装

订

线

高级语言程序设计实验报告
——彩球游戏的设计与实现

计算机一班
1850059
杨志远
2018年12月25日

1 题目及基本要求

本程序为彩球游戏的实现。彩球游戏的规则如下：在一个 9*9 的棋盘上，共有 7 颜色的不同小球。游戏开始时，有 5 个不同随机颜色的小球随机分步在棋盘上，玩家将一个小球从一个点移到另一个点。若在此处有 5 个（或更多）相同颜色的横、竖、斜 8 个方向的小球，则消除这一条或多条线上的小球，否则在棋盘上随机添加 3 个小球，并消除恰好连成线的小球。循环往复，直到棋盘被彩球填满，此时游戏结束。

具体到本程序来说，本程序的主要功能包括两部分：非图形化、输入指令的彩球游戏，以及伪图形化、鼠标（或键盘）操作的彩球游戏。除此以外，还有两者分别对应的单步移动和棋盘的绘制功能，由于其功能从属于前两者，故在此不做赘述。

1.1 非图形化彩球游戏

非图形化的彩球游戏完全依靠向控制台输出彩色的 ASCII 码字符来模拟小球，可以看到当前的分数和下一回合的小球颜色。每回合输入两句指令，输入的指令由一个从 A 到 I 的字符和一个从 1 到 9 的字符组成。程序将第一句指令所对应的点上的彩球移动第二句指令所对应的点，并进行包括添加新的小球、检查是否产生同色小球连线、重新输出画面之类的操作。循环往复，直到游戏结束。

1.2 伪图形化彩球游戏

伪图形化的彩球游戏使用制表符和圆圈之类的符号表示分隔符号和小球，可以看到当前的分数、下一回合的小球颜色和棋盘的详细统计信息（包括某个颜色的小球的数量、在棋盘中的百分占比、已经消除的小球数量）。每回合点击鼠标（键盘按键），选择一个小球，再选择一个空位，程序将小球从前一个点移动到后一个点，并进行包括添加新的小球、检查是否产生同色小球连线之类的操作。循环往复，直到游戏结束。

2 整体设计思路

非图形化和伪图形化的彩球游戏的实现思路大同小异。以伪图形化的彩球游戏为例，具体的流程如下：

（1） 初始化各种变量、数组等
（2） 计算下回合的 3 个小球的颜色（未计算小球的位置）
（3） 绘制棋盘，显示分数与下一次的 3 个小球的颜色
（4） 循环，直到判定到游戏结束：
    （1） 显示分数、下一次的 3 个小球的颜色（与棋盘统计信息）
    （2） 输入指令（键盘输入/鼠标）（伪图形化模式下，如果按下了鼠标右键，则退出循环（使用 break））

（3） 计算路径，如果输入了错误的起点、终点、无法形成可行的路径，则要求用户重新输入，直到输入正确为止
（4） 如果在移动后的小球的位置上出现了同色 5 球（或更多）连线，则计入分数，并重新执行一次循环（使用 continue）
（5） 为之前已经计算好颜色的 3 个小球分配棋盘上空余的位置
（6） 分别判断这些小球是否在各自所在的点形成同色 5 球（或更多）连线，如果有则消除这些连成线的小球，并计入分数
（7） 计算下回合的 3 个小球的颜色（未计算小球的位置）
（8） 在棋盘上绘制小球
（5） 进行游戏结束后的收尾工作

非图形化的彩球游戏为了显示出正确的画面，部分流程与上述有所不同，但整体思路没有区别。

3 主要功能的实现

本程序中的一个高难度的核心部分是寻路算法的实现。在本程序中，在综合考虑数据规模（9×9 个点）和编写代码的难度后，使用了广度优先搜索（Breadth First Search）。通过寻路函数 find_path，可以得到路径的总步数和每一步的坐标。具体的流程如下：

（1） 创建三组变量（以下简称为 openList、closeList、allList），各包括四个整型数组（两个数组存储某个点的 X、Y 坐标，两个数组存储该点的父节点的 X、Y 坐标）和一个整型变量
（2） 记录起点的颜色信息，并将起点设置为没有彩球的状态
（3） 将起点加入 openList 和 allList
（4） 循环，追到 openList 为空：
    （1） 创建一组变量（以下简称为 surroundList），包括四个大小为 4 的整型数组和一个整型变量
    （2） 将 openList 此时的第一个节点 P 加入 closeList
    （3） （关键步骤：）获取节点 P 四周所有可以到达的节点（没有越界且没有彩球），并将这些节点的坐标与它们的父节点的坐标存入 surroundList
    （4） 将节点 P 从 openList 中删除
    （5） 将 suroundList 中所有出现在 closeList 中的节点删除
    （6） 将 surroundList 中所有剩下的节点加入 openList 和 allList
    （7） 判断终点是否在 openList 中，如果不是，则继续循环；如果不是，实行以下步骤：
        （1） 将起点还原为刚开始的状态
        （2） 获取终点在 allList 中的索引
        （3） 循环操作，直到索引对应的节点的父节点为空：
            （1） 将索引对应的节点点添加到函数外部的数组 pointList
            （2） 更新索引，令其指向该节点的父节点

（4） 倒转 pointList
（5） 函数返回循环次数，该循环次数为路径的总步数

（5） 程序执行到此处，说明没有找到可行的路径，将起点还原为刚开始的状态，函数返回值为 0

本程序的另一个核心部分是判定并消除同色 5 球（或更多）连线的算法，为了减少程序的复杂度，只需要将某个点作为中心进行判断。具体实现如下：

（1） 创建一个二维常数组，存储八个代表横、竖、斜八个方向的向量。
（2） 循环 8 次，每次使用不同的向量：
  （1） 从向量的反方向第 9 个格子开始，找到第一个没有越界并且与目标点同色的点
  （2） 从这个点开始，根据目前的向量移动，统计没有越界并且与目标点 P 同色的连续的点的数量 N，直到找到一个不符合要求的点为止
  （3） 如果 N 大于 5：
    （1） 从向量的反方向第 9 个格子开始，找到第一个没有越界并且与目标点同色的点
    （2） 从这个点开始，根据目前的向量移动，将除了 P 点以外每个点的颜色设置为 0（伪图形模式中还需要擦除此处的圆圈），直到找到一个越界或者与目标点不同色的点为止
    （3） 根据 N，计算这个模式下获取的分数
（3） 如果 N 大于 5 的情况出现一次及以上，则将 P 点的颜色设置为 0（伪图形模式中还需要擦除此处的圆圈）
（4） 更新总分数，加上本次程序计算出的分数
（5） 返回 N 大于 5 的情况出现的次数

