装

订

线

Color-Linez游戏实现报告

班级：计算机一班

姓名：王哲源

学号：1652228

完成日期：2017.1.5

1.题目及其描述

题目主要目的在于通过 CMD 及 API 实现小游戏 color-linez 的伪图形化，其中有以下分问题

1.1 利用内部数组随机生成 5 个球

其为本游戏实现的最基本步骤，即随机生成初始状况

1.2 利用内部数组随机生成 60%的球并实现单步寻路

其为本游戏实现的另一基本步骤，即在任意情况下对指定起点到终点是否存在可行路径并显示

1.3 利用内部数组实现游戏

利用 cmd 原始输出界面实现简单的游戏界面，并支持游戏基本的统分功能

1.4 在 1.1 的基础上绘制无分割线网格图

其为实现游戏伪图形化的第一步，即实现最简单的色块打印

1.5 在 1.4 的基础上绘制有分割线网格图

该小题为实现伪图形化界面的进一步改进

1.6 融合 1.2 与 1.4，实现单步展示

其为实现伪图形化的基础，即支持鼠标点触控制及路线伪图形化展示

1.7 伪图形化界面完整版

该步为伪图形化的最终版本，即原版游戏的 cmd 简易版本

*1.8 同时支持鼠标/键盘读入

对提供的函数进行修改，在 1.7 基础上实现原版游戏的键盘快捷键控制

*1.9 支持鼠标双击操作

对 1.8 中函数进行进一步修改，使其对于单击/双击操作有辨识能力

1.10 退出

2.整体设计思路

以下对于程序的整体设计思路进行概述，由于 1.8/1.9 非强制性要求，因此此处主要以 1.7 为主对设计进行概述（后同）

2.1 关于界面的存储

由于游戏操作界面为 9×9 的二维矩阵，因此此处利用一个 n*m(n,m<=15)的二维数组 map[i][j]表示第 i 行第 j 列位置的球的颜色。球颜色由 1-7 进行表示，空白位置则由 0 作为缺省值。

2.2 关于球的生成

考虑到，出事情况为一次性生成 5 个球，则可认为其是每次操作生成三个球的特殊情况，因此两者可共同调用同一个随机函数对球的颜色及位置进行生成。

2.3 关于寻路函数

为了实现移动路径的展示，且针对界面为二维矩阵图的特性，可以利用由指定起始点进行宽度优先搜索(以下简称 bfs)的方式对整张图进行搜索，若 bfs 结束目标点仍未被访问，则认为无合法路径。同时 bfs 过程可以不停记录每个点由哪个点拓展而来，便于之后的路径提取。

2.4 关于消除判断

球的消除情况分为两步，即第一步向 4 对共线的 8 个方向进行统计判断成线的同色球共多少颗，若存在满足消除条件的线，第二步再则对其进行消除并将分数进行统计。同时应当注意，由于每次移动/加球前图中必然不存在可以消除的球，因此我们只需要每次对移动过/新添加的球进行消除判断即可。

2.5 关于游戏结束的判断

游戏结束时，图中不存在可新添加球的位置，因此可利用这一特性，利用一个变量统计剩余空格个数，当该变量为 0 且处于新添加球步骤时宣布游戏结束。同时应当注意，当剩余空格不足 3 个时应当只生成剩余空格数量的球，以免陷入死循环。

2.6 关于图形界面的打印

鉴于每次移动/生成球之后，棋盘、总分统计、数据统计均有可能发生变化，因此采用每次操作之后先对内部数组进行更新后对整张图进行清空重新打印的方法实现图形界面的更新。同时对于左侧棋盘和右侧统分界面进行分别打印，以便支持 1.8 中的 FN 操作——即利用两个 bool 变量存储界面的显示/关闭情况，并根据其值判断是否打印统计部分。

