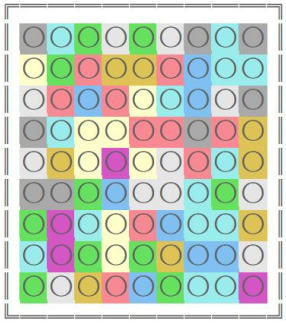
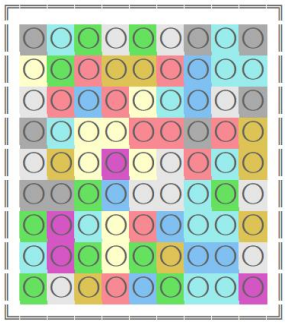
同济大学计算机系

高级语言程序设计报告-2

Project 2　MAGIC BALL



学　号　：　2152131
姓　名　：　吴洪蕊
专　业　：　计算机科学与技术
项目名称　：　MAGIC BALL
日期　：　2024-6-15



5 号写完？15 号发现没交上？？

1. 项目问题重述

和消消乐基本一个规则的 MagicBall 游戏，其区域为 5*5~9*9，共有 9 种颜色的彩球随机出现，初始随机填满，随机概率相同。消除规则为横向/纵向连续颜色超过 3 个，每消除 1 个球计 1 分，如果横纵向同时存在，则分别计算是否超过三个。

消除后，彩球下落填补空位，最上方的空位再随机填满。并如此循环的判断是否还存在初始可以自动符合消除的项，则立即消除/填充/再消除/再填充…

当无初始可消除项才停止，**停止前的所有消除项不计分。**

遍历整个游戏区域，给出可以互换构造新的可消除项的彩球位置（成对标出），用鼠标选择可互换的球(再按一次则取消选择)，再选择邻近的另一个可互换的球，进行交换。

交换后进行消除/填充…（进入之前的循环判断，检查是否有初始可消除项），检查结束后给出提示，继续游戏模式，游戏模式中可以有记录分数和检查游戏是否进入死局以及右键退出等互动模式。

```
------------------------------------------------------------
 1.内部数组，生成初始状态，寻找初始项
 2.内部数组，消除初始项后下落并用 0 填充
 3.内部数组，消除初始项后查找消除提示
 4.n*n 的框架(无分隔线)，显示初始状态
 5.n*n 的框架(有分隔线)，显示初始状态
 6.n*n 的框架(无)，显示初始及初始可消除项
 7.n*n 的框架，消除初始项后显示消除提示
 8.cmd 图形界面完整版(有)，鼠标可移动
 9.cmd 图形界面完整版
------------------------------------------------------------
```

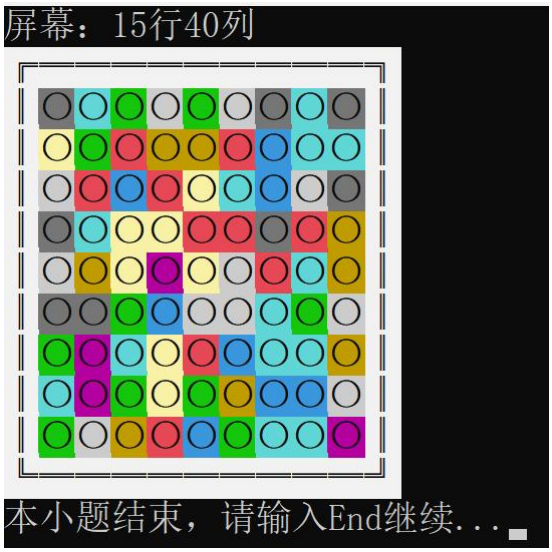


图 1 图形化页面

2. 整体设计思路

以第八个选项为例，文字化表述程序执行过程。和汉诺塔作业类似，main 函数的处理逻辑时接收菜单选择跳转对应的选择函数执行完成之后，输入 End(无论大小写)返回菜单选择页面，进行下一次。

有所调整的是，汉诺塔的一本参数的选择是放在选择函数的内部的，汉诺塔是因为并不是所有菜单选项需要输入基本参数，但是这次的魔法球是需要所有选项输入行数和列数且只有这两个。所以将 input 函数和二维数组的初始化与随机数的填充都放在了 base 程序中。（第八个选项不涉及纯数组的打印这个放在后文中介绍）

完成准备工作之后，进入第一个循环检察环节——检查是否有初始可消除项目。如果检查到了就进行消除并将彩球下落，空位进行填补，再一次检查是否有初始可以消除项；如果没有检查到初始可

以消除的项，进入给出提示环节，将可以互换形成消除项的彩球进行标注。此时进入悬浮模式，下方可以返回此时鼠标所在位置的合法性和合法坐标，如果是选项九，则再选项八的基础上进行彩球的交换，移除消除项的逻辑和上一个循环检查的逻辑一致，查找到消除项目→消除→下落→填补→消除检查→检查循环，如果循环结束没有初始可以消除的项则显示提示进行下一次的可互换球的游戏模式，如果已经没有提示了，则说明游戏进入死局，输入 end 返回菜单选择页面。下方是流程图。

