同济大学

实验报告

“合成十”小游戏

班级：信息类11班

学号：2151294

姓名：马威

完成日期：2021.12.28

1. 题目：“合成十”的实现

1.1.总体玩法

1.初始在一个M*N的区域内产生随机值1-3，选定一个坐标进行第2项及以后的操作（这里M、N可手动调整）。

2.找到与该坐标相邻的所有相同值（不只是该坐标周围一圈中的元素，而是包括该坐标在内的连成一整片的所有相同值）。

3.将所有找到的相同值合并，指定位置+1，其余位置消除为0

4.每一列的值跳过0下落，让0元素留在最上方

5.所有为零的位置随机产生新值（概率按当前数组的最大值确定）

6.重复2-5项，直到某个位置的元素达到合成目标。

7.第6项达成后游戏不结束，提示继续向更高目标进发，每一次达到更高目标提示一次。

8.重复上述所有操作（除第1项），直到所有元素都不能合成，则提示游戏失败。

1.2.分数累加规则

每次新增得分=本次消除值*本次消除数量*3

1.3.子题目1：命令行方式找出可合并项并标识（非递归）

1.键盘输入行列数，产生随机数组，然后输入要合并的行列坐标，找出并标记所有可合成项

2.本小题中，找可合成项用一个非递归函数完成

1.4.子题目2：命令行方式找出可合并项并标识（递归）

本小题中，找可合成项用一个递归函数完成，其余要求同子题目1

1.5.子题目3：命令行方式完成一次合成（分步骤显示）

1.找出可合成项后，用指令确定下一步行动。

“Y”：继续完成合并

“N”：放弃本次操作，重新输入坐标并继续

“Q”：放弃本次游戏

2.若某次合并后无可合成项，则提示游戏结束

3.若选择完成合并，逐步打印可合成项查找结果、完成合并、计算得分、下落0、产生新值

1.6.子题目4：命令行完整版（分步骤显示）

1.完成多次合成的操作，每次合成要求同子题目3，且需要分步演示

2.某个位置的元素达到合成目标后，给出提示并继续游戏。

1.7.子题目5：伪图形界面显示初始数组（无分隔线）

1.根据键盘输入的行列数，在cmd窗口中以M*N个色块形式显示内部数组值

2.窗口大小需随着行列数动态变化

3.画图过程中适当加上延时，看清楚制图的整个过程

1.8.子题目6：伪图形界面显示初始数组（有分隔线）

色块间需要有分隔线，其余要求同子题目5

1.9.子题目7：伪图形界面下用箭头键/鼠标选择当前色块

1.在子题目6基础上，用键盘或鼠标实现选择色块

2.键盘使用箭头键移动（初始位置为左上角、边界回绕），回车键确认

3.支持鼠标选择（当前位置信息实时变化），左键确认

1.10.子题目8：伪图形界面完成一次合成（分步骤）

1.在子题目7的基础上，选定一个色块，标出所有可合成项

2.选定后，鼠标移动/按箭头键取消本次选定，重新选择。鼠标左键/按回车键则合成

3.逐步显示消除可合并项、下落0、产生新色块、计算得分等操作，下落0必须有动画效果

1.11.子题目9：伪图形界面完整版（支持鼠标）

1、完成多次合成的操作，确定合成后的后续操作不需逐步显示，但需要有动画效果

2.某个位置的元素达到合成目标后，给出提示并继续游戏

3.窗口上下有一个状态栏，显示得分、合成目标、操作提示等

1.12.子题目A：命令行界面（网络自动解-基本版）

1.以网络交互的形式完成一次游戏

2.不同于单机版的手动选择坐标，本小题的每次合成中，需按某种算法找出你认为最优解的坐标并上传给服务器

3.结束条件：无可合并项或合成到16

1.13.子题目B：伪图形界面（网络自动解-基本版）

在子题目A基础上，将交互过程展示为伪图形界面下的色块移动

1.14.退出

按下数字0能直接退出程序

2. 整体设计思路

1、整个程序以一个永真循环为背景，通过菜单函数来控制菜单选择。

2、若菜单项为零，直接跳出循环，程序结束。

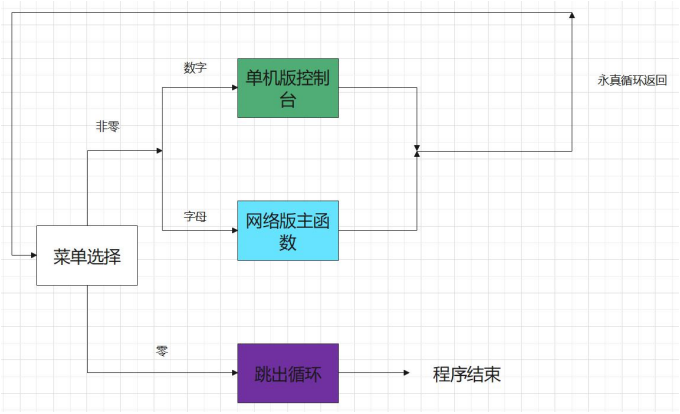
3、若菜单项不为零，调用一个总的输入函数（包含正确性处理的函数），输入信息后再调用一个总控函数（单机版和网络版各一个，里面根据菜单项选择调用不同函数，作用相当于一个控制台）实现主要功能。

4、单机版中各菜单项的实现，很多项都是在前一个项完成后的延伸，代码（尤其是每一项刚开始的代码）重复率接近100%，体现出很强的递进性。所以总控函数应当以循环为背景，设置多个循环结束点，一旦达到符合当前菜单项的要求便从此结束。