3.主要功能实现

以下对整个程序的具体实现进行较为详细的介绍

3.1 预处理部分

在读入长(n)宽(m)之后，首先对于图进行清空，重置格子颜色的缺省值以及剩余空格数目，同时对矩阵的边框部分赋值-1，以防之后的操作发生非法访问的情况。

3.2 主体函数部分

主体函数先随机生成 5 个球，并对接下来产生的三个球颜色进行随机，存储在一个一维数组中，然后对于初始游戏界面进行打印。接下来执行一个直到型函数，其 bool 表达式为剩余空格(last)变量不为 0。循环内部首先调用鼠标操作判断，利用 API 获取鼠标输入内容，决定退出游戏/进行移动。在移动之后调用消除函数，并对总分及调用消除函数前总分进行对比，若发生消除，则不对接下来三个球的颜色进行随机，否则将新产生的球放置入 map 中，并减去对应空格个数，同时产生三个新的颜色的球。

3.3 寻路部分

寻路部分采用 bfs 的方法，以给定起始点开始，对四个方向进行拓展。此处四个方向可通过定义全局常量函数 fx[4],fy[4]对数组下标变化进行初步定义，以简化拓展时的代码量。根据 bfs 的特性，最先被拓展到的点必然经过起始点到该点最短的路径，因此可以同时实现最移动路径的寻找。此处同时应当注意，对于已经被访问过的点，不可再进行二次拓展，否则可能陷入死循环。对于每个点保存其由哪个点拓展而来，当寻路结束后利用链表特性从终止点反推即可得到路径。（此处提供一个更好的方法，即以终止点作为寻路起始点，便可省去逆推的步骤）

3.4 鼠标输入部分

利用控制台 API——ReadConsoleInput(hin, &mouseRec, 1, &res)可读取鼠标状况，其读取内容保存于结构体 mouseRec 下，利用成员 mouseRec.EventType 可判断是否为鼠标事件。而其成员 mouseRec.Event.MouseEvent.dwMousePosition 可获取当前鼠标所处的横纵坐标，因此可以实现鼠标位置的实时显示。同时，利用成员 mouseRec.Event.MouseEvent.dwEventFlags 可以判断鼠标的移动/点击事件，当移动时，当鼠标移动时，其值为 1；单击时，其值为 0，双击则为 2。而鼠标的具体点击情况则可根据成员 mouseRec.Event.MouseEvent.dwButtonState 的值进行判断，具体为： 0-无按键，1-左键按下，2-右键按下。利用这些内容可以确定返回值，并执行相应的鼠标操作

3.5 获取鼠标读入后的操作

根据函数返回值，当读入鼠标右键的事件时，直接退出循环，打印游戏结束界面。

否则，对于鼠标左键选取的点进行判断，分为以下 4 种情况：

1.若当前未选取球，且点击的位置为空，则不进行之后操作，重新读取

2.若当前未选取球，但点击的位置非空，则将被选取球位置的图形进行修改，同时将起始位置赋为当前点击的位置

3.若当前选取了球，但点击的位置非空，则不进行之后操作，重新读取

4.若当前选取了球，但点击的位置为空，则将被点击位置赋给终止点，并运行寻路函数，并根据寻路函数决定之后的操作。

3.6 消除函数

如 2.4 提及，消除函数主要由两部分组成。首先先时 Calc()函数，通过以被移动/新产生的球为中心，向 4 条线方向向外统计同色球的个数，如果同颜色球个数超过消除标准，则将 last 减去对应个数，再执行 Delete()函数，将 Calc()中的统计改为将 map 中的球色进行重置，以到达消除的目的。

此处应当注意两点：第一，统计时不能将中心球的个数算入，否则当存在两条以上可消除线时，中心球会被重复计算；第二，为了简便代码，我们可以参考二维矩阵 bfs 中的防线函数，即定义两个全局常量函数 gx[8],gy[8]，以下标 2*i 与 2*i+1 为一对作为同一条线上(例如上与下，左与右)二维下标的变化量，通过 for 循环可以实现代码的简便实现。

