同济大学 实验报告

装

订

线

高级语言程序设计实验报告

——合成十游戏

班级：[redacted]

学号：[redacted]

姓名：[redacted]

完成日期：[redacted]

1. 题目（黑体，四号，段前段后间距各1行）

实现一个合成十综合演示。菜单如下：

1) 命令行找出可合成项并标识（非递归）

2) 命令行找出可合成项并标识（非递归）

3) 命令行完成一次合成（分步骤显示）

4) 命令行完整版（分步骤显示）

5) 伪图形界面显示初始数组（无分隔线）

6) 伪图形界面显示初始数组（有分割线）

7) 伪图形界面下用箭头键选择当前色块

8) 伪图形界面完成一次合成（分步骤）

9) 伪图形界面完整版

0) 退出

整个项目由7个文件组成

① cmd_console_tools.cpp

② cmd_console_tools.h

③ 90-b2.h

④ 90-b2-main.cpp

⑤ 90-b2-base.cpp

⑥ 90-b2-console.cpp

⑦ 90-b2-tools.cpp

1.1.菜单项1：

① 键盘输入行列数，随机产生数组，随后输入要合并的行列坐标，找出所有可合并的位置，并用不同颜色显示。

② 矩阵中找相邻位置要求用一个函数实现，该函数的实现为非递归形式。

1.2.菜单项2：

① 键盘输入行列数，随机产生数组，随后输入要合并的行列坐标，找出所有可合并的位置，并用不同颜色显示。

② 矩阵中找相邻位置要求用一个函数实现，该函数的实现为递归形式。

1.3.菜单项3：

① 找出可合并位置后确定是否合并

“N”：放弃本次操作，重新输入坐标并继续

“Y”：完成本次合并

“Q”：放弃游戏

如果某次合并后无法找到可合并位置，则自动提示游戏结束

② 每次合成包括查找相邻项、合并相邻项、计算得分、下落消除0、在0位置产生新数据，逐步打印。

1.4.菜单项4：

① 合成要求同菜单项3

② 合成到预期目标值后，给出提示信息，但不结束，可继续进行游戏

1.5.菜单项5：

① 键盘输入行列数，随机产生数组，将数组的内容在cmd窗口中用伪图形显示出来

② cmd窗口大小对行列数动态变化

③ 不同数字的前景色/背景色不同

④ 画图过程中适当加延时

1.6.菜单项6：

同菜单项5

1.7.菜单项7：

① 在6的基础上，箭头键实现“当前选择”色块的选择，按回车确认。

② 初始位置定为左上角，用不同颜色标注“当前选择色块”

③ 越过边界后采用环绕的方式

④ 延时可取消

1.8.菜单项8：

① 箭头键选择色块，回车键选定合成位置，并将所有可合成的色块标注出来

② 选定后再次按箭头键则取消本次选定，重新选择，按回车则进行一次合成

③ 一次合成操作包括合并相邻项、下落消除0、0位置产生新数据、计算得分操作，下落必须有动画效果

1.9.菜单项9：

① 达成合成目标后游戏不结束，合成目标+1后游戏继续

② 若某次合并后无法找到可合并位置，则游戏结束

1.10. 菜单项10：

退出程序

2. 整体设计思路

整体思路如下：

90-b2-main：放主函数以及菜单部分函数

90-b2-base：放内部数组方式实现的各函数

90-b2-console：放cmd伪图形界面方式实现的各函数

90-b2-tools：放一些内部数组/图形方式公用的函数

3. 主要功能的实现

3.1. 随机数字的生成

使用num_generate函数，主要作用有两个：第一个是生成初始的最大值为3的界面，第二个是在下落0之后产生新的随机数。生成的数组满足各个最大值的概率分布。在生成初始的界面的时候，随机生成在界面内的横纵坐标（若这个坐标已经产生了数字，则重新生成另一个横纵坐标，如此循环直到找到没有生成数字的坐标）。初始数组通过这个方式直接生成，后来补全的数组由生成随机坐标在这个数组中任意选择一个数字填到空白处。

3.2. 非递归形式求相邻区域

使用一个和输出数组等大的标记数组，用于标记选中坐标周围和该坐标的值相等的区域，若遍历到的数字等于选中坐标的数字且该位置的标记数组位置没有被置为1，则将标记数组的该位置置为1，并用一个bool变量记录下该位置周围有出现目标数字。若bool变量显示没有目标数字了，则退出。

3.3. 递归形式求相邻区域

使用一个简单的dfs算法，临界条件为横纵坐标到达行列的最小值和最大值或者该区域的标记数组已经到1或者该位置的数字不是目标数字，然后将其他不满足临界条件的坐标置为1。

