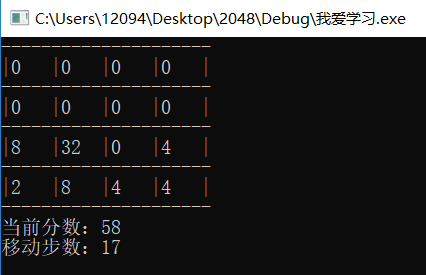
**2048**



**代码**

/\*

2048游戏

游戏规则：随机生成几个数，选择上下左右移动，相同的数字可以相加变为更大的数。

每移动一个数，在空白位置随机再生成一个2或者4。

需要解决的问题：

通过键盘控制移动（可以循环+不断刷新画面） √

难点问题：

如何在随机位置随机生成一个数字？

百度搜索得知：头文件加入<stdlib.h>

若要获得a~b之间的随机整数，使用（rand()%(b-a+1）)+a;

大致思路：

首先需要设置一个4\*4的地图，初始有两个格子被数字填充（2或4）

移动问题：循环嵌套解决

3/25 18:30:创建游戏工程，大致构造主体

3/26 10:00:继续完成主体

3/26 12:20:修复随机数bug，完成游戏主体

3/26 12:50:更改2和4出现的概率

3/26 14:00:增加分数系统

3/26 17:14:增加步数显示，更改游戏过程中界面显示

3/26 17:17:修复初始数字对于分数的影响

3/26 23:00:修复字符对不齐问题（调用库iomanip）

3/27 13:25:美化游戏界面，即加上界面字符边框

3/31 20:10:隐藏了输入光标

4/17 18:50:修改变量传递问题

基础功能已经完成，现在我想对它进行美化。

有什么方法可以使数组中每一个元素不是通过单纯的“数”来表示，而是通过图片来显示呢？

\*/

#include<iostream>

#include<stdlib.h>

#include<conio.h>

#include<time.h>

#include<iomanip>

#include<Windows.h>

using namespace std;

//定义函数

int score(int map[][5],int startscore);

int rands() //随机从1~4中选择一个

{

return (rand() % 4) + 1;

}

int rands2() //随机2(概率为2/3)或者4（概率为1/3）

{

int l = rand() % 3 + 1;

if (l == 3)

return 4;

else

return 2;

}

//随机从未出现数字的位置选择一个出现2或者4的函数

void randing(int map[][5])

{

int x;

int y;

while (1) //循环，直到该位置可以放数字

{

x = rands();

y = rands();

if (map[x][y] == 0)

{

map[x][y] = rands2();

break;

}

}

}

void reflesh(int map[][5],int step,int startscore) //刷新地图

{

system("cls");

cout << "---------------------" << endl;

for (int i = 1; i <= 4; i++)

{

cout << "|";

for (int j = 1; j <= 4; j++)

cout << setw(4) << setiosflags(ios::left) << map[i][j] << setw(1) << "|"; //输出数组，占5个宽度，并且左对齐

cout <<endl<< "---------------------" << endl;

}

cout << "当前分数：" << score(map,startscore) << endl << "移动步数：" << step << endl;

}

int checking(int map[][5]) //检查地图，判断是否已经出现2048（胜利判断）

{

for (int i = 1; i <= 4; i++)

for (int j = 1; j <= 4; j++)

if (map[i][j] == 2048)

return 1;

return 0;

}

int checking2(int map[][5]) //检查地图，判断是否已经没有位置可以放置新增的数组（失败判断）

{

for (int i = 1; i <= 4; i++)

for (int j = 1; j <= 4; j++)

if (map[i][j] == 0)

return 0;

return 1;

}

void left(int map[][5]) //检查向左移动的情况

{

//一行一行的开始检查，从右上角开始。

//这里需要注意一个问题，如果第一列是空格的话，第四列还是需要再次被检查

/\*

研究一下检索的方法。

向左移动的话，应该先检索第二列，看第二列和第一列的关系，如果第一列是空格，那么第二列可以直接跑到第一列去。或者第一列是

相同数字，那么两个数字就可以合并。然后第三列就可以跑到第二列来，也有可能和第一列合并。

或者还有一种情况，第四列可能和第三列合并后再和第二列合并，最后和第一列合并。

所以，应该要判断下面的一些相对关系：

第二列和第一列的关系

第三列和第二列的关系：如果第三列和第二列成功合并（数字相同或者第二列为空白），追加判断一次第二列和第一列的关系

第四列和第三列的关系：如果第四列和第三列成功合并，则判断第三列和第二列的关系；如果第三列和第二列成功合并，则判断第二列和

第一列的关系。

\*/

for (int i = 1; i <= 4; i++) //四行需要判断

{

if (map[i][1] == map[i][2] || map[i][1] == 0) //第二列和第一列的关系

{

map[i][1] += map[i][2];

map[i][2] = 0;

}

//第三列开始判断

if (map[i][2] == map[i][3] || map[i][2] == 0)

{

map[i][2] += map[i][3];

map[i][3] = 0;

if (map[i][1] == map[i][2] || map[i][1] == 0) //追加判断第一列和第二列的关系

{

map[i][1] += map[i][2];

map[i][2] = 0;

}

}

//第四列开始判断

if (map[i][3] == map[i][4] || map[i][3] == 0)