本次程序的主要输入函数负责了输入行数、列数、退出指令、起点指令、终点指令，根据输入的模式选择输入的内容。具体的实现如下：

（1） 读取目前的光标位置，根据输入的提示信息进行调整
（2） 设置一个布尔型变量，称其为 secondInput，初始值为 false
（3） 循环执行以下操作：
  （1） 移动到指定位置(0, y + secondInput)，输出提示信息
  （2） 输出一串空格，再次移动到指定位置 gotoxy(x, y + secondInput)
  （3） 输入字符串，并清除缓冲区，将输入的字符串转化为小写
  （4） 如果字符串没有通过基于输入模式的格式检查，则移动到指定位置(0, y + 1 + secondInput)，并输出错误提示信息，并回到循环的起点（使用 continue）
  （5） 如果选择了输入行数，则修改输入模式，将 secondInput 设置为 true，完成行数的输入，回到循环的起点（使用 continue）
  （6） 如果选择了输入列数的模式，完成行数的输入

（7） 如果选择了输入起点指令和终点指令，则完成指令的输入，之后会调用其他函数进行后续处理

（8） 执行到这里就说明完已经完成了输入操作，跳出循环（使用 break）

本程序中伪图形模式的鼠标键盘操作函数 move_in_console 负责了所有与鼠标键盘的读入有关的操作。具体实现如下：

（1） 创建多个变量，存储当前鼠标所在的 XY 坐标、当前键盘操作的 XY 坐标、当前被选择的点的 XY 坐标、已经被选择的点的 XY 坐标

（2） 打开鼠标操作

（3） 循环执行如下步骤：

（1） 隐藏光标

（2） 监听鼠标和键盘事件

（3） 如果监听到了鼠标事件：

（1） 如果移动了鼠标并且鼠标所在的坐标在棋盘内：在棋盘下方输出此时鼠标对应的点的位置。

（2） 如果按下了鼠标右键：本函数返回 false

（3） 如果按下了鼠标左键：

（1） 将当前鼠标对应的坐标转化为棋盘上的点

（2） 执行函数 move_in_console_left_button_or_enter，处理按下鼠标左键后的操作

（3） 如果该函数返回值为 true，则本函数返回 true

（4） 如果监听到了键盘事件：

（1） 显示光标

（2） 如果按下了 ESC 键并且已经选择了某个点：将已经选择的点的特殊标识擦除，取消选择这个点

（3） 如果按下了方向键：

（1） 根据按下的方向键调整键盘所在的点的坐标

（2） 在棋盘下方输出此时键盘操作下的点的位置

（4） 如果按下了回车键：

（1） 执行函数 move_in_console_left_button_or_enter，处理按下回车后的操作（回车键相当于鼠标左键）

（2） 如果该函数返回值为 true，则本函数返回 true

（5） 隐藏光标，进行收尾工作

本程序中处理按下鼠标左键和键盘回车的函数的具体实现如下：

（1） 创建两个数组，分别存储寻路算法得到的路径的 X、Y 坐标

（2） 如果鼠标的位置上存在小球：

（1） 如果当前鼠标对应的点上有小球并且没有已经被选择的点：

（1） 将该点设置为已经被选择的点

（2） 使用特殊的图形在棋盘上表示被选择的点
（2） 如果当前鼠标对应的点上有小球并且已经存在被选择的点：
（1） 使用特殊的图形在棋盘上表示当前的点
（2） 使用某个图形在棋盘上表示之前被选择的点
（3） 将该点设置为已经被选择的点
（3） 如果当前鼠标对应的点上没有小球并且已经存在被选择的点
（1） 传入数组，执行寻路函数，得到路径，记录步数
（2） 如果步数为 0：在棋盘下方输出提示信息
（3） 如果步数不为 0：
（1） 在棋盘下方输出移动信息
（2） 根据得到的路径，动画演示移动小球的移动
（3） 将小球在内部数组上从起点移动到终点
（4） 将被选择的点设置为空
（5） 本函数返回 true
（3） 本函数返回 false

装 订 线

4 调试过程遇到的问题

调试过程中最大的问题在于寻路算法的实现。我原本打算使用 A 星算法，但在不允许使用结构体和 STL 容器的情况下，每个点需要 4 个整型变量、3 个浮点型变量来储存信息，整体需要四组这样的点组成的数组来存储全部的信息。这也就是说在寻路函数中总共需要创建 7×4=28 个数组，不仅大大增加了编写代码的难度，而且对于本题的数据规模而言，A 星算法之类的启发式搜索算法获得的性能提升非常有限。因此，为了简化程序，我将 A 星算法中的评估操作全部删除，将 A 星算法退化为广度优先搜索算法。由于数组在删除某个数据的开销较大，对于需要频繁进行删除操作的数组，我使用了两个变量来实现了一个简单的列表。但即使是这样，编写代码的难度依旧不小。冗长的函数形参列表让人眼花缭乱，对于每个数组都需要细致的操作，才能保证不出错。

一开始，我编写的程序在执行时频繁出错，只要移动的距离较长，就会进入死循环，最后因为非法访问内存而报错。在仔细研究了各个数组中的内容后，我发现有一组数组中的内容有很大的问题。大量重复的信息被不断写入数组，最终导致了数组溢出。经过了大量的排查，我发现了某句代码出现了错误，原因为函数实参传入时传入了错误的参数。在函数实参传入时，因为形参列表过长，就不可避免地会出现这些问题。

