**同济大学计算机系**

**高级程序设计大作业报告**

**II**



**学 号 1950000**

**姓 名 一二三**

**专 业 计算机类**

**班 级 18 级计算机二班**

**授课老师 陈宇飞**

**完成日期 2020/1/1**

**拯救陈博士大作战大作业**

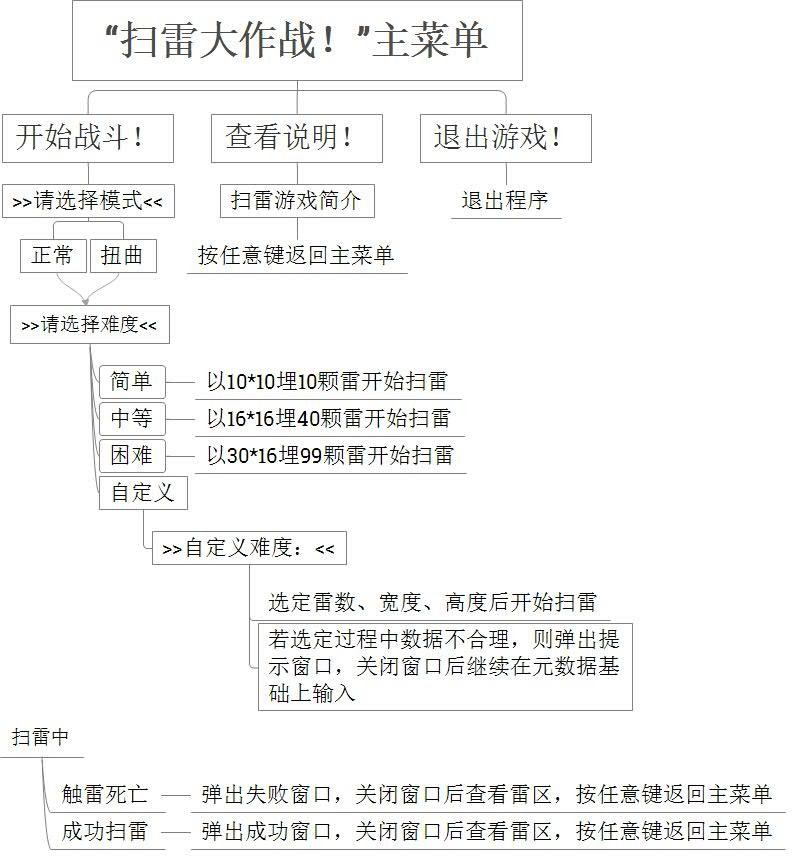
# 一、实验内容

根据 OJ 所提供的框架代码，编写并完成一个扫雷游戏，从而达到营救陈博士的最终目的。

# 二、设计思路与功能描述

## 菜单设计

* + 1. **菜单结构说明**

本次大作业项目的菜单结构与使用逻辑设计如下：

## 菜单样式说明

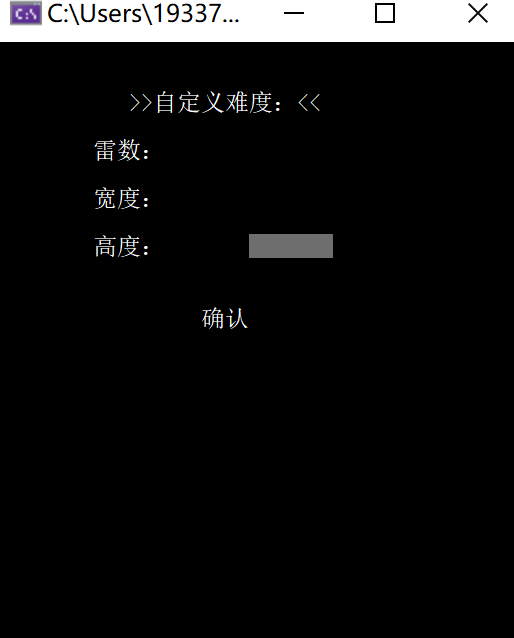
* + - 1. **与鼠标的交互**

本项目各级菜单的设计都遵从了类似于 OJ 所发布的框架的样式——即随着鼠标的移动，对应可选项的背景变白，而当鼠标移开后，该对应可选项又将会恢复至黑底白字的状态。

该功能的实现主要是将鼠标所指向位置与当前页面各个选项的网格区域所划分的位置进行对比，从而识别出当前鼠标所指向的位置。之后再将所指选项的信息传递到对应的函数，使刚被指到的选项背景变为渐变的白色，使被指向之后的选项背景变为黑色。

通过上述网格对照的方法，不仅可以达到与鼠标移动的交互动画效果，还能得到鼠标按键被按下时，用户所想要选定的目标选项。从而完成菜单界面与用户的鼠标交互。

交互的截屏展示如下：



## 自定义难度页面自由选择设置元素

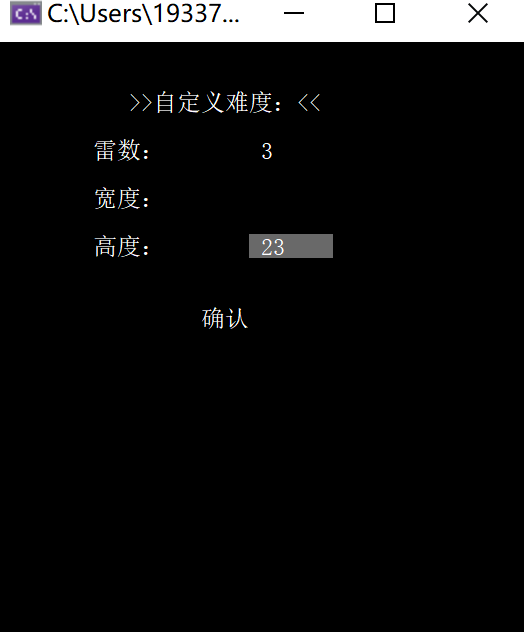
考虑到一般游戏中用户在设定自选数据时，往往会按照自己习惯的顺序对游戏参数进行设定与更改，所以在本项目自定义难度的设计环节中，加入了用户可自由选择对应数据设定的操作方式。即用户只需要单机对应的输入位置，就可直接从键盘输入对应的参数。

该功能的实现主要是预设了三个为 0 的参数（在本项目中其实是一个长度为3 的数组），当参数为 0 时，不显示该参数。而光标的位置信息可传输到数组中表示对某个数组元素的修改。

用户每输入一个数字，就将对应位置数组元素乘以 10 再加上刚才的数字， 并对页面进行一次刷新，从而达到实时输入的效果。

在用户最后按下“确认键”后，会首先对数据内容进行核查，确认数据合法后再一并赋值给对应存储该数据的变量。

自由输入的截屏展示如下：



## 弹出样式设计

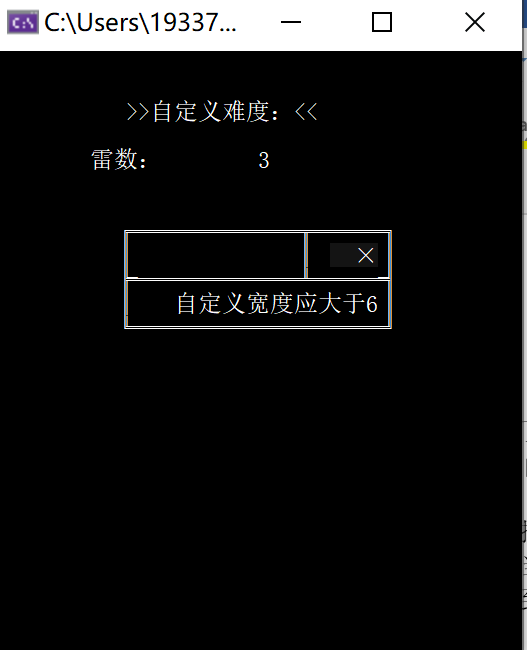
考虑到一般情况下，提醒类反馈都会采用弹窗的形式进行告知而非进入一个新菜单，所以这里对于用户输入非法数据的提醒采用一个类似于弹出小窗口的形式进行说明。

考虑到新弹出一个命令行窗口颇有几分“跳戏”的感觉，所以在这里，采用一种“伪弹窗”的方式进行实现。其具体实现方式在于，保存当前所有输出信息。以当前界面为底板，在该底板上中央偏上的位置画出弹出的内容，未被“弹窗” 覆盖的区域保持原样。

由于这里其实是生成了一个新的窗口，所以原有窗口与鼠标的交互被全部取消，可以设置新的弹窗界面的交互位置。而由于弹窗界面保持了与自定义界面较高的相似信息，所以在人的主观感受上，会觉得弹出仅仅是占用了命令行窗口的一小部分。

同样的，该弹出菜单的对应函数也被应用于扫雷过程中胜利或是失败的窗口提示。

该弹出样式的截屏展示如下：



## 扫雷游戏设计

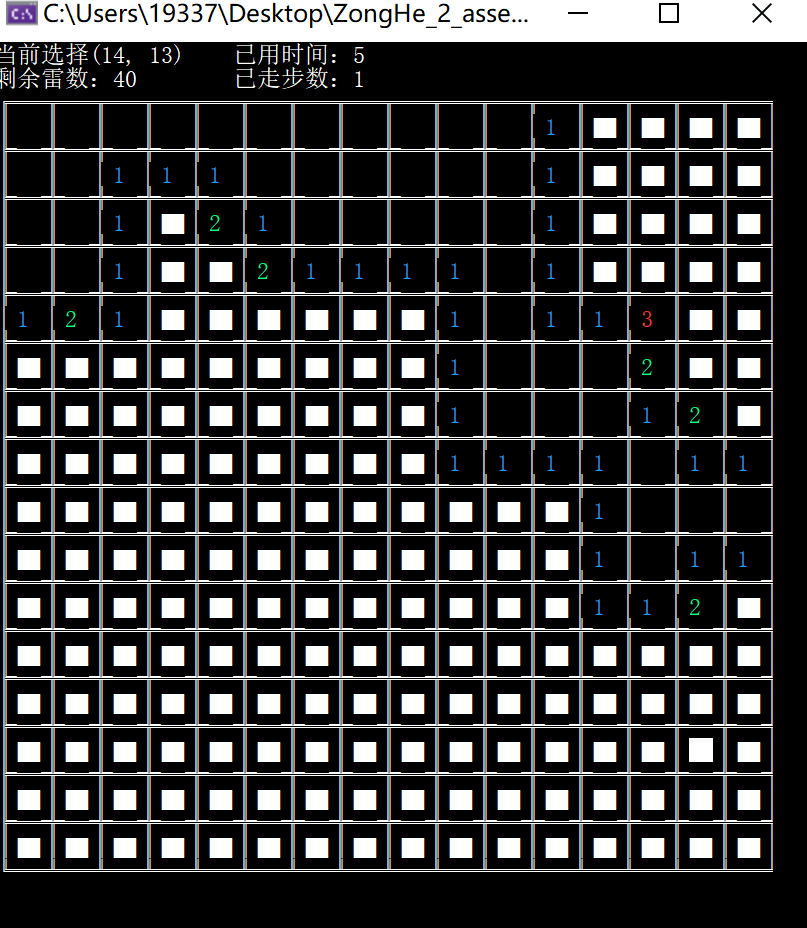
* + 1. **扫雷界面展示**

本次项目中，使用像素块“▇”来代表未被翻开的格子；使用“☆”来代表旗帜；使用“⊙”来代表地雷；使用“?”来代表不确定的格子。

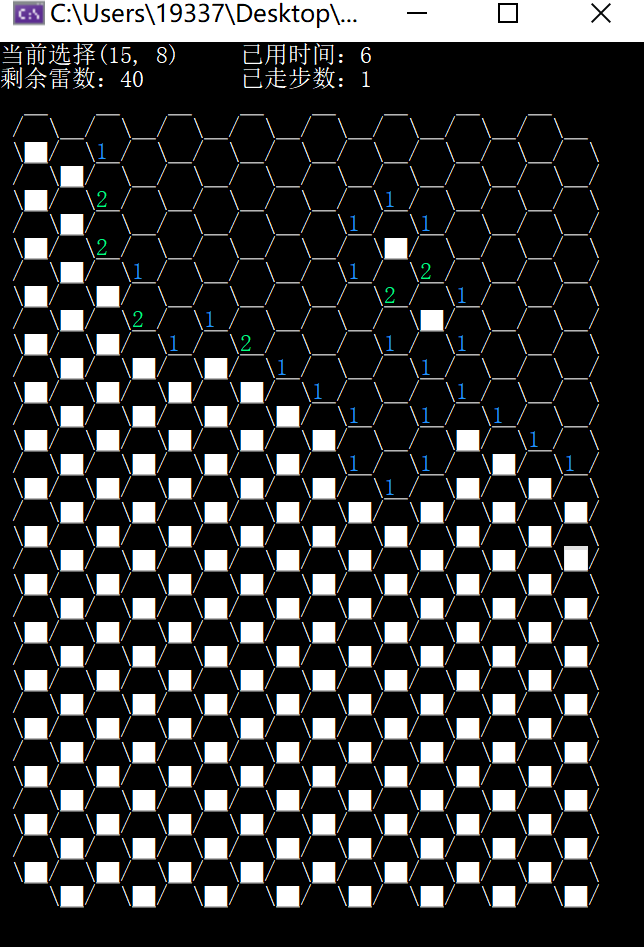
同时，扫雷边框的设定根据扫雷模式的不同选择了不同的符号来表示。标准模式下，使用专用的制表符号进行方块格的区分，例如“╬”、“═”等；而在扭曲模式下，则使用正反斜杠与上划线进行区分。

而数字的显示则是根据OJ 原有的模块设定输出，并未进行更改。

标准模式下扫雷的界面展示为：



扭曲模式下扫雷的界面展示为：



## 扫雷地图生成

由于扫雷要求第一下点击的格子与周围的一圈格子内不能有雷，所以扫雷地图应当是在用户完成第一次点击后生成的。

地图生成的具体实现方式为，将用户所选择的位置定义为安全区中心，将该安全区中心所毗邻的格子定义为安全区，之后开始进入循环体随机生成雷块。

每次埋雷方块的选择是先获得一个小于方块总数减去安全区大小、再减去已经埋雷的个数的随机非负数。然后从第（0，0）个方块开始，每逢可以埋雷的方块该随机数减一，每逢安全区或是已经埋雷的方块就跳过。直至该随机数减到 0， 则再对应的方块埋入一颗雷。此外，每当一颗雷被埋入后，就将该颗雷周围的毗邻非雷方格的数字自增 1，用于提示该数字方块周围的雷数。