{

map[i][3] += map[i][4];

map[i][4] = 0;

//追加判断

if (map[i][2] == map[i][3] || map[i][2] == 0)

{

map[i][2] += map[i][3];

map[i][3] = 0;

if (map[i][1] == map[i][2] || map[i][1] == 0) //追加判断第一列和第二列的关系

{

map[i][1] += map[i][2];

map[i][2] = 0;

}

}

}

}

}

void right(int map[][5]) //检查向右移动的情况

{

for (int i = 1; i <= 4; i++) //四行需要判断,和向左移动情况的判断方法恰好相反

{

if (map[i][4] == map[i][3] || map[i][4] == 0) //第四列和第三列的关系

{

map[i][4] += map[i][3];

map[i][3] = 0;

}

//第二列开始判断

if (map[i][3] == map[i][2] || map[i][3] == 0)

{

map[i][3] += map[i][2];

map[i][2] = 0;

if (map[i][4] == map[i][3] || map[i][4] == 0) //追加第三列和第二列的关系

{

map[i][4] += map[i][3];

map[i][3] = 0;

}

}

//第一列开始判断

if (map[i][2] == map[i][1] || map[i][2] == 0)

{

map[i][2] += map[i][1];

map[i][1] = 0;

//追加判断

if (map[i][3] == map[i][2] || map[i][3] == 0)

{

map[i][3] += map[i][2];

map[i][2] = 0;

if (map[i][4] == map[i][3] || map[i][4] == 0) //追加第三列和第二列的关系

{

map[i][4] += map[i][3];

map[i][3] = 0;

}

}

}

}

}

void upmove(int map[][5]) //检查向上移动的情况

{

for (int i = 1; i <= 4; i++) //移动有四列需要判断

{

if (map[1][i] == map[2][i] || map[1][i] == 0)

{

map[1][i] += map[2][i];

map[2][i] = 0;

}

if (map[2][i] == map[3][i] || map[2][i] == 0)

{

map[2][i] += map[3][i];

map[3][i] = 0;

if (map[1][i] == map[2][i] || map[1][i] == 0)

{

map[1][i] += map[2][i];

map[2][i] = 0;

}

}

if (map[3][i] == map[4][i] || map[3][i] == 0)

{

map[3][i] += map[4][i];

map[4][i] = 0;

if (map[2][i] == map[3][i] || map[2][i] == 0)

{

map[2][i] += map[3][i];

map[3][i] = 0;

if (map[1][i] == map[2][i] || map[1][i] == 0)

{

map[1][i] += map[2][i];

map[2][i] = 0;

}

}

}

}

}

void downmove(int map[][5]) //检查向下移动的情况

{

for (int i = 1; i <= 4; i++) //移动有四列需要判断

{

if (map[4][i] == map[3][i] || map[4][i] == 0)

{

map[4][i] += map[3][i];

map[3][i] = 0;

}

if (map[3][i] == map[2][i] || map[3][i] == 0)

{

map[3][i] += map[2][i];

map[2][i] = 0;

if (map[4][i] == map[3][i] || map[4][i] == 0)

{

map[4][i] += map[3][i];

map[3][i] = 0;

}

}

if (map[2][i] == map[1][i] || map[2][i] == 0)

{

map[2][i] += map[1][i];

map[1][i] = 0;

if (map[3][i] == map[2][i] || map[3][i] == 0)

{

map[3][i] += map[2][i];

map[2][i] = 0;

if (map[4][i] == map[3][i] || map[4][i] == 0)

{

map[4][i] += map[3][i];

map[3][i] = 0;

}

}

}

}

}

int score(int map[][5],int startscore)//分数统计

{

int m = 0;

for (int i = 1; i <= 4; i++)

for (int j = 1; j <= 4; j++)

m += map[i][j];

return m - startscore;

}

int main()

{

//隐藏输入光标

CONSOLE\_CURSOR\_INFO cursor\_info = { 1, 0 };

SetConsoleCursorInfo(GetStdHandle(STD\_OUTPUT\_HANDLE), &cursor\_info);

//伪随机变量获得时间

srand(time(NULL));

//初始变量

int map[5][5] = { 0 };

int x = 0, y = 0;

int firsttry = 0;

char moving;

int step = 0;

int startscore = 0;

//初始化地图

randing(map);

//进入循环，直到出现2048

while (checking(map) != 1)

{

//该轮游戏的初始化，如果randing无法成功，则游戏失败

if (checking2(map) == 0)

randing(map);

else //游戏失败的情况

{

system("cls");

cout << "游戏失败！" << endl;

cout << "你的分数为:" << score(map,startscore) << endl;

cout << "移动步数为：" << step << endl;

system("pause");

return 0;

}

if (firsttry == 0) //消除初始数字格的影响

{

startscore = score(map,startscore);

firsttry = 1;

}

//刷新画面

reflesh(map,step,startscore);

//操作控制，这里用一个循环来判断输入是否合理

while (1)

{

moving = \_getch();

if (moving == 'w' || moving == 'W' || moving == 'a' || moving == 'A' || moving == 's' || moving == 'S' || moving == 'd' || moving == 'D')

break;

}

step += 1; //移动步数+1

//根据按下按键，进行相应函数操作

switch (moving)