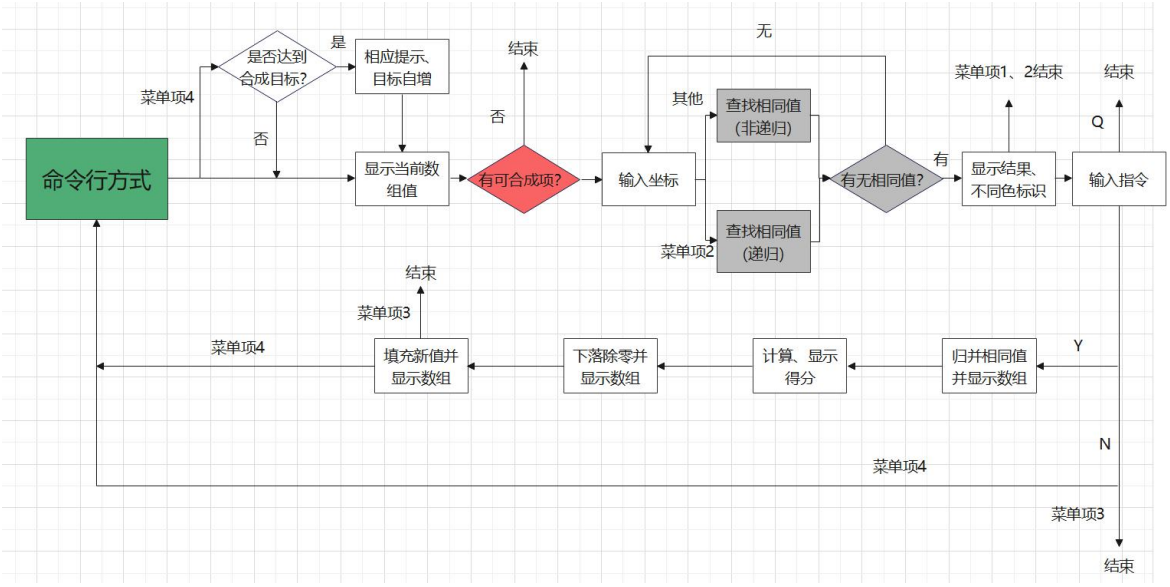
5、根据不同的主要功能，有针对性地写出完成各种功能的函数。

3. 主要功能的实现

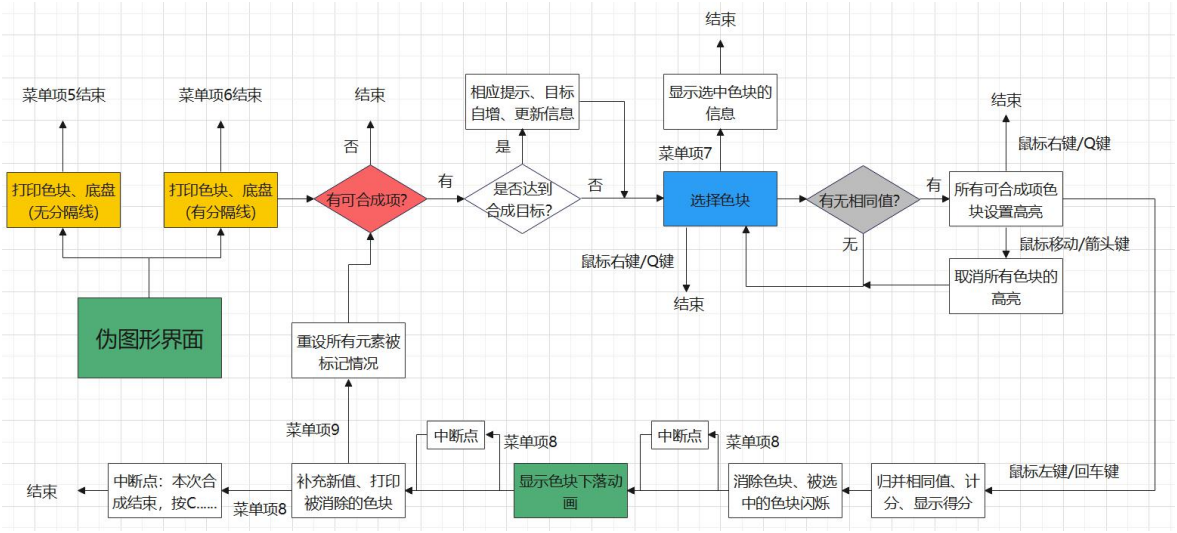
3.1.主函数



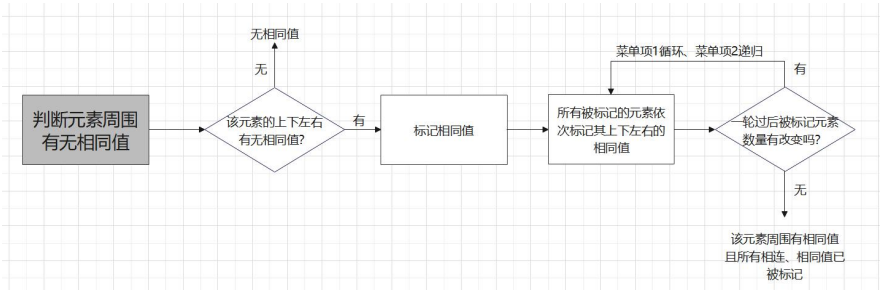
3.2. 命令行方式总体



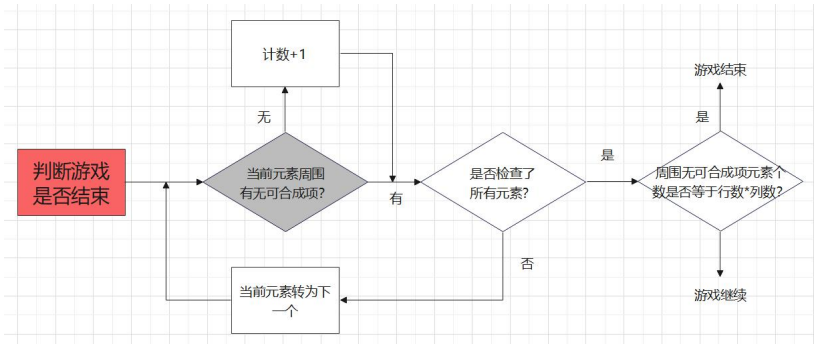
3.3. 伪图形界面总体



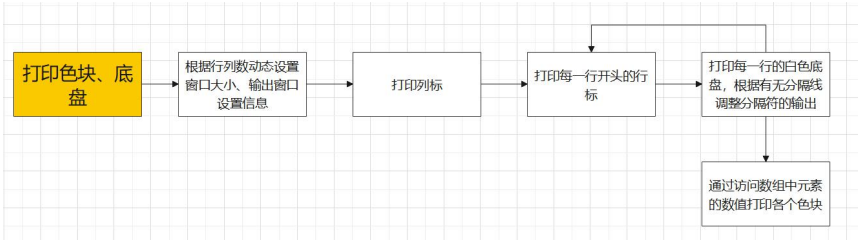
3.4. 判断元素周围有无相同值



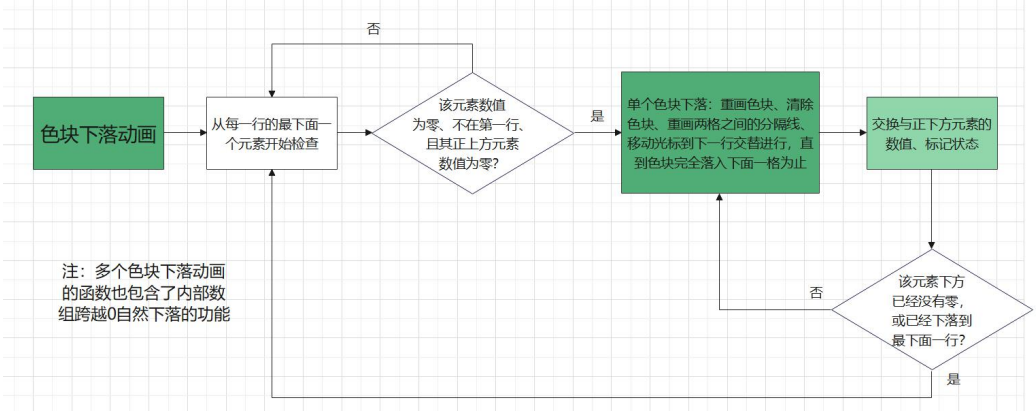
3.5. 判断游戏是否结束



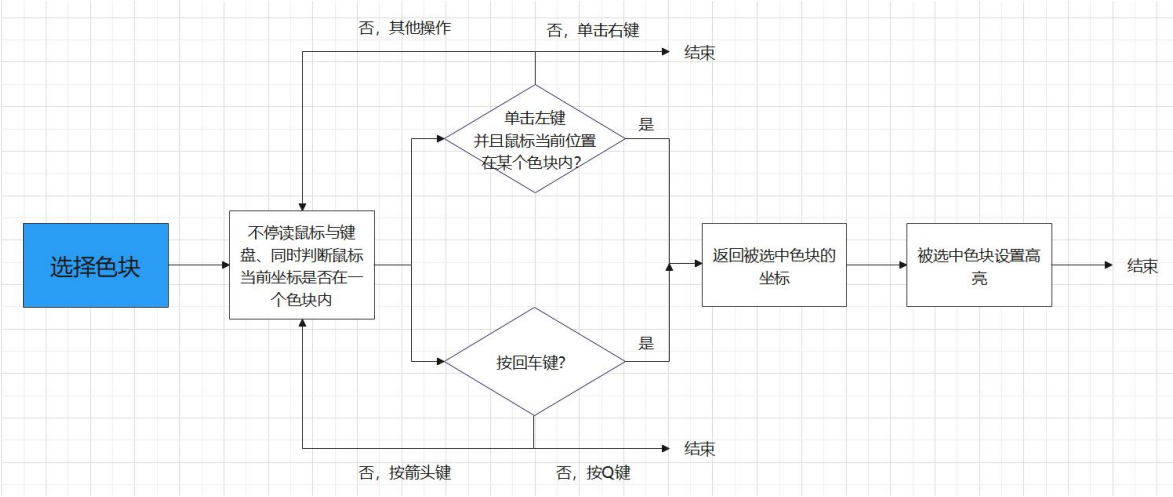
3.6. 打印色块、底盘



3.7. 显示色块下落动画



3.8.键盘/鼠标选择色块



4. 调试过程碰到的问题

问题：写子题目8的时候，第一次选定色块后（只单击了一次鼠标左键），所有被标记的色块又立刻自动恢复原样，无法进入下一步的确认是否继续合成。

原因：标记所有可合成色块的函数延时过短，在鼠标左键抬起前已经运行完毕。导致此时程序确实已经执行到下一步的读鼠标/键盘（原本的功能是确认是否继续合成），结果读到了鼠标左键的抬起而判断为不继续合成，最终取消所有色块的标记而回到色块选择的步骤。

解决：按每次可合成色块数量控制标记函数的延时，色块数量较少则延时加长，保证在鼠标左键抬起后才执行完毕，抬起动作不被第二次读鼠标的函数读到，顺利进入下一环节。同时色块数量较多时延时不能太长，否则影响观感。

5. 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.本次作业很多子题目都是前一个子题目的延续，对前一个子题目的代码利用率很高，接近完全。但写下来发现整个过程中，代码难免会显得杂乱（一会调用自己写的函数，一会cout，一会移动光标.....），而且有些代码块虽然相同，但执行次数完全不同（如子题目3只需分步演示1次，但子题目4要执行多次）而导致外套循环情况也不尽相同。

2.从3可以看出，第一次写选择调用不同函数的代码时，若过早地规划这段代码中的循环位置、间断点位置、各子题目的结束点等，反而会让问题更加的复杂化。不如先按顺序结构或仅有一层的循环结构，按各个子题目的流程写出每个小题的执行过程，完成各个小题后再提炼出各小题的相同或相似的部分即可。

3.本次作业中涉及到许多用参数区分差异的地方，如命令行方式下输出每一次的查找结果数组与当前数组、伪图形界面下显示内部数组的有无分隔线等，这些功能的实现表面上看差别很大，但实际上本质几乎是一样的。应当用同一个函数，设置其形参的不同以至于一个函数就能实现多

个类似功能。

4.对于5，有的时候可以设形参为某个函数的某种模式，调用时实参可以为某种宏定义，这样调用时就能清晰地认识到当前函数实现的是诸多类似功能中的哪一种。

5.2.分为若干小题的方式是否对你的设计起到了一定的提示作用？

是的。分为若干小题可以让我更为直观地了解一个功能繁多的程序是由哪几个部分组成，在设计时可以按照这样的顺序将各类实现逐个击破。换言之，分为若干小题给到了题目的切入点以及完成了题目的分解，降低了解决问题的难度，对程序设计起到了很大的提示作用。

5.3.与汉诺塔相比，你在函数的分解上是否做到了更合理？介绍一下你的几个重点函数的分类方法及使用情况（特别是涉及到用参数区分差异的部分）

是的。随着作业变得愈发复杂，需实现的功能也逐渐变多，在函数上的划分也就要越合理，才能在编写程序的过程中保持思路的清晰，使程序的逻辑性更强。否则就会导致自己都不会用自己写的函数的尴尬境地，程序也会显得杂乱无章。

1、伪图形界面下的打印白色背景的每一行

介绍：由于每一行使用到的制表符不同，故可以通过设置多个参数，用同一个函数来实现多种行的打印。设置的参数有：第一个制表符、中间的特殊制表符、中间的一般制表符、最后一个制表符。

使用：根据不同行的样式改变实参。

若要打印这种行：

则四个实参为“ ┣”、“╋”、“━”、“┫ ”

若要打印这种行：

则四个实参为“ ┏”、“┳”、“━”、“┓ ”

若要打印这种行：

则四个实参为“ ┃”、“ ┃”、“ ”、“ ┃”

无分隔线情况下，设置第二个实参对应的制表符不输出。

2、命令行方式下的显示数组情况

介绍：由于程序需要实现显示当前数组值以及显示查找数组结果的两个频繁交替使用的功能，故可以在显示数组值的函数中设置一个表示打印模式的形参，通过其决定显示数组值的方式。

使用：根据不同的显示方式改变实参。

若要显示查找数组结果，将宏定义ARRAY_SHOW_RESULT作为实参，则遇到被标记的元素时输出星号，其余元素输出0。

若要显示当前数组值，将宏定义ARRAY_SHOW_CURRENT作为实参，则打印所有元素的数值，被标记的还可用不同色标识。

3、伪图形界面下特殊提示信息的输出

介绍：伪图形界面下进行游戏时会遇到多种事件需要给出提示，例如操作说明等。每一种提示内容都不尽相同，但输出位置是一致的。故可以在显示特殊提示的函数中设置一个表示提示内容的形参，通过其决定输出的内容。