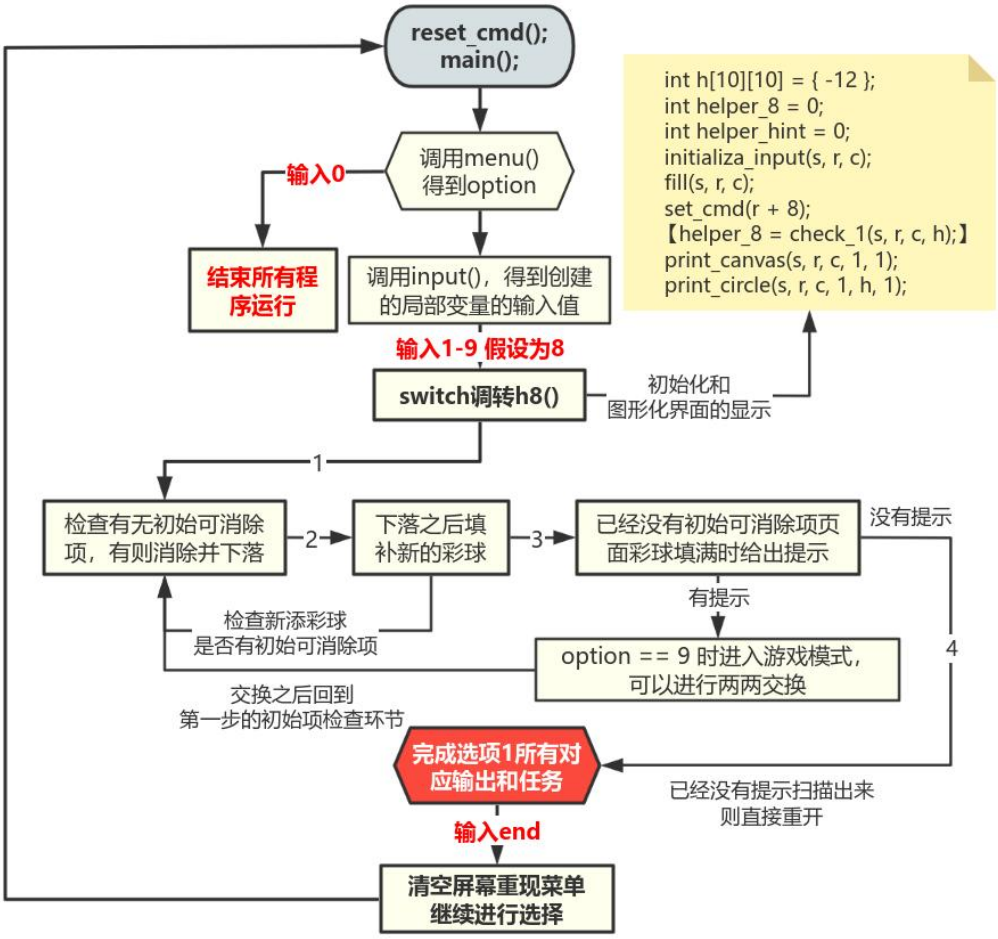


图 2 流程图

3. 主要功能的实现

输入、初始化、数字版本的打印大同小异这里不详细介绍如果后面页数不够回过头来写，主要介绍的是：**检查可消除项、去除消除项填充为零并以新数值覆盖、给出提示、彩球下落、鼠标悬浮与游戏模式**这几个关键功能函数的设计思路。

检查可消除项：

这里使用了一个h[10][10]伴随数组来用作位置标记，s[i][j]数组我既需要这个位置原本填的是什么数字也需要标记这个地方是否是可以消除的对象，由于写的时候没有学结构体就设置了一个伴随二位数组。

此外要懊悔的是当时设置数组大小的时候选择的是[10][10]但是后面有一个找提示的环节发现遍历的时候出现了超过边界的情况，但是就要因此所有的s[10][10]改成s[13][13]乃至更大吗？显然也是不方便的，但是当时设置数组大小的时候确实也没考虑太多。

检查思路为遍历所有的位置，该位置能否为横向遍历方向的后两个数字相同，和纵向遍方向的下面两个数字相同？如果是则在伴随函数h[10][10]相同位置由0标为1，如此完成全部位置的可消除项的检查。

```
if (s[i][j] == s[i][j + 1] && s[i][j] == s[i][j + 2] && s[i][j] != -1) {
    h[i][j] = h[i][j + 1] = h[i][j + 2] = 1; // 伴随数组同位置做上标记
    helper = 1; // 标记确实有可消除项
}
```

去除消除项填充为0并以新值填补：

这里的填充为0更多的是为数字打印服务的，和后面图形化设计下落函数关系不大。在上一个环节完成初始项检查之后，数字打印步骤是让所有所有消除项变为0并向上浮动的。

所以按照单列去遍历，从下往上遍历，遇到标记处就从所在位置开始上面的所有数字下移——等于自己上面位置的数字，而顶部位置标为0 ，遍历标号自减1以防止下落的数字覆盖所遍历处的数字也还是被标记的数字（e.g.比如竖着的消除项组）此时可以顺便把h[10][10]伴随数组给消除标记，当初始化了（这也是我觉得写的不好的一个地方，因为h是函数到处公用的用于标记的伴随数组，有的时候没有及时初始化会出现不同功能的标记交叉冲突的情况）

图 3 填补为零数字下落实现思路图

如此在s数组中得到了一堆需要新数值填补的0，所以设计一个fill_new函数来遍历找s[i][j]处为0的地方，随机函数填上新的数值继续返回查找初始可消除项，循环检查函数如下，这里的检查函数是有返回值的，也就是一直循环检查到没有发现任何可消除项，再进入下一个环节。

```
helper_2 = check_1(s, r, c, h);
if (helper_2) {
    while (helper_2) {
```

```
            print_out(s, r, c, 10, h);
            enter_option(2, 0);
            remove_0(s, r, c, h);
            print_out(s, r, c, 11, h);
            enter_option(3, 0);
            fill_new(s, r, c, h, 0);
            print_out(s, r, c, 12, h);
            helper_2 = check_1(s, r, c, h);
        }
    }
    else {
        cout << "本小题无法测试，请再次运行" << endl;
    }
```

（在这个函数里面可以顺便画个图）：

在选项8设计图形化彩球下落时，其中得到空白的下落空位之后的新彩球的填补我这里是和fill_new函数公用了遍历过程，增设了一个int print_new_circle选项， 如果是0那么就是一般的前三个选项的只需要打印数字就可以的可视化，如果是1就是需要顺便画上彩球空位。

给出提示：

该环节的设计我感觉有点受到文档的误导了？开始我觉得是需要稍微带一些穷举或者考虑很多文档里面没有列举出来的情况，但是画了几张图图之后就发现文档那样的分类和叠加是比较需要遍历讨论特殊情况的，我还是太懒了，想着有没有一招鲜的思路：

给出提示的环节，全局是没有任何三个相连的排列的，此外一个球的移动无非是上下左右**（边界球后面有特殊讨论）**，我们假设一个球向上移动就可以使得 与它交换的球的消除**（假设是横向的）**，那么这个交换的球是不是一定是串消除项组的某个位置的成员**（假设只促成了三个相消）**，这个被换过来的球无非是这个三个串组的第一个、第二个、第三个（行列大小分次序，横向的）。那么只要考虑交换后是否满足下图得到某一种情况就可以判定这里是否需要打上标记符号，（这里语言叙述有点抽象，画一张图解释一下）