{

case 'a'://向左移动

left(map);

break;

case 'A'://向左移动

left(map);

break;

case 'd'://向右移动

right(map);

break;

case 'D'://向右移动

right(map);

break;

case 'w'://向上移动

upmove(map);

break;

case 'W'://向上移动

upmove(map);

break;

case 's'://向下移动

downmove(map);

break;

case 'S'://向下移动

downmove(map);

break;

}

}

//胜利提示

system("cls");

cout << "游戏胜利！" << endl;

cout << "你的分数为:" << score(map,startscore) << endl;

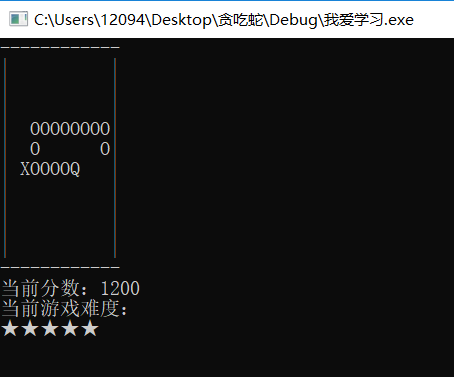
cout << "移动步数为：" << step << endl;

system("pause");

return 0;

}

**贪吃蛇**



**代码**

/\*

贪吃蛇

需要重点解决的问题：

1.食物的随机（这个问题可以解决了，通过随机数）

2.身体所在的位置（可以通过再创建一个地图数组，用来专门描绘蛇的身体）

3.蛇的不断前进与按键判定的问题：如何处理在不前进的过程中，判断按键是否被按下？（重点需要解决的问题）

3/31 12:15：创建游戏工程，设计游戏思路

3/31 13:30：完善基本内容

3/31 15:00：解决多线程问题

3/31 16:00：完成游戏失败判定

3/31 16:30：完成蛇头和蛇尾的判定

3/31 17:00：完成食物判定，游戏主体完成

3/31 19:30：增加分数系统

3/31 19:50：美化地图

3/31 20:00：增加难度选项

3/31 20:10：隐藏了输入光标

3/31 21:10：更改难度显示样式

3/31 21:20：修复蛇尾与蛇头接触的问题

3/31 21:30：更改蛇头样式

4/17 19:50：修改变量传递

\*/

#include<iostream>

#include<stdlib.h>

#include<conio.h>

#include<time.h>

#include<iomanip>

#include<Windows.h>

using namespace std;

int randing()//从1~10中随机生成一个数

{

return rand() % 10 + 1;

}

void fooding(char map[][12],int &food\_x,int &food\_y)//随机生成一个食物

{

while (1)//循环，直到成功生成食物

{

int i = randing();

int j = randing();

if (map[i][j] != 'O' && map[i][j] != 'Q')//食物出现的位置不能为蛇所在位置

{

//食物坐标代入

food\_x = i;

food\_y = j;

map[food\_x][food\_y] = 'X';

return;

}

}

}

void reflesh(char map[][12],int &food\_x, int &food\_y,int &foodgetting,int &length,int speed)//刷新地图

{

//画面重置

system("cls");

//在食物已经被吃掉后再次生成食物

if (foodgetting == 1)

{

fooding(map,food\_x, food\_y);

foodgetting = 0;

}

//地图绘制

for (int i = 0; i <= 11; i++)

{

for (int j = 0; j <= 11; j++)

cout << map[i][j];

cout << endl;

}

cout << "当前分数：" << (length - 3) \* 100 << endl;

cout << "当前游戏难度：" << endl;

for (int i = 1; i <= (1000 - speed) / 200 + 1; i++)

{

cout << "★";

if (i == (1000 - speed) / 200 + 1)

cout << endl;

}

return;

}

int gamefail(char map[][12],int face,int head\_x,int head\_y)//游戏是否失败的判定

{

switch (face)

{

case 1:

if (map[head\_x][head\_y + 1] == '-' || map[head\_x][head\_y + 1] == 'O' || map[head\_x][head\_y + 1] == '|')

return 1;

else

return 0;

case 2:

if (map[head\_x + 1][head\_y] == '-' || map[head\_x + 1][head\_y] == 'O' || map[head\_x + 1][head\_y] == '|')

return 1;

else

return 0;

case 3:

if (map[head\_x][head\_y - 1] == '-' || map[head\_x][head\_y - 1] == 'O' || map[head\_x][head\_y - 1] == '|')

return 1;

else

return 0;

case 4:

if (map[head\_x - 1][head\_y] == '-' || map[head\_x - 1][head\_y] == 'O' || map[head\_x + 1][head\_y] == '|')

return 1;

else

return 0;

}

return 0;

}

int foodgettings(char map[][12],int face, int head\_x, int head\_y)//是否吃到食物的判定

{

switch (face)

{

case 1:

if (map[head\_x][head\_y + 1] == 'X')

return 1;

else

return 0;

case 2:

if (map[head\_x + 1][head\_y] == 'X')

return 1;

else

return 0;

case 3:

if (map[head\_x][head\_y - 1] == 'X')

return 1;

else

return 0;

case 4:

if (map[head\_x - 1][head\_y] == 'X')

return 1;

else

return 0;

}

return 0;

}

//蛇头的检测和处理

void headchecking(char map[][12], int &head\_x, int &head\_y,int &head\_num,int routine[][12],int face)