使用：根据不同的提示内容改变实参。

若达到了合成目标，将宏定义INFO_GOAL_ACHIEVED作为实参，则输出“已经合成到5（举例），按回车键/单击左键继续向更高目标进发”。

若已经无可合成的项且游戏结束，将宏定义INFO_END作为实参，则输出“无可合并的项，游戏结束!按Q退出”。

若当前选定色块周围无相同值，将宏定义INFO_NO_CLEAR作为实参，则输出“周围无相同值，箭头键/鼠标移动，回车键/单击左键选择，Q/单击右键结束”。

5.4.以本次作业为例，谈谈编写一个相对复杂的程序的心得体会

1.总体思路要清晰。什么时候调用什么函数、各类功能的实现步骤、每个菜单项开始、结束的时间点.....这些问题在编写程序时要随时思考，增强源程序的逻辑性。当然，思考这些问题的前提是把题目化整为零，化大为小，从小切口进入。本次作业中，将一个相对复杂的程序划分为若干个子题目就是一个非常好的例子。

2.函数划分要合理。一个函数就实现某一特定的功能，不要一股脑地往下写其他功能的实现，否则后续调用时会让自己手忙脚乱。例如本次作业中的伪图形界面下键盘/鼠标选择色块，该函数只要选择、标记、色块设置高亮即可，要把查找结果、所有相同值色块设置高亮交给其他函数处理，否则调用时自己一定会忘记这些“多余”功能，从而造成不符合预期的结果。

6. 附件：源程序

1.90-b2-base.cpp

```
void array_initialize(MATRIX matrix[][ROW_MAX], const int line, const int row)
{
	int i, j;  /*计数变量*/

	for (i = 0; i < line; i++) {
		for (j = 0; j < row; j++) {
			matrix[i][j].num = rand() % 3 + 1;  /*随机赋值*/
			matrix[i][j].mark = 0;              /*标记为零*/
		}
	}
}

void array_show(MATRIX matrix[][ROW_MAX], const int line, const int row, const int show_mode)
{
	int i, j;  /*计数变量*/
	int X, Y;  /*横纵坐标*/
	cout << resetiosflags(ios::right);  /*解除输出右对齐*/

	/*1、输出首行的列标记（数字）*/
	cout << "  |  ";
	for (i = 0; i < row; i++)
		cout << setw(3) << setiosflags(ios::left) << i;

	/*2、输出第二行的分割线*/
	cout << endl << "--+--";
	for (i = 0; i < 3 * row - 1; i++)
		cout << "-";
	cout << endl;

	/*3、逐行输出元素*/
	for (i = 0; i < line; i++) {

		/*3.1、输出首列的行标记（字母）及分割线*/
		cout << char('A' + i) << "  |  ";

		/*3.2、同一行内逐个输出元素*/
		for (j = 0; j < row; j++) {

			/*3.2.1、若为被标记元素*/
			if (matrix[i][j].mark) {
				if (show_mode == ARRAY_SHOW_CURRENT) {  /*输出当前数组模式：改变颜色后输出*/
					cct_getxy(X, Y);
```

```
                                        cct_showch(X, Y, char(matrix[i][j].num + '0'), COLOR_HYELLOW, COLOR_BLACK, 1);
                                        cct_setcolor();
                                        cout << "  ";
                                }
                                else  /*输出查找数组结果模式：输出星号*/
                                        cout << "*  ";
                        }

                        /*3.2.2、若为未标记元素*/
                        else {
                                if (show_mode == ARRAY_SHOW_CURRENT)  /*输出当前数组模式：输出元素数值*/
                                        cout << setw(3) << setiosflags(ios::left) << matrix[i][j].num;
                                else  /*输出查找数组结果模式：输出字符零*/
                                        cout << "0  ";
                        }

                        /*3.2.3、输出完一行后换行*/
                        if ((j + 1) % row == 0)
                                cout << endl;
                }
        }
}

void array_single_scan(MATRIX matrix[][ROW_MAX], const int I, const int J, const int line, const int row)
{
        if (matrix[I][J].mark) {  /*只有在目标元素也被标记的情况下才开始检查*/
                if (I) {  /*若目标元素不在第一列，检查其上面一个元素*/
                        if (matrix[I][J].num == matrix[I - 1][J].num)
                                matrix[I - 1][J].mark = 1;
                }

                if (J) {  /*若目标元素不在第一行，检查其左边一个元素*/
                        if (matrix[I][J].num == matrix[I][J - 1].num)
                                matrix[I][J - 1].mark = 1;
                }

                if (I < line - 1) {  /*若目标元素不在最后一行，检查其下面一个元素*/
                        if (matrix[I][J].num == matrix[I + 1][J].num)
                                matrix[I + 1][J].mark = 1;
                }

                if (J < row - 1) {  /*若目标元素不在最后一列，检查其右边一个元素*/
                        if (matrix[I][J].num == matrix[I][J + 1].num)
                                matrix[I][J + 1].mark = 1;
                }
        }
}

void array_multiple_scan1(MATRIX matrix[][ROW_MAX], const int line, const int row, char coord[], const int menu)
{
        const int I = coord[0] - 'A';  /*将坐标由字符转换为数字*/
        const int J = coord[1] - '0';  /*将坐标由字符转换为数字*/
        matrix[I][J].mark = 1;  /*先标记目标元素*/
        array_single_scan(matrix, I, J, line, row);  /*再以目标元素为中心进行第一次检查*/

        /*若第一次处理完后，数组中被标记元素的个数仍为 1，说明无连续相同值*/
        if (array_mark_count(matrix, line, row) == 1) {

                if (menu <= 4)  /*若菜单函数返回值不大于 4，则此时为命令行方式完成，输出提示*/
                        cout << "输入的矩阵坐标位置处无连续相同值，请重新输入" << endl;

                matrix[I][J].mark = 0;  /*取消目标元素标记，避免影响后续操作*/
        }

        /*若第一次处理完后，数组中被标记元素的个数不为 1，说明有连续相同值，进行下一步检查*/
        else {
                int i, j;  /*计数变量*/
                int count1 = array_mark_count(matrix, line, row), count2 = 0;  /*每一次处理前后数组中被标记元素个数*/

                while (count1 != count2) {  /*若一次处理前后标记元素个数相等，则说明已检查出所有连续相同值，检查结束*/

                        count1 = array_mark_count(matrix, line, row);  /*每次处理前的个数*/

                        /*以所有数组中的元素为目标元素，每一项都进行单项检查*/
                        for (i = 0; i < line; i++) {
                                for (j = 0; j < row; j++) {
                                        array_single_scan(matrix, i, j, line, row);
                                }
                        }

                        count2 = array_mark_count(matrix, line, row);  /*每次处理后的个数*/
                }
        }
}

void array_multiple_scan2(MATRIX matrix[][ROW_MAX], const int line, const int row, char coord[])
{
        const int I = coord[0] - 'A';  /*将坐标由字符转换为数字*/
        const int J = coord[1] - '0';  /*将坐标由字符转换为数字*/
        matrix[I][J].mark = 1;  /*先标记目标元素*/
        array_single_scan(matrix, I, J, line, row);  /*再以目标元素为中心进行第一次检查*/

        /*若第一次处理完后，数组中被标记元素的个数仍为 1，说明无连续相同值，输出提示*/
        if (array_mark_count(matrix, line, row) == 1) {
                cout << "输入的矩阵坐标位置处无连续相同值，请重新输入" << endl;
                matrix[I][J].mark = 0;
        }

        /*若第一次处理完后，数组中被标记元素的个数不为 1，说明有连续相同值，进行下一步检查*/
        else {
                int i, j;  /*计数变量*/
                int count1 = array_mark_count(matrix, line, row), count2 = 0;  /*本次处理前数组中被标记元素个数，处理后个数先置为 0*/

                /*以所有数组中的元素为目标元素，每一项都进行单项检查*/
                for (i = 0; i < line; i++) {
                        for (j = 0; j < row; j++) {
                                array_single_scan(matrix, i, j, line, row);
                        }
                }

                count2 = array_mark_count(matrix, line, row);  /*统计本次处理后的个数*/

                if (count1 != count2)  /*若一次处理前后标记元素个数不等，则说明未检查出所有连续相同值，用递归方式继续检查*/
                        array_multiple_scan2(matrix, line, row, coord);
        }
}

void array_clear(MATRIX matrix[][ROW_MAX], const int line, const int row, const char coord[])
{
        int i, j;  /*计数变量*/
        int num = matrix[coord[0] - 'A'][coord[1] - '0'].num;  /*取目标元素数值*/

        /*将数组中所有被标记的元素数值置 0*/
        for (i = 0; i < line; i++) {
                for (j = 0; j < row; j++) {
                        if (matrix[i][j].mark)
                                matrix[i][j].num = 0;
                }
        }

        matrix[coord[0] - 'A'][coord[1] - '0'].num = num + 1;  /*将先前取好的值自增，赋给目标元素*/
}

int array_getscore(MATRIX matrix[][ROW_MAX], const int line, const int row)
{
        int count = array_mark_count(matrix, line, row);  /*统计已被标记元素数量（消除数量）*/
        int num = 0, i, j;  /*num 为目标元素数值-1（消除值）*/

        for (i = 0; i < line; i++) {
                for (j = 0; j < row; j++) {
                        if (matrix[i][j].num != 0 && matrix[i][j].mark == 1) {  /*若某个元素既被标记又数值不为零，则其一定为目标元素*/
                                num = matrix[i][j].num - 1;  /*消除值为当前目标元素数值-1*/
                                matrix[i][j].mark = 0;  /*将目标元素的标记去除*/
                                break;
                        }
                }

                if (num)  /*若 num 不为零，说明已取到消除值，结束检查*/
                        break;
        }

        return count * num * 3;  /*分数=消除值*消除数量*3*/
}

void array_move_zero(MATRIX matrix[][ROW_MAX], const int line, const int row)
{
        int i, j, t;  /*计数变量、中间变量*/

        /*逐列判断，一列中从下往上判断*/
        for (j = 0; j < row; j++)
                for (i = line - 1; i >= 0; i--) {

                        /*若检查出数值为零的元素*/
                        if (matrix[i][j].num == 0) {
                                int I = i;  /*取元素所在行数*/
                                MATRIX m[LINE_MAX][ROW_MAX];  /*中间数组*/
                                array_copy(m, matrix, line, row);  /*将当前数组复制到中间数组去*/

                                /*该元素不断与其上面的元素交换属性，直到该元素到达第一行为止*/
                                while (I > 0) {
                                        t = matrix[I][j].num;
                                        matrix[I][j].num = matrix[I - 1][j].num;
                                        matrix[I - 1][j].num = t;
                                        t = matrix[I][j].mark;
                                        matrix[I][j].mark = matrix[I - 1][j].mark;
                                        matrix[I - 1][j].mark = t;
                                        I--;  /*每交换一次 I 自减*/
                                }

                                /*将处理前和处理后的数组比较，若有不同，则可能本列需继续处理，将 i 置为 line，从本列的最后一行重新检查*/
                                if (!array_compare(m, matrix, line, row))
                                        i = line;
                        }
                }
}

void array_mark_set(MATRIX matrix[][ROW_MAX], const int line, const int row)
{
        int i, j;  /*计数变量*/
        for (i = 0; i < line; i++) {
                for (j = 0; j < row; j++) {
                        matrix[i][j].mark = 0;  /*元素标记状态为 0*/
                }
        }
}

int array_mark_count(MATRIX matrix[][ROW_MAX], const int line, const int row)
{
        int i, j, count = 0;

        for (i = 0; i < line; i++) {
                for (j = 0; j < row; j++) {
                        if (matrix[i][j].mark)  /*若已被标记，计数变量自增*/
                                count++;
                }
        }

        return count;
```

```
}

void array_random(MATRIX matrix[][ROW_MAX], const int line, const int row, const int MaxNum)
{
	int i, j;  /*计数变量*/

	/*逐个检查数组中元素的数值*/
	for (i = 0; i < line; i++) {
		for (j = 0; j < row; j++) {

			/*若检查出数值为零的元素*/
			if (matrix[i][j].num == 0) {
				int random = rand();  /*随机等概率生成一个 0-32767 之间的数*/

				switch (MaxNum) {
				case 3:
					random = rand() % 3 + 1;
					break;
				case 4:
					if (random > 0 && random <= 3277)
						random = 4;
					else
						random = rand() % 3 + 1;
					break;
				case 5:
					if (random > 0 && random <= 3277)
						random = 5;
					else if (random > 3277 && random <= 8192)
						random = 4;
					else
						random = rand() % 3 + 1;
					break;
				case 6:
					if (random > 0 && random <= 1638)
						random = 6;
					else if (random > 1638 && random <= 6554)
						random = 5;
					else
						random = rand() % 4 + 1;
					break;
				default:
					if (random > 0 && random <= 1638)
						random = MaxNum;
					else if (random > 1638 && random <= 3277)
						random = MaxNum - 1;
					else if (random > 3277 && random <= 6554)
						random = MaxNum - 2;
					else
						random = rand() % (MaxNum - 3) + 1;
					break;
				}

				matrix[i][j].num = random;
			}
		}  //end of for(1)
	}  //end of for(2)
}

void command_goal_achieved(int& goal)
{
	int X, Y;
	cout << endl;
	cct_getxy(X, Y);
	cct_showstr(X, Y, "已经合成到", COLOR_HYELLOW, COLOR_RED);
	cct_getxy(X, Y);
	cct_showch(X, Y, char(goal + '0'), COLOR_HYELLOW, COLOR_RED);

	cct_setcolor();
	cout << endl << "按回车键继续向更高目标进发..." << endl;

	goal++;  /*goal 自增，合成目标上升*/
}

void command_end()
{
	int X, Y;
	cct_getxy(X, Y);
	cct_showstr(X, Y, "无可合并的项，游戏结束!", COLOR_HYELLOW, COLOR_RED);
	cct_setcolor();
	cout << endl;
}
```