此外，值得说明的是，该项目中共有两个地图数组，一个用于存储用户所看 到的界面，而另一个数组则用于存储该地图中实际的埋雷位置与非雷区块的数字。

所以用户操作的其实只是可见的数组，而操作的反馈判定则是依照实际的地图数组产生的。

## 扫雷过程的实现

用户的操作主要分为左击与右击，所以只要追踪用户的鼠标位置与执行的操作，再根据地图情况对用户的操作进行反馈，即可实现扫雷所需的用户操作。

当用户左击翻开格子时，首先判定该格子是否为雷，如果是雷的话，则弹出“游戏失败”的界面，结束游戏；如果该格子不是雷，则判定剩余未翻开的格子数是否等于雷数，如果等于的话，则说明雷已经被扫完，弹出“游戏成功”的界面，结束游戏。

若上述两个判定都通过了的话，则判定被翻开的格子是否为数字。若被翻开的格子是数字的话，则在用户可见数组内修改该格子的可见属性，同时刷新页面， 完成“翻开格子”的操作。

若被翻开的是空位置，则对该格子的周围一圈格子进行“翻开格子”操作， 如此往复迭代，直至翻开数字格或是到达边界。

而当用户右击时，则判断目前用户所指的格子的可见属性，并实现三种不同状态的转移，即（方格→旗帜→问号→方格→…）。每一次可见属性的改变都对应着一次页面的刷新，从而达到实时交互的效果。

## 提示 UI 的实现

本项目包含了 OJ 所要求的“已用时间”、“剩余雷数”以及“已走步数”的提示公告。其逐一的实现方式在于：

1. 已用时间的实现：在第一次点击左键的同时记录一次系统时间，之后每一页面刷新时，就使用当前时间与开始时间作差并输出在对应的位置。当踩雷或是扫完雷游戏结束后，则在判定为结束的那一刻记录为结束时间，时间结束之后就不再在刷新时再次计算时间，而是使用结束时间与开始时间的差值。
2. 剩余雷数的实现：本项目记录了用户的插旗数，每当用户进行插旗或是拔旗操作时，都会对相应的数值进行记录刷新。而剩余雷数就是根据系统的总雷数减去插旗数得到的。
3. 已走步数的实现：本项目在每次用户按下左键或是右键后会将已走步数的记录加一，如此往复，每当用户进行操作时，就对该数值进行一次刷新，并进行一次页面刷新，从而完成刷新展示的效果。

# 三、项目亮点与情况说明

## 项目亮点

①本项目的实现过程中均未使用 mapArray 二维数组，而是采用直接操作

mapCanvas 实现；

②本项目可选择难度等级与自定义难度；

③本项目完成了六边形扫雷加分项，并且自认为取得了较好的展示效果；

④本项目自定义难度页面设计的自认为比较具有人性化与便捷性。

⑤提示窗口采用“伪弹出窗口”的设计，自认为较为美观真实。

## 诸多情况说明

①本次报告并未详细说明代码的每个函数与具体变量细节。

相较于上一次大作业，本次项目的代码规模较大，且函数之间的逻辑重复度较高，如果是逐一说明的话，可能会占据极大的篇幅且使得本次报告较为冗长。所以在上面的代码实现模块仅对代码的实现逻辑进行了说明，并未延伸到本次项目中的具体变量名与函数名。

②本次项目代码中有较多重复度比较高的函数。

该现象在GameMenu.cpp 文件中尤为明显，甚至是完全近乎相似的鼠标背景变换的交互函数也设置了不同的函数引用。

这样做的考虑是为了模拟实际的开发环境，例如当用户后续需要主菜单和次级菜单采用不一样的显示方式或是特效交互时，运用这种相互分离的方法能更为方便的进行修改与补充，避免了大段代码重写的问题。

虽然本次大作业的要求不会再发生变动，但考虑到变量名与函数名都采取了较为规范化的命名。本着“做戏就要做全套”的原则，本项目也坚持采用了这种不同菜单采用不同交互函数各自调用实现的方法。

# 四、遇到的问题与解决方法

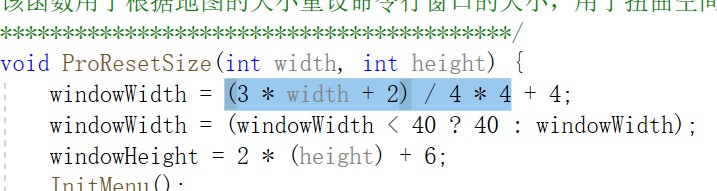
## 遇到的问题

在对扭曲模式进行胜利或是失败的弹窗提示的设计过程中，发现会存在弹窗输出错位的情况，例如下图展示的情况：

## 解决方法

对于这个问题的调试，我自认为还是花费了一番功夫的——

首先是通过不断地调试与尝试，我发现当我在扭曲模式下开启中等难度并需要弹出窗口时，就不会出现错位，而在简单与困难模式下就会出现错位。于是我开始寻找中等难度相较于其他两者的区别。我意识到可能是由于中等难度有 16

个格子，是 4 的倍数，而简单和困难模式就没有这一特点。所以我在设定窗口宽度时，作出了如下更改：

即将宽度强制定义为 4 的倍数——经过这一修改，窗口弹窗的输出就变得正常了。

反思这个问题出现的原因，我认为是我在移动光标绘制弹出窗口时，是根据当前窗口的位置进行居中计算而得到的——即采用的是相对位置，而非其他部件一样的绝对位置。

但移动光标后所输出的信息，是要占据两个字符宽度的制表符号，而每一行的制表符号与汉字的个数也有所出入。这就导致，当我使用相对位置最终却得出一个奇数开头的位置时，不同行会根据其不同的输出字符串特性进行输出，进而导致不同行之间出现错位的情况。

而之所以这一问题并未在标准模式下显露出来，是因为标准模式下，由于单元格的宽度就是 4 个字节，所以屏幕的宽度能够保持为 4 的倍数，从而使得相对位置换算后得到的开始打印弹出窗口的位置必为偶数，凑巧错开了问题；但当在扭曲模式下，将单元格宽度设置为 3 时，窗口的宽度无法保证为 4 的倍数，弹出错位的错误也就开始显露了。

最后附上解决问题之后的截屏：

# 五、心得体会

# 六、源代码

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Controller.h

Author: Root Date: 2019/12/09 Description:

定义了控制框架的头文件，用Controller命名空间防止同名污染 各函数说明：

void FlushInput() 更新输入缓冲区

char GetKeyHit() 获取键盘输入

COORD GetCursorPos() 获取鼠标当前坐标（在控制台内的坐标）

COORD GetCursorHitPos(DWORD type = 1) 获取当前鼠标点击的坐标，type将决定是获取左键/右键还是左键双击的坐标

void InitController() 初始化控制器

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ #pragma once

#ifndef CONTROLLER\_H #define CONTROLLER\_H #include <windows.h>

namespace Controller { void FlushInput(); char GetKeyHit();

COORD GetCursorPos();

COORD GetCursorHitPos(DWORD type = 1); void InitController();

}

#endif

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* GameEngine.h

Author: Root Date: 2019/12/09 Description:

定义了程序游戏模块的头文件，用GameEngine命名空间防止同名污染 各函数说明：

void InitGame() 初始化游戏

void Play() 开始游戏

void DestroyGame() 结束游戏

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ #pragma once

#ifndef GAMEENGINE\_H #define GAMEENGINE\_H #include "Graphic.h" #include "Controller.h" #include "GameMenu.h" #include <time.h> namespace GameEngine {

void InitGame(int getWidth, int getHeight, int getBoom); void Play();

void DestroyGame();

}

#endif // GAMEENGINE\_H

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* GameEnginePro.h

Author: 1950000 一二三Date: 2019/12/29 Description:

定义了程序扭曲游戏模块的头文件，用GameEnginePro命名空间防止同名污染 各函数说明：

void InitGame() 初始化游戏

void Play() 开始游戏

void DestroyGame() 结束游戏

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ #pragma once

#ifndef GAMEENGINEPRO\_H #define GAMEENGINE\_H #include "Graphic.h"

#include "Controller.h" #include "GameMenu.h" #include <time.h> namespace GameEnginePro {

void InitGame(int getWidth, int getHeight, int getBoom); void Play();

void DestroyGame();

}

#endif // GAMEENGINEPRO\_H

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* GameMenu.h

Author: Root Date: 2019/12/09 Description:

定义了程序菜单的渲染/管理模块的头文件，用GameMenu命名空间防止同名污染 各函数说明：

void InitMenu() 初始化菜单，并渲染菜单选项

void WaitChoice() 等待用户输入（选择菜单等）

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ #pragma once

#ifndef GAMEMENU\_H #define GAMEMENU\_H #include "Graphic.h" #include "Controller.h" #include "GameEngine.h" #include "GameEnginePro.h" #include <time.h>

namespace GameMenu { void InitMenu(); void WaitChoice();

void outChat(const char outString[]); void WaitChatChoice();

}

#endif // GAMEMENU\_H

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Graphic.h

Author: Root Date: 2019/12/09 Description:

定义了渲染框架的头文件，用Graphic命名空间防止同名污染 各函数说明：

void MovePos() 将当前坐标更新为入参

void ModChar() 将当前位置的字符改为新的newChar

void ModColor() 将当前位置的字符的颜色/背景(具体修改哪个由mode指定)改为新的颜色值

void PutChar() 将当前位置放置一个outChar

void PutCharWithColor() 将当前位置放置一个outChar，并设置为指定的颜色

void PutString() 将当前位置放置一个字符串outString

void PutStringWithColor() 将当前位置放置一个字符串outString，并设置颜色

void DrawRect() 将当前位置绘制一个长为width，宽为height的矩形 void DrawRectWithColor() 将当前位置绘制一个长为width，宽为height的矩形，并带颜色void ClearScreen() 强制清屏操作

void InitGraphic() 初始化图形操作，将屏幕长宽设置为width和height void DestroyGraphic() 销毁操作

void Update() 将所有操作更新至屏幕，也就是说在这里做的所有操作都不是实时的，只有调用这个函数后屏幕才 会真正的更新

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ #pragma once

#ifndef GRAPHIC\_H #define GRAPHIC\_H #include <stdio.h> #include <windows.h>

namespace Graphic {

void MovePos(SHORT x, SHORT y); void ModChar(const char newChar);

void ModColor(DWORD mode, const BYTE textR, const BYTE textG, const BYTE textB, const BYTE backR, const BYTE backG, const BYTE backB);

void PutChar(const char outChar);

void PutCharWithColor(const char outChar, const BYTE textR, const BYTE textG, const BYTE textB, const BYTE backR, const BYTE backG, const BYTE backB);

void PutString(const char\* outString);

void PutStringWithColor(const char\* outString, const BYTE textR, const BYTE textG, const BYTE textB, const BYTE backR, const BYTE backG, const BYTE backB);

void DrawRect(const SHORT width, const SHORT height);

void DrawRectWithColor(const SHORT width, const SHORT height, const BYTE textR, const BYTE textG, const BYTE textB, const BYTE backR, const BYTE backG, const BYTE backB);

void ClearScreen();

void InitGraphic(int width, int height); void DestroyGraphic();

void Update();

}

#endif

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Controller.cpp

Author: Root Date: 2019/12/09 Description:

定义了控制器框架，用Controller命名空间防止同名污染

！\*！注意：除非你能完全读懂内部都在做什么操作，否则不建议修改这个文件！\*！

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ #include "Controller.h"

namespace Controller {

COORD pos = { 0 }; // 鼠标坐标

HANDLE hStdIn = NULL; // 标准输入句柄INPUT\_RECORD inRecArr[256] = { 0 }; // 输入缓冲区数组INPUT\_RECORD inRec = { 0 }; // 最终需要获取的输入

DWORD dRead = 0; // 读取到的输入缓冲区长度

char key = '0'; // 键盘按键

const DWORD MOVE = 0, LEFT\_CLICK = 1, RIGHT\_CLICK = 2, \_DOUBLE\_CLICK = 3; // 一些常量

false

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function: updateInput(DWORD type, DWORD hitMode = MOVE) Parameter: DWORD type, hitMode

Return: bool Description:

将inRec变量设置为需要获取的输入类型type(鼠标或键盘)，如果找到需要的输入，则返回true并设置inRec变量，否则返回

hitMode将指定鼠标的点击模式，是左键单击/右键单击或双击

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool updateInput(DWORD type, DWORD hitMode = MOVE) {

// 从输入缓冲区的最后一个数据开始循环搜索

for (int i = (int)dRead; i > 0; i--) {

// 如果这个数据的类型符合输入

if (inRecArr[i - 1].EventType == type) {

// 检查hitMode 如果是MOVE则跳过switch switch (hitMode) {

case LEFT\_CLICK:

// 左键单击

if (inRecArr[i - 1].Event.MouseEvent.dwButtonState != 1) { continue;

}

break;

case RIGHT\_CLICK:

// 右键单击

if (inRecArr[i - 1].Event.MouseEvent.dwButtonState != RIGHTMOST\_BUTTON\_PRESSED) { continue;

}

break;

case \_DOUBLE\_CLICK:

// 双击

if (inRecArr[i - 1].Event.MouseEvent.dwEventFlags != DOUBLE\_CLICK) { continue;

}

break;

}

// 设置inRec并返回true表示找到需要的输入

inRec = inRecArr[i - 1]; return true;

}

}

return false;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: FlushInput()

Parameter: None(void) Return: None(void) Description:

刷新输入缓冲区读取缓冲区的全部数据

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ void FlushInput() {

if (WaitForSingleObject(hStdIn, 0) == WAIT\_OBJECT\_0) {

// 如果缓冲区内有数据，则读出数据，最大256个

ReadConsoleInputA(hStdIn, &inRecArr[0], 256, &dRead);

}

else {

}

// 否则将Read置为0 dRead = 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: GetKeyHit()

Parameter: None(void) Return: char Description:

获取键盘输入，如果没有键盘击中，则返回'\0'