{

head\_num += 1;

map[head\_x][head\_y] = 'O';

switch (face)

{

case 1:

head\_y += 1;

map[head\_x][head\_y] = 'Q';

routine[head\_x][head\_y] = head\_num;

return;

case 2:

head\_x += 1;

map[head\_x][head\_y] = 'Q';

routine[head\_x][head\_y] = head\_num;

return;

case 3:

head\_y -= 1;

map[head\_x][head\_y] = 'Q';

routine[head\_x][head\_y] = head\_num;

return;

case 4:

head\_x -= 1;

map[head\_x][head\_y] = 'Q';

routine[head\_x][head\_y] = head\_num;

return;

}

}

//蛇尾的检测和处理

void tailchecking(char map[][12],int &tail\_num,int &getting,int routine[][12],int &tail\_x,int &tail\_y)

{

if (getting == 1)

{

map[tail\_x][tail\_y] = 'O';

getting = 0;

return;

}

else

map[tail\_x][tail\_y] = ' ';

tail\_num += 1;

if (routine[tail\_x][tail\_y + 1] == tail\_num)

{

tail\_y += 1;

map[tail\_x][tail\_y] = 'O';

}

if (routine[tail\_x][tail\_y - 1] == tail\_num)

{

tail\_y -= 1;

map[tail\_x][tail\_y] = 'O';

}

if (routine[tail\_x + 1][tail\_y] == tail\_num)

{

tail\_x += 1;

map[tail\_x][tail\_y] = 'O';

}

if (routine[tail\_x - 1][tail\_y] == tail\_num)

{

tail\_x -= 1;

map[tail\_x][tail\_y] = 'O';

}

getting = 0;

return;

}

int face=1;

//多线程判断按键是否被按下

DWORD WINAPI Fun(LPVOID lpParameter)

{

//循环，不断判断按键是否被按下

while (1)

{

char moving;

//操作控制，这里用一个循环来判断输入是否合理

while (1)

{

moving = \_getch();

if (moving == 'w' || moving == 'W' || moving == 'a' || moving == 'A' || moving == 's' || moving == 'S' || moving == 'd' || moving == 'D')

break;

}

//根据按下按键，进行相应变换

switch (moving)

{

case 'a'://向左移动

face = 3;

break;

case 'A'://向左移动

face = 3;

break;

case 'd'://向右移动

face = 1;

break;

case 'D'://向右移动

face = 1;

break;

case 'w'://向上移动

face = 4;

break;

case 'W'://向上移动

face = 4;

break;

case 's'://向下移动

face = 2;

break;

case 'S'://向下移动

face = 2;

break;

}

}

return 0L;

}

//游戏主体部分

int main()

{

//伪随机变量获得时间

srand(time(NULL));

int step = 0;//蛇行走的步数

int head\_x = 1, head\_y = 3;//蛇头的位置

int tail\_x = 1, tail\_y = 1;//蛇尾的位置

int food\_x = 0, food\_y = 0;//食物的位置

int length = 3;//蛇的长度

int foodgetting = 1;//食物是否被吃掉，被吃掉为1，否则为0

int head\_num = 3;//当前蛇头位点

int tail\_num = 1;//当前蛇尾位点

int getting = 0;//判断本轮是否吃到食物的变量

int speed = 1000;//游戏难度，即蛇前进的速度

//地图变量

char map[12][12] =

{

{'-','-','-','-','-','-','-','-','-','-','-','-'},

{'|',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ','|'},

{'|',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ','|'},

{'|',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ','|'},

{'|',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ','|'},

{'|',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ','|'},

{'|',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ','|'},

{'|',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ','|'},

{'|',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ','|'},

{'|',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ','|'},

{'|',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ','|'},

{'-','-','-','-','-','-','-','-','-','-','-','-'}

};

//用来表示蛇走过的路线的变量

int routine[12][12] = { 0 };

//蛇初始所在的位点

for (int i = 1; i <= 3; i++)

{

map[1][i] = 'O';

routine[1][i] = i;

}

//蛇头单独显示

map[1][3] = 'Q';

int over=0;

//游戏开始界面和难度选择

while (1)

{

cout <<

"------------------------------------------------------\n" <<

"| |\n" <<

"|游戏名称：贪吃蛇 |\n" <<

"|版本号：v0.1 |\n" <<

"|游戏简介：新手练手作品，可能会有很多漏洞 |\n" <<

"|操作说明：W向上 A向左 S向右 D向下 |\n" <<

"| |\n" <<

"------------------------------------------------------\n" << endl;

cout << "当前游戏难度：";

for (int i = 1; i <= (1000 - speed) / 200 + 1; i++)

{

cout << "★";

if (i == (1000 - speed) / 200 + 1)

cout << endl;

}

cout <<

"1.开始游戏\n2.难度选择\n3.退出游戏" << endl;

int i;

cin >> i;

switch (i)

{

case 1:

over = 1;

break;

case 2:

while (1)

{

system("cls");

cout << "请选择游戏难度（1~5）：" << endl;

int k;

cin >> k;

if (k >= 1 && k <= 5)

{

speed =1000-(k - 1) \* 200;

break;

}

else

cout << "不合理的游戏难度哦。" << endl;

}

break;

case 3:

return 0;

}

if (over == 1)

break;

system("cls");

}

//多线程相关定义

HANDLE hThread = CreateThread(NULL, 0, Fun, NULL, 0, NULL);