3.4. 下落0操作的实现

按照从下而上、从左而右的顺序对数组进行遍历，若有一个坐标上的值不为0而其下方的坐标对应值为0，则对在这个元素进行下落操作（下方坐标变为该坐标的值，该坐标的值变为0）。当这一列的数字操作完毕，则对下一列进行操作。对于伪图形界面的下落0操作，则多增加一个数字的下落动画。

3.5. 打印数字方块

数字方块是3*3大小的矩形，通过cct_gotoxy函数进行坐标跳转，从左到右，从上到下依次打印各个组成矩形的字符。

3.6. 下落0动画的实现

主要使用cct工具函数，打印出不同移动时刻方块的位置。可是由于之前打印在屏幕上的尾部部分会有残留，需要再打印空白部分将残留覆盖。在到达最终目的坐标的时候，需要用表格的分割线用来覆盖残留。

4. 调试过程碰到的问题

调试过程中主要碰到的问题如下：

① 在进行下落0的操作时，总是会出现下落结果不正确的情况，有时会下落随机负数，则出现数组越界的情况。随后改变了思路，纠正了这个错误

② 在进行命令行部分的操作的时候，会碰到由于缓存区未清空导致输入错误的情况，于是对缓存区清空并进行修正，解决了缓存区错误的问题。

5. 心得体会

在合成十游戏制作过程中，我有以下一些心得体会：

1、在写程序之前，对程序的整体框架要有完整的把握。起初在写这个程序之前，我并没有对这个程序有一个比较完整的概念，只是想到哪写到哪，导致写出来的最初成品非常杂乱，函数之间的嵌套关系非常混乱模糊，给后期的代码修改和维护造成了很大的不便，浪费了很多无谓的时间和精力。

2、对于复杂的程序，在把握全局的同时，也要将程序分成各个小部分来进行，使程序清晰、有条理。我们这次的合成十作业就是这么进行的，由起初的非常基本的汉合成十程序，渐渐的添加 修饰，变成了一个丰富完整的合成十演示程序。

3、养成写注释的习惯非常重要。我在之前没有写注释的习惯，在写这个程序之前也没有写很多注释。因此在写程序的时候，过两天就忘了自己前两天写的是什么了。整体的感受就是思路非常杂乱无章，为写后续程序带来了障碍。

4、对于函数的设计，将一个很大的程序分割成很多有着特定功能的小函数效果会更佳。以 此次合成十程序为例，打印命令行内部数组、打印数字方块、下落0、递归函数等的功能各自分配给多个函数， 将不同的功能通过分割成不同的函数，使思路更加清晰明了，有利于理解程序并更好地实现功能， 后期修改和完善也更加方便。

5、此次的作业中，很多后续的菜单选项都可以用到之前的函数，可以有效地减少重复冗余的代码。以此次的合成十程序为例，寻找相邻区域递归函数、输入参数函数、打印数组方块等功能都可以被各个菜单项反复利用，尤其是合成十递归函数，几乎贯穿了我们整个编写合成十程序的过程。

6. 附件：源程序

随机数字的生成：

```
void num_generate(int grid[][maxn], int row,
int col, int max_num)
{
    int generate[20] = { 0 };
    int size = row * col;//总共的格子数
    switch (max_num)
    {
        case 3:
        {
            generate[1] = int(size / 3);
            generate[2] = int(size / 3);
            generate[3] = size - generate[1]
- generate[2];
            break;
        }
        case 4:
        {
```

```
            for (int i = 1; i <= 3; i++)
                generate[i] =
int(size*0.3);
            generate[4] = size-generate[1]
* 3;
            break;
        }
        case 5:
        {
            for (int i = 1; i <= 3; i++)
                generate[i] =
int(size*0.2);
            generate[4] = int(size*0.15);
            generate[5] =
size-generate[4]-generate[1]*3;
            break;
        }
        case 6:
        {
            for (int i = 1; i <= 4; i++)
                generate[i] =
int(size*0.2);
            generate[5] = int(size*0.15);
            generate[6] = size - generate[1]
* 4 - generate[5];
            break;
        }
        default:
        {
            for (int i = 1; i <= max_num-3;
i++)
                generate[i] =
int(size*0.8/(max_num-3));
            generate[max_num-2] =
int(size*0.1);
            generate[max_num - 1] =
int(size*0.05);
            generate[max_num] = size;
            for (int i = 1; i <= max_num -
1; i++)
                generate[max_num] -=
generate[i];
            break;
        }
    }

    for (int i = 1; i <= max_num; i++)
    {
        for (int j = 1; j <= generate[i];
j++)
        {
            while (true)
            {
                int x = 0, y = 0;
                x = rand() % row+1;
                y = rand() % col+1;//随机
生成坐标
                if(!grid[x][y])
                {
                    grid[x][y] = i;
                    break;//若原来的坐标
上没有赋值，则给当前坐标赋值，否则继续生成
新坐标
                }
            }
        }
    }
}
```