2.90-b2-console.cpp

```
void block_single_print(const int num, const int bg_color, const int fg_color)
{
	int X, Y;		/*横纵坐标*/
	cct_getxy(X, Y);  /*取光标当前坐标*/

	cct_showstr(X, Y, "┌──┐", bg_color, fg_color);  /*打印上边界*/
	cct_showstr(X, Y + 1, "│  │", bg_color, fg_color);  /*打印左边界*/

	/*若 num 为一位数，输出该数及一个空格*/
	if (num < 10) {
		cct_showch(X + 2, Y + 1, char(num + '0'), bg_color, fg_color);
		cct_showch(X + 3, Y + 1, ' ', bg_color, fg_color);
	}
	/*若 num 为两位数，输出该数两个不同位上的数*/
	else {
		cct_showch(X + 2, Y + 1, char((num - num % 10) / 10 + '0'), bg_color, fg_color);
		cct_showch(X + 3, Y + 1, char(num % 10 + '0'), bg_color, fg_color);
	}

	cct_showstr(X + 4, Y + 1, "│", bg_color, fg_color);  /*打印右边界*/
	cct_showstr(X, Y + 2, "└──┘", bg_color, fg_color);  /*打印下边界*/
	cct_setcolor();  /*重置颜色*/
}

void console_print_background(const int line, const int row, const int print_mode)
{
	char ch = 'A';  /*行标，初值为 A*/
	int Y;		/*纵坐标*/

	background_number(row, print_mode);  /*按列数输出列标*/

	/*列标位于第一行，下面从第二行开始，逐行打印内容*/
	for (Y = 2; Y <= (3 + print_mode) * line + (3 - print_mode); Y++) {
		background_letter(Y, ch, print_mode);  /*输出行标*/

		if (Y % (3 + print_mode) == (1 - print_mode))  /*每打印一个行标，行标自增*/
			ch++;

		background_multiple_lines(line, row, print_mode);  /*输出不同形式的每一行*/

		cout << " ";  /*每行末尾的空格*/
	}
}

void background_number(const int row, const int print_mode)
{
	int i;
	cout << resetiosflags(ios::left);  /*解除输出左对齐*/

	for (i = 0; i < row; i++) {
		if (i)  /*i 不为零时，按打印模式设置宽度*/
			cout << setw(6 + print_mode * 2) << setiosflags(ios::right) << i;
		else    /*i 为零时，设置宽度为 6*/
			cout << setw(6) << setiosflags(ios::right) << i;
	}

	cout << endl;
}

void background_letter(const int Y, const char ch, const int print_mode)
{
	if (Y % (3 + print_mode) == (1 - print_mode))  /*当前为指定位置，输出列标及一个空格*/
		cout << ch << " ";
	else  /*当前不为指定位置，输出两个空格*/
		cout << "  ";
}

void background_line(int row, int Y, const char firstch[], const char midch[], const char midspecialch[], const char lastch[], int print_mode)
{
	int i;		/*计数变量*/
	int X = 2;  /*横坐标，初始为 2*/

	/*1、输出第一个制表符*/
	cct_showstr(X, Y, firstch, COLOR_WHITE, COLOR_BLACK);
	X = X + 2;

	/*2、输出中间的多个制表符*/
	for (i = 0; i < 3 * row; i++) {

		/*2.1、输出中间的一般制表符*/
		cct_showstr(X, Y, midch, COLOR_WHITE, COLOR_BLACK);
		X = X + 2;

		/*2.2、在指定位置打印特殊制表符*/
		if (print_mode) {
			if (X % 8 == 2 && X != 2 + row * 8) {
				cct_showstr(X, Y, midspecialch, COLOR_WHITE, COLOR_BLACK);
				X = X + 2;
			}
		}

		/*2.3、每执行三次停止 1，控制打印速度*/
		if (i % 3 == 1)
			Sleep(1);
	}

	/*3、输出最后一个制表符并重置颜色*/
	cct_showstr(X, Y, lastch, COLOR_WHITE, COLOR_BLACK);
	cct_setcolor();
}

void background_multiple_lines(const int line, const int row, const int print_mode)
{
	int X, Y;		/*横纵坐标*/
	cct_getxy(X, Y);  /*取当前光标的坐标*/

	/*首行的输出*/
	if (Y == 2)
		background_line(row, Y, "┌", "──", "┬", "┐", print_mode);

	/*最后一行的输出*/
	else if (Y == (3 + print_mode) * line + (3 - print_mode))
		background_line(row, Y, "└", "──", "┴", "┘", print_mode);

	/*特殊行的输出*/
	else if (Y % 4 == 2 && print_mode == 1)
		background_line(row, Y, "├", "──", "┼", "┤", PRINT_WITH_LINE);

	/*一般行的输出*/
	else
		background_line(row, Y, "│", "  ", "│", "│", print_mode);
}

void console_print_base(MATRIX matrix[][ROW_MAX], const int line, const int row, const int print_mode)
{
	const int LENGTH = row * (6 + print_mode * 2) + (7 - print_mode * 2);  /*根据列数取长度*/
	const int WIDTH = line * (3 + print_mode) + (8 - print_mode);		/*根据行数取宽度*/
	int i, j;  /*计数变量*/

	cct_setconsoleborder(LENGTH, WIDTH);  /*根据取得的长度及宽度，动态设置窗口大小*/
	cout << "屏幕当前设置为：" << LENGTH << "行" << WIDTH << "列" << endl << ' ';  /*输出屏幕设置信息*/
	console_print_background(line, row, print_mode);  /*打印白色背景*/

	/*通过访问结构体数组中的内容，逐个打印色块*/
	for (i = 0; i < line; i++) {
```

```
                for (j = 0; j < row; j++) {
                        cct_gotoxy(j * (6 + print_mode * 2) + 4, i * (3 + print_mode) + 3);
                        block_single_print(matrix[i][j].num, matrix[i][j].num,
COLOR_BLACK);
                }
                Sleep(1);  /*每打印完一行停止 1，控制打印速度*/
        }

        cout << "\n\n";  /*输出完成后调整光标位置*/
}

void block_single_activate(MATRIX matrix[][ROW_MAX], const char coordinate[])
{
        const int I = coordinate[0] - 'A';  /*将行标转换为数字*/
        const int J = coordinate[1] - '0';  /*将列标转换为数字*/
        cct_gotoxy(J * 8 + 4, I * 4 + 3);   /*移动光标至指定位置，便于接下来重画色块*/
        block_single_print(matrix[I][J].num, matrix[I][J].num, COLOR_WHITE);  /*重画色块，底
色不变，前景色为白色*/
        cct_setcolor();  /*重置颜色*/
}

void block_single_default(MATRIX matrix[][ROW_MAX], const char coordinate[])
{
        bool prime;  /*对应坐标是否为初始值*/
        if (coordinate[0] == '\0' && coordinate[1] == '\0')
                prime = false;
        else
                prime = true;

        /*若坐标为初始值（都为'\0'），则不执行重画操作，防止发生数组越界*/
        if (prime) {
                const int I = coordinate[0] - 'A';  /*将行标转换为数字*/
                const int J = coordinate[1] - '0';  /*将列标转换为数字*/
                cct_gotoxy(J * 8 + 4, I * 4 + 3);   /*移动光标至指定位置，便于接下来重画色
块*/
                block_single_print(matrix[I][J].num, matrix[I][J].num, COLOR_BLACK);  /*重
画色块，底色不变，前景色为白色*/
        }

        cct_setcolor();  /*重置颜色*/
}

void block_multiple_activate(MATRIX matrix[][ROW_MAX], const int line, const int row)
{
        int i, j;
        char coord[2];  /*存放元素坐标的字符数组*/

        /*根据数组中被标记元素总数取延时时长*/
        int count = array_mark_count(matrix, line, row);
        unsigned int delay;

        if (count <= 5)
                delay = 100;
        else
                delay = 25;

        /*逐个检查数组中元素，若被标记则设置高亮*/
        for (i = 0; i < line; i++) {
                for (j = 0; j < row; j++) {
                        if (matrix[i][j].mark) {
                                coord[0] = i + 'A';
                                coord[1] = j + '0';
                                block_single_activate(matrix, coord);
                                Sleep(delay);  /*每设置一个色块的高亮就延时，控制速度*/
                        }
                }
        }
}

void block_multiple_default(MATRIX matrix[][ROW_MAX], const int line, const int row)
{
        int i, j;
        char coord[2];  /*存放元素坐标的字符数组*/

        for (i = 0; i < line; i++) {
                for (j = 0; j < row; j++) {
                        if (matrix[i][j].mark) {
                                coord[0] = i + 'A';
                                coord[1] = j + '0';
                                block_single_default(matrix, coord);
                                Sleep(1);  /*每取消一个色块的高亮就延时，控制速度*/
                        }
                }
        }
}

void block_single_clear()
{
        int X, Y;
        cct_getxy(X, Y);
        cct_showstr(X, Y,     "      ", COLOR_WHITE);      /*用白色擦除第一行*/
        cct_showstr(X, Y + 1, "      ", COLOR_WHITE);      /*用白色擦除第二行*/
        cct_showstr(X, Y + 2, "      ", COLOR_WHITE);      /*用白色擦除第三行*/
        cct_setcolor();  /*重置颜色*/
}

void block_multiple_clear(MATRIX matrix[][ROW_MAX], const int line, const int row)
{
        int i, j;

        for (i = 0; i < line; i++) {
                for (j = 0; j < row; j++) {

                        /*被标记且数值不为零，则为目标元素，重画当前色块*/
                        if (matrix[i][j].mark == 1 && matrix[i][j].num != 0) {
                                cct_gotoxy(8 * j + 4, 4 * i + 3);
                                block_single_print(matrix[i][j].num, matrix[i][j].num,
COLOR_BLACK);

                                Sleep(1);  /*延时 1，控制速度*/