那么上下交换的三种横向与纵向唯一的三个相同球的相消情况就考虑完全了，我们再来考虑**三个假设**——如何把特殊情况去掉变成一般假设。

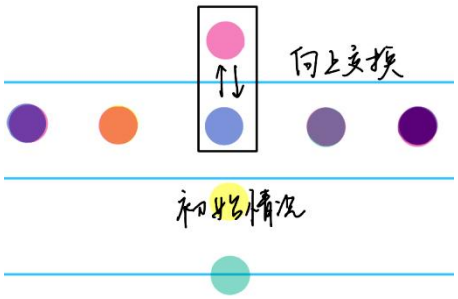
第一个是只有三个相同球的消除，其实这个假设是不必要的。因为我们检查的逻辑实现是上一个check_1()函数来检查，如果是四个函数是会标记的，而且不管是四个相消还是五个相消都是从三个相同球的消除子问题叠加发展来的，所以问题不大；

第二个假设是上下交换。这个也问题不大，考虑到这个是一个十分对称的问题，所以我能够给出上下交换的情况也可以画出左右交换的情况；

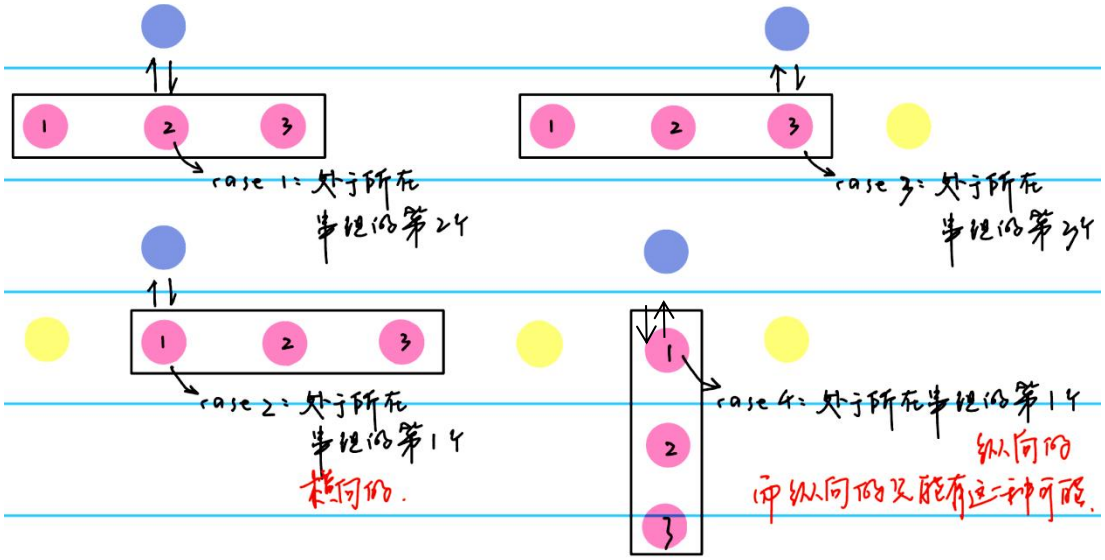
最后一个假设是写代码的时候需要考虑的边界球没有全部的移动方向的选择，同时要来考虑的是这个遍历范围实际上是原本大小的二维数组向外拓展两圈（如果遍历条件的i j还是从1到c/r的话）

如何解决边界球移动方向可能触发的越界访问数组空间的情况呢？（当时写的时候就在懊悔这个数组大小当时设置的时候没有考虑好，但是后面是这样解决的就是我在函数内部给设置一个转移的复写数组，这个数组更大！）这就有点像书法作品裱到一张更大的纸张上面好做展览的感觉。

如此就可以解决边界方向可以不用限制的问题，注意要在判断是否数字相同时候去除元素不可以为0，因为边界扩充的填补元素是0，不参与相消判断的。



初始情况 （举例为上下交换）



横向的三个情况和纵向的唯一情况

图 4 给出提示实现思路图

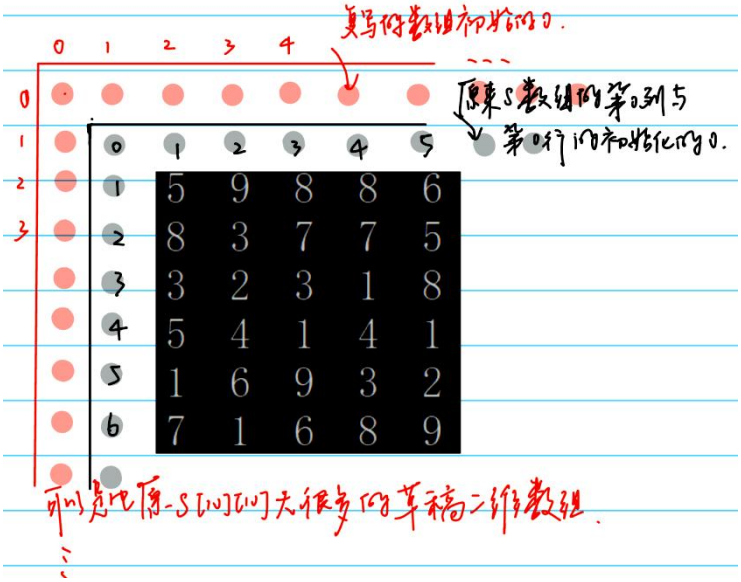


图 5 边界球方向限制的解决思路图

彩球下落：

这个无疑是我写的最痛苦的地方，也是我切切实实第一次体会到debug魅力的地方（第二次体会是实验课小测的第二道题）

在这里就先不写我第一版本的思路了简直就是与臭又长 毫无逻辑 东平西凑 惨不忍睹！最终的实现思路是：需要记录此时此刻需要下落球的能够下落的最低坐标，此时下落球的初始位置，already_up = 2 * r; 和 already_bottom = 2 * r。前者防止下落过度导致覆盖，后者记录哪些位置的球已经下落过了，不需要重复下落。