3.7 图形界面打印

打印的基本思路为，先打印左侧棋盘，再根据 F3/F5 的情况决定是否打印右侧的统计情况。其中，我们需要一个变量 tot 用于计算总分，以及一个数组 sum_del[]用于存储当前我们一共消除了多少个某种颜色的球，其值的修改同 last 传入消除函数进行统计即可。

当球发生移动时，我们通过寻路函数获得路径，然后从起始点开始逐点移动。这里可以利用一个单位变量，通过判断下一个点是在当前点左/右分别赋值-2/2 以节省代码量（注意，上下移动的单位为-1/1）。同时，对于每次循环我们只移动一格，这样既方便边寻路边打印，同时也方便移动之后对于棋盘边框的复原。

而对于界面的更新，我们采用的是将界面清空之后重新打印一次。但这样带来的问题则是频繁的清屏会导致屏幕闪烁严重，因此为了避免太过频繁的闪烁，我们在整个移动操作执行结束一次之后，再对屏幕进行重新打印，这样可以省去在界面更新后再写其他函数对被修改位置的覆盖，同时利用运行结束所有数据更新完毕的特性直接将新的界面打印出来。

4. 调试过程遇到的问题

以下将对调试过程中遇到的几个主要问题进行概述

由于 1.7 部分调试错误不多，因此将涉及 1.8 及 1.9 中内容

4.1 起始球只剩一个

这一问题出现概率非常小，导致一开始并未考虑到。由于一开始生成球的函数仅仅考虑如果生成的球会发生消去则重新生成，并未发现其会对于还需生成的球个数(x)也会产生影响，结果就导致了当生成的 5 个球颜色一样且满足消除条件时（这个概率极其极其的小），虽然会把这些

球消去，但 x 此时仍然为 1，因此就导致了起始就只剩一个球的情况。

解决方法则是，由于起始生成球所调用的与后续生成球的函数相同，因此保存 last 前值，并将 last 传入 check 函数判断是否满足消除，当返回发生消除时，将两者做差便可确认消除了多少球，并通过其值使 x 加上对应值达到复原目的。

*4.2 无法同时支持键盘操作及鼠标操作

在下发的读取鼠标函数 read_mouse()中忽略了所有非鼠标事件，导致无法做到对鼠标/键盘同时读取的操作。

通过观察发现，read_mouse ()中读取当 mouseRec.EventType != MOUSE_EVENT 即输入事件非鼠标事件则跳过，因此初步判定需要在此处进行修改。通过查看发现 INPUT_RECORD 结构体即 mouseRec 下含成员.Event，其中包含.KeyEvent，而当 mouseRec.EventType 值为 KEY_EVENT 时则说明读入事件为键盘事件，利用这一点便可以提取出非鼠标事件中所有的键盘事件。同时，mouseRec.Event.KeyEvent 下的成员.wVirtualKeyCode 提供了键盘的虚拟按键值(VK_CODE)，通过这个可以直接获取 FN 的值，而非通过 ascii 的方式。

但更改函数之后又发现，每次 F3/F5 按下之后右侧统计栏并不会消去，同时还发现 FN 按下之后会导致屏幕闪烁两次（3.7 中清屏的打印方法为 DEBUG 提供了很大的帮助），这说明读取了两次按键值，通过输出中间变量发现的确如此。因此推断对于键盘的 VK_CODE 读取，ReadConsoleInput 会读取两次，即按下和弹起分别读取一次，而 mouseRec.Event.KeyEvent 下的.bKeyDown 成员又恰好提供了判断键位当前处于按下/弹起的状态。因此利用这一值可以将键位弹起时所读取到的 VK_CODE 过滤，即实现了游戏同时支持键盘和鼠标操作的要求。