                        }

                        /*被标记且数值为零，则为数值为零元素，清除当前色块*/
                        else if (matrix[i][j].mark == 1 && matrix[i][j].num == 0) {
                                cct_gotoxy(8 * j + 4, 4 * i + 3);
                                block_single_clear();
                                Sleep(1);  /*延时 1，控制速度*/
                        }
                }
        }
}

void block_single_fall(MATRIX matrix[][ROW_MAX], const char coordinate[])
{
        const int I = coordinate[0] - 'A';  /*将行标转换为数字*/
        const int J = coordinate[1] - '0';  /*将列标转换为数字*/
        const int X = J * 8 + 4;  /*取起始横坐标*/
        const int Y = I * 4 + 3;  /*取起始纵坐标*/

        /*总共下落五行*/
        for (int i = 0; i < 5; i++) {

                /*每下移一行就打印当前色块*/
                cct_gotoxy(X, Y + i);
                block_single_print(matrix[I][J].num, matrix[I][J].num, COLOR_BLACK);

                /*每下移一行清除色块，循环最后一次不执行，避免最后一次画出的色块被清除*/
                if (i < 4) {
                        cct_gotoxy(X, Y + i);
                        block_single_clear();
                }

                /*重画上下格子之间的分割线*/
                cct_showstr(X, Y + 3, "—", COLOR_WHITE, COLOR_BLACK, 3);

                /*每下移两行延时 1，控制速度*/
                if (i % 2 == 0)
                        Sleep(1);
        }

        cct_setcolor();  /*重设颜色*/
}

void block_multiple_fall(MATRIX matrix[][ROW_MAX], const int line, const int row)
{
        int i, j, t;    /*计数变量、中间变量*/
        char coord[2];  /*存放下落色块坐标的字符数组*/

        /*逐列检查，从最下面一行开始检查*/
        for (j = 0; j < row; j++) {
                for (i = line - 1; i >= 0; i--) {

                        /*若数组中这一色块对应元素数值为零（它不在第一行），其正上方色块对
应元素数值不为零，则执行下落操作*/
                        if (matrix[i][j].num == 0 && matrix[i - 1][j].num != 0 && i != 0)
{

                                /*一直下落直到 i 行 j 列元素不为零（即该元素当前位置正下方
没有零）为止*/
                                while (matrix[i][j].num == 0) {
                                        coord[0] = i - 1 + 'A';  /*将行标转换为字符*/
                                        coord[1] = j + '0';      /*将列标转换为字符*/
                                        block_single_fall(matrix, coord);  /*使色块下
落一格*/

                                        /*交换数值，注意：交换前，i-1 行才是要下落的元
素，i 行是其下方的 0*/
                                        matrix[i][j].num = matrix[i - 1][j].num;
                                        matrix[i - 1][j].num = 0;

                                        /*交换标记状态，注意：交换前，i-1 行才是要下落
的元素，i 行是其下方的 0*/
                                        t = matrix[i - 1][j].mark;
                                        matrix[i - 1][j].mark = matrix[i][j].mark;
                                        matrix[i][j].mark = t;

                                        /*若还没下落到最下面一行，i 自增，进行再次下落
的操作*/
                                        if (i < line - 1)
                                                i++;
                                }
                        }
                }
        }
}

void block_multiple_print(MATRIX matrix[][ROW_MAX], const int line, const int row)
{
        int i, j;

        for (i = 0; i < line; i++) {
                for (j = 0; j < row; j++) {
                        if (matrix[i][j].mark) {  /*该元素被标记才重画对应色块，提高运行
效率*/
                                cct_gotoxy(8 * j + 4, 4 * i + 3);
                                block_single_print(matrix[i][j].num, matrix[i][j].num,
COLOR_BLACK);
                                Sleep(1);  /*每画完一个色块延时 1，控制速度*/
                        }
                }
        }
}

void console_blink_block(MATRIX matrix[][ROW_MAX], const char coordinate[])
{
        const int I = coordinate[0] - 'A';  /*将行标转换为数字*/
        const int J = coordinate[1] - '0';  /*将列标转换为数字*/
        const int rpt = 10;       /*重复次数：10 次*/

        for (int i = 0; i < rpt; i++) {
                block_single_activate(matrix, coordinate);  /*设置高亮*/
                Sleep(10);  /*延时 10，控制速度*/

                block_single_default(matrix, coordinate);   /*高亮取消*/
                Sleep(10);  /*延时 10，控制速度*/
        }
}

void special_info(const int line, int& goal, const int info_name)
```

```
{
	int X = 0;			/*取起始横坐标*/
	int Y = 4 * line + 3;	/*取起始纵坐标*/

	/*擦除原有内容*/
	cct_gotoxy(X, Y);
	for (int i = 0; i < 85; i++)
		cout << ' ';
	cct_gotoxy(X, Y);

	/*1、需要改变文字颜色的提示*/
	switch (info_name) {
	case INFO_NO_CLEAR:
		cct_showstr(X, Y, "周围无相同值，", COLOR_BLACK, COLOR_HYELLOW);
		break;
	case INFO_GOAL_ACHIEVED:
		cct_showstr(X, Y, "已经合成到", COLOR_BLACK, COLOR_HYELLOW);
		cct_getxy(X, Y);
		cct_showch(X, Y, char(goal + '0'), COLOR_BLACK, COLOR_HYELLOW);
		cct_getxy(X, Y);
		cct_showstr(X, Y, "，", COLOR_BLACK, COLOR_HYELLOW);
		break;
	case INFO_END:
		cct_showstr(X, Y, "无可合并的项，游戏结束!", COLOR_BLACK, COLOR_HYELLOW);
		break;
	case MENU8_CONTINUE:
		cct_showstr(X, Y, "本次合成结束，按 C/单击左键继续新一次的合成", COLOR_BLACK,
COLOR_HYELLOW);
		break;
	}

	cct_setcolor();  /*重设颜色*/

	/*2、使用正常颜色的提示*/
	switch (info_name) {
	case INFO_RULES:
		cout << "箭头键/鼠标移动，回车键/单击左键选择，Q/单击右键结束";
		break;
	case INFO_NO_CLEAR:
		cout << "箭头键/鼠标移动，回车键/单击左键选择，Q/单击右键结束";
		break;
	case INFO_GOAL_ACHIEVED:
		goal++;  /*合成目标自增*/
		cout << "按回车键/单击左键继续向更高目标进发";
		break;
	case INFO_MAKE_SURE:
		cout << "箭头键/鼠标移动取消当前选择，回车键/单击左键合成";
		break;
	case INFO_END:
		cout << "按 Q 退出";
		break;
	case MENU8_FALL:
		cout << "合成完成，回车键/单击左键下落 0";
		break;
	case MENU8_FILL:
		cout << "下落 0 完成，回车键/单击左键填充新值";
		break;
	}
}

int mouse_and_keyboard(MATRIX matrix[][ROW_MAX], const int line, const int row, char
final_coord[])
{
	int X = 0, Y = 0;		/*横纵坐标，初始均为零*/
	int ret, maction;
	int keycode1, keycode2;
	int loop = 1;
	int count = 0;			/*用来处理特殊情况的计数器*/
	int RET = 0;			/*函数的返回值*/
	char coord[2] = { 0,0 };  /*鼠标控制、或鼠标位置非法时键盘控制的时候，当前选中色块的
坐标*/
	char keyboard_coord[2] = { 0,0 };  /*鼠标位置合法时键盘控制的时候，当前选中色块的坐
标*/

	cct_enable_mouse();  /*启用鼠标*/
	cct_setcursor(CURSOR_INVISIBLE);  /*关闭光标*/

	while (loop) {
		ret = cct_read_keyboard_and_mouse(X, Y, maction, keycode1, keycode2);  /*
读鼠标/键盘*/
		bool BOOL = if_in_block(X, Y, line, row, coord);  /*判断此时鼠标的位置是否
在某个色块中，并处理当前读出的鼠标坐标*/

		/*1、若返回的是鼠标事件*/
		if (ret == CCT_MOUSE_EVENT) {
			count = 0;					/*正常情况，计数器
置零*/
			block_single_default(matrix, keyboard_coord);  /*将特殊情况下被选
中的色块取消高亮*/
			mouse_act(matrix, line, coord, BOOL);		/*打印鼠标当前情况
*/

			/*若鼠标单击左键，且当前位置在某个色块中，则结束循环*/
			if (maction == MOUSE_LEFT_BUTTON_CLICK && BOOL == true) {
				select_act(matrix, final_coord, coord);  /*标记选中元素，
给 final_coord[]赋值*/

				loop = 0;
			}

			else if (maction == MOUSE_RIGHT_BUTTON_CLICK && BOOL == true) {
				final_coord[0] = coord[0];  /*赋给 final_coord[]行信息*/
				final_coord[1] = coord[1];  /*赋给 final_coord[]列信息*/
				RET = MK_RET_MOUSE_RIGHT;   /*单击右键是特殊情况，函数返
回值不为零*/

				loop = 0;
			}
		}

		/*2、若返回的是键盘事件，同时鼠标位置非法*/
		else if (ret == CCT_KEYBOARD_EVENT && BOOL == false) {
			count = 0;					/*正常情况，计数
器置零*/
			block_single_default(matrix, keyboard_coord);  /*将特殊情况下被选
中的色块取消高亮*/

			/*若正常情况下操作的坐标都为初始值，则将其变为 A0，避免发生数组越
界*/
			if (coord[0] == '\0' && coord[1] == '\0') {
				coord[0] = 'A';
				coord[1] = '0';
			}

			/*根据键码进行的操作*/
			switch (keycode1) {
			case 'Q':
			case 'q':
				RET = MK_RET_KEYBOARD_Q;  /*按 Q 键是特殊情况，函数返回值
不为零*/
			case '\r':
				select_act(matrix, final_coord, coord);  /*标记选中元素，
给 final_coord[]赋值*/
				loop = 0;
				break;
			case 224:
				keyboard_multiple_act(matrix, line, row, coord, keycode2);
/*按第二个键码决定下一步行动*/
				break;
			}
		}

		/*3、若返回的是键盘事件，同时鼠标位置合法*/