为什么需要注意重复下落呢？因为我这里的遍历是从底部向上遍历，如果上面三个都是空的，上面第四个不是空的球下落之后，i取什么呢？当时我继续i++来移动一个来遍历，那么就可能出现一个非空位置重复下落的情况，此时的s二维数组和h伴随数组都是没有归零的，还是原来填满的纯数字数组，所以就需要设置一个already_up参数来记录已经下落到哪一步了，但是后面我设计了这样一行来解决没有意义的对空白遍历的环节，不再局限于i++，而是直接跳转到最近一个下落球的位置继续向上遍历：这里i是数组行号，后者/2是从坐标换算为行号。

```
i = min(i, already_up / 2);
```

需要注意的几个问题，一是下边界不要覆盖的问，还有一个就是上边界不要超出，因为这里的下移动画效果是单独外包成一个小函数封装出去的：基本逻辑就是先下移动一行画一个相同的彩球→**把原来的彩球空格覆盖**→继续下移一行→把边框修补回来，所以在调用前需要判断该彩球是否是顶部的，防止动画越出。

```
void lan(int (*s)[10], int j, int w, int lie,
int hang, int helper_down)
    {
        int m = 5;
        int color[10] = { -1, 6, 7, 8, 9, 10,
11, 12, 13, 14 };
        cct_setcolor(color[s[w][j]], 0);
        cct_gotoxy(lie, hang);
        cout << "○";
        Sleep(m);
        if (hang - 1 > 0) {
            cct_setcolor(15, 0);
            cct_gotoxy(lie, hang - 1);
            cout << "  ";
            Sleep(m);
        }

        cct_setcolor(color[s[w][j]], 0);
        cct_gotoxy(lie, hang + 1);
        cout << "○";
        Sleep(m);
        cct_setcolor(15, 0);
        cct_gotoxy(lie, hang);
        cout << "=";
    }
```

写出来这个下落真是我最有成就感的一个但是是我死磕耗了一下午晚上三点钟气急败坏删掉重新写五分钟写出来的函数，悲喜交加！

游戏模式：

6.04写出来了，开心！

首先得到坐标，但是需要坐标组也就是一次需要传入两个坐标对进行互换，故设计了一个单双阀？就是一个计数器，单数就把此时左键鼠标事件传入x31和y31，双数就传入x32 y32中并触发交换判断：是否是原地点了两下；是否换了不可以造成消除对就是消除对扎堆出现选的两个的虽然相邻但是不是相互匹配的；不要跳跃互换，只能上下左右互换。同时要体验出选中的时圆圈从黑色变成白色，取消选择又变成黑色。判断流程和游戏模式的循环大概如下：

→ 此时是鼠标左键的第双数次触发，已经传入了合法的可以交换的两对坐标对

→ s交换两个数字
→ 扫描s查找可消除项的数量，来计算得分的增加
→ 除去零，彩球下落，空位补齐（这里要注意cct返回的x y是列和行 与s[i][j]的行列顺序不一样！）
→ 检查初始可消除项的循环
→ 重新标上提示符号（相当于重组了所以先前的提示可能就不好使了要remake）
→ 如果没有提示存在了就结束循环说明游戏结束进入死局
→ x31 y31 x32 y32 归零初始化，鼠标事件执行次数++
→ 进行下一次的鼠标左键事件第单次

由于需要判断是不是两个合法位置交换（选择错误是不匹配的合法位置的情况）比较方便的方法我想的是遍历h伴随二维矩阵遍历到等于1就分数+1同时可以设置一个返回值0/1如果一个1都没遍历到就说明选的不是匹配的交换项。

这个计算分数的函数嵌入在交换函数当中，其返回值在判断交换函数中如果是可以交换分数增加再触发实际的图形上的交换这里的交换没有延时，同样hint提示的重新绘制也是没有延时的需要重新加一个传入参数int option来说明option == 9时候Sleep(0)，其余的就是注意颜色要每次变换回来都要换回来，不然就出现下面的提示字五颜六色。

其他：

```
void enter_option(int option, int r) {
    cout << endl;
    if (option == 6 || option == 7 || option == 5)
        clear_instrcution(r);
    if (option == 1) {
        cout << "按回车键进行寻找初始可消除项的操作..." << endl; // 1
    }
    else if (option == 2 || option == 5) {
        cout << "按回车键进行数组下落除 0 操作..." << endl;
        if (option == 2)
            cout << "下落除 0 后的数组(不同色标识):" << endl;
    }
    else if (option == 3 || option == 6) {
        cout << "按回车键进行新值填充..." << endl;
        if (option == 3)
            cout << "新值填充后的数组(不同色标识):" << endl;
    }
    else if (option == 4) {
        cout << "按回车键显示图形..."<<endl;
    }
    else if (option == 7) {
        cout << "按回车键显示消除提示..." << endl;
    }

    int x;
    while (1)
    {
        x = _getch();
        if (x == 13)
            break;
    }
}
```

4. 调试过程碰到的问题

在写下落动画的时候，由于第一版本写的思路究极弯弯绕绕，只能 Debug 一步一步查，也感受到了监视窗口要设置 1 2 3 4 这么多的必要性！

```
void lan(int (*s)[10], int j, int w, int lie, int hang, int helper_down)
```

之所以给这个函数命名为 lan 是因为在写下落第一版错误思路函数 debug 的时候这个环节我基本上每次调试逐过程都不需要看这个下落具体单步动画效果的参数改变，加之这个环节确实没有参数的改变只有 cmd 窗口的图形覆盖变化，所以就封装出去了，称之为我懒得点击逐过程的懒函数。

当时在写 Debug 函数的时候还认为监视窗口和局部变量没有自动变量窗口万能，但还是能发现其显示的内容侧重不同。多几个监视窗口还是很有必要的！

鼠标悬浮处的 Debug 不太方便，或许是我没有掌握好方法，只能触发首次的断点之前的鼠标点击事件，之后在应该触发的时候就直接跳过了不能连续的读取新的点击处的坐标，我是通过打印的方式来替代 Debug 的，但是 Debug 还是太方便了，如果改动不是很大那么微小改动可以一边调试一边进行但还是会有前后执行遗漏的风险！

由于我解决不了 for 循环的逐语句的麻烦，所以就在初始化 fill 的时候固定一个地方是需要的数字来第一行或者第一列遍历的时候就触发。不过整个游戏还是需要多思考可能情况和多尝试来遍历所有可能的情况！调用了 demo 很多次，就是为了找到某些比较罕见的情况，demo 的示范处理是什么样子的！

5. 心得体会

项目完成时间线记录：

→ 5.26　13:00-17:00

→ 5.27　10:30-17:00 （此时完成选项 1 2 4 5 6， 考虑到选项三考虑项比较多准备晚上睡前先思考思考再去设计，希望这周可以想出一个七七八八，因为周六要出去看欧冠决赛加上计组 31 条 MIPS 指令的实验又发布了很抓狂!!）

→ 5.28　这天写了什么记不得了...可能去看数据库了

→ 5.29　08:30-10:00 和一下午实现找到提示环节，这个环节自己觉得自己思路还蛮合理的！

→ 5.30　很烦写了一整个大下午的下落环节，但是第一次的思路不对晚上全部推到重来五分钟解决...