*4.3 双击操作前单击操作的过滤及双键操作的读取

通过尝试发现，当前的 read_mouse()函数内 ReadConsoleInput(hin, &mouseRec, 1, &res)对于鼠标双击事件的读取会将双击的第一次当作单击读入，再读入第二次双击；而当左右双键同时按下时，则会先读去一边的键位，然后读取出双键，最终又马上读入一侧的键位值。为了解决该问题，进行了以下的尝试。

首先是在 read_mouse() 函数中输出非鼠标移动操作，发现当双击时，mouseRec.Event.MouseEvent.dwEventFlags 的值变化为 1，0，1，2，即双击时不可避免的会读入单击事件，因此想到的解决方法是通过限制一定的卡时来过滤该段时间内的非点击操作，若在该段时间发生了同键单击时间，则认为其为双击事件，退出卡时。但由于 ReadConsoleInput 的限制，在读入缓冲区无数据时会进行等待，因此导致卡时无效。对于此，我们采用了另一个控制台 API 函数 PeekConsoleInpout(hin, &mouseRec, 1, &res)，其会读取读入缓冲区中的数据且不对其进行清空，当缓冲区中不存在数据时立刻返回 res=0。利用这一点我们可以先判断缓冲区中是否有数据，再对其进行读入判断。但随之而来的问题是有时双键读取会失效为单键，根据 4.2 中.bKeyDown 的经验，猜测到可能是鼠标按下与弹起时同键盘一样会有两个键值，此时发现 mouseRec.Event.MouseEvent.dwEventFlags 中可以针对键位按下/弹起判断，具体已在 3.4 中有所提及。因此我们只需要将第一次读取到的 0 忽略掉，即可过滤由于键位弹起而造成的误读。至此，区分单双击的问题已解决。

而接下来的问题则是左右键双键问题，参考单双击的问题，可以推断，由于按键有一定的延时，即无法做到真正的同时按下双键，因此其读取类似双击操作，因而在双击延时判断中加入了返回最后点击的键位值，便可以根据这个值进行判断双键操作。但这样虽然解决了双键前单击的清除，但有时候仍然会被判断为左键双击/右键双击。通过加长延时，尝试不同组合双键的按松方式，发现当双键读入时一侧先抬起时 mouseRec.Event.MouseEvent.dwButtonState 的值会仍然保持 1 而非 0，即仍然会读取处于按下的键位的值。同时通过研究 demo 中按键返回值，发现当双键的响应为按下马上响应而非抬起时，同时当双键中单侧抬起，无论鼠标如何移动，等待多长时间，再次按下松开的键，得到的仍然是双键的响应。因此解决方法采用了一个固定变量 mouse_pre_event 用于存储上一次是否为双键操作，若是，且当前读取到的仍然是单键操作，则认为当前键位仍未松开，即等待下一次双键操作，忽略这期间一切的单键操作。最终通过在主函数中添加鼠标读取返回值验证，该方法可行，且基本达到了和 demo 一致的操作。

5. 总结部分

5.1 一些体会

经过上一次 hanoi 的编写，且由于这次 color_linez 的二维实现难度不大，因此编写过程中并未出现过多的巨大失误，对于变量的初始值/中间变量处理以及各个函数间配合均有了一定程度的提升。但仍有一些不足，例如在 1.1 中随机生成球的函数出现 4.1 中提及的严重逻辑 bug，若非极小概率事件发生，该问题可能很长时间都无法被发现。另外，对于 1.2 中 bfs 寻路采用了较为复杂的方法（3.3 中提及），有时候逆向思维也是一种很好的思路。

这次程序难度于 1.6 加入鼠标操作之后有一定的提升，但配合模板总体编写难度不大，1.8 解决也是在阅读工具函数时可以确认大致方向的,因此整个程序难度主要集中于 1.9 的键位判断，以及系统函数特性、系统结构体的尝试。而最好的方法就是通过输出中间变量值查看函数运作规律，这一点对于之后的代码学习及源代码查看、API 调用有着巨大的帮助。