非递归形式求相邻区域：

```
void search_result(int grid[][maxn], int
search[][maxn], int row, int col, int x, int
y)
{
    search_clear(search, row, col);
    int target = grid[x][y];
    search[x][y] = 1;
    bool flag = 0;
    while (true)
    {
        flag = false;
        for (int i = 1; i <= row; i++)
        {
            for (int j = 1; j <= col; j++)
            {
                if (search[i][j] == 1)
                {
                    if (grid[i - 1][j] ==
target && search[i - 1][j] != 1)
                    {
                        search[i - 1][j]
= 1;
                        flag = true;
                    }
                    if (grid[i][j - 1] ==
target && search[i][j - 1] != 1)
                    {
                        search[i][j - 1]
= 1;
                        flag = true;
                    }
                    if (grid[i][j + 1] ==
target && search[i][j + 1] != 1)
                    {
                        search[i][j + 1]
= 1;
                        flag = true;
```

```
                    }
                    if (grid[i + 1][j] ==
target && search[i + 1][j] != 1)
                    {
                        search[i + 1][j]
= 1;
                        flag = true;
                    }
                }
            }
        }
        if (!flag)
            break;
    }

}
```

递归形式求相邻区域：

```
void search_recursive(int grid[][maxn], int
search[][maxn], int row, int col, int x, int
y,const int target)
{
    int dir[4][2] =
{ {0,1},{0,-1},{1,0},{-1,0} };
    int ix, iy;
    if (x == 0 || y == 0 || x == row+1 || y
==
col+1||search[x][y]||grid[x][y]!=target)
        return;
    search[x][y] = 1;
    for (int i = 0; i < 4; i++)
    {
        ix = x + dir[i][0];
        iy = y + dir[i][1];
       search_recursive(grid, search, row,
col, ix, iy,target);
    }
    return;
}
```

打印数字方块：

```
void print_block_internal(int grid[][maxn],
int i, int j)
{
    cct_gotoxy(6 * j - 4 + 2 * (j - 1), 3 *
i - 1 + (i - 1));
    cct_setcolor(grid[i][j] + 6,
COLOR_BLACK);
    cout << "┌──┐";
    cct_gotoxy(6 * j - 4 + 2 * (j - 1), 3 *
i + (i - 1));
    cout << "│" <<
static_cast<char>(grid[i][j] % 10 + '0') <<
"│";
    cct_gotoxy(6 * j - 4 + 2 * (j - 1), 3 *
i + 1 + (i - 1));
    cout << "└──┘";
    cct_setcolor(COLOR_BLACK,
COLOR_WHITE);
}
```

下落 0 操作：

```
void fall_whole(int grid[][maxn], const int
row, const int col)
{
    for (int j =1;j<=col;j++ )
    {
        for (int i = row; i >=1; i--)
        {
            while (grid[i][j]&&grid[i +
1][j] == 0&&(i!=row))
            {
                grid[i + 1][j] =
grid[i][j];
                print_block_fall(grid, i,
j);
                grid[i][j] = 0;
                i++;
            }

        }
    }
}
```

下落 0 动画实现：

```
void print_block_fall(int grid[][maxn], int
i, int j)
{
    for (int cnt = 0; cnt < 4; cnt++)
    {
        Sleep(100);
        cct_gotoxy(6 * j - 4 + 2 * (j - 1),
3 * i - 1 + (i - 1)+cnt);

        if(cnt==3)
            cct_setcolor(COLOR_HWHITE,
COLOR_BLACK);
        else
        cct_setcolor(COLOR_HWHITE,
COLOR_HWHITE);
        cout << "══════";
        cct_gotoxy(6 * j - 4 + 2 * (j - 1),
3 * i + (i - 1) + cnt);
        cct_setcolor(grid[i][j] + 6,
COLOR_BLACK);
        cout << "┌──┐";
```

```
            cct_gotoxy(6 * j - 4 + 2 * (j - 1),
3 * i + i + cnt);
            cout << "┃" <<
static_cast<char>(grid[i][j] % 10 + '0') <<
"┃";
            cct_gotoxy(6 * j - 4 + 2 * (j - 1),
3 * i + 1 + i + cnt);
            cout << "┗━━┛";
            cct_setcolor(COLOR_BLACK,
COLOR_WHITE);
        }
}
```

装 订 线