		else if (ret == CCT_KEYBOARD_EVENT && BOOL == true) {
			/*特殊情况，每次 keyboard_coord[]接收 coord[]的值时，count 自增，循
环只在 count 为零时进行，确保只赋一次值*/
			while (count == 0) {
				keyboard_coord[0] = coord[0];
				keyboard_coord[1] = coord[1];
				count++;
			}

			/*根据键码进行的操作同上一种情况，不过只对 keyboard_coord[]进行操
作*/
			switch (keycode1) {
			case 'Q':
			case 'q':
				RET = MK_RET_KEYBOARD_Q;  /*按 Q 键是特殊情况，函数返回值
不为零*/
			case '\r':
				select_act(matrix, final_coord, keyboard_coord);  /*标记
选中元素，给 final_coord[]赋值*/
				loop = 0;
				break;
			case 224:
				keyboard_multiple_act(matrix, line, row, keyboard_coord,
keycode2);  /*按第二个键码决定下一步行动*/
				break;
			}
		}
	}

	cct_disable_mouse();		//禁用鼠标

	return RET;
}

int mk_select()
{
	int X = 0, Y = 0;		/*横纵坐标，初始均为零*/
	int ret, maction;
	int keycode1, keycode2;
	int loop = 1;
	int RET;  /*函数返回值*/

	cct_enable_mouse();  /*启用鼠标*/
	cct_setcursor(CURSOR_INVISIBLE);  /*关闭光标*/

	while (loop) {
		ret = cct_read_keyboard_and_mouse(X, Y, maction, keycode1, keycode2);

		if (ret == CCT_MOUSE_EVENT) {
			switch (maction) {
			case MOUSE_LEFT_BUTTON_CLICK:   /*单击左键*/
				RET = MK_RET_MOUSE_LEFT;
				loop = 0;
				break;
			case MOUSE_RIGHT_BUTTON_CLICK:  /*单击右键*/
				RET = MK_RET_MOUSE_RIGHT;
				loop = 0;
				break;
			case MOUSE_ONLY_MOVED:          /*仅移动*/
				RET = MK_RET_MOUSE_MOVE;
				loop = 0;
				break;
			}
		}

		else if (ret == CCT_KEYBOARD_EVENT) {
			switch (keycode1) {
			case '\r':
				RET = MK_RET_KEYBOARD_ENTER;  /*按回车*/
				loop = 0;
				break;
			case 224:
				RET = MK_RET_KEYBOARD_ARROW;  /*按箭头键*/
				loop = 0;
				break;
			case 'C':
			case 'c':
				RET = MK_RET_KEYBOARD_C;      /*按 C 键*/
				loop = 0;
				break;
			case 'Q':
			case 'q':
				RET = MK_RET_KEYBOARD_Q;      /*按 Q 键*/
				loop = 0;
				break;
			}
		}
	}
```

```
        }

        return RET;
}

void mk_pause(const int pause_mode)
{
        int loop = 1;

        while (loop) {
                int ret = mk_select();  /*用 ret 接收 mk_select()的返回值*/

                switch (pause_mode) {
                case MK_ENTER_CONTINUE:  /*鼠标单击左键或键盘按回车继续*/
                        if (ret == MK_RET_MOUSE_LEFT || ret == MK_RET_KEYBOARD_ENTER)
                                loop = 0;
                        break;
                case MK_C_CONTINUE:      /*鼠标单击左键或键盘按 C 键继续*/
                        if (ret == MK_RET_MOUSE_LEFT || ret == MK_RET_KEYBOARD_C)
                                loop = 0;
                        break;
                }
        }
}

void mouse_act(MATRIX matrix[][ROW_MAX], const int line, const char coordinate[], const bool prime)
{
        cct_gotoxy(0, 4 * line + 3);  /*根据操作行数移动光标位置*/
        cout << "[当前鼠标] ";

        /*若鼠标当前位置在某个色块中*/
        if (prime) {

                /*擦除原有内容，输出新内容*/
                int X, Y;
                cct_getxy(X, Y);
                for (int i = 0; i < 100; i++)
                        cout << ' ';
                cct_gotoxy(X, Y);
                cout << coordinate[0] << "行" << coordinate[1] << "列";

                /*使当前选中的色块高亮*/
                block_single_activate(matrix, coordinate);
        }

        /*若鼠标当前位置不在某个色块中*/
        else {
                cout << "位置非法";  /*输出提示*/
                block_single_default(matrix, coordinate);  /*取消上一次被选中色块的高亮*/
        }
}

void select_act(MATRIX matrix[][ROW_MAX], char final_coord[], const char selected_coord[])
{
        matrix[selected_coord[0] - 'A'][selected_coord[1] - '0'].mark = 1;  /*对最终选中的色块对应的数组元素做标记*/
        final_coord[0] = selected_coord[0];  /*赋给 final_coord[]行信息*/
        final_coord[1] = selected_coord[1];  /*赋给 final_coord[]列信息*/
}

void keyboard_single_act(MATRIX matrix[][ROW_MAX], const int line, const int row, char operate_coord[], const int move_mode)
{
        block_single_default(matrix, operate_coord);  /*将当前操作坐标对应的色块高亮取消*/
        keyboard_coord_move(line, row, operate_coord, move_mode);  /*根据移动方式改变坐标值*/
        print_keyboard_info(line, operate_coord);       /*打印键盘当前状态*/
        block_single_activate(matrix, operate_coord);  /*将当前操作坐标对应的色块设置为高亮*/
}

void keyboard_multiple_act(MATRIX matrix[][ROW_MAX], const int line, const int row, char operate_coord[], const int keycode)
{
        switch (keycode) {
        case KB_ARROW_UP:     /*向上，以上移方式进行操作*/
                keyboard_single_act(matrix, line, row, operate_coord, KEYBOARD_MOVE_UP);
                break;
        case KB_ARROW_DOWN:   /*向下，以下移方式进行操作*/
                keyboard_single_act(matrix, line, row, operate_coord, KEYBOARD_MOVE_DOWN);
                break;
        case KB_ARROW_LEFT:   /*向左，以左移方式进行操作*/
                keyboard_single_act(matrix, line, row, operate_coord, KEYBOARD_MOVE_LEFT);
                break;
        case KB_ARROW_RIGHT:  /*向右，以右移方式进行操作*/
                keyboard_single_act(matrix, line, row, operate_coord, KEYBOARD_MOVE_RIGHT);
                break;
        }
}

void keyboard_coord_move(const int line, const int row, char coordinate[], const int move_mode)
{
        const int I = coordinate[0] - 'A';  /*将行标转换为数字*/
        const int J = coordinate[1] - '0';  /*将列标转换为数字*/

        switch (move_mode) {
        case KEYBOARD_MOVE_UP:     /*向上移动：若为上边界，边界回绕；否则行标自减*/
                if (I == 0)
                        coordinate[0] = line - 1 + 'A';
                else
                        coordinate[0]--;
                break;
        case KEYBOARD_MOVE_DOWN:   /*向下移动：若为下边界，边界回绕；否则行标自增*/
                if (I == line - 1)
                        coordinate[0] = 'A';
                else
                        coordinate[0]++;
                break;
        case KEYBOARD_MOVE_LEFT:   /*向左移动：若为左边界，边界回绕；否则列标自减*/
                if (J == 0)
                        coordinate[1] = row - 1 + '0';
                else
                        coordinate[1]--;
                break;
        case KEYBOARD_MOVE_RIGHT:  /*向右移动：若为右边界，边界回绕；否则列标自增*/
                if (J == row - 1)
                        coordinate[1] = '0';
                else
                        coordinate[1]++;
                break;
        }
}

bool if_in_block(const int X, const int Y, const int line, const int row, char coordinate[])
{
        int i, j;  /*计数变量*/
        bool if_x = false, if_y = false;  /*横、纵坐标是否在某个色块中，初始均为假*/

        /*对横坐标检查，若横坐标在任意一个色块中，则为真并停止检查*/
        for (j = 0; j < row; j++) {
                if (X >= 8 * j + 4 && X <= 8 * j + 9) {
                        if_x = true;
                        break;
                }
        }

        /*对纵坐标检查，若纵坐标在任意一个色块中，则为真并停止检查*/
        for (i = 0; i < line; i++) {
                if (Y >= 4 * i + 3 && Y <= 4 * i + 5) {
                        if_y = true;
                        break;
                }
        }

        /*若横、纵坐标都在某一色块中，则位置合法*/
        if (if_x == true && if_y == true) {
                coordinate[0] = i + 'A';  /*将转换后的色块行标赋给 coordinate[]*/
                coordinate[1] = j + '0';  /*将转换后的色块列标赋给 coordinate[]*/
                return true;  /*位置合法，返回真*/
        }

        /*若横、纵坐标有一个不在某一色块中，则位置非法*/
        else
                return false;  /*位置非法，返回假*/
}

void print_keyboard_info(const int line, const char coordinate[])
{
        cct_gotoxy(0, 4 * line + 3);  /*根据操作行数移动光标位置*/
        cout << "[当前键盘] ";

        /*擦除原有内容，输出新内容*/
        int X, Y;
        cct_getxy(X, Y);
        for (int i = 0; i < 100; i++)
                cout << ' ';

        cct_gotoxy(X, Y);
        cout << coordinate[0] << "行" << coordinate[1] << "列";
}
```