之后，我又发现在伪图形化模式下，只要将某个点从较远的距离移动到左上角时，就会发现该点瞬移到了左上角。在经历了一系列排查后，我发现原因在于数组的初始化阶段，一些设置为 0 的数据与棋盘上的 A1 点（0，0）发生冲突导致的。这些细节上的一点点差错，在排查问题的根源时却需要花费大量的时间和精力去解决。

除此以外，在编写动画演示移动彩球的函数时，我发现小球的位置出现了错误。经过一系列尝试后，我发现，制表符与汉字一样，在横向占用了两个小格，在纵向占用了一个小格。因

此，横向移动需要绘制和消除 4 次小球，纵向移动需要绘制和消除 2 次小球。并且，每一格的移动都需要明确地指定正确的坐标。否则，动画就会出现不连贯的现象。

另外，在编写判断彩球是否连成一条线的函数时，我最初认为只要使用了四个向量表示方向，就能涵盖所有情况。然而，在经过一系列尝试后，我发现，小球在某些情况可以被消除，但在某些情况下则无法消除。在重新检查代码的过程中，我发现了在一些方向上的小球无法被消除。由此，我想到了，如果补齐方向向量的数量，是否就可以解决这个问题。在调整了向量后，这个问题得到了解决。

5 心得体会

5.1 完成本次作业的心得体会、经验教训

这次的程序，从规模和实现细节上来看，比汉诺塔的程序更加复杂，涉及了一些更加高级的算法，以及使用鼠标键盘来进行操作。将程序分为多个源文件，就是为了更加清晰地分隔程序的功能，以达到简化程序的目的。

也正由于程序的复杂程度提升，很多函数的形参列表很长，传入大量参数的过程增加了出错的可能性，一些隐性的问题也会被无限放大，从而极大地增加了纠错的难度，尤其是在使用了指针之后，很有可能发生修改了不应该修改的数据的情况。

在编写寻路函数时，我就遇到了这样的问题。这样的错误很难被发现，因此要想解决这些问题，需要花费大量的精力和时间来排查出错的地方。如果在传入参数时更多地使用常数组，或许可以规避一些类似的问题。但我在编写代码时，没有注意到这一点。更重要的是，这样的方法无法根除这样的问题，归根结底，还是要依靠编写代码时的小心谨慎，才能尽可能地减少错误的代码。

5.2 前后小题的关系、有效利用已完成的代码及重用代码

这次的程序中，我在编写函数时，留下了一定的修改余地。并且在那些公用的函数中，我尽可能地少使用输出语句，画面输出由非图形模式和伪图形模式对应的函数实现。这样，就可以尽可能多的重复利用已有的函数。

在为后续程序的实现，向函数添加新的功能时，我添加了很多带有默认值的形式参数，比如使用 bool 变量来调整函数的功能，使用 NULL 指针代表不需要传入某个数组。这样就可以在尽可能不变动之前的函数调用语句的同时，扩充函数的内容，从而实现更多的功能。

另外，我在每个文件中，广泛地使用静态函数来封装一些小规模的函数，实现重要函数的一些功能，在不打乱原本可以被调用的函数的同时，减少代码的重复。可以说，我在编写这次的程序的时候，比编写汉诺塔程序时有了很大的进步，在代码和函数重复利用方面花费的时间大幅度减少。

装 订 线

6 源程序（伪图形化）

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <iomanip>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>
#include <conio.h>
#include "cmd_console_tools.h"
using namespace std;
const int MinRow = 7;         //棋盘最小行
const int MinCol = 7;         //棋盘最小列
const int MaxRow = 9;         //棋盘最大行
const int MaxCol = 9;         //棋盘最大列
const int MaxColors = 7;      //颜色最大值
const int FontSize1 = 16;     //模式 1-3 字体大小
const int FontSize2 = 36;     //模式 4-8 字体大小
const int WindowXSize1 = 80; //初始窗口横向大小
const int WindowYSize1 = 26; //初始窗口纵向大小
const int WindowXSize2 = 35; //模式 4 窗口横向大小
const int WindowYSize2 = 13; //模式 4 窗口纵向大小
const int WindowXSize3 = 39; //模式 5 窗口横向大小
const int WindowYSize3 = 19; //模式 5 窗口纵向大小
const int MaxPath = 200;   //寻路算法最多存储的点
const int BallsEachStep = 3; //每回合增加的彩球数
const int MinBallsCount = 5; //连成线的最小彩球数
const int Delay1 = 25;        //动画延迟
const char helpInformation[6][80] = {
      "输入错误，请重新输入",
      "请输入列数(7-9)：",
      "请输入行数(7-9)：",
      "输入 End 结束：",
      "请以字母+数字形式[例：c2]输入要移动球的矩阵坐标：",
      "请以字母+数字形式[例：c2]输入要移动球的目的坐标："
};
void input_information(int winX, int inputMode, int mode, int
*row, int *col, char *instruction, int realRow, int realCol)
{
      char tempStr[200] = "";
      bool secondInput = false;
      int x, y;
      getxy(x, y);
      x += winX;
      while (1)
      {
            gotoxy(0, y + secondInput);
            cout << helpInformation[inputMode];
            cout << "                          ";
            gotoxy(x, y + secondInput);
            cin >> tempStr;
            cin.ignore(99999, '\n');
            _strlwr(tempStr);
            if (!check_input_information(inputMode, tempStr,
realRow, realCol))
            {
                  gotoxy(0, y + 1 + secondInput);
                  cout << helpInformation[0] << endl;
                  continue;
            }
            if (inputMode == 2)
            {
                  inputMode--;
                  secondInput = true;
                  *row = tempStr[0] - '0';
                  continue;
            }
            if (inputMode == 1)
                  *col = tempStr[0] - '0';
            if (inputMode == 4 || inputMode == 5)
                  strcpy(instruction, tempStr);
            break;
      }
}
void input_instruction(int mode, int map[][MaxCol + 1], int
*sX, int *sY, int *eX, int *eY ,int row ,int col)
{
      char instructionA[5] = "", instructionB[5] = "";
      while (1)
      {
            input_information(strlen(helpInformation[4]), 4,
mode, 0, 0, instructionA, row, col);
            *sX = instructionA[1] - '1';
            *sY = instructionA[0] - 'a';
            if (map[*sY][*sX] == 0)
            {
                  cout << "起点错误，请重新输入    " <<
endl;
                  continue;
            }
            else
                  cout << "输入为" << char(*sY + 'A') << "行
" << (*sX) + 1 << "列          " << endl;
            break;
      }
      while (1)
      {
            input_information(strlen(helpInformation[5]), 5,
mode, 0, 0, instructionB, row, col);
            *eX = instructionB[1] - '1';
            *eY = instructionB[0] - 'a';
            if (map[*eY][*eX] != 0)
            {
                  cout << "终点错误，请重新输入    " <<
endl;
                  continue;
            }
            else
                  cout << "输入为" << char(*eY + 'A') << "行
" << *eX << "列          " << endl;
            break;
      }
}
static bool check_input_information(int inputMode, char
tempStr[], int realRow, int realCol)
{
      if (inputMode == 1 || inputMode == 2)
      {
            if (strlen(tempStr) != 1 || tempStr[0] < MinRow +
'0' || tempStr[0] > MaxRow + '0')
                  return false;
      }
      if (inputMode == 3)
      {
            if (strlen(tempStr) != 3 || strcmp(tempStr,
"end") != 0)
                  return false;
      }
      if (inputMode == 4 || inputMode == 5)
      {
            if (strlen(tempStr) != 2 || tempStr[0] < 'a' ||
tempStr[0] > 'a' + realRow - 1 || tempStr[1] < '1' ||
tempStr[1] > '1' + realCol - 1)
                  return false;
      }
      return true;
}
void set_balls_color(int ballsColor[], int colors)
{
      for (int i = 0; i < colors; i++)
            ballsColor[i] = rand() % MaxColors + 1;
}
void create_balls(int map[][MaxCol + 1], int row, int col, int
balls, int ballsColor[], int ballsX[], int ballsY[])
{
      int x, y;
      for (int i = 0; i < balls; i++)
      {
            do
            {
                  y = rand() % row;
                  x = rand() % col;
            } while (map[y][x] != 0 && !game_over(map, row,
col));
```