CloseHandle(hThread);

//隐藏输入光标

CONSOLE\_CURSOR\_INFO cursor\_info = { 1, 0 };

SetConsoleCursorInfo(GetStdHandle(STD\_OUTPUT\_HANDLE), &cursor\_info);

//循环游戏

while (1)

{

//游戏刷新

reflesh(map, food\_x, food\_y,foodgetting,length,speed);

//等待一段时间

Sleep(speed);

//是否吃到食物的判定

if (foodgettings(map,face,head\_x,head\_y) == 1)

{

foodgetting = 1;

getting = 1;

length += 1;//蛇的长度加一

}

//蛇尾的处理（放在这里是为了防止还没有移走的蛇尾对游戏失败判定产生影响）

tailchecking(map,tail\_num,getting,routine,tail\_x,tail\_y);

//游戏是否失败的判定

if (gamefail(map,face, head\_x, head\_y) == 1)

break;

//蛇头的处理

headchecking(map, head\_x, head\_y,head\_num, routine,face);

}

//游戏失败的判定

system("cls");

cout << "游戏失败！" << endl;

cout << "你的分数为：" << (length - 3) \* 100 << endl;

Sleep(3000);

system("pause");

return 0;

}

**俄罗斯方块**



**代码**

/\*

游戏：俄罗斯方块

首先需要查找一个俄罗斯方块标准模式的一些相关数据

游戏实际可移动场景大小为：长20，宽10

一共有七种方块：

OO 00 000 0 00 00 00

OO 0 0 0 00 0 00

0 0 0

0

把每一种方块用二维数组（3X3）表示，旋转的时候绕着中心点顺时针旋转

因此，七种方块首先每一种应当使用一个数组来进行表示，旋转方法经过百度搜索，实质上就是

改变下标的相对位置：

0 1 2 旋转 2 5 8

3 4 5 -> 1 4 7

6 7 8 0 3 6

因此，可以用一个循环来将下标进行变换。

关于地图，用一个char型数组来存储地图，同时使用一个int型数组来存储地图块的类型

（这种方法可以简单地判断每一行相加之和为10则可以进行消除）

后面还有几个比较难以解决的问题：

1.方块的平移

2.方块与方块间的接触判定

3.方块消除后，其他所有方块的相对移动如何实现

4.方块旋转和能否旋转的判定

5.方块的随机生成问题

首先，多线程肯定是需要使用到的：

一个线程判断按键，另一个线程用来等待方块移动

其次，使用随机数来出现不同的方块

关于按键的种类：

↑旋转

↓加速下降

←向左移动

→向右移动

其他可能会出现的问题暂且先不讨论

4/3 18:00：创建游戏工程，寻找设计思路

4/3 18:30：基础规则资料搜集完毕

4/3 19:00：游戏初步思路完成

4/3 19:50：基本地图和七种方块绘制完毕

4/3 20:00：导入多线程和隐藏光标代码

4/3 21:00：完成方块产生函数

4/3 21:20：完成方块消除函数

4/3 21:40：完成方块下移函数

4/3 22:50：删除七种方块，重新绘制方块

4/4 20:00：删除全部代码，重新开始

4/4 20:15：新思路完成，开始编写

4/4 21:20：完成七种方块的绘制

4/4 22:00：完成方块生成函数

4/4 22:30：基础界面完成

4/4 23:10：测试完成初始界面及方块生成

4/5 09:20：初步完成方块下移函数

4/5 10:00：寻找出现问题的原因

4/5 13:50：初步完成左右移动函数

4/5 14:10：寻找莫名其妙出现的bug的原因

4/5 14:20：产生原因为变量名的重复，成功修复

4/5 15:00：完成方块消除函数

4/5 15:25：除旋转外的基本功能成功实现

4/5 16:45：完成六种方块的旋转

4/5 17:00：解决当前存在的问题

4/5 17:10：基础功能全部完成，但仍存在许多问题

4/5 17:50：实现瞬间降落功能

4/5 22:30：修复旋转时存在的问题

4/5 22:45：更改图案样式

4/6 14:10：增加游戏失败判定

4/7 13:40：增加下一个方块提示，完善游戏界面

4/7 16:50：修复了画面崩坏的问题

4/8 12:00：继续修复画面崩坏的问题

\*/

#include<iostream>

#include<stdlib.h>

#include<conio.h>

#include<time.h>

#include<iomanip>

#include<Windows.h>

using namespace std;

//已经放置好的方块

int mapsaving[22][12] = { 0 };

//正在移动的方块

int mapmoving[22][12] = { 0 };

//地图上实际存在的方块

int map[22][12] = { 0 };

//用来模拟旋转的数组

int mapping[22][12] = { 0 };

//用来存储当前方块类型的数组

int basesaving1[3][3];

int basesaving2[4][4];

int basesaving3[3][3];

int basesaving4[4][4];

int type;//当前正在下落的方块类型

int typed;//下一个方块类型

int x, y;//当前正在下落的方块的中心坐标

int moving = 0;//是否存在下落的方块（1存在，2不存在）

char acting;//存储按键变量

int speedup = 0;//加速下滑状态是否开启

int point = 0;//分数

int control = 0;//是否处于控制状态

int rand(int a, int b)//生成a~b之间的随机数

{

return (rand() % (b - a + 1)) + a;