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ char GetKeyHit() {

if (updateInput(KEY\_EVENT)) {

if (inRec.Event.KeyEvent.bKeyDown) {

key = (char)inRec.Event.KeyEvent.wVirtualKeyCode; return key;

}

}

return '\0';

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: GetCursorPos()

Parameter: None(void) Return: COORD Description:

获取鼠标输入，将一直返回上一次更新的坐标

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ COORD GetCursorPos() {

if (updateInput(MOUSE\_EVENT)) {

pos = inRec.Event.MouseEvent.dwMousePosition;

}

return pos;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: GetCursorHitPos()

Parameter: DWORD type Return: COORD Description:

获取鼠标点击输入，type将指定是左键单击(1)右键单击(2)双击(3)，若没有点击，则将一直返回0,0

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ COORD GetCursorHitPos(DWORD type) {

if (updateInput(MOUSE\_EVENT, type)) {

return inRec.Event.MouseEvent.dwMousePosition;

}

return { 0,0 };

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: InitController()

Parameter: None(void) Return: None(void) Description:

初始化控制器

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ void InitController() {

hStdIn = GetStdHandle(STD\_INPUT\_HANDLE); // 获取标准输入句柄

SetConsoleMode(hStdIn, ENABLE\_EXTENDED\_FLAGS | ENABLE\_WINDOW\_INPUT | ENABLE\_MOUSE\_INPUT);

// 将控制台设置为允许鼠标输入

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* GameEngine.cpp

Author: Root Date: 2019/12/09 Description:

定义了程序游戏模块，用GameEngine命名空间防止同名污染，是你主要需要修改的文件

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ #include "GameEngine.h"

namespace GameEngine {

// 引入两个命名空间using namespace Graphic; using namespace Controller;

size\_t mapWidth = 0, mapHeight = 0; // 地图宽度和高度

UCHAR mapCanvas[50 \* 25]; // 地图数组指针，若做基础项请不要直接尝试操作这个

指针

COORD pos = { 0 }, posOld = { 0 }; // 鼠标的位置COORD posChoice = { 0 }, posChoiceOld = { 0 }; // 选中地图坐标的位置char key = '\0'; // 键盘的按键

bool gameFlag = false; // 游戏运行状态

size\_t frame = 0; // 已渲染帧数

clock\_t tic = clock(); // 游戏开始时刻

int operation = 0; // 执行的操作

bool isFirst = true; // 是否为第一步

//int mapArray[50][50]; // 这里直接操作mapCanvas，不使用mapArray数组

// 如果你有新加的变量，建议加在下方和预置变量做区别

int boomTotal = 0; // 定义雷的总数

int safeZone = 0; // 定义不可能为雷的安全区的中心位置

UCHAR mapVisible[50 \* 25]; // 定义当前可见的地图信息

int remainBlock = 0; // 定义未点开的砖块数

int flagNum = 0; // 定义已经插旗的数目

int alreadySlice = 1; // 定义已走的步数

double timeStart, timeNow, timeEnd = 1000; // 定义开始时的时间

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: renderMap()

Parameter: None(void) Return: None(void) Description:

绘制当前地图

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ void renderMap() {

// 在0,0处放置当前选择位置的字符串，注意结尾有空格留白，可以思考为什么要加这么多空格

MovePos(0, 1);

char strPos[32] = "";

sprintf\_s(strPos, "当前选择(%u, %u) ", posChoice.X, posChoice.Y); PutString(strPos);

Update();

// 在0,20处放置字符串，显示时间

MovePos(20, 1);

char strTime[32] = ""; if (isFirst) {

sprintf\_s(strTime, "已用时间：%u ", 0);

}

else {

}

sprintf\_s(strTime, "已用时间：%u ", (int)(timeEnd - timeStart) / 1000);

PutString(strTime); Update();

// 在1,0处放置字符串，显示剩余雷数

MovePos(0, 2);

char strBoom[32] = "";

sprintf\_s(strBoom, "剩余雷数：%d ", boomTotal - flagNum); PutString(strBoom);

Update();

// 在1,20处放置字符串，显示已走步数

MovePos(20, 2);

char strSlice[32] = "";

sprintf\_s(strSlice, "已走步数：%u ", (isFirst ? 0 : alreadySlice));

PutString(strSlice); Update();

// 从行开始遍历

for (size\_t i = 0; i < mapHeight + 1; i++) {

// 移动坐标至行首 并绘制行首的字符

MovePos(0, (SHORT)i \* 2 + 3);

if (i == 0) {

PutString("╔");

}

else if (i == mapHeight) { PutString("╚");

}

else {

}

PutString("╠");

// 遍历列，绘制边界

for (size\_t j = 0; j < mapWidth; j++) {

// 移动至确切的坐标，绘制方格的边界

MovePos(2 + (SHORT)j \* 8, (SHORT)i \* 2 + 3);

if (i == 0) {

if (j < mapWidth - 1) {

PutString("═══╦");

}

else {

}

}

PutString("═══╗");

else if (i == mapHeight) {

if (j < mapWidth - 1) {

PutString("═══╩");

}

else {

}

else {

}

PutString("═══╝");

if (j < mapWidth - 1) {

PutString("═══╬");

}

else {

}

}

}

PutString("═══╣");

将其转换成字符串

// 绘制地雷地图

if (i > 0 && i < mapHeight + 1) {

// 移动至行首，绘制边界字符

MovePos(0, (SHORT)i \* 2 + 2);

PutString("║");

// 遍历列 绘制地雷

for (size\_t j = 0; j < mapWidth; j++) {

MovePos(2 + (SHORT)j \* 5, (SHORT)i \* 2 + 2); // 移动至确切坐标

const size\_t mapIndex = (i - 1) \* mapWidth + j; // 确定地图数组的下标

char numMap[8] = " "; // 确定数字字符串

numMap[1] = '0' + mapCanvas[mapIndex]; // 当mapVisible[mapIndex]为1到8时，

switch (mapVisible[mapIndex]) { case 0:

switch (mapCanvas[mapIndex]) { case 0:

case 1:

case 2:

case 3:

case 4:

// 0的时候放置空白PutString(" "); break;

// 从1开始绘制数字 带颜色PutStringWithColor(numMap, 30, 144, 255, 0, 0, 0); break;

PutStringWithColor(numMap, 0, 255, 127, 0, 0, 0); break;

PutStringWithColor(numMap, 255, 48, 48, 0, 0, 0); break;

PutStringWithColor(numMap, 72, 61, 139, 0, 0, 0);

case 5:

case 6:

case 7:

case 8:

break;

PutStringWithColor(numMap, 255, 105, 180, 0, 0, 0); break;

PutStringWithColor(numMap, 148, 0, 211, 0, 0, 0); break;

PutStringWithColor(numMap, 139, 0, 0, 0, 0, 0); break;

PutStringWithColor(numMap, 139, 34, 82, 0, 0, 0); break;

case 11:

// 11为地雷

PutStringWithColor(" ⊙", 255, 215, 0, 0, 0, 0); break;

case 9:

}

break;

// 9为砖块PutString(" ▇"); break;

case 10:

// 10为旗帜

PutStringWithColor(" ☆", 255, 255, 255, 0, 0, 0); break;

case 12:

//12为暂定

PutStringWithColor(" ？", 255, 255, 255, 0, 0, 0); break;

}

MovePos(5 + (SHORT)j \* 5, (SHORT)i \* 2 + 2);

PutString("║");

}

}

}

Update(); // 将地图更新到屏幕

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: updateMap()

Parameter: None(void) Return: None(void) Description:

将二维数组的数据复制到一维数组中

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*void updateMap() {

memcpy\_s(mapCanvas, mapWidth \* mapHeight, mapArray, mapWidth \* mapHeight);

}\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function: InitGame(int getWidth,int getHeight, int getBoom) Parameter: getWidth(int),getHeight(int),getBoom(int)

Return: None(void) Description:

初始化游戏

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void InitGame(int getWidth,int getHeight, int getBoom) {

ClearScreen(); // 清屏

mapWidth = getWidth;

mapHeight = getHeight; // 设置宽高

boomTotal = getBoom;

for (int i = 0; i < mapWidth \* mapHeight; i++) { // 初始化两个数组

mapCanvas[i] = 0;

mapVisible[i] = 9;

}

//updateMap(); // 更新地图remainBlock = mapWidth \* mapHeight; renderMap(); // 绘制地图

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: renderChoiceBackground(COORD choice) Parameter: COORD choice

Return: None(void) Description:

绘制选中的地块的背景

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ void renderChoiceBackground(COORD choice) {

const SHORT x = choice.X, y = choice.Y;

const BYTE alpha = 255 - (BYTE)((frame % 50) \* 5); // 定义背景白色当前的颜色值，通过已渲染的帧数进行运算（此处也可以改成使用时间）

const size\_t mapIndex = (size\_t)y \* mapWidth + (size\_t)x; // 确定地图下标

MovePos(2 + x \* 4, y \* 2 + 3);

// 以下内容同renderMap中绘制地雷部分相同，不详做介绍

char numMap[8] = " ";

numMap[0] = '0' + mapCanvas[mapIndex]; switch (mapVisible[mapIndex]) {

case 0:

switch (mapCanvas[mapIndex]) { case 0:

PutStringWithColor(" ", 255, 255, 255, alpha, alpha, alpha); break;

case 1:

PutStringWithColor(numMap, 30, 144, 255, alpha, alpha, alpha); break;

case 2:

case 3:

case 4:

case 5:

case 6:

case 7:

case 8:

PutStringWithColor(numMap, 0, 255, 127, alpha, alpha, alpha); break;

PutStringWithColor(numMap, 255, 48, 48, alpha, alpha, alpha); break;

PutStringWithColor(numMap, 72, 61, 139, alpha, alpha, alpha); break;

PutStringWithColor(numMap, 255, 105, 180, alpha, alpha, alpha); break;

PutStringWithColor(numMap, 148, 0, 211, alpha, alpha, alpha); break;

PutStringWithColor(numMap, 139, 0, 0, alpha, alpha, alpha); break;

PutStringWithColor(numMap, 139, 34, 82, alpha, alpha, alpha); break;

case 11:

PutStringWithColor("⊙", 255, 215, 0, alpha, alpha, alpha); break;

case 9:

}

break;

PutStringWithColor("▇", 255, 255, 255, alpha, alpha, alpha); break;

case 10:

PutStringWithColor("☆", 255, 255, 255, alpha, alpha, alpha); break;

case 12:

PutStringWithColor("？", 255, 255, 255, alpha, alpha, alpha); break;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: clearChoiceBackground(COORD choice) Parameter: COORD choice

Return: None(void) Description:

清除choice中指示位置的背景，功能同上，不详做介绍

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ void clearChoiceBackground(COORD choice) {

const SHORT x = choice.X, y = choice.Y;

const size\_t mapIndex = (size\_t)y \* mapWidth + (size\_t)x; MovePos(2 + x \* 4, y \* 2 + 3);

char numMap[8] = " ";

numMap[0] = '0' + mapCanvas[mapIndex]; switch (mapVisible[mapIndex]) {

case 0:

switch (mapCanvas[mapIndex]) { case 0:

case 1:

case 2:

case 3:

case 4:

case 5:

case 6:

case 7:

case 8:

PutStringWithColor(" ", 255, 255, 255, 0, 0, 0); break;

PutStringWithColor(numMap, 30, 144, 255, 0, 0, 0); break;

PutStringWithColor(numMap, 0, 255, 127, 0, 0, 0); break;

PutStringWithColor(numMap, 255, 48, 48, 0, 0, 0); break;

PutStringWithColor(numMap, 72, 61, 139, 0, 0, 0); break;

PutStringWithColor(numMap, 255, 105, 180, 0, 0, 0); break;

PutStringWithColor(numMap, 148, 0, 211, 0, 0, 0); break;

PutStringWithColor(numMap, 139, 0, 0, 0, 0, 0); break;

PutStringWithColor(numMap, 139, 34, 82, 0, 0, 0); break;

case 11:

PutStringWithColor("⊙", 255, 215, 0, 0, 0, 0); break;

case 9:

}

break;

PutStringWithColor("▇", 255, 255, 255, 0, 0, 0); break;

case 10:

PutStringWithColor("☆", 255, 255, 255, 0, 0, 0); break;

case 12:

PutStringWithColor("？", 255, 255, 255, 0, 0, 0); break;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: checkChoice()

Parameter: None(void) Return: None(void) Description:

检查输入和操作，并设置相关变量

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ void checkChoice() {

FlushInput(); // 刷新输入缓冲区

pos = GetCursorPos(); // 获取鼠标坐标COORD hitLeftPos = GetCursorHitPos(); // 获取左键单击坐标COORD hitRightPos = GetCursorHitPos(2); // 获取右键单击坐标key = GetKeyHit(); // 获取键盘输入

operation = 0; // 当前操作 (0无操作，1为挖开地块，2为标值旗帜，或

者自己指定)

// 检查键盘输入

switch (key) {

case VK\_ESCAPE:

// ESC键，将游戏运行状态置否以退出游戏gameFlag = false;

break;

// 上下左右键移动选中的坐标

case VK\_UP:

if (posChoice.Y > 0)posChoice.Y--; break;

case VK\_RIGHT:

if (posChoice.X < (SHORT)mapWidth - 1)posChoice.X++; break;

case VK\_DOWN:

if (posChoice.Y < (SHORT)mapHeight - 1)posChoice.Y++; break;

case VK\_LEFT:

if (posChoice.X > 0)posChoice.X--; break;

// 回车和空格分别对应操作1和2 case VK\_RETURN:

operation = 1; break;

case VK\_SPACE:

operation = 2; break;

}

// 将鼠标控制台的坐标放缩至地图坐标

const size\_t mouseY = (size\_t)((size\_t)pos.Y + 1) / 2 - 2; const size\_t mouseX = (size\_t)((size\_t)pos.X - 1) / 4;

if (mouseY < mapHeight && mouseX < mapWidth) {

// 若没有超边界，更新选择的坐标posChoice.X = (SHORT)mouseX; posChoice.Y = (SHORT)mouseY;

}

// 左键和右键分别对应操作1和2

if (hitLeftPos.X == pos.X && hitLeftPos.Y == pos.Y) { operation = 1;

}

else if (hitRightPos.X == pos.X && hitRightPos.Y == pos.Y) { operation = 2;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: isInArray()