→ 5.31　晚上胡写了鼠标悬浮

→ 6.01　11:00-12:30 解决昨晚的鼠标悬浮问题，调整了以下坐标位置，没有实现长按一直显示不可选择这个细节，看比赛前先把前八个的报告写了！

构思一下第九个选项的思路：交换位置 / 记录分数 / 判断持续执行到判断分数结束的 pipline

→ 6.04 写完全部！撒完结花！ 好像是有 BUG 的但是比较隐蔽

由于篇幅限制，主要的感想还是在感慨 Debug 的辅助作用的强大！其次是几个判断函数的设计动用的数学巧思(?)如果设计的真的有没有理论错误的话还是暗自有小有成就的，报告写在 2024 高考前，动用一些比较传统的数学思维解决问题的感觉很熟悉很怀念!! 这是转过来之后必经之路的第一个像模像样的大作业，以后还会有很多吧估计，继续努力！

6. 附件：源程序

以选项 9 为例部分源码：按照调用函数的顺序给出，部分片段没有缩进，因为保留全部缩进这个换行会很奇怪...

```
void h9(int s[10][10], int r, int c)
{
    //无可消除项，游戏结束!最终分数:51)
    // 记录分数

    int h[10][10] = { -12 };
    int helper_9 = 0;
    int helper_hint = 0;
    int x, y;
    initializa_input(s, r, c);
    fill(s, r, c);
    set_cmd(r + 8);
    helper_9 = check_1(s, r, c, h, 0);
    print_canvas(s, r, c, 1, 1);
    print_circle(s, r, c, 1, h, 1, 0);

    if (helper_9) {
```

```
        while (helper_9) {
            remove_circle(s, r, c, h);
            falling_down(s, r, c, h); // 下落动画
            remove_0(s, r, c, h); //把零取消 为填充提供识别项
            fill_new(s, r, c, h, 1);
            helper_9 = check_1(s, r, c, h, 0);
        }
    }
    helper_hint = hint(s, r, c, h, 1); // 无可消除项，游戏结束!
    if (!helper_9 && !helper_hint) {
        cct_getxy(x, y);
        cct_gotoxy(14, 0);
        cout << "（无可消除项，游戏结束!）";
        cct_gotoxy(x, y);
        return;
    }
    else {
        cct_getxy(x, y);
        cct_gotoxy(14, 0);
        cout << "（当前分数：0 右键退出）";
        cct_gotoxy(x, y);
        location(s, r, c, h, 9);
    }
}

void input(int *r, int *c)
void initializa_input(int s[10][10], int r, int c) // 数组定义必须是常数
void fill(int s[10][10], int r, int c)
void set_cmd(int r)
以上准备工作略

/*------------------------------ 查找区 ----------------------------------*/
int check_1(int (*s)[10], int r, int c, int (*h)[10], int option)
{
     int helper = 0;
     int helper_check_1 = 0;
     for (int i = 1; i <= r; i++) {
          for (int j = 1; j <= c; j++) {
               if (s[i][j] == s[i][j + 1] && s[i][j] == s[i][j + 2] && s[i][j] != -1) {
                    h[i][j] = h[i][j + 1] = h[i][j + 2] = 1;
                    helper = 1;
               }
          }
     }
     for (int j = 1; j <= c; j++) {
          for (int i = 1; i <= r; i++) {
               if (s[i][j] == s[i + 1][j] && s[i][j] == s[i + 2][j] && s[i][j] != -1) {
                    h[i][j] = h[i + 1][j] = h[i + 2][j] = 1;
                    helper = 1;
               }
          }
     }
     helper_check_1 = helper;
     if (option != 9) {
          if (helper == 0) {
               cout << "初始已无可消除项" << endl;
               return helper_check_1;
          }
          else {
               cout << "初始可消除项（不同色标识）：" << endl; // 1
          }
     }

     return helper_check_1;
}

void remove_0(int (*s)[10], int r, int c, int(*h)[10])
{
     for (int i = 1; i <= r; i++) {
          for (int j = 1; j <= c; j++) {
               if (h[i][j] == 1) {
                    s[i][j] = 0;
                    h[i][j] = 0; //初始化为后面零的位置做记号做准备
               }
          }
     }

     for (int j = 1; j <= c; j++) {
          for (int i = r; i > 0; i--) {
               if (s[i][j] == 0) {
                    for (int w = i; w > 1; w--) {
                         s[w][j] = s[w - 1][j];
                    }
                    s[1][j] = 0;
                    int helper_i = 0;
                    for (int w = i; w > 0; w--) {
                         if (s[w][j] == 0) {
                              helper_i = 0;
                         }
                         else {
                              helper_i = 1;
                              break;
                         }
                    }
                    if (helper_i) {
                         i++;
                    }
               }
          }
     }
}

/*------------------------------ 填补区 ----------------------------------*/
void fill_new(int (*s)[10], int r, int c, int(*h)[10], int print_new_circle)
{
     int color[10] = { -1, 6, 7, 8, 9, 10, 11, 12, 13, 14 };
     srand((unsigned)time(NULL));
     for (int i = 1; i <= r; i++) {
          for (int j = 1; j <= c; j++) {
               if (s[i][j] == 0) {
                    //srand((unsigned)time(NULL));
                    s[i][j] = rand() % 9 + 1;
                    if (print_new_circle) {
                         cct_gotoxy(4 * j - 2, 2 * i);
                         cct_setcolor(color[s[i][j]], 0);
                         cout << "o"; // ●
```

```
                    Sleep(200);
                }
            }
        }
    }
    if (print_new_circle) {
        cct_setcolor(0);
        cct_gotoxy(0, 2 * r + 2);
        cout << "                              ";
        cct_gotoxy(0, 2 * r + 2);
    }

}

/*------------------------------ 提示区
---------------------------------*/
int hint(int(*s)[10], int r, int c, int(*h)[10], int