5.2 关于程序的高效编写

上一次 hanoi 编写已经展现了分块对于高效编写/查错的影响。而这次则展现了源程序拆分对代码整体简介度的影响。通过按类型拆分源程序,可以在 debug 时迅速定位到需要的函数之中，同时可以降低每个源程序中的冗长程度，对于阅读有很大的帮助。

同时，依然应当注意编写函数时对后续题的考虑。这次编写 1-3 时由于一开始想尝试将其进行整合,导致了花费近三个小时,但最终还是采用拆分函数的方式。而后续函数的调用也证明了，将关系并非非常紧密的部分进行函数拆分对于程序的后续添加有着很大的帮助。也正因为这一点，这次编写函数，例如寻路函数/打印函数时会考虑后续小题中是否需要传入别的参数，并为此预先设置好了形参，以便后续小题的对接，而其具体体现在当 1.3 与 1.6 编写完毕之后，程序基本主体已完成，剩余小题只需要对已经写好的函数进行调用即可，极大的减少了编写的时间。

6. 附件：程序（此处以 1.7 为主，且将贴入 1.9 改进后的读取函数，不包含打印函数）

主函数部分：

```
int work_7(const int mode, int(*map)[Maxm], const int n, const int m, int &last)
{
    const int Clear = n*m;

    bool setting[2] = { 1,1 };
    reset_map(5, map, n, m, last);

    bool flag = 1;
    int col[4], tot = 0;
    int sum_del[8] = { 0 };

    For(i, 1, 3)
        col[i] = rand() % Maxc + 1;
    reset_pic(mode, map, n, m);
    reset_menu(tot, col, map, n, m, sum_del, setting);

    do
    {
        int pretot = tot, res;
        if (res = mouse_work(mode, map, n, m, last, tot, sum_del, setting))
        {
            if (res == 1)
                break;
            else if (res == 2)
                return 1;
            else
            {
                reset_pic(mode, map, n, m);
                reset_menu(tot, col, map, n, m, sum_del, setting);
                continue;
            }
        }
        reset_menu(tot, col, map, n, m, sum_del, setting);
        if (!(tot - pretot))
        {
            int lim = Min(3, last), cnt = 1;
            while (cnt <= lim)
            {
                if (rand_ball(map, n, m, col[cnt], last))
                {
                    col[cnt] = rand() % Maxc + 1;
                    continue;
                }
                ++cnt;
            }
            For(i, 1, 3)
                col[i] = rand() % Maxc + 1;
        }
        reset_pic(mode, map, n, m);
        reset_menu(tot, col, map, n, m, sum_del, setting);
    } while (last > 0 && last != Clear);
    gotoxy(hout, 0, 16 + 2 * (n - 7));
    setcolor(hout, 0, 7);
    printf("
");
    gotoxy(hout, 0, 16 + 2 * (n - 7));
    printf("游戏结束!!!!! \n");
    return 0;
}
```