Parameter: ins(int),arr(int[9]) Return: 1/0 (bool) Description:

判断ins是否和arr中的某个元素相等，相等的话返回true

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ bool isInArray(int ins, int arr[]) {

for (int i = 0; i < 9; i++) { if (ins == arr[i]) {

return true;

}

}

return false;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: initMap()

Parameter: None(void) Return: None(void) Description:

初始化函数，请通过posChoice变量判断操作，并操作mapArray数组，请不要操作其他程序预置好的变量

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ void initMap() {

int boomCounter = 0; //定义已经埋下雷的计数器

int tempBomb = 0; //定义可能为雷的工作点

UCHAR\* pin = mapCanvas; //定义埋雷工作指针

int safePlace[9] = { -1,-1,-1,-1,-1,-1,-1,-1,-1 }; //定义安全区具体范围if (safeZone >= mapWidth) {

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

safePlace[i] = safeZone - mapWidth - 1 + i;

}

}

for (int i = 3; i < 6; i++) {

safePlace[i] = safeZone - 4 + i;

}

if (safeZone < (mapHeight - 1) \* mapWidth) { for (int i = 6; i < 9; i++) {

safePlace[i] = safeZone + mapWidth - 7 + i;

}

}

if (safeZone % mapWidth == 0) { for (int i = 0; i < 3; i++) {

safePlace[3 \* i] = -1;

}

}

if (safeZone % mapWidth == mapWidth - 1) {

for (int i = 0; i < 3; i++) { safePlace[3 \* i + 2] = -1;

}

}

int safeSize = 9; //定义安全区的大小

if (posChoice.X == 0 || posChoice.X == mapWidth - 1) { safeSize -= 3;

}

if (posChoice.Y == 0 || posChoice.Y == mapHeight - 1) { safeSize = (safeSize == 9 ? 6 : 4);

}

srand((unsigned)time(NULL)); //定义时间作为随机数种子for (;;) {

pin = mapCanvas;

tempBomb = rand() % (mapWidth \* mapHeight - safeSize - boomCounter); for (int i = 0;;) {

for (;;) {

if (\*pin >= 9) { //判断没有埋雷pin++;

}

else if (isInArray((int)(pin-mapCanvas),safePlace)) {

pin++; //防止把雷埋在安全区

}

else {

}

}

break;

行赋值

1));

if (i == tempBomb) { //这里根据埋雷的情况对周围位块进

\*pin = 11;

if ((pin - mapCanvas) % mapWidth == 0) {

if ((pin - mapCanvas) >= 0 && (pin - mapCanvas) < mapWidth) {

\*(pin + 1) = (\*(pin + 1) == 11 ? 11 : (\*(pin + 1) + 1));

\*(pin + mapWidth) = (\*(pin + mapWidth) == 11 ? 11 : (\*(pin + mapWidth) + 1));

\*(pin + mapWidth + 1) = (\*(pin + mapWidth + 1) == 11 ? 11 : (\*(pin + mapWidth + 1) +

mapHeight \* mapWidth) {

1));

}

else if ((pin - mapCanvas) >= mapWidth \* (mapHeight - 1) && (pin - mapCanvas) <

\*(pin + 1) = (\*(pin + 1) == 11 ? 11 : (\*(pin + 1) + 1));

\*(pin - mapWidth) = (\*(pin - mapWidth) == 11 ? 11 : (\*(pin - mapWidth) + 1));

\*(pin - mapWidth + 1) = (\*(pin - mapWidth + 1) == 11 ? 11 : (\*(pin - mapWidth + 1) +

1));

1));

}

else {

}

}

\*(pin - mapWidth) = (\*(pin - mapWidth) == 11 ? 11 : (\*(pin - mapWidth) + 1));

\*(pin - mapWidth + 1) = (\*(pin - mapWidth + 1) == 11 ? 11 : (\*(pin - mapWidth + 1) +

\*(pin + 1) = (\*(pin + 1) == 11 ? 11 : (\*(pin + 1) + 1));

\*(pin + mapWidth) = (\*(pin + mapWidth) == 11 ? 11 : (\*(pin + mapWidth) + 1));

\*(pin + mapWidth + 1) = (\*(pin + mapWidth + 1) == 11 ? 11 : (\*(pin + mapWidth + 1) +

1));

else if ((pin - mapCanvas) % mapWidth == mapWidth - 1) {

if ((pin - mapCanvas) >= 0 && (pin - mapCanvas) < mapWidth) {

\*(pin - 1) = (\*(pin - 1) == 11 ? 11 : (\*(pin - 1) + 1));

\*(pin + mapWidth) = (\*(pin + mapWidth) == 11 ? 11 : (\*(pin + mapWidth) + 1));

\*(pin + mapWidth - 1) = (\*(pin + mapWidth - 1) == 11 ? 11 : (\*(pin + mapWidth - 1) +

mapHeight \* mapWidth) {

1));

}

else if ((pin - mapCanvas) >= mapWidth \* (mapHeight - 1) && (pin - mapCanvas) <

\*(pin - 1) = (\*(pin - 1) == 11 ? 11 : (\*(pin - 1) + 1));

\*(pin - mapWidth) = (\*(pin - mapWidth) == 11 ? 11 : (\*(pin - mapWidth) + 1));

\*(pin - mapWidth - 1) = (\*(pin - mapWidth - 1) == 11 ? 11 : (\*(pin - mapWidth - 1) +

1));

1));

}

else {

}

\*(pin - mapWidth) = (\*(pin - mapWidth) == 11 ? 11 : (\*(pin - mapWidth) + 1));

\*(pin - mapWidth - 1) = (\*(pin - mapWidth - 1) == 11 ? 11 : (\*(pin - mapWidth - 1) +

\*(pin - 1) = (\*(pin - 1) == 11 ? 11 : (\*(pin - 1) + 1));

\*(pin + mapWidth) = (\*(pin + mapWidth) == 11 ? 11 : (\*(pin + mapWidth) + 1));

\*(pin + mapWidth - 1) = (\*(pin + mapWidth - 1) == 11 ? 11 : (\*(pin + mapWidth - 1) +

}

else if ((pin - mapCanvas) >= 0 && (pin - mapCanvas) < mapWidth) {

\*(pin - 1) = (\*(pin - 1) == 11 ? 11 : (\*(pin - 1) + 1));

\*(pin + 1) = (\*(pin + 1) == 11 ? 11 : (\*(pin + 1) + 1));

\*(pin + mapWidth) = (\*(pin + mapWidth) == 11 ? 11 : (\*(pin + mapWidth) + 1));

\*(pin + mapWidth - 1) = (\*(pin + mapWidth - 1) == 11 ? 11 : (\*(pin + mapWidth - 1) + 1));

\*(pin + mapWidth + 1) = (\*(pin + mapWidth + 1) == 11 ? 11 : (\*(pin + mapWidth + 1) + 1));

mapWidth) {

}

else if ((pin - mapCanvas) >= mapWidth \* (mapHeight - 1) && (pin - mapCanvas) < mapHeight \*

\*(pin - 1) = (\*(pin - 1) == 11 ? 11 : (\*(pin - 1) + 1));

\*(pin + 1) = (\*(pin + 1) == 11 ? 11 : (\*(pin + 1) + 1));

\*(pin - mapWidth) = (\*(pin - mapWidth) == 11 ? 11 : (\*(pin - mapWidth) + 1));

\*(pin - mapWidth - 1) = (\*(pin - mapWidth - 1) == 11 ? 11 : (\*(pin - mapWidth - 1) + 1));

\*(pin - mapWidth + 1) = (\*(pin - mapWidth + 1) == 11 ? 11 : (\*(pin - mapWidth + 1) + 1));

}

else {

}

}

}

else {

}

break;

pin++; i++;

\*(pin - 1) = (\*(pin - 1) == 11 ? 11 : (\*(pin - 1) + 1));

\*(pin + 1) = (\*(pin + 1) == 11 ? 11 : (\*(pin + 1) + 1));

\*(pin - mapWidth) = (\*(pin - mapWidth) == 11 ? 11 : (\*(pin - mapWidth) + 1));

\*(pin - mapWidth - 1) = (\*(pin - mapWidth - 1) == 11 ? 11 : (\*(pin - mapWidth - 1) + 1));

\*(pin - mapWidth + 1) = (\*(pin - mapWidth + 1) == 11 ? 11 : (\*(pin - mapWidth + 1) + 1));

\*(pin + mapWidth) = (\*(pin + mapWidth) == 11 ? 11 : (\*(pin + mapWidth) + 1));

\*(pin + mapWidth - 1) = (\*(pin + mapWidth - 1) == 11 ? 11 : (\*(pin + mapWidth - 1) + 1));

\*(pin + mapWidth + 1) = (\*(pin + mapWidth + 1) == 11 ? 11 : (\*(pin + mapWidth + 1) + 1));

boomCounter++;

if (boomCounter == boomTotal) { break;

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: gameOver()

Parameter: Nnoe(void) Return: None(void) Description:

游戏失败

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ void gameOver() {

timeEnd = GetTickCount64(); GameMenu::outChat("很遗憾，您失败了 "); GameMenu::WaitChatChoice(); ClearScreen();

for (int i = 0; i < mapWidth \* mapHeight; i++) { if (\*(mapCanvas + i) == 11) {

\*(mapVisible + i) = 0;

}

}

renderMap();

bool quitFlag = true; // 强制退出

while (quitFlag) {

FlushInput(); // 刷新输入缓冲区

pos = GetCursorPos(); // 获取鼠标坐标COORD hitLeftPos = GetCursorHitPos(); // 获取左键单击坐标COORD hitRightPos = GetCursorHitPos(2); // 获取右键单击坐标key = GetKeyHit(); // 获取键盘输入

// 检查键盘输入

if (key) {

quitFlag = false;

}

// 将鼠标控制台的坐标放缩至地图坐标

const size\_t mouseY = (size\_t)((size\_t)pos.Y + 1) / 2 - 2; const size\_t mouseX = (size\_t)((size\_t)pos.X - 1) / 4;

if (mouseY < mapHeight && mouseX < mapWidth) {

// 若没有超边界，更新选择的坐标posChoice.X = (SHORT)mouseX; posChoice.Y = (SHORT)mouseY;

}

// 左键和右键分别对应操作1和2

if (hitLeftPos.X == pos.X && hitLeftPos.Y == pos.Y) { quitFlag = false;

}

else if (hitRightPos.X == pos.X && hitRightPos.Y == pos.Y) { quitFlag = false;

}

}

gameFlag = false;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: digBlock(int digPlace)

Parameter: digPlace(int) Return: None(void) Description:

实现翻开操作

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ void digBlock(int digPlace) {

if (isFirst) {

// 如果是第一步走，则先初始化地图，注意不要在落点设置一个地雷safeZone = posChoice.Y \* mapWidth + posChoice.X; //设立安全区initMap();

isFirst = false; // 将第一步设置为否

timeStart = GetTickCount64();

}

if (\*(mapVisible + digPlace) == 0) { return;

}

if (\*(mapCanvas + digPlace) == 11) { gameOver();

}

else if (\*(mapCanvas + digPlace) >= 1 && \*(mapCanvas + digPlace) <= 8) {

\*(mapVisible + digPlace) = 0; remainBlock--;

renderMap();

}

else if (\*(mapCanvas + digPlace) == 0){ //这里用函数迭代的方式翻开邻近无雷方块

\*(mapVisible + digPlace) = 0; remainBlock--;

renderMap();

if (digPlace >= mapWidth) { digBlock(digPlace - mapWidth);

if (digPlace % mapWidth != mapWidth - 1) { digBlock(digPlace - mapWidth + 1);

}

if (digPlace % mapWidth != 0) { digBlock(digPlace - mapWidth - 1);

}

}

if (digPlace < mapWidth \* (mapHeight - 1)) { digBlock(digPlace + mapWidth);

if (digPlace % mapWidth != mapWidth - 1) { digBlock(digPlace + mapWidth + 1);

}

if (digPlace % mapWidth != 0) { digBlock(digPlace + mapWidth - 1);

}

}

if (digPlace % mapWidth != mapWidth - 1) { digBlock(digPlace + 1);

}

if (digPlace % mapWidth != 0) { digBlock(digPlace - 1);

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: flagBlock(int flagPlace)

Parameter: flagPlace(int) Return: None(void) Description:

实现插旗操作

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ void flagBlock(int flagPlace) {

if (\*(mapVisible + flagPlace) == 0) { return;

}

switch(\*(mapVisible + flagPlace)) { case 9:

\*(mapVisible + flagPlace) = 10; flagNum++;

break; case 10:

\*(mapVisible + flagPlace) = 12; flagNum--;

break; case 12:

\*(mapVisible + flagPlace) = 9; break;

}

renderMap();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: judgeWin()

Parameter: None(void) Return: winFlag(bool) Description:

用于判断是否完全清空雷

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ bool judgeWin() {

if (remainBlock == boomTotal) { timeEnd = GetTickCount64(); renderMap();

GameMenu::outChat("恭喜您，营救成功 "); GameMenu::WaitChatChoice(); ClearScreen();

for (int i = 0; i < mapWidth \* mapHeight; i++) { if (\*(mapCanvas + i) == 11) {

\*(mapVisible + i) = 0;

}

}

renderMap();

bool quitFlag = true; // 强制退出

while (quitFlag) {

FlushInput(); // 刷新输入缓冲区

pos = GetCursorPos(); // 获取鼠标坐标COORD hitLeftPos = GetCursorHitPos(); // 获取左键单击坐标COORD hitRightPos = GetCursorHitPos(2); // 获取右键单击坐标key = GetKeyHit(); // 获取键盘输入

operation = 0; // 当前操作 (0无操作，1为挖开地块，2为标

值旗帜，或者自己指定)

// 检查键盘输入

if (key) {

quitFlag = false;

}

// 将鼠标控制台的坐标放缩至地图坐标

const size\_t mouseY = (size\_t)((size\_t)pos.Y + 1) / 2 - 2; const size\_t mouseX = (size\_t)((size\_t)pos.X - 1) / 4;

if (mouseY < mapHeight && mouseX < mapWidth) {

// 若没有超边界，更新选择的坐标posChoice.X = (SHORT)mouseX; posChoice.Y = (SHORT)mouseY;