```
            map[y][x] = (ballsColor != NULL) ?
ballsColor[i] : rand() % MaxColors + 1;
            if (ballsY != NULL)
            {
                  ballsX[i] = x;
                  ballsY[i] = y;
            }
      }
}
void move_map(int map[][MaxCol + 1], int dX, int dY, int sX,
int sY)
{
      map[dY][dX] = map[sY][sX];
      map[sY][sX] = 0;
}
bool game_over(int map[][MaxCol + 1], int row, int col)
{
      for (int i = 0; i < row; i++)
            for (int j = 0; j < col; j++)
                  if (map[i][j] == 0)
                        return false;
      return true;
}
static bool in_map(int map[][MaxCol + 1], int row, int col,
int x, int y)
{
      if (x < 0 || x >= col || y < 0 || y >= row)
            return false;
      else
            return true;
}
static int check_map_find_i_start(int map[][MaxCol + 1], int
row, int col, const int direction[][2], int d, int value, int
maxRange, int x, int y)
{
      int i = -maxRange;
      while (++i < 0)
            if (in_map(map, row, col, x + i *
direction[d][0], y + i * direction[d][1]) && map[y + i *
direction[d][1]][x + i * direction[d][0]] == value)
                  break;
      return i;
}
int check_map(int map[][MaxCol + 1], int row, int col, int x,
int y, int *score, int *scoreAll, bool consoleMode, int
ballsRemoved[])
{
      const int direction[8][2] = { {0, 1}, {1, 1}, {1, 0},
{-1, 1} , {-1, 0}, {-1, -1}, {0, -1}, {1, -1} };
      int count = 0, result = 0, value = map[y][x], maxRange
= max(MaxRow, MaxCol);
      *score = 0;
      for (int d = 0; d < 8; d++)
      {
            count = 0;
            int i = check_map_find_i_start(map, row, col,
direction, d, value, maxRange, x, y);
            for (; i <= maxRange; i++)
            {
                  if (in_map(map, row, col, x +
direction[d][0], y + direction[d][1]) && map[y + i *
direction[d][1]][x + i * direction[d][0]] == value)
                        count++;
                  else
                        break;
            }
            if (count >= MinBallsCount)
            {
                  if (ballsRemoved != NULL)
                        ballsRemoved[map[y][x]] = count;
                  result++;
                  i = check_map_find_i_start(map, row, col,
direction, d, value, maxRange, x, y);
                  for (; i <= maxRange; i++)
                  {
                        if (i != 0 && in_map(map, row, col,
x + direction[d][0], y + direction[d][1]) && map[y + i *
direction[d][1]][x + i * direction[d][0]] == value)
                        {
                              map[y + i *
direction[d][1]][x + i * direction[d][0]] = 0;
                              if (consoleMode)
                                    showstr((x + i *
direction[d][0]) * 4 + 2, (y + i * direction[d][1]) * 2 + 2, "
", 15, 0);
                        }
                        else if(i != 0)
                              break;
                  }
                  *score += result * (count - 1) * (count -
2);
            }
      }
      if (result > 0)
      {
            map[y][x] = 0;
            if (consoleMode)
                  showstr(x * 4 + 2, y * 2 + 2, "  ", 15,
0);
      }
      *scoreAll += *score;
      return result;
}
void print_console_border(int mode)
{
      int winX, winY, winBufferX, winBufferY, x, y;
      getxy(x, y);
      getconsoleborder(winX, winY, winBufferX, winBufferY);
      gotoxy(0, 0);
      cout << "屏幕: " << winY << "行" << winX << "列" <<
((mode >= 6)? "(鼠标右键退出)": "") << endl;
      gotoxy(x, y);
}
int statistics_balls(int map[][MaxCol + 1], int row, int col,
int color)
{
      int result = 0;
      for (int i = 0; i < row; i++)
            for (int j = 0; j < col; j++)
                  if (color == map[i][j])
                        result++;
      return result;
}
int find_path(int row, int col, int startX, int startY, int
endX, int endY, int map[][MaxCol + 1], int *pointListX, int
*pointListY)
{
      int startPoint = map[startY][startX];
      int index_of_myPoint = 0, openListStart = 0,
openListEnd = 0, closeListLength = 0, allListLength = 0,
surroundLength = 0;
      int openListX[MaxPath] = { startX }, openListY[MaxPath]
= { startY }, openListFatherX[MaxPath] = { -1 },
openListFatherY[MaxPath] = { -1 };
      int closeListX[MaxPath], closeListY[MaxPath],
closeListFatherX[MaxPath], closeListFatherY[MaxPath];
      int allListX[MaxPath] = { startX }, allListY[MaxPath] =
{ startY }, allListFatherX[MaxPath] = { -1 },
allListFatherY[MaxPath] = { -1 };
      map[startY][startX] = 0;
      openListEnd++;
      allListLength++;
      while (openListEnd - openListStart > 0 && allListLength
< MaxPath - 5)
      {
            int surroundX[4] = { -1, -1, -1, -1 },
surroundY[4] = { -1, -1, -1, -1 }, surroundFatherX[4] = { -1,
-1, -1, -1 }, surroundFatherY[4] = { -1, -1, -1, -1 };
            PushPointFromListToList(closeListX, closeListY,
closeListFatherX, closeListFatherY, &closeListLength,
openListX, openListY, openListFatherX, openListFatherY,
openListStart);
            surroundLength = getSurroundPoints(map, row, col,
openListStart, surroundX, surroundY, surroundFatherX,
surroundFatherY, openListX, openListY);
            openListStart++;
            for (int i = 0; i < surroundLength; i++)
```

装 订 线