鼠标读入部分：

```
int mouse_work(const int mode, int(*map)[Maxm], const int n, const int m, int &last, int &score, int *sum_del, bool*setting)
{
    const int size = 28 - 4 * (n - 7);
    bool select = 0;
    int mouse[2] = { 0 }, prex, prey;

    while (1)
    {
        int mouse_init_f = mouse_init(mode, map, n, m, mouse, setting);
        if (mouse_init_f)
            return mouse_init_f;
        int x = mouse[0], y = mouse[1];
        if (!map[x][y] && !select)
            continue;
        if (map[x][y] && !select)
        {
            select = 1;
            gotoxy(hout, 4 * y - 2, 2 * x);
            setfontsize(hout, L"新宋体", size);
            setcolor(hout, 7 + map[x][y], 0);
            printf("◎");
            prex = x, prey = y;
            setcolor(hout, 0, 7);
            continue;
```

```
            }
            if (map[x][y] && select)
            {
                setfontsize(hout, L"新宋体", size);
                setcolor(hout, 7 + map[prex][prey], 0);
                gotoxy(hout, 4 * prey - 2, 2 * prex);
                printf("○");
                setcolor(hout, 7 + map[x][y], 0);
                gotoxy(hout, 4 * y - 2, 2 * x);
                printf("◎");
                prex = x, prey = y;
                setcolor(hout, 0, 7);
                continue;
            }

            bool bfs(int(*map)[Maxm], const int sx, const int sy, const int ex, const int ey, int(*tox)[Maxm], int(*toy)[Maxm]);
            int tox[Maxn][Maxn] = { 0 };
            int toy[Maxn][Maxn] = { 0 };
            if (!bfs(map, prex, prey, x, y, tox, toy))
                continue;
            map[x][y] = map[prex][prey];
            map[prex][prey] = 0;
            ball_moving(map, n, m, prex, prey, x, y, tox, toy, 7 + map[x][y]);
            score += check(map, x, y, last, sum_del);

            return 0;
        }
    }
    int mouse_init(const int mode, int(*map)[Maxm], const int n, const int m, int *mouse, bool*setting)
    {
        int X = 0, Y = 0;
        enable_mouse(hin);
        setcursor(hout, CURSOR_INVISIBLE);
        while (1)
        {
            int mouse_action;
            if (mode <= 7)
                mouse_action = read_mouse(hin, X, Y);
            else
            {
                mouse_action = read_mouse_and_key(hin, mode, X, Y);
                if (mouse_action == VK_F3)
                {
                    setting[0] ^= 1;
                    return 3;
                }
                else if (mouse_action == VK_F5)
                {
                    setting[1] ^= 1;
                    return 3;
                }
                else if (mouse_action == VK_F4)
                    return 2;
            }
            /*************************

            gotoxy(hout, 0, 16 + 2 * (n - 7));
            setcolor(hout, 0, 7);
            if ((X >= 2 && X <= 4 * m - 1 && ((X % 4 == 2) || (X % 4) == 3)) && (Y >= 2 && Y <= 2 * n && (Y % 2) == 0))
            {
                printf("[当前光标位置]：%c行%d列", Y / 2 + 'A' - 1, (X + 2) / 4);
                if (mouse_action == MOUSE_RIGHT_BUTTON_CLICK)
                    return 1;
                if (mouse_action == MOUSE_LEFT_BUTTON_CLICK)
                {
                    mouse[0] = Y / 2, mouse[1] = (X + 2) / 4;
                    return 0;
                }
            }
        }
    }
    int read_mouse_and_key(const HANDLE hin,const int mode, int &X, int &Y, const int enable_read_mouse_moved)
    {
        INPUT_RECORD mouseRec;
        DWORD res;
        COORD crPos;

        while (1)
        {
            static int mouse_pre_event = MOUSE_MOVED;
            ReadConsoleInput(hin, &mouseRec, 1, &res);

            if (mouseRec.EventType != MOUSE_EVENT)
                if (mouseRec.EventType == KEY_EVENT && mouseRec.Event.KeyEvent.bKeyDown
```

```
//avoid the twice of input
                        &&
(mouseRec.Event.KeyEvent.wVirtualKeyCode   ==
VK_F3
                            ||
mouseRec.Event.KeyEvent.wVirtualKeyCode    ==
VK_F4
                            ||
mouseRec.Event.KeyEvent.wVirtualKeyCode    ==
VK_F5))
                        return
mouseRec.Event.KeyEvent.wVirtualKeyCode;
                    else
                        continue;

                crPos                           =
mouseRec.Event.MouseEvent.dwMousePosition;
                X = crPos.X;
                Y = crPos.Y;

                if (mode==9 && mouse_pre_event
==      (FROM_LEFT_1ST_BUTTON_PRESSED      |
RIGHTMOST_BUTTON_PRESSED))
                {
                    if
(mouseRec.Event.MouseEvent.dwButtonState)
                        return
MOUSE_ONLY_MOVED;
                    mouse_pre_event          =
MOUSE_MOVED;
                }
                if (enable_read_mouse_moved &&
mouseRec.Event.MouseEvent.dwEventFlags     ==
MOUSE_MOVED)
                    return MOUSE_ONLY_MOVED;

                int Double_Click_Res;
                if
(mouseRec.Event.MouseEvent.dwButtonState   ==
(FROM_LEFT_1ST_BUTTON_PRESSED              |
RIGHTMOST_BUTTON_PRESSED))
                {
                    mouse_pre_event          =
(FROM_LEFT_1ST_BUTTON_PRESSED              |
RIGHTMOST_BUTTON_PRESSED);
                    return
MOUSE_LEFTRIGHT_BUTTON_CLICK;
                }
                else                          if
(mouseRec.Event.MouseEvent.dwButtonState   ==
FROM_LEFT_1ST_BUTTON_PRESSED)
                {
                    if
(mouseRec.Event.MouseEvent.dwEventFlags    ==
DOUBLE_CLICK)
                        return
MOUSE_LEFT_BUTTON_DOUBLE_CLICK;
                    else  if  (mode  ==  9  &&
(Double_Click_Res                          =
Check_Double_Click(FROM_LEFT_1ST_BUTTON_PRES
SED)))
                    {
                        if  (Double_Click_Res
== RIGHTMOST_BUTTON_PRESSED)
                        {
                            mouse_pre_event =
(FROM_LEFT_1ST_BUTTON_PRESSED              |
RIGHTMOST_BUTTON_PRESSED);
                            return
MOUSE_LEFTRIGHT_BUTTON_CLICK;
                        }
                        else
                            return
MOUSE_LEFT_BUTTON_DOUBLE_CLICK;
                    }
                    else
                        return
MOUSE_LEFT_BUTTON_CLICK;
                }
                else                          if
(mouseRec.Event.MouseEvent.dwButtonState   ==
RIGHTMOST_BUTTON_PRESSED)
                {
                    if
(mouseRec.Event.MouseEvent.dwEventFlags    ==
DOUBLE_CLICK)
                        return
MOUSE_RIGHT_BUTTON_DOUBLE_CLICK;
                    else  if  (mode  ==  9  &&
(Double_Click_Res                          =
Check_Double_Click(RIGHTMOST_BUTTON_PRESSED)
))
                    {
                        if  (Double_Click_Res
== FROM_LEFT_1ST_BUTTON_PRESSED)
                        {
                            mouse_pre_event =
(FROM_LEFT_1ST_BUTTON_PRESSED              |
RIGHTMOST_BUTTON_PRESSED);
                            return
MOUSE_LEFTRIGHT_BUTTON_CLICK;
                        }
                        else
                            return
MOUSE_RIGHT_BUTTON_DOUBLE_CLICK;
                    }
                    else
                        return
MOUSE_RIGHT_BUTTON_CLICK;
                }
                else
                    ;
```

```
        }
        return MOUSE_NO_ACTION;
    }
    int Check_Double_Click(int pre_mouse_event, const int enable_read_mouse_moved)
    {
        const int Time_Limit = 200;
        long time = clock();

        INPUT_RECORD    mouseRec;
        DWORD           res;

        bool release = 1;
        while(clock()-time<Time_Limit)
        {
            PeekConsoleInput(hin, &mouseRec, 1, &res);
            if (res > 0)
            {
                ReadConsoleInput(hin, &mouseRec, 1, &res);
                if (mouseRec.EventType != MOUSE_EVENT)
                    continue;
                if (enable_read_mouse_moved && mouseRec.Event.MouseEvent.dwEventFlags == MOUSE_MOVED)
                    continue;
                if (mouseRec.Event.MouseEvent.dwButtonState == (FROM_LEFT_1ST_BUTTON_PRESSED | RIGHTMOST_BUTTON_PRESSED))
                {
                    if (pre_mouse_event == FROM_LEFT_1ST_BUTTON_PRESSED)
                        return RIGHTMOST_BUTTON_PRESSED;
                    return FROM_LEFT_1ST_BUTTON_PRESSED;
                }
                if (!mouseRec.Event.MouseEvent.dwEventFlags)
                {
                    release ^= 1;
                    if (!release)
                        continue;
                }
                return mouseRec.Event.MouseEvent.dwButtonState;
            }
        }
        return 0;
    }
```