}

void reflesh()//刷新界面

{

if (control == 1)

return;

control = 1;

system("cls");

cout << "○○○○○○○○○○○○○○○○○○○○" << endl;

for (int i = 1; i < 21; i++)

{

cout << "○";

for (int j = 1; j < 11; j++)

{

if (mapmoving[i][j] == 1)

cout << "□";

else

if (mapsaving[i][j] == 1)

cout << "■";

else

cout << " ";

}

cout << "○";

switch (i)

{

case 2:

cout << "下一个方块： ○" << endl;

break;

case 4:

switch (typed)

{

case 1:

cout << " □ ○" << endl;

break;

case 2:

cout << " □ ○" << endl;

break;

case 3:

cout << " □□ ○" << endl;

break;

case 4:

cout << " □□ ○" << endl;

break;

case 5:

cout << " □ ○" << endl;

break;

case 6:

cout << " □□ ○" << endl;

break;

case 7:

cout << " ○" << endl;

break;

}

break;

case 5:

switch (typed)

{

case 1:

cout << " □□□ ○" << endl;

break;

case 2:

cout << " □□□ ○" << endl;

break;

case 3:

cout << " □□ ○" << endl;

break;

case 4:

cout << " □□ ○" << endl;

break;

case 5:

cout << " □□□ ○" << endl;

break;

case 6:

cout << " □□ ○" << endl;

break;

case 7:

cout << " □□□□ ○" << endl;

break;

}

break;

case 7:

cout << "--------------○" << endl;

break;

case 8:

cout << "当前分数： ○" << endl;

break;

case 10:

cout << setw(14) << setiosflags(ios::left) << point << "○" << endl;

break;

case 11:

cout << "--------------○" << endl;

break;

case 12:

cout << "操作说明： ○" << endl;

break;

case 14:

cout << " W旋转 ○" << endl;

break;

case 15:

cout << " A左移 D右移 ○" << endl;

break;

case 16:

cout << " S降落 ○" << endl;

break;

default:

cout << " ○" << endl;

}

}

cout << "○○○○○○○○○○○○○○○○○○○○" << endl;

control = 0;

}

void spin()//旋转方块

{

if (type == 6)//第六种无法旋转

return;

else

{

if (type == 7)//第七种需要将一个四阶矩阵进行旋转

{

//内层旋转

basesaving4[1][2] = basesaving2[1][1];

basesaving4[2][2] = basesaving2[1][2];

basesaving4[2][1] = basesaving2[2][2];

basesaving4[1][1] = basesaving2[2][1];

//外层旋转

basesaving4[0][2] = basesaving2[1][0];

basesaving4[0][3] = basesaving2[0][0];

basesaving4[1][3] = basesaving2[0][1];

basesaving4[2][3] = basesaving2[0][2];

basesaving4[3][3] = basesaving2[0][3];

basesaving4[3][2] = basesaving2[1][3];

basesaving4[3][1] = basesaving2[2][3];

basesaving4[3][0] = basesaving2[3][3];

basesaving4[2][0] = basesaving2[3][2];

basesaving4[1][0] = basesaving2[3][1];

basesaving4[0][0] = basesaving2[3][0];

basesaving4[0][1] = basesaving2[2][0];

//假设可以旋转

for (int i = 0; i < 4; i++)

for (int j = 0; j < 4; j++)

mapping[x - 1 + i][y - 1 + j] = basesaving4[i][j];

for (int i = 0; i <= 21; i++)

for (int j = 0; j <= 11; j++)

{

mapping[i][j] += mapsaving[i][j];

if (mapping[i][j] == 2)//如果为2，则表示出现了重合的情况

{

for (int k = 0; k <= 21; k++)//把mapping数组清空

for (int l = 0; l <= 11; l++)

mapping[k][l] = 0;

return;

}

}

//如果没有重合，则可以正常旋转

for (int i = 0; i < 4; i++)

for (int j = 0; j < 4; j++)

basesaving2[i][j] = basesaving4[i][j];//注意改变存储的数据

for (int i = 0; i < 4; i++)

for (int j = 0; j < 4; j++)

mapmoving[x - 1 + i][y - 1 + j] = basesaving4[i][j];

for (int k = 0; k <= 21; k++)//把mapping数组清空

for (int l = 0; l <= 11; l++)

mapping[k][l] = 0;

//刷新画面

system("cls");

}

else

{

//临时数组用来存储旋转后的情况

basesaving3[0][2] = basesaving1[0][0];

basesaving3[1][2] = basesaving1[0][1];

basesaving3[2][2] = basesaving1[0][2];

basesaving3[2][1] = basesaving1[1][2];

basesaving3[2][0] = basesaving1[2][2];

basesaving3[1][0] = basesaving1[2][1];

basesaving3[0][0] = basesaving1[2][0];

basesaving3[0][1] = basesaving1[1][0];

basesaving3[1][1] = basesaving1[1][1];

//假设可以旋转

for (int i = 0; i < 3; i++)

for (int j = 0; j < 3; j++)

mapping[x - 1 + i][y - 1 + j] = basesaving3[i][j];

for (int i = 0; i <= 21; i++)

for (int j = 0; j <= 11; j++)