}

// 左键和右键分别对应操作1和2

if (hitLeftPos.X == pos.X && hitLeftPos.Y == pos.Y) { quitFlag = false;

}

else if (hitRightPos.X == pos.X && hitRightPos.Y == pos.Y) { quitFlag = false;

}

}

}

else {

}

return true; return false;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: Play()

Parameter: None(void) Return: None(void) Description:

开始游戏

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ void Play() {

gameFlag = true; while (gameFlag) {

checkChoice(); // 检查输入

// 查看当前坐标是否需要更新背景

if (posChoice.X != posChoiceOld.X || posChoice.Y != posChoiceOld.Y) { clearChoiceBackground(posChoiceOld);

posChoiceOld = posChoice;

}

renderChoiceBackground(posChoice);

// 在0,0处放置当前选择位置的字符串，注意结尾有空格留白，可以思考为什么要加这么多空格

MovePos(0, 0);

char strPos[32] = "";

sprintf\_s(strPos, "当前选择(%u, %u) ", posChoice.X, posChoice.Y); PutString(strPos);

Update();

// 在0,20处放置字符串，显示时间

MovePos(20, 0);

timeNow = GetTickCount64(); char strTime[32] = "";

sprintf\_s(strTime, "已用时间：%u ", (isFirst ? 0 : (int)(timeNow - timeStart) / 1000)); PutString(strTime);

Update();

// 在1,0处放置字符串，显示剩余雷数

MovePos(0, 1);

char strBoom[32] = "";

sprintf\_s(strBoom, "剩余雷数：%d ", boomTotal - flagNum); PutString(strBoom);

Update();

// 在1,20处放置字符串，显示已走步数

MovePos(20, 1);

char strSlice[32] = "";

sprintf\_s(strSlice, "已走步数：%u ", (isFirst ? 0 : alreadySlice)); PutString(strSlice);

Update();

// 执行相应操作switch (operation) { case 1:

// 翻开地块

if (!isFirst) {

alreadySlice++;

}

digBlock(posChoice.Y \* mapWidth + posChoice.X); operation = 0;

if (judgeWin()) {

gameFlag = false;

case 2:

}

break;

// 标记为地雷

if (!isFirst) {

alreadySlice++;

}

flagBlock(posChoice.Y \* mapWidth + posChoice.X); operation = 0;

break;

}

// 以下内容不建议修改 处理每帧的事务

//updateMap(); // 更新地图画板

Update(); // 更新操作到屏幕

frame++; // 渲染帧数自增

clock\_t elapsed = 25 - (clock() - tic); // 检查上一帧渲染时间，并计算与25的差值

Sleep(elapsed > 0 ? elapsed : 0); // 若差值大于零，则休眠该差值的毫秒数，以确保每帧渲染不超过50帧

tic = clock(); // 更新上一次记录的时间

}

}

void DestroyGame() {

for (int i = 0; i < mapWidth \* mapWidth; i++) { // 初始化两个数组

mapCanvas[i] = 0;

mapVisible[i] = 9;

}

isFirst = true; remainBlock = 0;

flagNum = 0;

alreadySlice = 1;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* GameEnginePro.cpp

Author: 1950000 一二三Date: 2019/12/29 Description:

定义了程序扭曲游戏模块，用GameEngine命名空间防止同名污染，是你主要需要修改的文件

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ #include "GameEnginePro.h"

namespace GameEnginePro {

// 引入两个命名空间using namespace Graphic; using namespace Controller;

size\_t mapWidth = 0, mapHeight = 0; // 地图宽度和高度

UCHAR mapCanvas[50 \* 25]; // 地图数组指针，若做基础项请不要直接尝试操作这个

指针

COORD pos = { 0 }, posOld = { 0 }; // 鼠标的位置COORD posChoice = { 0 }, posChoiceOld = { 0 }; // 选中地图坐标的位置char key = '\0'; // 键盘的按键

bool gameFlag = false; // 游戏运行状态

size\_t frame = 0; // 已渲染帧数

clock\_t tic = clock(); // 游戏开始时刻

int operation = 0; // 执行的操作

bool isFirst = true; // 是否为第一步

//int mapArray[50][50]; // 这里直接操作mapCanvas，不使用mapArray数组

// 如果你有新加的变量，建议加在下方和预置变量做区别

int boomTotal = 0; // 定义雷的总数

int safeZone = 0; // 定义不可能为雷的安全区的中心位置

UCHAR mapVisible[50 \* 25]; // 定义当前可见的地图信息

int remainBlock = 0; // 定义未点开的砖块数

int flagNum = 0; // 定义已经插旗的数目

int alreadySlice = 1; // 定义已走的步数

double timeStart, timeNow, timeEnd; // 定义开始时的时间

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: renderMap()

Parameter: None(void) Return: None(void) Description:

绘制当前地图

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ void renderMap() {

// 在0,0处放置当前选择位置的字符串，注意结尾有空格留白，可以思考为什么要加这么多空格

MovePos(0, 1);

char strPos[32] = "";

sprintf\_s(strPos, "当前选择(%u, %u) ", posChoice.X, posChoice.Y); PutString(strPos);

Update();

// 在0,20处放置字符串，显示时间

MovePos(20, 1);

char strTime[32] = ""; if (isFirst) {

sprintf\_s(strTime, "已用时间：%u ", 0);

}

else {

}

sprintf\_s(strTime, "已用时间：%u ", (int)(timeEnd - timeStart) / 1000);

PutString(strTime); Update();

// 在1,0处放置字符串，显示剩余雷数

MovePos(0, 2);

char strBoom[32] = "";

sprintf\_s(strBoom, "剩余雷数：%d ", boomTotal - flagNum); PutString(strBoom);

Update();

// 在1,20处放置字符串，显示已走步数

MovePos(20, 2);

char strSlice[32] = "";

sprintf\_s(strSlice, "已走步数：%u ", (isFirst ? 0 : alreadySlice));

PutString(strSlice); Update();

// 从行开始遍历

for (size\_t i = 0; i < 2 \* mapHeight + 1; i++) {

// 移动坐标至行首 并绘制行首的字符

MovePos(1, (SHORT)i + 4); if (i % 2 == 0) {

if (i != 2 \* mapHeight) { PutString("/");

}

else {

}

}

PutString("\\");

// 遍历列，绘制边界

for (size\_t j = 0; j < mapWidth; j++) {

// 移动至确切的坐标，绘制方格的边界MovePos(2 + (SHORT)j \* 3, (SHORT)i + 4); if (i % 2 == 0) {

PutString(j % 2 == 0 ? (i == 2 \* mapHeight ? (j == mapWidth - 1 ? "￣ " : "￣\\") : "￣\\") : ((mapHeight % 2 == 0) && (i == 0) && (j == mapWidth - 1) ? " " : " /"));

}

else {

PutString(j % 2 == 1 ? "￣\\" : " /");

}

}

}

换成字符串

// 绘制地雷地图

for (size\_t i = 0; i < mapHeight + 1; i++) {

// 遍历列 绘制地雷

for (size\_t j = 0; j < mapWidth; j++) {

MovePos(2 + (SHORT)j \* 3, (SHORT)i \* 2 + 3 + (j % 2 == 0 ? 0 : 1)); // 移动至确切坐标

const size\_t mapIndex = (i - 1) \* mapWidth + j; // 确定地图数组的下标

char numMap[3] = " "; // 确定数字字符串

numMap[0] = '0' + mapCanvas[mapIndex]; // 当mapVisible[mapIndex]为1到8时，将其转

switch (mapVisible[mapIndex]) { case 0:

switch (mapCanvas[mapIndex]) { case 0:

case 1:

case 2:

case 3:

case 4:

case 5:

case 6:

case 7:

case 8:

// 0的时候放置空白PutString(" "); break;

// 从1开始绘制数字 带颜色PutStringWithColor(numMap, 30, 144, 255, 0, 0, 0); break;

PutStringWithColor(numMap, 0, 255, 127, 0, 0, 0); break;

PutStringWithColor(numMap, 255, 48, 48, 0, 0, 0); break;

PutStringWithColor(numMap, 72, 61, 139, 0, 0, 0); break;

PutStringWithColor(numMap, 255, 105, 180, 0, 0, 0); break;

PutStringWithColor(numMap, 148, 0, 211, 0, 0, 0); break;

PutStringWithColor(numMap, 139, 0, 0, 0, 0, 0); break;

PutStringWithColor(numMap, 139, 34, 82, 0, 0, 0); break;

case 11:

// 11为地雷

PutStringWithColor("⊙", 255, 215, 0, 0, 0, 0); break;

case 9:

}

break;

// 9为砖块PutString("▇"); break;

case 10:

// 10为旗帜

PutStringWithColor("☆", 255, 255, 255, 0, 0, 0); break;

case 12:

//12为暂定

PutStringWithColor("？", 255, 255, 255, 0, 0, 0); break;

}

}

}

// 绘制最下面一层的底部。

for (size\_t j = 1; j < mapWidth; j = j + 2) {

// 移动至确切的坐标，绘制方格的边界

MovePos(2 + (SHORT)j \* 3, (SHORT)2 \* mapHeight + 5); PutString("￣");

}

MovePos(0, 2 \* mapHeight + 5);

Update(); // 将地图更新到屏幕

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: updateMap()

Parameter: None(void) Return: None(void) Description:

将二维数组的数据复制到一维数组中

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*void updateMap() {

memcpy\_s(mapCanvas, mapWidth \* mapHeight, mapArray, mapWidth \* mapHeight);

}\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function: InitGame(int getWidth,int getHeight, int getBoom) Parameter: getWidth(int),getHeight(int),getBoom(int)

Return: None(void) Description:

初始化游戏

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void InitGame(int getWidth, int getHeight, int getBoom) {

ClearScreen(); // 清屏

mapWidth = getWidth;

mapHeight = getHeight; // 设置宽高

boomTotal = getBoom;

for (int i = 0; i < mapWidth \* mapHeight; i++) { // 初始化两个数组

mapCanvas[i] = 0;

mapVisible[i] = 9;

}

//updateMap(); // 更新地图remainBlock = mapWidth \* mapHeight; renderMap(); // 绘制地图

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: renderChoiceBackground(COORD choice) Parameter: COORD choice

Return: None(void) Description:

绘制选中的地块的背景

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ void renderChoiceBackground(COORD choice) {

const SHORT x = choice.X, y = choice.Y;

const BYTE alpha = 255 - (BYTE)((frame % 50) \* 5); // 定义背景白色当前的颜色值，通过已渲染的帧数进行运算（此处也可以改成使用时间）

const size\_t mapIndex = (size\_t)y \* mapWidth + (size\_t)x; // 确定地图下标

MovePos(2 + (SHORT)x \* 3, 4 + (((x % 2 == 0) && ((SHORT)y % 2 == 1) ? (SHORT)y \* 2 + 1 : (SHORT)y \* 2) - 2) / 2 \*

2 + 2 + (x % 2 == 0 ? 0 : 1));

// 以下内容同renderMap中绘制地雷部分相同，不详做介绍

char numMap[8] = " ";

numMap[0] = '0' + mapCanvas[mapIndex]; switch (mapVisible[mapIndex]) {

case 0:

switch (mapCanvas[mapIndex]) { case 0:

case 1:

case 2:

case 3:

case 4:

case 5:

case 6:

case 7:

case 8:

PutStringWithColor(" ", 255, 255, 255, alpha, alpha, alpha); break;

PutStringWithColor(numMap, 30, 144, 255, alpha, alpha, alpha); break;

PutStringWithColor(numMap, 0, 255, 127, alpha, alpha, alpha); break;

PutStringWithColor(numMap, 255, 48, 48, alpha, alpha, alpha); break;

PutStringWithColor(numMap, 72, 61, 139, alpha, alpha, alpha); break;

PutStringWithColor(numMap, 255, 105, 180, alpha, alpha, alpha); break;

PutStringWithColor(numMap, 148, 0, 211, alpha, alpha, alpha); break;

PutStringWithColor(numMap, 139, 0, 0, alpha, alpha, alpha); break;

PutStringWithColor(numMap, 139, 34, 82, alpha, alpha, alpha); break;

case 11:

PutStringWithColor("⊙", 255, 215, 0, alpha, alpha, alpha); break;

case 9:

}

break;

PutStringWithColor("▇", 255, 255, 255, alpha, alpha, alpha); break;

case 10:

PutStringWithColor("☆", 255, 255, 255, alpha, alpha, alpha); break;

case 12:

PutStringWithColor("？", 255, 255, 255, alpha, alpha, alpha); break;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: clearChoiceBackground(COORD choice) Parameter: COORD choice

Return: None(void) Description:

清除choice中指示位置的背景，功能同上，不详做介绍

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ void clearChoiceBackground(COORD choice) {

const SHORT x = choice.X, y = choice.Y;

const size\_t mapIndex = (size\_t)y \* mapWidth + (size\_t)x;

MovePos(2 + (SHORT)x \* 3, 4 + (((x % 2 == 0) && ((SHORT)y % 2 == 1) ? (SHORT)y \* 2 + 1 : (SHORT)y \* 2) - 2) / 2 \*

2 + 2 + (x % 2 == 0 ? 0 : 1));

char numMap[8] = " ";

numMap[0] = '0' + mapCanvas[mapIndex]; switch (mapVisible[mapIndex]) {

case 0:

switch (mapCanvas[mapIndex]) { case 0:

case 1:

case 2:

case 3:

PutStringWithColor(" ", 255, 255, 255, 0, 0, 0); break;

PutStringWithColor(numMap, 30, 144, 255, 0, 0, 0); break;

PutStringWithColor(numMap, 0, 255, 127, 0, 0, 0); break;

PutStringWithColor(numMap, 255, 48, 48, 0, 0, 0);

case 4:

case 5:

case 6:

case 7:

case 8:

break;

PutStringWithColor(numMap, 72, 61, 139, 0, 0, 0); break;

PutStringWithColor(numMap, 255, 105, 180, 0, 0, 0); break;

PutStringWithColor(numMap, 148, 0, 211, 0, 0, 0); break;

PutStringWithColor(numMap, 139, 0, 0, 0, 0, 0); break;

PutStringWithColor(numMap, 139, 34, 82, 0, 0, 0); break;

case 11:

PutStringWithColor("⊙", 255, 215, 0, 0, 0, 0); break;

case 9:

}

break;

PutStringWithColor("▇", 255, 255, 255, 0, 0, 0); break;

case 10:

PutStringWithColor("☆", 255, 255, 255, 0, 0, 0); break;

case 12:

PutStringWithColor("？", 255, 255, 255, 0, 0, 0); break;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: checkChoice()

Parameter: None(void) Return: None(void) Description:

检查输入和操作，并设置相关变量

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ void checkChoice() {

FlushInput(); // 刷新输入缓冲区

pos = GetCursorPos(); // 获取鼠标坐标COORD hitLeftPos = GetCursorHitPos(); // 获取左键单击坐标COORD hitRightPos = GetCursorHitPos(2); // 获取右键单击坐标key = GetKeyHit(); // 获取键盘输入

operation = 0; // 当前操作 (0无操作，1为挖开地块，2为标值旗帜，或

者自己指定)

// 检查键盘输入

switch (key) {

case VK\_ESCAPE:

// ESC键，将游戏运行状态置否以退出游戏gameFlag = false;

break;

// 上下左右键移动选中的坐标

case VK\_UP:

if (posChoice.Y > 0)posChoice.Y--; break;

case VK\_RIGHT:

if (posChoice.X < (SHORT)mapWidth - 1)posChoice.X++; break;

case VK\_DOWN:

if (posChoice.Y < (SHORT)mapHeight - 1)posChoice.Y++; break;

case VK\_LEFT:

if (posChoice.X > 0)posChoice.X--; break;

// 回车和空格分别对应操作1和2 case VK\_RETURN:

operation = 1; break;

case VK\_SPACE:

operation = 2; break;

}

// 将鼠标控制台的坐标放缩至地图坐标

const size\_t mouseX = (size\_t)((size\_t)pos.X - 1) / 3;

const size\_t mouseY = ((size\_t)((size\_t)pos.Y - (mouseX % 2 == 0 ? 0 : 1)) / 2) - 2; if (mouseY < mapHeight && mouseX < mapWidth) {

// 若没有超边界，更新选择的坐标posChoice.X = (SHORT)mouseX; posChoice.Y = (SHORT)mouseY;

}

// 左键和右键分别对应操作1和2

if (hitLeftPos.X == pos.X && hitLeftPos.Y == pos.Y) { operation = 1;

}

else if (hitRightPos.X == pos.X && hitRightPos.Y == pos.Y) { operation = 2;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: isInArray()

Parameter: ins(int),arr(int[9]) Return: 1/0 (bool) Description:

判断ins是否和arr中的某个元素相等，相等的话返回true

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ bool isInArray(int ins, int arr[]) {

for (int i = 0; i < 9; i++) { if (ins == arr[i]) {

return true;

}

}

return false;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: initMap()

Parameter: None(void) Return: None(void) Description:

初始化函数，请通过posChoice变量判断操作，并操作mapArray数组，请不要操作其他程序预置好的变量

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ void initMap() {

int boomCounter = 0; //定义已经埋下雷的计数器

int tempBomb = 0; //定义可能为雷的工作点

UCHAR\* pin = mapCanvas; //定义埋雷工作指针

int safePlace[9] = { -1,-1,-1,-1,-1,-1,-1,-1,-1 }; //定义安全区具体范围if (safeZone >= mapWidth) {

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

safePlace[i] = safeZone - mapWidth - 1 + i;

}

}

for (int i = 3; i < 6; i++) {

safePlace[i] = safeZone - 4 + i;

}

if (safeZone < (mapHeight - 1) \* mapWidth) { for (int i = 6; i < 9; i++) {

safePlace[i] = safeZone + mapWidth - 7 + i;

}

}

if (safeZone % mapWidth == 0) { for (int i = 0; i < 3; i++) {

safePlace[3 \* i] = -1;

}

}

if (safeZone % mapWidth == mapWidth - 1) { for (int i = 0; i < 3; i++) {

safePlace[3 \* i + 2] = -1;

}

}

if ((safeZone % mapWidth) % 2 == 0) { safePlace[6] = safePlace[8] = -1;

}

else{

}

safePlace[0] = safePlace[2] = -1;

int safeSize = 9; //定义安全区的大小

if (posChoice.X == 0 || posChoice.X == mapWidth - 1) { safeSize -= 3;

}

if (posChoice.Y == 0 || posChoice.Y == mapHeight - 1) { safeSize = (safeSize == 9 ? 6 : 4);

}

srand((unsigned)time(NULL)); //定义时间作为随机数种子for (;;) {

pin = mapCanvas;

tempBomb = rand() % (mapWidth \* mapHeight - safeSize - boomCounter); for (int i = 0;;) {

for (;;) {

if (\*pin >= 9) { //判断没有埋雷pin++;

}

else if (isInArray((int)(pin - mapCanvas), safePlace)) {

pin++; //防止把雷埋在安全区

}

else {

}

}

break;

行赋值

if (i == tempBomb) { //这里根据埋雷的情况对周围位块进

\*pin = 11;

if ((pin - mapCanvas) % mapWidth == 0) {

if ((pin - mapCanvas) >= 0 && (pin - mapCanvas) < mapWidth) {

\*(pin + 1) = (\*(pin + 1) == 11 ? 11 : (\*(pin + 1) + 1));

\*(pin + mapWidth) = (\*(pin + mapWidth) == 11 ? 11 : (\*(pin + mapWidth) + 1));

mapHeight \* mapWidth) {

1));

}

else if ((pin - mapCanvas) >= mapWidth \* (mapHeight - 1) && (pin - mapCanvas) <

\*(pin + 1) = (\*(pin + 1) == 11 ? 11 : (\*(pin + 1) + 1));

\*(pin - mapWidth) = (\*(pin - mapWidth) == 11 ? 11 : (\*(pin - mapWidth) + 1));

\*(pin - mapWidth + 1) = (\*(pin - mapWidth + 1) == 11 ? 11 : (\*(pin - mapWidth + 1) +

1));

}

else {

}

}

\*(pin - mapWidth) = (\*(pin - mapWidth) == 11 ? 11 : (\*(pin - mapWidth) + 1));

\*(pin + 1) = (\*(pin + 1) == 11 ? 11 : (\*(pin + 1) + 1));

\*(pin + mapWidth) = (\*(pin + mapWidth) == 11 ? 11 : (\*(pin + mapWidth) + 1));

\*(pin - mapWidth + 1) = (\*(pin - mapWidth + 1) == 11 ? 11 : (\*(pin - mapWidth + 1) +

- 1) + 1));

else if ((pin - mapCanvas) % mapWidth == mapWidth - 1) {

if ((pin - mapCanvas) >= 0 && (pin - mapCanvas) < mapWidth) {

\*(pin - 1) = (\*(pin - 1) == 11 ? 11 : (\*(pin - 1) + 1));

\*(pin + mapWidth) = (\*(pin + mapWidth) == 11 ? 11 : (\*(pin + mapWidth) + 1)); if (((pin - mapCanvas) % mapWidth) % 2 == 1) {

\*(pin + mapWidth - 1) = (\*(pin + mapWidth - 1) == 11 ? 11 : (\*(pin + mapWidth

}

}

mapHeight \* mapWidth) {

1) + 1));

else if ((pin - mapCanvas) >= mapWidth \* (mapHeight - 1) && (pin - mapCanvas) <

\*(pin - 1) = (\*(pin - 1) == 11 ? 11 : (\*(pin - 1) + 1));

\*(pin - mapWidth) = (\*(pin - mapWidth) == 11 ? 11 : (\*(pin - mapWidth) + 1)); if (((pin - mapCanvas) % mapWidth) % 2 == 0) {

\*(pin - mapWidth - 1) = (\*(pin - mapWidth - 1) == 11 ? 11 : (\*(pin - mapWidth -

- 1) + 1));

}

else {

}

\*(pin - mapWidth) = (\*(pin - mapWidth) == 11 ? 11 : (\*(pin - mapWidth) + 1));

\*(pin - 1) = (\*(pin - 1) == 11 ? 11 : (\*(pin - 1) + 1));

\*(pin + mapWidth) = (\*(pin + mapWidth) == 11 ? 11 : (\*(pin + mapWidth) + 1)); if (((pin - mapCanvas) % mapWidth) % 2 == 1) {

\*(pin + mapWidth - 1) = (\*(pin + mapWidth - 1) == 11 ? 11 : (\*(pin + mapWidth

1) + 1));

}

else {

}

}

}

\*(pin - mapWidth - 1) = (\*(pin - mapWidth - 1) == 11 ? 11 : (\*(pin - mapWidth -

1));

1));

else if ((pin - mapCanvas) >= 0 && (pin - mapCanvas) < mapWidth) {

\*(pin - 1) = (\*(pin - 1) == 11 ? 11 : (\*(pin - 1) + 1));

\*(pin + 1) = (\*(pin + 1) == 11 ? 11 : (\*(pin + 1) + 1));

\*(pin + mapWidth) = (\*(pin + mapWidth) == 11 ? 11 : (\*(pin + mapWidth) + 1)); if (((pin - mapCanvas) % mapWidth) % 2 == 1) {

\*(pin + mapWidth - 1) = (\*(pin + mapWidth - 1) == 11 ? 11 : (\*(pin + mapWidth - 1) +

\*(pin + mapWidth + 1) = (\*(pin + mapWidth + 1) == 11 ? 11 : (\*(pin + mapWidth + 1) +

}

}

mapWidth) {

1));

1));

else if ((pin - mapCanvas) >= mapWidth \* (mapHeight - 1) && (pin - mapCanvas) < mapHeight \*

\*(pin - 1) = (\*(pin - 1) == 11 ? 11 : (\*(pin - 1) + 1));

\*(pin + 1) = (\*(pin + 1) == 11 ? 11 : (\*(pin + 1) + 1));

\*(pin - mapWidth) = (\*(pin - mapWidth) == 11 ? 11 : (\*(pin - mapWidth) + 1)); if (((pin - mapCanvas) % mapWidth) % 2 == 0) {

\*(pin - mapWidth - 1) = (\*(pin - mapWidth - 1) == 11 ? 11 : (\*(pin - mapWidth - 1) +

\*(pin - mapWidth + 1) = (\*(pin - mapWidth + 1) == 11 ? 11 : (\*(pin - mapWidth + 1) +

1));

1));

}

else {

}

\*(pin - 1) = (\*(pin - 1) == 11 ? 11 : (\*(pin - 1) + 1));

\*(pin + 1) = (\*(pin + 1) == 11 ? 11 : (\*(pin + 1) + 1));

\*(pin - mapWidth) = (\*(pin - mapWidth) == 11 ? 11 : (\*(pin - mapWidth) + 1)); if (((pin - mapCanvas) % mapWidth) % 2 == 0) {

\*(pin - mapWidth - 1) = (\*(pin - mapWidth - 1) == 11 ? 11 : (\*(pin - mapWidth - 1) +

\*(pin - mapWidth + 1) = (\*(pin - mapWidth + 1) == 11 ? 11 : (\*(pin - mapWidth + 1) +

1));

1));

}

\*(pin + mapWidth) = (\*(pin + mapWidth) == 11 ? 11 : (\*(pin + mapWidth) + 1)); if (((pin - mapCanvas) % mapWidth) % 2 == 1) {

\*(pin + mapWidth - 1) = (\*(pin + mapWidth - 1) == 11 ? 11 : (\*(pin + mapWidth - 1) +

\*(pin + mapWidth + 1) = (\*(pin + mapWidth + 1) == 11 ? 11 : (\*(pin + mapWidth + 1) +

}

else {

}

}

}

}

break;

pin++; i++;

boomCounter++;

if (boomCounter == boomTotal) { break;

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: gameOver()

Parameter: Nnoe(void) Return: None(void) Description:

游戏失败

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ void gameOver() {

timeEnd = GetTickCount64(); GameMenu::outChat("很遗憾，您失败了 "); GameMenu::WaitChatChoice(); ClearScreen();

for (int i = 0; i < mapWidth \* mapHeight; i++) { if (\*(mapCanvas + i) == 11) {

\*(mapVisible + i) = 0;

}

}

renderMap();

bool quitFlag = true; // 强制退出

while (quitFlag) {

FlushInput(); // 刷新输入缓冲区

pos = GetCursorPos(); // 获取鼠标坐标COORD hitLeftPos = GetCursorHitPos(); // 获取左键单击坐标COORD hitRightPos = GetCursorHitPos(2); // 获取右键单击坐标

key = GetKeyHit(); // 获取键盘输入

// 检查键盘输入

if (key) {

quitFlag = false;

}

// 将鼠标控制台的坐标放缩至地图坐标

const size\_t mouseY = (size\_t)((size\_t)pos.Y + 1) / 2 - 2; const size\_t mouseX = (size\_t)((size\_t)pos.X - 1) / 4;

if (mouseY < mapHeight && mouseX < mapWidth) {

// 若没有超边界，更新选择的坐标posChoice.X = (SHORT)mouseX; posChoice.Y = (SHORT)mouseY;

}

// 左键和右键分别对应操作1和2

if (hitLeftPos.X == pos.X && hitLeftPos.Y == pos.Y) { quitFlag = false;

}

else if (hitRightPos.X == pos.X && hitRightPos.Y == pos.Y) { quitFlag = false;

}

}

gameFlag = false;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: digBlock(int digPlace)

Parameter: digPlace(int) Return: None(void) Description:

实现翻开操作

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ void digBlock(int digPlace) {

if (isFirst) {

// 如果是第一步走，则先初始化地图，注意不要在落点设置一个地雷safeZone = posChoice.Y \* mapWidth + posChoice.X; //设立安全区initMap();

isFirst = false; // 将第一步设置为否

timeStart = GetTickCount64();

}

if (\*(mapVisible + digPlace) == 0) { return;

}

if (\*(mapCanvas + digPlace) == 11) { gameOver();

}

else if (\*(mapCanvas + digPlace) >= 1 && \*(mapCanvas + digPlace) <= 8) {

\*(mapVisible + digPlace) = 0; remainBlock--;

renderMap();

}

else if (\*(mapCanvas + digPlace) == 0) { //这里用函数迭代的方式翻开邻近无雷方块

\*(mapVisible + digPlace) = 0; remainBlock--;

renderMap();

if (digPlace >= mapWidth) { digBlock(digPlace - mapWidth);

if ((digPlace % mapWidth) % 2 == 0) {

if (digPlace % mapWidth != mapWidth - 1) { digBlock(digPlace - mapWidth + 1);