生成及判断部分

```
    int rand_ball(int(*map)[Maxm], const int n, const int m, const int col, int &last)
    {
        int tmp[8] = { 0 };
        while (1)
        {
            int i = rand() % n + 1;
            int j = rand() % m + 1;
            if (map[i][j])
                continue;

            map[i][j] = col;
            --last;
            return check(map, i, j, last, tmp);
        }
    }
    int check(int(*map)[Maxm], const int x, const int y, int &last, int *sum_del)
    {
        int A = 1, B = 0;
        For(i, 0, 3)
        {
            int tmp = Calc(map, x, y, map[x][y], gx[2 * i], gy[2 * i])
                + Calc(map, x, y, map[x][y], gx[2 * i + 1], gy[2 * i + 1]);
            if (tmp >= 4)
            {
                Delete(map, x, y, map[x][y], gx[2 * i], gy[2 * i]);
                Delete(map, x, y, map[x][y], gx[2 * i + 1], gy[2 * i + 1]);
                A += tmp;
                ++B;
            }
        }
        if (A == 1)
            return 0;
        int res = 10 + 2 * (A + B - 6)*(A + B - 6);
        last += A;
        sum_del[map[x][y]] += A;
        map[x][y] = 0;
        return res;
    }
    int Calc(int(*map)[Maxm], int x, int y, const int col, const int fx, const int fy)
    {
        int res = 0;
        x += fx, y += fy;
        while (map[x][y] == col)
            ++res, x += fx, y += fy;
        return res;
    }
```

```
    void Delete(int(*map)[Maxm], int x, int y,
const int col, const int fx, const int fy)
    {
        x += fx, y += fy;
        while (map[x][y] == col)
            map[x][y] = 0, x += fx, y += fy;
    }
```