{

mapping[i][j] += mapsaving[i][j];

if (mapping[i][j] == 2)//如果为2，则表示出现了重合的情况

{

for (int k = 0; k <= 21; k++)//把mapping数组清空

for (int l = 0; l <= 11; l++)

mapping[k][l] = 0;

return;

}

}

//如果没有重合，则可以正常旋转

for (int i = 0; i < 3; i++)

for (int j = 0; j < 3; j++)

basesaving1[i][j] = basesaving3[i][j];//注意改变存储的数据

for (int i = 0; i < 3; i++)

for (int j = 0; j < 3; j++)

mapmoving[x - 1 + i][y - 1 + j] = basesaving3[i][j];

for (int k = 0; k <= 21; k++)//把mapping数组清空

for (int l = 0; l <= 11; l++)

mapping[k][l] = 0;

//刷新画面

system("cls");

}

}

}

void godowner(int here)//将here行以上的方块全部下移一格

{

//从here-1行开始判断

for (int godowner\_i = here - 1; godowner\_i >= 1; godowner\_i--)

for (int godowner\_j = 1; godowner\_j <= 10; godowner\_j++)

if (mapsaving[godowner\_i][godowner\_j] == 1)

{

mapsaving[godowner\_i + 1][godowner\_j] = 1;

mapsaving[godowner\_i][godowner\_j] = 0;

}

}

void disappear()//消除方块

{

int disappear\_i, disappear\_j;

//从最底层开始判断能否发生消除

for (disappear\_i = 20; disappear\_i > 0; disappear\_i--)

{

for (disappear\_j = 1; disappear\_j <= 10; disappear\_j++)

{

if (mapsaving[disappear\_i][disappear\_j] != 1)//该位置存在方块

break;//跳过本轮判断

if (disappear\_j == 10)//j为10表示本行全部为方块

{

//发生消除

for (int disappear\_k = 1; disappear\_k <= 10; disappear\_k++)

mapsaving[disappear\_i][disappear\_k] = 0;

//将本行以上的方块全部下移一个单位

godowner(disappear\_i);

//分数增加

point += 100;

//如果本轮成功消除，则i的值保持不变

disappear\_i += 1;

break;

}

}

}

}

void construction(int basing1[][3], int basing2[][3], int basing3[][3], int basing4[][3], int basing5[][3], int basing6[][4], int basing7[][4])//随机生成方块

{

moving = 1;//新方块生成

type = typed;

typed = rand(1, 7);

int k = rand(0, 7);

x = 2;

y = k + 2;

switch (type)

{

case 1:

for (int i = 1; i <= 3; i++)

for (int j = 1; j <= 3; j++)

{

mapmoving[i][j + k] += basing1[i - 1][j - 1];

basesaving1[i - 1][j - 1] = basing1[i - 1][j - 1];

}

break;

case 2:

for (int i = 1; i <= 3; i++)

for (int j = 1; j <= 3; j++)

{

mapmoving[i][j + k] += basing2[i - 1][j - 1];

basesaving1[i - 1][j - 1] = basing2[i - 1][j - 1];

}

break;

case 3:

for (int i = 1; i <= 3; i++)

for (int j = 1; j <= 3; j++)

{

mapmoving[i][j + k] += basing3[i - 1][j - 1];

basesaving1[i - 1][j - 1] = basing3[i - 1][j - 1];

}

break;

case 4:

for (int i = 1; i <= 3; i++)

for (int j = 1; j <= 3; j++)

{

mapmoving[i][j + k] += basing4[i - 1][j - 1];

basesaving1[i - 1][j - 1] = basing4[i - 1][j - 1];

}

break;

case 5:

for (int i = 1; i <= 3; i++)

for (int j = 1; j <= 3; j++)

{

mapmoving[i][j + k] += basing5[i - 1][j - 1];

basesaving1[i - 1][j - 1] = basing5[i - 1][j - 1];

}

break;

case 6:

k = rand(0, 8);

x = 1;

y = 2 + k;

for (int i = 1; i <= 4; i++)

for (int j = 0; j < 4; j++)

mapmoving[i][j + k] += basing6[i - 1][j];

break;

case 7:

k = rand(0, 6);

x = 1;

y = 2 + k;

for (int i = 0; i < 4; i++)

for (int j = 1; j <= 4; j++)

{

mapmoving[i][j + k] += basing7[i][j - 1];

basesaving2[i][j - 1] = basing7[i][j - 1];

}

break;

}

for (int construction\_over1 = 1; construction\_over1 <= 20; construction\_over1++)

for (int construction\_over2 = 1; construction\_over2 <= 10; construction\_over2++)

if (mapsaving[construction\_over1][construction\_over2] + mapmoving[construction\_over1][construction\_over2] == 2)

{

system("cls");

cout << "游戏结束" << endl << "分数：" << point << endl;

Sleep(10000);

}

}

void godown()//方块下移

{

int godownchecking = 0;

for (int i = 20; i > 0; i--)//从最底层开始判断是否存在方块

for (int j = 1; j < 11; j++)//从第一列开始判断

if (mapmoving[i][j] == 1)//1表示该位置存在方块

if (mapsaving[i + 1][j] == 1)//如果存在方块的正下方存在方块

godownchecking = 1;//准备碰撞

if (godownchecking == 1)//开始相连

{

for (int i = 1; i <= 20; i++)

for (int j = 1; j <= 10; j++)