```
                DeletePointFromCloseList(surroundX,
surroundY, i, closeListX, closeListY, closeListLength);
            for (int i = 0; i < surroundLength; i++)
            {
                if (surroundX[i] != -1)
                {
                    if (FindPointInList(allListX,
allListY, 0, allListLength, surroundX[i], surroundY[i]) == -1)

      PushPointFromListToList(openListX, openListY,
openListFatherX, openListFatherY, &openListEnd, surroundX,
surroundY, surroundFatherX, surroundFatherY, i);
                    if (FindPointInList(allListX,
allListY, 0, allListLength, surroundX[i], surroundY[i]) == -1)

      PushPointFromListToList(allListX, allListY,
allListFatherX, allListFatherY, &allListLength, surroundX,
surroundY, surroundFatherX, surroundFatherY, i);
                }
            }
            if (FindPointInList(openListX, openListY,
openListStart, openListEnd, endX, endY) > -1)//判断何时退出
            {
                map[startY][startX] = startPoint;
                return find_path_output(endX, endY,
pointListX, pointListY, allListX, allListY, allListFatherX,
allListFatherY, allListLength);;
            }
      }
      map[startY][startX] = startPoint;
      return 0;
}
inline void exchange(int &x, int &y)
{
      int temp = x;
      x = y;
      y = temp;
}
static int FindPointInList(int *theListX, int *theListY, int
start, int end, int x, int y)
{
      for (int i = start; i < end; i++)
      {
            if (theListX[i] == x && theListY[i] == y)
            {
                return i;
            }
      }
      return -1;
}
static void PushPointFromListToList(int *dX, int *dY, int
*dFX, int *dFY, int *dLength, int *sX, int *sY, int *sFX, int
*sFY, int index)
{
      dX[(*dLength)] = sX[index];
      dY[(*dLength)] = sY[index];
      dFX[(*dLength)] = sFX[index];
      dFY[(*dLength)] = sFY[index];
      (*dLength)++;
}
static void DeletePointInOpenList(int *openListX, int
*openListY, int *openListFatherX, int *openListFatherY, int
*Length, int index)
{
      for (int i = index; i < *Length; i++)
      {
            openListX[i] = openListX[i + 1];
            openListY[i] = openListY[i + 1];
            openListFatherX[i] = openListFatherX[i + 1];
            openListFatherY[i] = openListFatherY[i + 1];
      }
      (*Length)--;
}
static bool DeletePointFromCloseList(int *surroundX, int
*surroundY, int surroundIndex, int *closeListX, int
*closeListY, int closeListLength)
{
      bool result = false;
      for (int i = 0; i < closeListLength; i++)
      {
            if (closeListX[i] == surroundX[surroundIndex] &&
closeListY[i] == surroundY[surroundIndex])
            {
                surroundX[surroundIndex] = -1;
                surroundY[surroundIndex] = -1;
                result = true;
            }
      }
      return result;
}
static int getSurroundPoints(int map[][MaxCol + 1], int row,
int col, int tempPointIndex, int *surroundX, int *surroundY,
int *surroundFatherX, int *surroundFatherY, int *openListX,
int *openListY)
{
      int tempPointsX[4] = { -1, -1, -1, -1 }, tempPointsY[4]
= { -1, -1, -1, -1 };
      bool tempPointsIsWall[4] = {false};
      int tempPointsLength = 0;
      if (openListY[tempPointIndex] < row - 1)
      {
            tempPointsX[1] = openListX[tempPointIndex];
            tempPointsY[1] = openListY[tempPointIndex] + 1;
            tempPointsIsWall[1] =
map[openListY[tempPointIndex]][openListX[tempPointIndex]];
      }
      if (openListY[tempPointIndex] > 0)
      {
            tempPointsX[0] = openListX[tempPointIndex];
            tempPointsY[0] = openListY[tempPointIndex] - 1;
            tempPointsIsWall[0] =
map[openListY[tempPointIndex]][openListX[tempPointIndex]];
      }
      if (openListX[tempPointIndex] > 0)
      {
            tempPointsX[2] = openListX[tempPointIndex] - 1;
            tempPointsY[2] = openListY[tempPointIndex];
            tempPointsIsWall[2] =
map[openListY[tempPointIndex]][openListX[tempPointIndex]];
      }
      if (openListX[tempPointIndex] < col - 1)
      {
            tempPointsX[3] = openListX[tempPointIndex] + 1;
            tempPointsY[3] = openListY[tempPointIndex];
            tempPointsIsWall[3] =
map[openListY[tempPointIndex]][openListX[tempPointIndex]];
      }
      for (int i = 0; i < 4; i++)
      {
            if (tempPointsIsWall[i] == false)
            {
                surroundX[i] = tempPointsX[i];
                surroundY[i] = tempPointsY[i];
                surroundFatherX[i] =
openListX[tempPointIndex];
                surroundFatherY[i] =
openListY[tempPointIndex];
                tempPointsLength++;
            }
      }
      return tempPointsLength;
}
static int find_path_output(int endX, int endY, int
pointListX[], int pointListY[], int allListX[], int
allListY[], int allListFatherX[], int allListFatherY[], int
allListLength)
{
      int index_of_return = FindPointInList(allListX,
allListY, 0, allListLength, endX, endY);
      int m_pointListSize = 0;
      for (; index_of_return != -1 && m_pointListSize <=
allListLength; m_pointListSize++)
      {
            pointListX[m_pointListSize] =
allListX[index_of_return];
            pointListY[m_pointListSize] =
allListY[index_of_return];
```

装 订 线