寻路部分

```
    bool bfs(int(*map)[Maxm], const int sx,
const int sy, const int ex, const int ey,
int(*tox)[Maxm], int(*toy)[Maxm])
    {
        int l = 1, r = 2;
        int Qx[Maxx] = { 0 }, Qy[Maxx] = { 0 };
        int frox[Maxn][Maxm] = { 0 },
froy[Maxn][Maxm] = { 0 };
        bool f[Maxn][Maxm] = { 0 };
        Qx[1] = sx, Qy[1] = sy;
        f[sx][sy] = 1;
        while (l < r)
        {
            int ux = Qx[l], uy = Qy[l];
            if (ux == ex && uy == ey)
            {
                f[ex][ey] = 1;
                break;
            }
            For(i, 0, 3)
            {
                int vx = ux + fx[i], vy = uy
+ fy[i];
                if    (map[vx][vy]    ||
f[vx][vy])
                    continue;
                frox[vx][vy] = ux;
                froy[vx][vy] = uy;
                Qx[r] = vx, Qy[r] = vy;
                f[vx][vy] = 1;
                ++r;
            }
            ++l;
        }
        if (!f[ex][ey])
            return 0;
        get_route(sx, sy, ex, ey, frox, froy,
tox, toy);
        return 1;
    }
    void get_route(const int sx, const int sy,
const int ex, const int ey, int(*frox)[Maxm],
int(*froy)[Maxm],          int(*tox)[Maxm],
int(*toy)[Maxm])
    {
        int curx = ex, cury = ey;
        while (curx != sx || cury != sy)
        {
            int prex = frox[curx][cury],
prey = froy[curx][cury];
            tox[prex][prey] = curx;
            toy[prex][prey] = cury;
            curx = prex, cury = prey;
        }
    }
```

装 订 线