}

if (digPlace % mapWidth != 0) { digBlock(digPlace - mapWidth - 1);

}

}

}

if (digPlace < mapWidth \* (mapHeight - 1)) { digBlock(digPlace + mapWidth);

if ((digPlace % mapWidth) % 2 == 1) {

if (digPlace % mapWidth != mapWidth - 1) { digBlock(digPlace + mapWidth + 1);

}

if (digPlace % mapWidth != 0) { digBlock(digPlace + mapWidth - 1);

}

}

}

if (digPlace % mapWidth != mapWidth - 1) { digBlock(digPlace + 1);

}

if (digPlace % mapWidth != 0) { digBlock(digPlace - 1);

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: flagBlock(int flagPlace)

Parameter: flagPlace(int) Return: None(void) Description:

实现插旗操作

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ void flagBlock(int flagPlace) {

if (\*(mapVisible + flagPlace) == 0) { return;

}

switch (\*(mapVisible + flagPlace)) { case 9:

\*(mapVisible + flagPlace) = 10; flagNum++;

break; case 10:

\*(mapVisible + flagPlace) = 12; flagNum--;

break; case 12:

\*(mapVisible + flagPlace) = 9; break;

}

renderMap();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: judgeWin()

Parameter: None(void) Return: winFlag(bool) Description:

用于判断是否完全清空雷

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ bool judgeWin() {

if (remainBlock == boomTotal) { timeEnd = GetTickCount64();

GameMenu::outChat("恭喜您，营救成功 "); GameMenu::WaitChatChoice(); ClearScreen();

for (int i = 0; i < mapWidth \* mapHeight; i++) { if (\*(mapCanvas + i) == 11) {

\*(mapVisible + i) = 0;

}

}

renderMap();

bool quitFlag = true; // 强制退出

while (quitFlag) {

FlushInput(); // 刷新输入缓冲区

pos = GetCursorPos(); // 获取鼠标坐标COORD hitLeftPos = GetCursorHitPos(); // 获取左键单击坐标COORD hitRightPos = GetCursorHitPos(2); // 获取右键单击坐标key = GetKeyHit(); // 获取键盘输入

operation = 0; // 当前操作 (0无操作，1为挖开地块，2为标

值旗帜，或者自己指定)

// 检查键盘输入

if (key) {

quitFlag = false;

}

// 将鼠标控制台的坐标放缩至地图坐标

const size\_t mouseY = (size\_t)((size\_t)pos.Y + 1) / 2 - 2; const size\_t mouseX = (size\_t)((size\_t)pos.X - 1) / 4;

if (mouseY < mapHeight && mouseX < mapWidth) {

// 若没有超边界，更新选择的坐标posChoice.X = (SHORT)mouseX; posChoice.Y = (SHORT)mouseY;

}

// 左键和右键分别对应操作1和2

if (hitLeftPos.X == pos.X && hitLeftPos.Y == pos.Y) { quitFlag = false;

}

else if (hitRightPos.X == pos.X && hitRightPos.Y == pos.Y) { quitFlag = false;

}

}

}

else {

}

return true; return false;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: Play()

Parameter: None(void) Return: None(void) Description:

开始游戏

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ void Play() {

gameFlag = true; while (gameFlag) {

checkChoice(); // 检查输入

// 查看当前坐标是否需要更新背景

if (posChoice.X != posChoiceOld.X || posChoice.Y != posChoiceOld.Y) { clearChoiceBackground(posChoiceOld);

posChoiceOld = posChoice;

}

renderChoiceBackground(posChoice);

// 在0,0处放置当前选择位置的字符串，注意结尾有空格留白，可以思考为什么要加这么多空格

MovePos(0, 0);

char strPos[32] = "";

sprintf\_s(strPos, "当前选择(%u, %u) ", posChoice.X, posChoice.Y); PutString(strPos);

Update();

// 在0,20处放置字符串，显示时间

MovePos(20, 0);

timeNow = GetTickCount64(); char strTime[32] = "";

sprintf\_s(strTime, "已用时间：%u ", (isFirst ? 0 : (int)(timeNow - timeStart) / 1000)); PutString(strTime);

Update();

// 在1,0处放置字符串，显示剩余雷数

MovePos(0, 1);

char strBoom[32] = "";

sprintf\_s(strBoom, "剩余雷数：%d ", boomTotal - flagNum); PutString(strBoom);

Update();

// 在1,20处放置字符串，显示已走步数

MovePos(20, 1);

char strSlice[32] = "";

sprintf\_s(strSlice, "已走步数：%u ", (isFirst ? 0 : alreadySlice)); PutString(strSlice);

Update();

// 执行相应操作switch (operation) { case 1:

// 翻开地块

if (!isFirst) {

alreadySlice++;

}

digBlock(posChoice.Y \* mapWidth + posChoice.X); operation = 0;

if (judgeWin()) {

gameFlag = false;

case 2:

}

break;

// 标记为地雷

if (!isFirst) {

alreadySlice++;

}

flagBlock(posChoice.Y \* mapWidth + posChoice.X); operation = 0;

break;

}

// 以下内容不建议修改 处理每帧的事务

//updateMap(); // 更新地图画板

Update(); // 更新操作到屏幕

frame++; // 渲染帧数自增

clock\_t elapsed = 25 - (clock() - tic); // 检查上一帧渲染时间，并计算与25的差值

Sleep(elapsed > 0 ? elapsed : 0); // 若差值大于零，则休眠该差值的毫秒数，以确保每帧渲染不超过50帧

tic = clock(); // 更新上一次记录的时间

}

}

void DestroyGame() {

for (int i = 0; i < mapWidth \* mapWidth; i++) { // 初始化两个数组

mapCanvas[i] = 0;

mapVisible[i] = 9;

}

isFirst = true; remainBlock = 0;

flagNum = 0;

alreadySlice = 1;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* GameMenu.cpp

Author: Root Date: 2019/12/09 Description:

定义了程序菜单的渲染/管理模块，用GameMenu命名空间防止同名污染

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ #include "GameMenu.h"

namespace GameMenu {

// 引入两个命名空间using namespace Graphic; using namespace Controller;

size\_t frame = 0; // 定义已渲染的帧数

COORD pos = { 0 }, hitPos = { 0 }; // 定义鼠标当前的坐标和左键点击的坐标

int choice = 0, oldChoice = 0; // 定义选中的菜单项

int subChoice = 0, subOldChoice = 0; //定义了在次级菜单中的选择菜单项int diyChoice = 0, diyOldChoice = 0; //定义了在自定义菜单中的输入项int chatChoice = 0, chatOldChoice = 0; //定义了在弹出窗口的输入项

int modeChoice = 0, modeOldChoice = 0; //定义了在模式窗口的输入项bool isChosen = false; // 定义是否做出选择

bool subIsChosen = false; // 定义在次级菜单是否做出选择

bool diyIsChosen = false; // 定义了自定义菜单中的输入项是否选择

bool chatIsChosen = false; // 定义了chat窗口的输入项是否选择

bool modeIsChosen = false; // 定义了mode窗口的输入项是否选择

char key = '\0'; // 定义当前键盘的按键

clock\_t tic = clock(); // 定义程序上一帧渲染的时刻

// 如果你有新加的变量，建议加在下方和预置变量做区别

int boomNum = 0; // 定义雷数

int mapWidth = 10; // 定义地图宽度

int mapHeight = 10; // 定义地图高度

int tempNum[3] = { 0,0,0 }; // 定义自定义的暂存数值:0 代表雷数 1 代表宽度 2 代表高度

int pin = 0, pin\_ = 0; // 输出时使用的工作指针

int windowWidth = 44, windowHeight = 25; //定义初始化的窗口大小bool modeFinallyFlag = 0; // 定义最终选择的扭曲模式与否

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: outChat()

Parameter: outString(char[]) Return: None(void) Description:

用于弹出对话框的框体，对话框文字即为outSring（长度必须为18）

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ void outChat(const char outString[]) {

MovePos(windowWidth / 2 / 2 \* 2 - 16, windowHeight / 2 - 6); PutString(" ");

MovePos(windowWidth / 2 / 2 \* 2 - 16, windowHeight / 2 - 5);

PutString(" ╔══════════════╦══════╗ "); MovePos(windowWidth / 2 / 2 \* 2 - 16, windowHeight / 2 - 4);

PutString(" ║ ║ × ║ "); MovePos(windowWidth / 2 / 2 \* 2 - 16, windowHeight / 2 - 3);

PutString(" ╠══════════════╩══════╣ "); MovePos(windowWidth / 2 / 2 \* 2 - 16, windowHeight / 2 - 2);

PutString(" ║ ║ "); MovePos(windowWidth / 2 / 2 \* 2 - 16, windowHeight / 2 - 1);

PutString(" ╚═════════════════════╝ "); MovePos(windowWidth / 2 / 2 \* 2 - 16, windowHeight / 2 );

PutString(" ");

MovePos(windowWidth / 2 / 2 \* 2 - 7, windowHeight / 2 - 2); PutString(outString);

Update();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: introMenu()

Parameter: Nnoe(void) Return: None(void) Description:

用于画出介绍菜单

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ void introMenu() {

ClearScreen(); // 清 屏

MovePos(16, 2); // 移动坐标到 14,2

PutString("扫雷游戏简介"); // 在这个坐标放置一个string MovePos(6, 4);

PutString(" 《扫雷》是一款大众类的益智小"); MovePos(6, 5);

PutString("游戏，于1992年发行。游戏目标是在"); MovePos(6, 6);

PutString("最短的时间内根据点击格子出现的数"); MovePos(6, 7);

PutString("字找出所有非雷格子，同时避免踩到"); MovePos(6, 8);

PutString("雷，踩到一个雷即全盘皆输。"); MovePos(6, 10);

PutString(" 按下任意键返回。。。。。。");

Update();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: outNum()

Parameter: outnum(int) Return: None(void) Description:

在自定义环节用于数字的输出

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ void outNum(int outnum) {

if (outnum / 10 == 0) {

}

else {

}

outNum(outnum / 10);

MovePos(pin\_, pin); PutChar((char)((outnum % 10 )+ '0')); pin\_++;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: renderMenu()

Parameter: None(void) Return: None(void) Description:

清屏并且打印标题和相关菜单选项

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ void renderMenu() {

ClearScreen(); // 清 屏

MovePos(14, 2); // 移动坐标到 14,2

PutString("扫雷大作战！"); // 在这个坐标放置一个string MovePos(15, 4);

PutString("开始战斗！"); MovePos(15, 6); PutString("查看说明！"); MovePos(15, 8); PutString("退出游戏！"); Update();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: FormModeChoice()

Parameter: None(void) Return: None(void) Description:

选择扫雷的模式扭曲与否

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ void FormModeChoice() {

ClearScreen(); // 清 屏

MovePos(12, 2);

PutString(">>请选择模式<<"); MovePos(16, 4);

PutString("正 常"); // 在这个坐标放置一个string MovePos(16, 6);

PutString("扭 曲"); Update();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: ModeChoice()

Parameter: None(void) Return: None(void) Description:

选择扫雷的难度或是自定义大小与雷数

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ void ModeChoice() {

ClearScreen(); // 清 屏

MovePos(12, 2);

PutString(">>请选择难度<<"); MovePos(16, 4);

PutString("简 单"); // 在这个坐标放置一个string MovePos(16, 6);

PutString(" 中 等 "); MovePos(16, 8); PutString(" 困 难 "); MovePos(15, 10); PutString("自 定 义"); Update();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: DiyChoice()

Parameter: None(void) Return: None(void) Description:

选择扫雷的难度或是自定义大小与雷数

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ void DiyChoice() {

ClearScreen(); //清屏MovePos(11, 2);

PutString(">>自定义难度：<<"); MovePos(8, 4);

PutString("雷数："); MovePos(8, 6); PutString("宽度："); MovePos(8, 8); PutString("高度："); MovePos(17, 11); PutString("确认");

if (tempNum[0] != 0) { pin\_ = 22;

pin = 4; outNum(tempNum[0]);

}

if (tempNum[1] != 0) { pin\_ = 22;

pin = 6; outNum(tempNum[1]);

}

if (tempNum[2] != 0) { pin\_ = 22;

pin = 8; outNum(tempNum[2]);

}

Update();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: InitMenu()

Parameter: None(void) Return: None(void) Description:

初始化菜单，同时初始化图形框架和控制器框架

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ void InitMenu() {

InitGraphic(windowWidth, windowHeight); // 初始化新的窗口大小为44,25 InitController(); // 初始化控制器

renderMenu(); // 绘制菜单

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: randerChoiceBackground(int choice) Parameter: int choice

Return: None(void) Description:

将当前选中的菜单项(int choice)背景渲染一个渐变消失的白色

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ void randerChoiceBackground(int choice) {

size\_t y = 0; // 定义当前选中的菜单项的实际y坐标

const BYTE alpha = 255 - (BYTE)((frame % 50) \* 5); // 定义背景白色当前的颜色值，通过已渲染的帧数进行运算

（此处也可以改成使用时间）

switch (choice) {

case 0:

case 1:

case 2:

}

y = 4;

break;

y = 6;

break;

y = 8;

break;

for (size\_t x = 10; x < 30; x++) {

MovePos((SHORT)x, (SHORT)y); // 将坐标移到目标点

ModColor(2, 255, 255, 255, alpha, alpha, alpha); // 修改背景色

}

Update(); // 更新界面

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: randerChatChoiceBackground(int choice) Parameter: int chatChoice

Return: None(void) Description:

将当前选中的chat菜单项(int chatChoice)背景渲染一个渐变消失的白色

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ void randerChatChoiceBackground(int chatChoice) {

int y = 0; // 定义当前选中的菜单项的实际y坐标

const BYTE alpha = 255 - (BYTE)((frame % 50) \* 5); // 定义背景白色当前的颜色值，通过已渲染的帧数进行运算

（此处也可以改成使用时间）

for (int x = (windowWidth / 2 / 2 \* 2 + 6); x < (windowWidth / 2 / 2 \* 2 + 10); x++) { MovePos((SHORT)x, windowHeight / 2 - 4); // 将坐标移到目标点ModColor(2, 255, 255, 255, alpha, alpha, alpha); // 修改背景色