```
            index_of_return = FindPointInList(allListX,
allListY, 0, allListLength, allListFatherX[index_of_return],
                  allListFatherY[index_of_return]);
      }
      for (int i = 0; i < m_pointListSize / 2; i++)
      {
            exchange(pointListX[m_pointListSize - i - 1],
pointListX[i]);
            exchange(pointListY[m_pointListSize - i - 1],
pointListY[i]);
      }
      return m_pointListSize;
}

void draw_framework(int row, int col, int printMode)
{
      if (printMode == 4)
            draw_framework_no_gap(row, col);
      else
            draw_framework_with_gap(row, col);
      setcolor();
      //cout << endl << endl;
}

static void draw_framework_no_gap(int row, int col)
{
      showstr(0, 1, "┌", 15, 0);
      showstr(2, 1, "─", 15, 0, col);
      showstr(2 * (col + 1), 1, "┐", 15, 0);
      for (int i = 0; i < row; i++)
      {
            showstr(0, i + 2, "│", 15, 0);
            showstr(2 * (col + 1), i + 2, "│", 15, 0);
      }
      showstr(0, row + 2, "└", 15, 0);
      showstr(2, row + 2, "─", 15, 0, col);
      showstr(2 * (col + 1), row + 2, "┘", 15, 0);
}
static void draw_framework_with_gap(int row, int col)
{
      showstr(0, 1, "┌", 15, 0);
      for (int j = 0; j < col; j++)
      {
            showstr(2 + 4 * j, 1, "─", 15, 0);
            showstr(2 + 4 * j + 2, 1, "┬", 15, 0);
      }
      showstr(4 * col, 1, "┐", 15, 0);
      for (int i = 0; i < row; i++)
      {
            showstr(0, 2 * i + 2, "│", 15, 0);
            for (int j = 0; j < col; j++)
            {
                  showstr(2 + 4 * j, 2 * i + 2, "  ", 15,
0);
                  showstr(2 + 4 * j + 2, 2 * i + 2, "│",
15, 0);
            }
            showstr(4 * col, 2 * i + 2, "│", 15, 0);
            showstr(0, 2 * i + 3, "├", 15, 0);
            for (int j = 0; j < col; j++)
            {
                  showstr(2 + 4 * j, 2 * i + 3, "─", 15,
0);
                  showstr(2 + 4 * j + 2, 2 * i + 3, "┼",
15, 0);
            }
            showstr(4 * col, 2 * i + 3, "┤", 15, 0);
      }
      showstr(0, 2 * row + 1, "└", 15, 0);
      for (int j = 0; j < col; j++)
      {
            showstr(2 + 4 * j, 2 * row + 1, "─", 15, 0);
            showstr(2 + 4 * j + 2, 2 * row + 1, "┴", 15, 0);
      }
      showstr(4 * col, 2 * row + 1, "┘", 15, 0);
}
//printMode=4:顺序 printMode=5:跳跃
void draw_points(const int map[][MaxCol + 1], int row, int
col, int printMode)
{
      if (printMode == 4)
      {
            for (int i = 0; i < row; i++)
            {
                  for (int j = 0; j < col; j++)
                  {
                        if (map[i][j] != 0)
                              showstr(2 * (j + 1), (i + 2),
"○", map[i][j], 15);
                        else
                              showstr(2 * (j + 1), (i + 2),
"  ", 15, 0);
                  }
            }
      }
      else
      {
            for (int i = 0; i < row; i++)
            {
                  for (int j = 0; j < col; j++)
                  {
                        if (map[i][j] != 0)
                              showstr(2 + 4 * j, 2 * i + 2,
"○", map[i][j], 15);
                  }
            }
      }
      setcolor();
      cout << endl << endl;
}
static bool is_point_xy(int X, int Y, int row, int col)
{
      int mapX = (X - 2) / 4;
      int mapY = (Y - 2) / 2;
      if (mapX >= col || mapY >= row || (fabs(((double)X - 2)
/ 4 - mapX) > (0.25 + 1e-4)) || (fabs((double)Y - 2) / 2 -
mapY) > 1e-4)
            return false;
      return true;
}
static bool get_point_xy(int X, int Y, int &mapX, int &mapY,
int row, int col)
{
      mapX = (X - 2) / 4;
      mapY = (Y - 2) / 2;
      return is_point_xy(X, Y, row, col);
}
static void draw_point_movement(int map[][MaxCol + 1], int
preX, int preY, int X, int Y, int color)
{
      if (X - preX != 0)
      {
            showstr(preX * 4 + 2, Y * 2 + 2, "  ", 15, 0);
            showstr(preX * 4 + 2 + (X - preX), Y * 2 + 2, "◎
", color, 15);
            Sleep(Delay1);
            showstr(preX * 4 + 2 + (X - preX), Y * 2 + 2, "
", 15, 0);
            showstr(preX * 4 + 2 + (X - preX) * 2, Y * 2 + 2,
"◎", color, 15);
            Sleep(Delay1);
            showstr(preX * 4 + 2 + (X - preX) * 2, Y * 2 + 2,
"  ", 15, 0);
            showstr(preX * 4 + 2 + (X - preX) * 3, Y * 2 + 2,
"◎", color, 15);
            Sleep(Delay1);
            showstr(X * 4 + 2 - (X - preX) * 2, Y * 2 + 2, "
│", 15, 0);
            showstr(X * 4 + 2, Y * 2 + 2, "◎", color, 15);
            Sleep(Delay1);
      }
      else if (Y - preY != 0)
      {
            showstr(X * 4 + 2, preY * 2 + 2, "  ", 15, 0);
            showstr(X * 4 + 2, preY * 2 + 2 + (Y - preY), "◎
", color, 15);
            Sleep(Delay1 * 2);
```

