_line][to_col] = 1;
63.             tail++;
64.             path_index_arr[tail][0] = to_line;
65.             path_index_arr[tail][1] = to_col;
66.             pre[to_line][to_col].line = line;
67.             pre[to_line][to_col].col = col;
68.         }
69.         if (arr[line][col - 1] == 0 && col > 1 && visited[line][col - 1] == 0)  // 向左
70.         {
71.             to_line = line;
72.             to_col = col - 1;
```

```
            visited[to_line][to_col] = 1;
            tail++;
            path_index_arr[tail][0] = to_line;
            path_index_arr[tail][1] = to_col;
            pre[to_line][to_col].line = line;
            pre[to_line][to_col].col = col;
        }
    }
    int p = 0;
    InsertPath(end_line, end_col, pre, path, p);
    int i = 0;
    return is_path;
}
```

主要用来获取鼠标的动作，移动、左键和右键等。

```
void MouseAction(int arr[11][11], int lines, int cols, int& pre_line, int& pre_col, int& cur_line, int& cur_col)
{
    cout << endl;
    cct_gotoxy(BASIS_COL, BASIS_LINE + 2 * lines + 1);
    int pos_x, pos_y;
    cct_getxy(pos_x, pos_y);
    cct_enable_mouse();
    int mouse_x, mouse_y, action, keycode1, keycode2;  // x 横 y 竖

    while (1)
    {
        cct_read_keyboard_and_mouse(mouse_x, mouse_y, action, keycode1, keycode2);

        if (action == MOUSE_RIGHT_BUTTON_CLICK)  // 右键退出
        {
            cur_line = -1;
            cur_col = -1;
            break;
        }

        int line = (mouse_y + 1 - BASIS_LINE) / 2;
        int col = (mouse_x + 2 - BASIS_COL) / 4;
        if (line >= 1 && line <= lines && col >= 1 && col <= cols)
        {
            cct_showch(pos_x, pos_y, ' ', COLOR_BLACK, COLOR_WHITE, 30);
            cct_gotoxy(pos_x, pos_y);
            cout << "[当前光标] " << (char)(line + 'A' - 1) << "行" << col << "列" << endl;
            if (action == MOUSE_LEFT_BUTTON_CLICK)  // 按一下左键
            {
                cur_line = line;
                cur_col = col;
                if (arr[cur_line][cur_col] != 0)  // 说明选中球
                {
                    if (pre_line != -1 && pre_col != -1)  // 原来选中的球 取消选中标记
                        RecoverMark(pre_line, pre_col, arr[pre_line][pre_col]);
                    DrawMark(cur_line, cur_col, arr[cur_line][cur_col]);
                    pre_line = cur_line;
                    pre_col = cur_col;
                }
                else  // 说明选中空白地方
                {
```

```
                    if (pre_line != -1 && pre_col != -1)  // 说明之前选中过球 此时就要寻找路径了
                    {
                        int flag = DrawMove(arr, lines, cols, pre_line, pre_col, cur_line, cur_col);
                        if (flag == 1)  // 说明有路径
                        {
                            arr[cur_line][cur_col] = arr[pre_line][pre_col];  // 这两行的顺序不能换
                            arr[pre_line][pre_col] = 0;
                            pre_line = cur_line;  // 只有有路径才更新 没路径 pre 不变
                            pre_col = cur_col;
                            break;
                        }
                    }
                }
            }
        }
    }
    cct_setcolor();  // 恢复默认颜色
}
```

每次移动完后需要随机生成 3 个球，并显示在伪图形界面上，这里我采用的方式是在指定位置生成这 3 个球。

```
void DrawThreeBall(int three_ball[3][2], int three_ball_color[3])
{
    for (int w = 0; w < 3; w++)
    {
        int i = three_ball[w][0];
        int j = three_ball[w][1];
        int color = three_ball_color[w];
        RecoverMark(i, j, color);
    }
}
```

在(line, col)画球 1.可以是空处画球；2.也可以恢复未选中状态。

别看这个函数只有一行，非常好用。〇

```
void RecoverMark(int line, int col, int color)
{
    cct_showstr(BASIS_COL + col * 4 - 2, BASIS_LINE + line * 2 - 1, "〇", color, COLOR_HWHITE);
}
```