print_hint_circle)
{
    int ts = 0;
    int helper_hint = 0;
    // 还用 h 会不会冲突 先用着吧
    // 很烦这个有边界限制
    // 我能不能读到另外一个二位数组上
    int ds[20][20] = { 0 };
    for (int i = 0; i < 20; i++) {
        for (int j = 0; j < 20; j++) {
            ds[i][j] = 0;
        }
    }
    for (int i = 1; i <= r; i++) {
        for (int j = 1; j <= c; j++) {
            ds[i + 1][j + 1] = s[i][j];
            h[i][j] = 0;
        }
    }

    for (int i = 2; i <= r; i++) {
        for (int j = 2; j <= c; j++) {
            // up
            if (ds[i - 1][j]) {
                ts = ds[i - 1][j];
                ds[i - 1][j] = ds[i][j];
                ds[i][j] = ts;
                if ((ds[i][j] == ds[i][j + 1] &&
ds[i][j] == ds[i][j - 1])
                    || (ds[i][j] == ds[i][j + 1] &&
ds[i][j] == ds[i][j + 2])
                    || (ds[i][j] == ds[i][j - 1] &&
ds[i][j] == ds[i][j - 2])
                    || (ds[i][j] == ds[i + 1][j] &&
ds[i][j] == ds[i + 2][j]) && ds[i][j] != 0) {
                    // 标记加还原
                    h[i - 1][j - 1] = 1;
                    h[i - 2][j - 1] = 1;
                    helper_hint = 1;
                }
                ts = ds[i - 1][j];
                ds[i - 1][j] = ds[i][j];
                ds[i][j] = ts;
            }

            // down
            if (ds[i + 1][j]) {
                ts = ds[i + 1][j];
                ds[i + 1][j] = ds[i][j];
                ds[i][j] = ts;
                if ((ds[i][j] == ds[i][j + 1] &&
ds[i][j] == ds[i][j - 1])
                    || (ds[i][j] == ds[i][j + 1] &&
ds[i][j] == ds[i][j + 2])
                    || (ds[i][j] == ds[i][j - 1] &&
ds[i][j] == ds[i][j - 2])
                    || (ds[i][j] == ds[i - 1][j] &&
ds[i][j] == ds[i - 2][j]) && ds[i][j]) {
                    // 标记加还原
                    h[i - 1][j - 1] = 1;
                    h[i][j - 1] = 1;
                    helper_hint = 1;
                }
                ts = ds[i + 1][j];
                ds[i + 1][j] = ds[i][j];
                ds[i][j] = ts;
            }

            // left
            if (ds[i][j - 1]) {
                ts = ds[i][j - 1];
                ds[i][j - 1] = ds[i][j];
                ds[i][j] = ts;
                if ((ds[i][j] == ds[i + 1][j] &&
ds[i][j] == ds[i - 1][j])
                    || (ds[i][j] == ds[i - 1][j] &&
ds[i][j] == ds[i - 2][j])
                    || (ds[i][j] == ds[i + 1][j] &&
ds[i][j] == ds[i + 2][j])
                    || (ds[i][j] == ds[i][j + 1] &&
ds[i][j] == ds[i][j + 2]) && ds[i][j]) {
                    // 标记加还原
                    h[i - 1][j - 1] = 1;
                    h[i - 1][j - 2] = 1;
                    helper_hint = 1;
                }
                ts = ds[i][j - 1];
                ds[i][j - 1] = ds[i][j];
                ds[i][j] = ts;
            }

            // right
            if (ds[i][j + 1]) {
                ts = ds[i][j + 1];
                ds[i][j + 1] = ds[i][j];
                ds[i][j] = ts;
                if ((ds[i][j] == ds[i + 1][j] &&
ds[i][j] == ds[i - 1][j])
                    || (ds[i][j] == ds[i - 1][j] &&
ds[i][j] == ds[i - 2][j])
                    || (ds[i][j] == ds[i + 1][j] &&
ds[i][j] == ds[i + 2][j])
                    || (ds[i][j] == ds[i][j - 1] &&
ds[i][j] == ds[i][j - 2]) && ds[i][j]) {
                    //// 标记加还原
                    h[i - 1][j - 1] = 1;
                    h[i - 1][j] = 1;
                    helper_hint = 1;
                }
                ts = ds[i][j + 1];
                ds[i][j + 1] = ds[i][j];
                ds[i][j] = ts;
            }
```

```
          }
     }

     if (print_hint_circle) {
          int color[10] = { -1, 6, 7, 8, 9, 10, 11, 12, 13,
14 };
          for (int i = 1; i <= r; i++) {
               for (int j = 1; j <= c; j++) {
                    if (h[i][j] == 1) {
                         cct_gotoxy(4 * j - 2, 2 * i);
                         cct_setcolor(color[s[i][j]],
0);
                         cout << "◎"; // ●◎
                    }
               }
          }
          clear_instrcution(r);
     }
     return helper_hint;
}

void print_canvas(int (*s)[10], int r, int c, int grid, int
speed)
void print_circle(int(*s)[10], int r, int c, int grid, int
(*h)[10], int speed, int option)

void remove_circle(int(*s)[10], int r, int c, int(*h)[10])
{
     // 通过反复尝试找到了终于可以证明 demo 是横向一个一个扫
描消除的
     int x, y;
     int m = 200;
     int color[10] = { -1, 6, 7, 8, 9, 10, 11, 12, 13, 14 };
     for (int i = 1; i <= r; i++) {
          cct_gotoxy(0, 2 * i);
          for (int j = 1; j <= c; j++) {
               cct_setcolor(color[s[i][j]], 0);
               cct_getxy(x, y);
               cct_gotoxy(x + 2, y);
               if (h[i][j] == 1) {
                    cout << "●"; // ● ¤ ◎