3.90-b2-net.cpp

```
void map_trans(MATRIX matrix[][ROW_MAX], const char* get_map, const int line, const int row, const int menu)
{
        int i, j;

        for (i = 0; i < line; i++) {
                for (j = 0; j < row; j++) {
                        if (*get_map >= '1' && *get_map <= '9')  /*1-9 的转换*/
                                matrix[i][j].num = *get_map - '0';
                        else if (*get_map >= 'A' && *get_map <= 'F')  /*A-F 的转换*/
                                matrix[i][j].num = *get_map - 'A' + 10;

                        if (menu == 'A')  /*若为命令行方式，初始化每一个元素的标记状态为0*/
                                matrix[i][j].mark = 0;

                        get_map++;
                }
        }
}

void find_best_solution(MATRIX matrix[][ROW_MAX], const int line, const int row, char final_coord[])
{
        /*若此时无可合成项，返回坐标为 00（无效坐标）*/
        if (if_end(matrix, line, row)) {
                final_coord[0] = '0';
                final_coord[1] = '0';
        }

        else {
                const int MAX = array_getmax(matrix, line, row);  /*取当前数组最大元素*/
                const int MIN = array_getmin(matrix, line, row);  /*取当前数组最小元素*/
                int i, j, k;
                bool loop = true;     /*最优解是否已找到*/
                MATRIX m[LINE_MAX][ROW_MAX];

                for (k = MIN; k <= MAX && loop; k++) {  /*从最小元素开始检查*/
                        for (i = 0; i < line && loop; i++) {  /*从第一行开始检查*/
                                for (j = 0; j < row && loop; j++) {  /*从第一列开始检查*/

                                        /*若检查出了与当前 k 相等的元素，将 matrix 复制到 m，模拟一次扫描，看该项是否为可合成项*/
                                        if (matrix[i][j].num == k) {
                                                array_copy(m, matrix, line, row);
                                                m[i][j].mark = 1;
                                                array_single_scan(m, i, j, line, row);

                                                /*若为可合成项，赋值给 final_coord[]后结束检查，得到最优解*/
                                                if (array_mark_count(m, line, row) > 1)
{
```

```
                                                final_coord[0] = i + 'A';
                                                final_coord[1] = j + '0';
                                                loop = false;
                                        }
                                }
                        }//end of for (j = 0...)
                }//end of for (i = 0...)
        }//end of for(k = MIN...)
}//end of else
}

void my_print_map(mtoIO_net_tools& client, MATRIX matrix[][ROW_MAX])
{
        cout << endl << "当前数组: " << endl;
        print_info(client, matrix);  /*同时输出信息*/
        cout << endl;
        array_show(matrix, client.get_row(), client.get_col(), ARRAY_SHOW_CURRENT);
        cout << endl;
}

int game_progress(mtoIO_net_tools& client, MATRIX matrix[][ROW_MAX], const int menu)
{
        char svrpack[RECVBUF_LEN];  //存放从 Server 端收到的数据
        char coordinate[2];  /*最优解坐标*/

        if (menu == 'B') {  /*伪图形界面的准备工作*/
                map_trans(matrix, client.get_map(), client.get_row(), client.get_col(), 'B');
/*转换地图*/
                console_print_base(matrix, client.get_row(), client.get_col(),
PRINT_WITH_LINE);  /*打印底盘、色块*/
                cct_gotoxy(0, 0);
                print_info(client, matrix);  /*打印初始信息*/
        }

        while (1) {
                /* --------------------
                        Client=>Server
                   -------------------- */
                /* 根据服务端发来的地图，返回一个解坐标（目前为手工输入形式，需要改为自
动求最佳解坐标）
                        提示：可以将 client.get_map()的返回值复制到字符数组中，再还原为
你自己的二维数组后求解
                */
                map_trans(matrix, client.get_map(), client.get_row(), client.get_col(),
menu);  /*将服务器发回的地图转换到 matrix 中的数值*/
                array_mark_set(matrix, client.get_row(), client.get_col());  /*重置所有元素
的标记状态*/

                if (menu == 'A')  /*命令行方式下的显示当前内部数组*/
                        my_print_map(client, matrix);

                find_best_solution(matrix, client.get_row(), client.get_col(), coordinate);
/*求解最优解坐标*/

                /* 游戏已结束则不再读键*/
                if (client.is_gameover())
                        break;

                if (menu == 'B')  /*伪图形界面下，移动光标至指定位置*/
                        cct_gotoxy(0, 4 * client.get_row() + 3);
                cout << "请输入行(A-J)列(0-9)坐标 : " << coordinate[0] << coordinate[1] <<
endl;  /*输出提示及最优解坐标*/

                client.send_coordinate(coordinate[0], coordinate[1] - '0');  /*上传最优解坐
标*/

                /* --------------------
                        Server=>Client
                   -------------------- */
                /* 等待 Server 端的 gameprogress */
                if (client.get_gameprogress_string(svrpack) < 0) {
                        return -1;
                }

                if (menu == 'B')  /*伪图形界面下，演示色块的移动*/
                        my_console_move(client, matrix, coordinate, 'B');
                else
                        cout << endl << "Server 应答 : " << endl << svrpack << endl;
        }//end of while(1)

        return 0;
}
```