{

if (mapmoving[i][j] == 1)

{

mapsaving[i][j] = 1;//下落的方块转换为固定的方块

mapmoving[i][j] = 0;//下落的方块消失

}

}

moving = 0;//不存在移动的方块

speedup = 0;//加速下落状态关闭

//如果方块成功相连接了，则需要判断是否可以发生消除

disappear();

}

else

//无法连接，则需要向下移动一个单位

{

//注意要从下往上扫描

for (int i = 20; i > 0; i--)

for (int j = 1; j < 11; j++)

if (mapmoving[i][j] == 1)//此位置存在方块

{

mapmoving[i + 1][j] = 1;//方块到下一格

mapmoving[i][j] = 0;//原位置方块消失

}

x += 1;

}

}

void left()//向左移动

{

int lefts = 0;

for (int i = 20; i > 0; i--)//从最底层开始判断是否存在方块

for (int j = 1; j < 11; j++)//从第一列开始判断

if (mapmoving[i][j] == 1)//1表示该位置存在这个在移动的方块

if (mapsaving[i][j - 1] == 1)//如果正在移动的方块左边存在方块

lefts = 1;

if (lefts == 1)//k=1表示移动方块左边存在方块，因此无法向左移动

{

reflesh();

}

else

{

//可以正常向左移动

for (int j = 1; j <= 10; j++)

for (int i = 20; i >= 1; i--)

{

if (mapmoving[i][j] == 1)

{

mapmoving[i][j - 1] = 1;

mapmoving[i][j] = 0;

}

}

y -= 1;

//刷新画面

reflesh();

}

}

void right()//向右移动

{

int rights = 0;

for (int i = 20; i > 0; i--)//从最底层开始判断是否存在方块

for (int j = 10; j > 0; j--)//从第十列开始判断

if (mapmoving[i][j] == 1)//1表示该位置存在方块

if (mapsaving[i][j + 1] == 1)//如果存在方块的右边存在方块

rights = 1;

if (rights == 1)//k=1表示右边存在方块，则无法向左移动

{

reflesh();

}

else

{

for (int j = 10; j >= 1; j--)

for (int i = 20; i >= 1; i--)

{

if (mapmoving[i][j] == 1)

{

mapmoving[i][j + 1] = 1;

mapmoving[i][j] = 0;

}

}

y += 1;

//刷新画面

reflesh();

}

}

DWORD WINAPI Action(LPVOID lpParameter)//多线程判断按键是否被按下

{

//循环，不断判断按键是否被按下

while (1)

{

//操作控制，这里用一个循环来判断输入是否合理

while (1)

{

acting = \_getch();

if (acting == 'w' || acting == 'W' || acting == 'a' || acting == 'A' || acting == 's' || acting == 'S' || acting == 'd' || acting == 'D')

break;

}

//根据按下按键，进行相应变换

switch (acting)

{

case 'a'://向左移动

left();

break;

case 'A'://向左移动

left();

break;

case 'd'://向右移动

right();

break;

case 'D'://向右移动

right();

break;

case 'w'://旋转

spin();

break;

case 'W'://旋转

spin();

break;

case 's'://加速下落

speedup = 1;

break;

case 'S'://加速下落

speedup = 1;

break;

}

}

return 0L;

}

int main()

{

//存储七种方块

int basing1[3][3] =

{

{1,0,0},

{1,1,1},

{0,0,0}

};

int basing2[3][3] =

{

{0,0,1},

{1,1,1},

{0,0,0}

};

int basing3[3][3] =

{

{1,1,0},

{0,1,1},

{0,0,0}

};

int basing4[3][3] =

{

{0,1,1},

{1,1,0},

{0,0,0}

};

int basing5[3][3] =

{

{0,1,0},

{1,1,1},

{0,0,0}

};

int basing6[4][4] =

{

{0,1,1,0},

{0,1,1,0},

{0,0,0,0},

{0,0,0,0}

};

int basing7[4][4] =

{

{0,0,0,0},

{1,1,1,1},

{0,0,0,0},

{0,0,0,0}

};

//边界的绘制

for (int i = 0; i <= 11; i++)

{

mapsaving[0][i] = 1;

mapsaving[21][i] = 1;

mapmoving[0][i] = 1;

mapmoving[21][i] = 1;

}

for (int i = 1; i <= 20; i++)

{

mapsaving[i][0] = 1;

mapsaving[i][11] = 1;

mapmoving[i][0] = 1;

mapmoving[i][11] = 1;

}

//伪随机变量获得时间

srand(time(NULL));

//多线程相关定义

HANDLE hThread = CreateThread(NULL, 0, Action, NULL, 0, NULL);

CloseHandle(hThread);

//隐藏输入光标

CONSOLE\_CURSOR\_INFO cursor\_info = { 1, 0 };

SetConsoleCursorInfo(GetStdHandle(STD\_OUTPUT\_HANDLE), &cursor\_info);

//初次方块生成

typed = rand(1, 7);

while (1)

{

//如果不存在方块，则随机生成方块

if (moving == 0)

construction(basing1, basing2, basing3, basing4, basing5, basing6, basing7);

if (speedup == 0)

{

//刷新画面

reflesh();

//等待

Sleep(1000);

}

//方块下移

godown();

}

system("pause");

return 0;

}