                    cct_gotoxy(x + 2, y);
                    Sleep(m / 2);
                    cout << "o"; // ● ¤ ◎
                    cct_gotoxy(x + 2, y);
                    Sleep(m/2);
                    cout << "¤"; // ● ¤ ◎
                    cct_gotoxy(x + 2, y);
                    Sleep(m/2);
                    cout << "o"; // ● ¤ ◎
                    cct_gotoxy(x + 2, y);
                    Sleep(m/2);
                    cout << "¤"; // ● ¤ ◎
                    cct_setcolor(15, 0);
                    cct_gotoxy(x + 2, y);
                    cout << "  ";
               }
               else {
                    cct_gotoxy(x + 4, y);
               }
          }
     }
     clear_instrcution(r);
}

void falling_down(int(*s)[10], int r, int c, int(*h)[10])
{
     int x, y, w;
     int lie, hang = 2 * r;
     int helper_down; // 已经下落的数量
     int already_up = 2 * r; // 最近一次下落的坐标
     int already_bottom = 2 * r; // 最后一次到底部的纵坐标
和上面的不一样
     int step = 0; // 需要下降的高差
     int m = 10;
     int color[10] = { -1, 6, 7, 8, 9, 10, 11, 12, 13, 14 };
     cct_gotoxy(1, 2 * r);
     for (int j = 1; j <= c; j++) {
          already_up = 2 * r;
          already_bottom = 2 * r;
          helper_down = 0;
          lie = j * 4 - 2;
          cct_gotoxy(lie, 2 * r); //到达底部往上遍历
          for (int i = r; i > 0; i--) {
               if (h[i][j] == 1) {
                    for (w = i; w > 0; w--) {
                         if (h[w][j] == 1) {
                              step++;
                         }
                         else {
                              break;
                         }
                    }
                    // 此时 w 是下移的彩球序号
                    hang = w * 2 + 1;
                    already_up = 2 * w;
                    for (int step_i = 1; step_i < 10;
step++) {
                         if (hang < already_bottom &&
hang > 1) {
                              lan(s, j, w, lie, hang,
helper_down);
                              hang += 2;
                              helper_down = 1;
                         }
                         else {
                              cct_getxy(x, y);
                              already_bottom = min(y,
already_bottom);
                              break;
                         }
                    }
               }
               else if (helper_down) {
                    hang = i * 2 + 1;
                    already_up = i * 2;
                    for (int step_i = 1; step_i < 10;
step++) {
                         if (hang < already_bottom &&
hang > 1) {
                              lan(s, j, i, lie, hang,
helper_down);
                              hang += 2;
                         }
                         else {
                              cct_getxy(x, y);
                              already_bottom = min(y,
already_bottom);
                              break;
                         }
```

```
                    }
               }
               else {
                    already_bottom -= 2;
               }
               i = min(i, already_up / 2);
          }
     }
     clear_instrcution(r);
}

void lan(int (*s)[10], int j, int w, int lie, int hang, int
helper_down)
{
     int m = 5;
     int color[10] = { -1, 6, 7, 8, 9, 10, 11, 12, 13, 14 };
     cct_setcolor(color[s[w][j]], 0);
     cct_gotoxy(lie, hang);
     cout << "o";
     Sleep(m);
     if (hang - 1 > 0) {
          cct_setcolor(15, 0);
          cct_gotoxy(lie, hang - 1);
          cout << "  ";
          Sleep(m);
     }

     cct_setcolor(color[s[w][j]], 0);
     cct_gotoxy(lie, hang + 1);
     cout << "o";
     Sleep(m);
     cct_setcolor(15, 0);
     cct_gotoxy(lie, hang);
     cout << "=";
}

/*------------------------------ 悬浮区
---------------------------------*/
     int location(int(*s)[10], int r, int c, int(*h)[10],
int option)
     {
     cct_gotoxy(0, r * 2 + 2);
     cout << "[当前光标]";
     int color[10] = { -1, 6, 7, 8, 9, 10, 11, 12, 13, 14 };
     int x = 0, y = 0;
     int x1 = 0, y1 = 0;
     int x3 = 0, y3 = 0;
     int game_choice = 0;
     int x31 = 0, y31 = 0;
     int x32 = 0, y32 = 0;
     cct_getxy(x1, y1);
     int ret, maction;
     int keycode1, keycode2;
     int loop = 1;
     int SCORE = 0;
     int check_swap = 0;
     cct_enable_mouse();
     cct_setcursor(CURSOR_INVISIBLE);
     while (loop) {
     cct_gotoxy(0, r * 2 + 2);
     cout << "[当前光标]";
     /* 读鼠标/键盘，返回值为下述操作中的某一种，当前鼠标位置
在<x,y>处 */
     ret = cct_read_keyboard_and_mouse(x, y, maction,
keycode1, keycode2);
     if (ret == CCT_MOUSE_EVENT) {
     cct_gotoxy(x1, y1);
     cout << "                                    ";
     if (y % 2 == 0 && (x % 4 == 2 || x % 4 == 3)) {
     if (x % 4 == 3) {
     x3 = (x + 1) / 4;
     x = x - 1;
     }
     else
     x3 = (x + 2) / 4;