```
            showstr(X * 4 + 2, Y * 2 + 2 - (Y - preY), "—", 15, 0);
            showstr(X * 4 + 2, Y * 2 + 2 - (Y - preY) * 2, "", 15, 0);
            showstr(X * 4 + 2, Y * 2 + 2, "◎", color, 15);
            Sleep(Delay1 * 2);
      }
}
static bool move_in_console_left_button_or_enter(int map[][MaxCol + 1], int row, int col, int &X, int &Y, int &mapX, int &mapY, int &mapXselected, int &mapYselected, int &eX, int &eY)
{
      int pointListX[MaxPath], pointListY[MaxPath];
      if (get_point_xy(X, Y, mapX, mapY, row, col))
      {
            if (map[mapY][mapX] != 0 && mapXselected == -1) //选中某个小球
            {
                  mapXselected = mapX;
                  mapYselected = mapY;
                  showstr(X - X % 2, Y, "◎", map[mapY][mapX], 15);
            }
            else if (map[mapY][mapX] == 0 && mapXselected != -1) //判断是否可移动
            {
                  int steps = find_path(row, col, mapXselected, mapYselected, mapX, mapY, map, pointListX, pointListY);
                  if (steps == 0)
                  {
                        gotoxy(0, 2 * row + 2);
                        cout << "没有可行的路线" << endl;
                        Sleep(150);
                  }
                  else
                  {
                        draw_help(mapXselected, mapYselected, mapX, mapY, row);
                        for (int i = 1; i < steps; i++)
                              draw_point_movement(map, pointListX[i - 1], pointListY[i - 1], pointListX[i], pointListY[i], map[mapYselected][mapXselected]);
                        move_map(map, mapX, mapY, mapXselected, mapYselected);
                        showstr(mapX * 4 + 2, mapY * 2 + 2, "○", map[mapY][mapX], 15);
                        mapXselected = -1;
                        mapYselected = -1;
                        eX = mapX;
                        eY = mapY;
                        return true;
                  }
            }
            else if (map[mapY][mapX] != 0 && mapXselected != -1)//选中其他小球
            {
                  showstr(mapXselected * 4 + 2, mapYselected * 2 + 2, "○", map[mapYselected][mapXselected], 15);
                  showstr(mapX * 4 + 2, mapY * 2 + 2, "◎", map[mapY][mapX], 15);
                  mapXselected = mapX;
                  mapYselected = mapY;
            }
      }
      return false;
}
static bool move_in_console_esc(int map[][MaxCol + 1], int row, int col, int &mapXselected, int &mapYselected)
{
      if (mapXselected != -1)
      {
            showstr(mapXselected * 4 + 2, mapYselected * 2 + 2, "○", map[mapYselected][mapXselected], 15);
            mapXselected = -1;
            mapYselected = -1;
            return true;
      }
      return false;
}
static bool move_in_console_arraw_pressed(int keycode, int map[][MaxCol + 1], int row, int col, int &X, int &Y, int &mapX, int &mapY)
{
      if (keycode == KB_ARROW_DOWN)
      {
            if (((Y - 2) / 2) >= row - 1)
                  return false;
            mapY++;
            Y += 2;
      }
      else if (keycode == KB_ARROW_UP)
      {
            if (((Y - 2) / 2) <= 0)
                  return false;
            mapY--;
            Y -= 2;
      }
      else if (keycode == KB_ARROW_LEFT)
      {
            if (((X - 2) / 4) <= 0)
                  return false;
            mapX--;
            X -= 4;
      }
      else if (keycode == KB_ARROW_RIGHT)
      {
            if (((X - 2) / 4) >= col - 1)
                  return false;
            mapX++;
            X += 4;
      }
      return true;
}
static bool move_in_console(int map[][MaxCol + 1], int row, int col, int &eX, int &eY, int &keyX, int &keyY, bool keyboard)
{
      int X = -1, Y = -1, ret, maction, keycode1, keycode2, mapX = 0, mapY = 0, mapXselected = -1, mapYselected = -1;
      while (1)
      {
            ret = read_keyboard_and_mouse(X, Y, maction, keycode1, keycode2);
            setcolor();
            if (ret == CCT_MOUSE_EVENT)
            {
                  setcursor(CURSOR_INVISIBLE);
                  switch (maction) {
                        case MOUSE_ONLY_MOVED:
                              if (is_point_xy(X, Y, row, col))
                              {
                                    gotoxy(0, 2 * row + 2);
                                    cout << "[当前光标]" << char((Y - 2) / 2 + 'A') << "行" << (X - 2) / 4 + 1 << "列" << endl;
                              }
                              break;
                        case MOUSE_LEFT_BUTTON_CLICK:
                  //按下左键
                              get_point_xy(X, Y, mapX, mapY, row, col);
                              if (move_in_console_left_button_or_enter(map, row, col, X, Y, mapX, mapY, mapXselected, mapYselected, eX, eY))
                                    return true;
                              break;
                        case MOUSE_RIGHT_BUTTON_CLICK:
                  //按下右键
                              return false;
                  }
            }
            else if (keyboard && ret == CCT_KEYBOARD_EVENT)
            {
                  setcursor(CURSOR_VISIBLE_NORMAL);
                  switch (keycode1)
```

装 订 线