4.90-b2-tools.cpp

```
void command_center(MATRIX matrix[][ROW_MAX], const int line, const int row, int goal, int score,
char coord[], const int menu)
{
        /*******************************命令行方式*******************************/
        if (menu >= 1 && menu <= 4) {
                int loop = 1;

                while (loop) {

                        if (menu != 4)  /*若不为菜单项 4，只执行一次，loop 置为 0*/
                                loop = 0;

                        else {  /*若为菜单项 4，先检查是否达到了当前合成目标*/
                                if (array_getmax(matrix, line, row) == goal) {
                                        command_goal_achieved(goal);
                                        pause();
                                }

                                array_mark_set(matrix, line, row);  /*重置数组标记情况*/
                        }

                        /*显示当前数组值*/
                        cout << endl << "当前数组: " << endl;
                        array_show(matrix, line, row, ARRAY_SHOW_CURRENT);
                        cout << endl;

                        /*检查是否仍有可合成项，判断游戏是否继续*/
                        if (if_end(matrix, line, row) == true) {
                                command_end();
                                break;
                        }

                        do {  /*输入坐标直到该坐标周围有相同值为止*/
                                input_coord(line, row, coord);

                                if (menu == 2)
                                        array_multiple_scan2(matrix, line, row, coord);
/*菜单项 2 用递归方式寻找可合成项*/
                                else
                                        array_multiple_scan1(matrix, line, row, coord,
menu);  /*其他项（包括菜单项 1）用非递归方式寻找可合成项*/

                        } while (array_mark_count(matrix, line, row) == 0);

                        /*两种方式显示查找结果*/
                        cout << endl << "查找结果数组: " << endl;
                        array_show(matrix, line, row, ARRAY_SHOW_RESULT);
                        cout << endl << endl << "当前数组(不同色标识): " << endl;
                        array_show(matrix, line, row, ARRAY_SHOW_CURRENT);

                        /*菜单项 3、4 继续输入指令，确认是否进行本次合成*/
                        if (menu == 3 || menu == 4) {
                                char command = clear_command(coord);  /*输入指令*/

                                /*输入 Y*/
                                if (command == 'Y') {

                                        /*归并相同值并显示归并后的数组*/
                                        array_clear(matrix, line, row, coord);
                                        cout << endl << "相同值归并后的数组(不同色标识):
" << endl;
                                        array_show(matrix, line, row,
ARRAY_SHOW_CURRENT);

                                        /*计算得分、打印得分*/
                                        int _getscore = array_getscore(matrix, line,
row);
                                        score = score + _getscore;
                                        print_score(_getscore, score, goal, menu);

                                        /*设置断点、除零、显示除零后的数组*/
                                        cout << endl << "按回车键进行数组下落除 0 操
作...";
                                        pause();
                                        array_move_zero(matrix, line, row);
                                        cout << endl << "除 0 后的数组(不同色标识): " <<
endl;
                                        array_show(matrix, line, row,
ARRAY_SHOW_CURRENT);

                                        /*设置断点、新值填充、显示新值填充后的数组*/
                                        cout << endl << "按回车键进行新值填充...";
                                        pause();
                                        array_random(matrix, line, row,
array_getmax(matrix, line, row));
                                        cout << endl << "新值填充后的数组(不同色标识):
" << endl;
                                        array_show(matrix, line, row,
ARRAY_SHOW_CURRENT);
                                        cout << endl;

                                        if (menu == 4) {  /*菜单项 4 的特别提示*/
                                                cout << "本次合成结束，按回车键继续新
一次的合成..." << endl;
                                                pause();
                                        }
                                }

                                /*菜单项 3 不为 Y、菜单项 4 为 Q 则在此结束循环*/
                                else if ((menu == 3 && command != 'Y') || (menu == 4 &&
command == 'Q'))
                                        break;
                                /*菜单项 4 为 N 继续循环*/
                                else
                                        continue;
                        }
                }

                end();  /*本小题结束，请输入 End 继续...*/
        }

        /*******************************伪图形界面*******************************/

        else if (menu >= 5 && menu <= 9) {
                if (menu == 5)  /*菜单项 5 无分隔线*/
                        console_print_base(matrix, line, row, PRINT_WITHOUT_LINE);
                else            /*其他项有分割线*/
                        console_print_base(matrix, line, row, PRINT_WITH_LINE);

                int MK_RET1, MK_RET2, loop1 = 1;  /*两个读键鼠的返回值*/

                while (loop1) {
                        if (menu == 5 || menu == 6)  /*菜单项 5、6 在此结束*/
                                break;
                        else if (menu == 7)  /*菜单项 7 在 loop1 内只执行 1 次，loop1 置为 0*/
                                loop1 = 0;

                        int loop2 = 1;

                        /*若菜单项为 9，判断是否结束*/
                        if (if_end(matrix, line, row) == true && menu == 9) {
                                special_info(line, goal, INFO_END);  /*结束提示*/

                                while (1) {  /*按 Q 退出*/
                                        char c = _getch();
                                        if (c == 'q' || c == 'Q')
                                                break;
                                }

                                break;
                        }
```

```
                        /*若菜单项为9，达到合成目标*/
                        if (array_getmax(matrix, line, row) == goal && menu == 9) {
                                special_info(line, goal, INFO_GOAL_ACHIEVED);  /*达成提示*/

                                while (1) {
                                        int MK_RET = mk_select();  /*单击鼠标左键或按回车键继续*/
                                        if (MK_RET == MK_RET_MOUSE_LEFT || MK_RET == MK_RET_KEYBOARD_ENTER) {

                                                /*清除原有内容*/
                                                cct_gotoxy(0, 4 * line + 3);
                                                for (int i = 0; i < 80; i++)
                                                        cout << ' ';

                                                /*输出操作说明以及当前信息*/
                                                special_info(line, goal, INFO_RULES);
                                                cct_gotoxy(0, 0);
                                                print_score(0, score, goal, menu);
                                                break;
                                        }
                                }
                        }

                        /*选择色块*/
                        while (loop2) {
                                MK_RET2 = mouse_and_keyboard(matrix, line, row, coord);

                                if (menu == 7) {  /*菜单项7显示选中的色块坐标*/
                                        cct_gotoxy(0, 4 * line + 3);
                                        cout << "                              ";
                                        cct_gotoxy(0, 4 * line + 3);
                                        cout << "选中了" << coord[0] << "行" << coord[1] << "列";

                                        loop1 = 0;  /*跳出 loop1*/
                                        break;      /*跳出 loop2*/
                                }

                                if (!MK_RET2) {  /*若读键鼠返回值为零，标记所有可合成项并输出相应提示*/
                                        array_multiple_scan1(matrix, line, row, coord, menu);

                                        block_multiple_activate(matrix, line, row);
                                        special_info(line, goal, INFO_MAKE_SURE);
                                }

                                else {  /*若读键鼠返回值不为零（即鼠标单击右键或按Q键），则跳出所有循环，结束程序*/
                                        loop1 = 0;
                                        break;
                                }

                                if (array_mark_count(matrix, line, row)) {  /*若所选色块周围有相同值*/
                                        MK_RET1 = mk_select();  /*再读一次键鼠*/

                                        switch (MK_RET1) {
                                        case MK_RET_MOUSE_LEFT:
                                        case MK_RET_KEYBOARD_ENTER:
                                                loop2 = 0;  /*单击鼠标左键或按回车键就跳出 loop2，往下进行合成操作*/
                                                break;
                                        case MK_RET_MOUSE_MOVE:
                                        case MK_RET_KEYBOARD_ARROW:
                                                block_multiple_default(matrix, line, row);  /*鼠标移动或按箭头键就取消所有色块的标记，回去重新选择色块*/
                                                array_mark_set(matrix, line, row);
                                                break;
                                        case MK_RET_MOUSE_RIGHT:
                                        case MK_RET_KEYBOARD_Q:
                                                loop1 = 0, loop2 = 0;  /*单击鼠标右键或按Q键就跳出所有循环，结束程序*/
                                                break;
                                        }
                                }

                                else  /*若所选色块周围无相同值，则输出相应提示，回去重新选择色块*/
                                        special_info(line, goal, INFO_NO_CLEAR);
                        }

                        /*若没有单击鼠标右键或按Q键，执行合成操作*/
                        if (loop1) {

                                /*归并相同值、计算得分、显示信息*/
                                array_clear(matrix, line, row, coord);
                                int _getscore = array_getscore(matrix, line, row);
                                score = score + _getscore;
                                cct_gotoxy(0, 0);
                                print_score(_getscore, score, goal, menu);

                                /*消除色块、使被选中的色块闪烁*/
                                block_multiple_clear(matrix, line, row);
                                console_blink_block(matrix, coord);

                                if (menu == 8) {  //合成完成，回车键/单击左键下落0
                                        special_info(line, goal, MENU8_FALL);
                                        mk_pause(MK_ENTER_CONTINUE);
                                }

                                block_multiple_fall(matrix, line, row);  /*显示色块下落动画*/

                                if (menu == 8) {  //下落0完成，回车键/单击左键填充新值
                                        special_info(line, goal, MENU8_FILL);
                                        mk_pause(MK_ENTER_CONTINUE);
                                }

                                /*补充新值、打印被消除的色块*/
                                array_random(matrix, line, row, array_getmax(matrix, line, row));
                                block_multiple_print(matrix, line, row);

                                if (menu == 8) {  //本次合成结束，按C/单击左键继续新一次的合成
                                        special_info(line, goal, MENU8_CONTINUE);
                                        mk_pause(MK_C_CONTINUE);
                                }

                                array_mark_set(matrix, line, row);  /*重设所有元素标记情况*/

                                if (menu == 8)  /*菜单项8在此结束*/
                                        break;
                        }
                }

                if (menu == 8 || menu == 9) {  /*菜单项8、9擦除原有内容*/
                        cct_gotoxy(0, 4 * line + 3);
                        for (int i = 0; i < 80; i++)
                                cout << ' ';
                }

                cct_gotoxy(0, 4 * line + 3);
                end();  /*本小题结束，请输入End继续...*/
        }
}

void input_coord(const int line, const int row, char coordinate[])
{
        cout << "请以字母+数字形式[例：c2]输入矩阵坐标：";
        int X, Y, j;    /*横纵坐标、计数变量*/
        char coord[2];  /*存放坐标的数组*/

        while (1) {
                coord[0] = coord[1] = 0;  /*坐标初始化*/
                int i = 0;  /*计数归零*/

                /*1、输入坐标*/
                while (1) {
                        char c = getchar();  /*逐个提取字符*/
                        if (c == '\n')    /*若有换行符，结束输入*/
                                break;
                        if (i == 0)       /*若输入未结束，第一个字符赋给 coord[0]*/
                                coord[0] = c;
                        else if (i == 1)  /*若输入未结束，第二个字符赋给 coord[1]*/
                                coord[1] = c;
                        i++;    /*每输入一个字符，计数变量自增，统计输入的字符数*/
                }

                /*2、调整 coord[0]的大小写*/
                if (coord[0] >= 'a' && coord[0] <= 'z')
                        coord[0] = coord[0] - 'a' + 'A';

                /*3、若坐标符合要求，输出结果并结束本次输入*/
                if (coord[0] - 'A' >= 0 && coord[0] - 'A' <= line && coord[1] - '0' >= 0 && coord[1] - '0' <= row) {
                        cout << "输入为" << coord[0] << "行" << coord[1] << "列" << endl;
                        break;
                }

                /*4、若坐标不符合要求，输出提示、擦除原有内容并重新输入*/
                else {
                        cout << "输入错误，请重新输入.";

                        cct_getxy(X, Y);   /*取当前坐标*/
                        Y--, X = X + 18;   /*调整为目标位置*/
                        cct_gotoxy(X, Y);  /*移动光标至目标位置*/

                        for (j = 0; j < 20; j++)  /*擦除内容*/
                                cout << " ";

                        cct_gotoxy(X, Y);  /*光标复位*/
                }
        }

        coordinate[0] = coord[0], coordinate[1] = coord[1];  /*将已处理好的坐标赋给参数*/
}

bool if_end(MATRIX matrix[][ROW_MAX], const int line, const int row)
{
        int i, j, count = 0;  /*count 为统计周围无相同值元素数量的变量*/

        for (i = 0; i < line; i++) {
                for (j = 0; j < row; j++) {
                        matrix[i][j].mark = 1;  /*标记当前元素*/
                        array_single_scan(matrix, i, j, line, row);    /*检查该元素周围有无相同值*/
                        if (array_mark_count(matrix, line, row) == 1)  /*若整个数组被标记元素数量仍为1，说明该元素周围无相同值*/
                                count++;  /*计数器自增*/
                        array_mark_set(matrix, line, row);  /*重设所有元素的标记情况*/
                }
        }
        if (count == line * row)  /*若周围无相同值元素数量=行数*列数，说明所有元素都不可合成，游戏结束*/
                return true;
        else                      /*若不等，说明仍有元素可以合成，游戏继续*/
                return false;
}
```