     y3 = y / 2;

     if (y3 <= r && x3 <= c) {
     cct_gotoxy(x1, y1);
     cout << setw(2) << char(y3 + 'A' - 1) << "行" << setw(2)
<< x3 << "列";
     }
     else {
     cct_gotoxy(x1, y1);
     cout << "位置非法";
     }
     }
     else {
     cct_gotoxy(x1, y1);
     cout << "位置非法";
     }

     switch (maction) {
     case MOUSE_ONLY_MOVED:
     break;
     case MOUSE_LEFT_BUTTON_CLICK:
     ret = cct_read_keyboard_and_mouse(x, y, maction,
keycode1, keycode2);
     if (x % 4 == 3) {
     x3 = (x + 1) / 4;
     x = x - 1;
     }
     else
     x3 = (x + 2) / 4;
     y3 = y / 2;
     // 选中    //当前选择 H 行 3 列  //不能选择 I 行 4 列
     if (y % 2 == 0 && (x % 4 == 2 || x % 4 == 3) && (y3
<= r && x3 <= c)) {
     if (h[y3][x3] == 1) {
     cct_gotoxy(0, r * 2 + 2);
     cout << "                                    ";
     cct_gotoxy(0, r * 2 + 2);
     cout << "当前选择" << setw(4) << char(y3 + 'A' - 1) <<
"行" << setw(2) << x3 << "列";
     if (option == 8) {
     Sleep(400);
     cct_gotoxy(0, r * 2 + 2);
     cout << "                                    ";
     cct_gotoxy(0, r * 2 + 2);
     return -2;
     }
     if (option == 9) {
     game_choice++;
     if (game_choice % 2 == 1) {
     x31 = x3;
     y31 = y3;
     }
     if (game_choice % 2 == 0) {
     x32 = x3;
     y32 = y3;
```

```
	}
	if (x31 != x32 && y32 != y31) {
	cct_gotoxy(x, y);
	cct_setcolor(color[s[y3][x3]], 15);
	cout << "◎"; // ●
	cct_setcolor(0);
	}
	else if (x31 == x32 && y32 == y31) {
	cct_gotoxy(x, y);
	cct_setcolor(color[s[y3][x3]], 0);
	cout << "◎"; // ●
	cct_setcolor(0);
	x31 = 0;
	y31 = 0;
	x32 = 0;
	y32 = 0;
	break;
	}

	if (game_choice % 2 == 0) {
	check_swap = game_swap(s, r, c, h, x31, y31, x32, y32,
&SCORE);
	if (!check_swap) {
	cct_gotoxy(0, r * 2 + 2);
	cout << "                                    ";
	cct_gotoxy(0, r * 2 + 2);
	cout << "不能交换                    ";
	Sleep(400);
	}
	else {
	int helper_9_game = check_1(s, r, c, h, 9);
	if (helper_9_game) {
	while (helper_9_game) {
	remove_circle(s, r, c, h);
	falling_down(s, r, c, h); // 下落动画
	remove_0(s, r, c, h); //把零取消 为填充提供识别项
	fill_new(s, r, c, h, 1);
	helper_9_game = check_1(s, r, c, h, 9);
	}
	}
	print_circle(s, r, c, 1, h, 1, 9);
	int helper_hint_game = hint(s, r, c, h, 1); // 无可
消除项，游戏结束!
	if (!helper_9_game && !helper_hint_game) {
	cct_gotoxy(0, 0); //无可消除项，游戏结束!最终分数:3)
	cout << "无可消除项，游戏结束!最终分数:" << SCORE << ")
";
	cct_gotoxy(0, r * 2 + 2);
	cout << "                                    ";
	cct_gotoxy(0, r * 2 + 3);

	return -4;
	}
	else {
	cct_gotoxy(14, 0);
	cout << "（当前分数：" << SCORE << " 右键退出）";
	}
	}
	x31 = 0;
	y31 = 0;
	x32 = 0;
	y32 = 0;
	}
	}
	}
	else {
	cct_gotoxy(0, r * 2 + 2);
	cout << "                                    ";
	cct_gotoxy(0, r * 2 + 2);
	cout << "不能选择" << setw(4) << char(y3 + 'A' - 1) <<
"行" << setw(2) << x3 << "列";
	Sleep(180); // 如何实现选中持续显示呢？
	}
	}
	break;
	case MOUSE_RIGHT_BUTTON_CLICK:
	if (y % 2 == 0 && (x % 4 == 2 || x % 4 == 3) && (y3
<= r && x3 <= c)) {
	loop = 0; // 右键点击退出循环
	cct_gotoxy(0, r * 2 + 2);
	cout << "                                    ";
	cct_gotoxy(0, r * 2 + 2);
	return -1;
	break;
	}
	break;
	default:
	break;
	}
	}
	}
	return 0;
	}

	/*------------------------------ 游戏区
----------------------------------*/
	int game_score(int(*s)[10], int r, int c, int(*h)[10],
int* SCORE)
	{
	int check_swap = 0;
	for (int i = 1; i <= r; i++) {
	for (int j = 1; j <= c; j++) {
	h[i][j] = 0;
	}
	}

	check_1(s, r, c, h, 9);
	for (int i = 1; i <= r; i++) {
	for (int j = 1; j <= c; j++) {
	if (h[i][j] == 1) {
	check_swap = 1;
	(*SCORE)++;
	}
	}
	}
	return check_swap;
	}

	int game_swap(int(*s)[10], int r, int c, int(*h)[10],
int x31, int y31, int x32, int y32, int* SCORE)
	{
	int color[10] = { -1, 6, 7, 8, 9, 10, 11, 12, 13, 14 };
	int x1, y1, x2, y2;
	y1 = 2 * y31;
	y2 = 2 * y32;
	x1 = 4 * x31 - 2;
	x2 = 4 * x32 - 2;
	if (abs(x31 - x32) == 1 || abs(y31 - y32) == 1) {
	if (!(abs(x31 - x32) == 1 && abs(y31 - y32))) {
	int ts;
```

```
ts = s[y31][x31];
s[y31][x31] = s[y32][x32];
s[y32][x32] = ts;

int check_swap = game_score(s, r, c, h, SCORE);
if (!check_swap) {
ts = s[y31][x31];
s[y31][x31] = s[y32][x32];
s[y32][x32] = ts;
cct_gotoxy(x1, y1);
cct_setcolor(color[s[y31][x31]], 0);
// ○
cout << "◎"; // ●
cct_gotoxy(x2, y2);
cct_setcolor(color[s[y32][x32]], 0);
cout << "◎"; // ●
cct_setcolor(0);

return 0;
}

// 绘制交换
    cct_gotoxy(x1, y1);
    cct_setcolor(color[s[y31][x31]], 0);
    cout << "o"; // ●
    cct_gotoxy(x2, y2);
    cct_setcolor(color[s[y32][x32]], 0);
    cout << "o"; // ●
    cct_setcolor(0);

    return 1;
    }
    }
    cct_gotoxy(x2, y2);
    cct_setcolor(color[s[y32][x32]], 0);
    cout << "◎"; // ●
    cct_gotoxy(x1, y1);
    cct_setcolor(color[s[y31][x31]], 0);
    cout << "◎"; // ●
    cct_setcolor(0);

    return 0;
}
```