```
                {
                        case 27: //ESC 键
                                move_in_console_esc(map, row,
col, mapXselected, mapYselected);
                                break;
                        case 13: //ENTER 键
                                if
(move_in_console_left_button_or_enter(map, row, col, keyX,
keyY, mapX, mapY, mapXselected, mapYselected, eX, eY))
                                        return true;
                                break;
                        case 224: //方向键

        move_in_console_arraw_pressed(keycode2, map, row, col,
keyX, keyY, mapX, mapY);
                                gotoxy(0, 2 * row + 2);
                                cout << "[当前位置]" <<
char((keyY - 2) / 2 + 'A') << "行" << (keyX - 2) / 4 + 1 << "
列       " << endl;
                                gotoxy(keyX, keyY);
                                break;
                }
            }
        }
        setcursor(CURSOR_INVISIBLE);
        return false;
}
static int draw_score(int map[][MaxCol + 1], int row, int col,
int eX, int eY, int *score, int *scoreAll, int ballsRemoved[])
{
        int x, y, result;
        result = check_map(map, row, col, eX, eY, score,
scoreAll, true, ballsRemoved);
        setcolor(15, 0);
        getxy(x, y);
        gotoxy(40, 1);
        cout << "┌──────────┐" << endl;
        gotoxy(40, 2);
        cout << "│得分: " << setw(4) << *scoreAll <<  "│" <<
endl;
        gotoxy(40, 3);
        cout << "└──────────┘" << endl;
        gotoxy(x, y);
        setcolor();
        return result;
}
static void draw_next_balls(int map[][MaxCol + 1], int row,
int col, int ballsColor[])
{
        int x, y;
        setcolor(15, 0);
        getxy(x, y);
        gotoxy(40, 5);
        cout << "┌──────────┐" << endl;
        gotoxy(40, 6);
        cout << "│";
        for (int i = 0; i < BallsEachStep; i++)
        {
                showstr(42 + i * 4, 6, "○", ballsColor[i], 15);
                setcolor(15, 0);
                cout << "│";
        }
        cout << endl;
        gotoxy(40, 7);
        cout << "└──────────┘" << endl;
        gotoxy(x, y);
}
static void draw_help(int sX, int sY, int eX, int eY, int row)
{
        int x, y, score = 0, scoreAll = 0;
        setcolor();
        getxy(x, y);
        gotoxy(0, row * 2 + 2);
        cout << "[提示]从" << char(sY + 'A') << sX + 1 << "移动
到" << char(eY + 'A') << eX + 1;
        gotoxy(x, y);
}
static void draw_statistics(int map[][MaxCol + 1], int row,
int col, int ballsRemoved[])
{
        int x, y;
        setcolor(15, 0);
        getxy(x, y);
        gotoxy(40, 9);
        cout << "┌──────────────────────┐";
        for (int i = 0; i < MaxColors + 1; i++)
        {
                gotoxy(40, 9 + i + 1);
                cout << "│                      │";
        }
        for (int i = 0; i < MaxColors + 1; i++)
        {
                int balls = statistics_balls(map, row, col, i);
                gotoxy(40, 9 + i + 1);
                cout << "│";
                showstr(42, 9 + i + 1, "○", ((i == 0) ? 15 : i),
15);
                setcolor(15, 0);
                cout << ":" << setfill('0') << setw(2) << balls
<< "/(" << setw(4) << setiosflags(ios::fixed) <<
setprecision(2) << (double)balls / row / col * 100 <<  "%)
del-" << ballsRemoved[i];
        }
        gotoxy(40, 9 + MaxColors + 2);
        cout << "└──────────────────────┘";
        setcolor();
        gotoxy(x, y);
}
void mode_7_and_8(int map[][MaxCol + 1], int row, int col, int
mode)
{
        int eX = -1, eY = -1, keyX = 2, keyY = 2,
ballsColor[BallsEachStep], score = 0, scoreAll = 0,
ballsX[BallsEachStep], ballsY[BallsEachStep],
ballsRemoved[MaxColors + 1] = { 0 };
        enable_mouse();
        setcursor(CURSOR_INVISIBLE);
        draw_score(map, row, col, eX, eY, &score, &scoreAll,
ballsRemoved);
        set_balls_color(ballsColor, BallsEachStep);
        draw_next_balls(map, row, col, ballsColor);
        draw_statistics(map, row, col, ballsRemoved);
        while (!game_over(map, row, col))
        {
                gotoxy(keyX, keyY);
                draw_statistics(map, row, col, ballsRemoved);
                if (!move_in_console(map, row, col, eX, eY, keyX,
keyY, mode - 7))
                        break;
                if (draw_score(map, row, col, eX, eY, &score,
&scoreAll, ballsRemoved) != 0)
                        continue;
                create_balls(map, row, col, BallsEachStep,
ballsColor, ballsX, ballsY);
                for (int i = 0; i < BallsEachStep; i++)
                        draw_score(map, row, col, ballsX[i],
ballsY[i], &score, &scoreAll, ballsRemoved);
                set_balls_color(ballsColor, BallsEachStep);
                draw_next_balls(map, row, col, ballsColor);
                draw_points(map, row, col, 7);
        }
        setcolor();
        setcursor(CURSOR_VISIBLE_NORMAL);
        disable_mouse();
}
```