}

Update(); // 更新界面

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: randerDiyChoiceBackground(int choice) Parameter: int diyChoice

Return: None(void) Description:

将当前选中的diy菜单项(int diyChoice)背景渲染一个渐变消失的白色

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ void randerDiyChoiceBackground(int diyChoice) {

size\_t y = 0; // 定义当前选中的菜单项的实际y坐标

const BYTE alpha = 255 - (BYTE)((frame % 50) \* 5); // 定义背景白色当前的颜色值，通过已渲染的帧数进行运算

（此处也可以改成使用时间）

switch (diyChoice) { case 0:

case 1:

case 2:

case 3:

}

y = 4;

break;

y = 6;

break;

y = 8;

break;

y = 11;

break;

for (size\_t x = (diyChoice == 3 ? 10 : 21); x < (diyChoice == 3 ? 30 : 28); x++) { MovePos((SHORT)x, (SHORT)y); // 将坐标移到目标点

ModColor(2, 255, 255, 255, alpha, alpha, alpha); // 修改背景色

}

Update(); // 更新界面

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: randerModeChoiceBackground(int choice) Parameter: int modeChoice

Return: None(void) Description:

将当前选中的mode菜单项(int modeChoice)背景渲染一个渐变消失的白色

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ void randerModeChoiceBackground(int modeChoice) {

size\_t y = 0; // 定义当前选中的菜单项的实际y坐标

const BYTE alpha = 255 - (BYTE)((frame % 50) \* 5); // 定义背景白色当前的颜色值，通过已渲染的帧数进行运算

（此处也可以改成使用时间）

switch (modeChoice) { case 0:

case 1:

}

y = 4;

break;

y = 6;

break;

for (size\_t x = 10 ; x < 30 ; x++) {

MovePos((SHORT)x, (SHORT)y); // 将坐标移到目标点

ModColor(2, 255, 255, 255, alpha, alpha, alpha); // 修改背景色

}

Update(); // 更新界面

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: randerSubChoiceBackground(int choice) Parameter: int subChoice

Return: None(void) Description:

将当前选中的次级菜单项(int subChoice)背景渲染一个渐变消失的白色

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ void randerSubChoiceBackground(int subChoice) {

size\_t y = 0; // 定义当前选中的菜单项的实际y坐标

const BYTE alpha = 255 - (BYTE)((frame % 50) \* 5); // 定义背景白色当前的颜色值，通过已渲染的帧数进行运算

（此处也可以改成使用时间）

switch (subChoice) { case 0:

case 1:

case 2:

case 3:

}

y = 4;

break;

y = 6;

break;

y = 8;

break; y = 10;

for (size\_t x = 10; x < 30; x++) {

MovePos((SHORT)x, (SHORT)y); // 将坐标移到目标点

ModColor(2, 255, 255, 255, alpha, alpha, alpha); // 修改背景色

}

Update(); // 更新界面

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: clearChoiceBackground(int choice) Parameter: int choice

Return: None(void) Description:

将上次选中的菜单项(int choice)背景恢复为黑色，内容大致和上一个函数一致，不多做阐述

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ void clearChoiceBackground(int choice) {

size\_t y = 0; switch (choice) { case 0:

case 1:

case 2:

}

y = 4;

break;

y = 6;

break;

y = 8;

break;

for (size\_t x = 10; x < 30; x++) { MovePos((SHORT)x, (SHORT)y); ModColor(2, 255, 255, 255, 0, 0, 0);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: clearChatChoiceBackground(int choice) Parameter: int diyChoice

Return: None(void) Description:

将上次选中的chat菜单项(int chatChoice)背景恢复为黑色，内容大致和上一个函数一致，不多做阐述

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void clearChatChoiceBackground(int chatChoice) { size\_t y = 0;

for (int x = (windowWidth / 2 / 2 \* 2 + 6); x < (windowWidth / 2 / 2 \* 2 + 10); x++) {

MovePos((SHORT)x, windowHeight / 2 - 4);

ModColor(2, 255, 255, 255, 0, 0, 0);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: clearDiyChoiceBackground(int choice) Parameter: int diyChoice

Return: None(void) Description:

将上次选中的diy菜单项(int diyChoice)背景恢复为黑色，内容大致和上一个函数一致，不多做阐述

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ void clearDiyChoiceBackground(int diyChoice) {

size\_t y = 0;

switch (diyChoice) { case 0:

case 1:

case 2:

case 3:

}

y = 4;

break;

y = 6;

break;

y = 8;

break; y = 11;

for (size\_t x = (diyChoice == 3 ? 10 : 21); x < (diyChoice == 3 ? 30 : 28); x++) { MovePos((SHORT)x, (SHORT)y);

ModColor(2, 255, 255, 255, 0, 0, 0);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: clearSubChoiceBackground(int choice) Parameter: int subChoice

Return: None(void) Description:

将上次选中的次级菜单项(int subChoice)背景恢复为黑色，内容大致和上一个函数一致，不多做阐述

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ void clearSubChoiceBackground(int subChoice) {

size\_t y = 0;

switch (subChoice) { case 0:

case 1:

case 2:

case 3:

}

y = 4;

break;

y = 6;

break;

y = 8;

break;

y = 10;

break;

for (size\_t x = 10; x < 30; x++) { MovePos((SHORT)x, (SHORT)y); ModColor(2, 255, 255, 255, 0, 0, 0);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: clearModeChoiceBackground(int choice) Parameter: int modeChoice

Return: None(void) Description:

将上次选中的mode菜单项(int modeChoice)背景恢复为黑色，内容大致和上一个函数一致，不多做阐述

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ void clearModeChoiceBackground(int modeChoice) {

size\_t y = 0;

switch (modeChoice) { case 0:

case 1:

y = 4;

break; y = 6;

break;

}

for (size\_t x = 10; x < 30; x++) { MovePos((SHORT)x, (SHORT)y); ModColor(2, 255, 255, 255, 0, 0, 0);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: renderIntro()

Parameter: None(void) Return: None(void) Description:

绘制游戏说明

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ void renderIntro() {

introMenu();

bool quitFlag = true; // 按下任意键退出

while (quitFlag) {

FlushInput(); // 刷新输入缓冲区

pos = GetCursorPos(); // 获取鼠标坐标COORD hitLeftPos = GetCursorHitPos(); // 获取左键单击坐标COORD hitRightPos = GetCursorHitPos(2); // 获取右键单击坐标key = GetKeyHit(); // 获取键盘输入

// 检查键盘输入

if (key) {

quitFlag = false;

}

// 左键和右键分别对应操作1和2

if (hitLeftPos.X == pos.X && hitLeftPos.Y == pos.Y) { quitFlag = false;

}

else if (hitRightPos.X == pos.X && hitRightPos.Y == pos.Y) { quitFlag = false;

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: checkSubChoice()

Parameter: None(void) Return: None(void) Description:

检查键盘/鼠标输入，并更新选中的次级菜单项

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ void checkSubChoice() {

FlushInput(); // 刷新输入缓冲区pos = GetCursorPos(); // 获取鼠标输入key = GetKeyHit(); // 获取键盘输入

hitPos = GetCursorHitPos(); // 获取鼠标单击输入

subIsChosen = false; // 重置选中状态

// 检查鼠标位置，更新选中项

if (!(pos.Y != 4 || pos.X < 10 || pos.X > 30)) { subChoice = 0;

}

else if (!(pos.Y != 6 || pos.X < 10 || pos.X > 30)) { subChoice = 1;

}

else if (!(pos.Y != 8 || pos.X < 10 || pos.X > 30)) { subChoice = 2;

}

else if (!(pos.Y != 10 || pos.X < 10 || pos.X > 30)) { subChoice = 3;

}

// 检查键盘按键，更新选中项

switch (key) { case VK\_UP:

// 上方向键

if (subChoice > 0)subChoice--; break;

case VK\_DOWN:

// 下方向键

if (subChoice < 3)subChoice++;

break;

case VK\_RETURN:

// 回车，更新选中subIsChosen = true; break;

}

// 检查鼠标点击的位置是否在菜单项中，如果是，更新选中

if (!(hitPos.Y != 4 || hitPos.X < 10 || hitPos.X > 30)) { subIsChosen = true;

}

else if (!(hitPos.Y != 6 || hitPos.X < 10 || hitPos.X > 30)) { subIsChosen = true;

}

else if (!(hitPos.Y != 8 || hitPos.X < 10 || hitPos.X > 30)) { subIsChosen = true;

}

else if (!(hitPos.Y != 10 || hitPos.X < 10 || hitPos.X > 30)) { subIsChosen = true;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: checkModeChoice()

Parameter: None(void) Return: None(void) Description:

检查键盘/鼠标输入，并更新选中的mode菜单项

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ void checkModeChoice() {

FlushInput(); // 刷新输入缓冲区pos = GetCursorPos(); // 获取鼠标输入key = GetKeyHit(); // 获取键盘输入

hitPos = GetCursorHitPos(); // 获取鼠标单击输入

subIsChosen = false; // 重置选中状态

// 检查鼠标位置，更新选中项

if (!(pos.Y != 4 || pos.X < 10 || pos.X > 30)) { modeChoice = 0;

}

else if (!(pos.Y != 6 || pos.X < 10 || pos.X > 30)) { modeChoice = 1;

}

// 检查键盘按键，更新选中项

switch (key) { case VK\_UP:

// 上方向键

if (modeChoice > 0)modeChoice--; break;

case VK\_DOWN:

// 下方向键

if (modeChoice < 1)modeChoice++; break;

case VK\_RETURN:

// 回车，更新选中modeIsChosen = true; break;

}

// 检查鼠标点击的位置是否在菜单项中，如果是，更新选中

if (!(hitPos.Y != 4 || hitPos.X < 10 || hitPos.X > 30)) { modeIsChosen = true;

}

else if (!(hitPos.Y != 6 || hitPos.X < 10 || hitPos.X > 30)) { modeIsChosen = true;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: checkChatChoice()

Parameter: None(void) Return: None(void) Description:

检查键盘/鼠标输入，并更新选中的次级菜单项

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ void checkChatChoice() {

FlushInput(); // 刷新输入缓冲区

pos = GetCursorPos(); // 获取鼠标输入

key = GetKeyHit(); // 获取键盘输入

hitPos = GetCursorHitPos(); // 获取鼠标单击输入

chatIsChosen = false; // 重置选中状态

// 检查鼠标位置，更新选中项

if (!(pos.Y != windowHeight / 2 - 4 || pos.X < windowWidth / 2 + 7 || pos.X > windowWidth / 2 + 12)) { chatChoice = 0;

}

// 检查键盘按键，更新选中项

switch (key) {

case VK\_RETURN:

// 回车，更新选中chatIsChosen = true; break;

}

// 检查鼠标点击的位置是否在菜单项中，如果是，更新选中

if (!(hitPos.Y != windowHeight / 2 - 4 || pos.X < windowWidth / 2 + 7 || pos.X > windowWidth / 2 + 12)) { chatIsChosen = true;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: WaitChatChoice()

Parameter: None(void) Return: None(void) Description:

chat菜单的循环，若不退出将一直渲染下去。

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ void WaitChatChoice() {

bool runChatFlag = true; while (runChatFlag) {

checkChatChoice(); // 检查输入

if (chatChoice != chatOldChoice) {

// 若新的选择与旧选择不一致，将旧选择的背景清空clearChatChoiceBackground(chatOldChoice); chatOldChoice = chatChoice;

}

randerChatChoiceBackground(chatChoice); // 绘制选中项的背景

// 如果选中菜单，执行操作

if (chatIsChosen) {

switch (chatChoice) { case 0:

runChatFlag = false; break;

}

}

// 处理每帧的事务

frame++; // 渲染帧数自增

clock\_t elapsed = 25 - (clock() - tic); // 检查上一帧渲染时间，并计算与25的差值

Sleep(elapsed > 0 ? elapsed : 0); // 若差值大于零，则休眠该差值的毫秒数，以确保每帧渲染不超过50

帧

tic = clock(); // 更新上一次记录的时间

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: checkDiyChoice()

Parameter: None(void) Return: None(void) Description:

检查键盘/鼠标输入，并更新选中的次级菜单项

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ void checkDiyChoice() {

diyIsChosen = false; // 重置选中状态

int tempPosition = 0; //定义输入的位置for (;;) {

FlushInput(); // 刷新输入缓冲区

pos = GetCursorPos(); // 获取鼠标输入

key = GetKeyHit(); // 获取键盘输入

hitPos = GetCursorHitPos(); // 获取鼠标单击输入

// 检查鼠标位置，更新选中项

if (!(pos.Y != 4 || pos.X < 20 || pos.X > 29)) { diyChoice = 0;

}

else if (!(pos.Y != 6 || pos.X < 20 || pos.X > 29)) { diyChoice = 1;

}

else if (!(pos.Y != 8 || pos.X < 20 || pos.X > 29)) { diyChoice = 2;

}

else if (!(pos.Y != 11 || pos.X < 10 || pos.X > 30)) { diyChoice = 3;

}

// 检查键盘按键，更新选中项

switch (key) { case VK\_UP:

// 上方向键

if (diyChoice > 0) {

diyChoice--;

if (diyChoice <= 2) {

tempPosition = diyChoice;

}

}

break;

case VK\_DOWN:

// 下方向键

if (diyChoice < 3) {

diyChoice++;

if (diyChoice <= 2) {

tempPosition = diyChoice;

}

}

break; case '0':

case VK\_NUMPAD0:

// 0键

if (diyChoice != 3) {

if (tempNum[tempPosition] != 0) {

tempNum[tempPosition] = tempNum[tempPosition] \* 10;

}

break; case '1':

}

DiyChoice();

case VK\_NUMPAD1:

if (diyChoice != 3) {

tempNum[tempPosition] = tempNum[tempPosition] \* 10 + 1; DiyChoice();

}

break; case '2':

case VK\_NUMPAD2:

if (diyChoice != 3) {

tempNum[tempPosition] = tempNum[tempPosition] \* 10 + 2; DiyChoice();

}

break; case '3':

case VK\_NUMPAD3:

if (diyChoice != 3) {

tempNum[tempPosition] = tempNum[tempPosition] \* 10 + 3; DiyChoice();

}

break; case '4':

case VK\_NUMPAD4:

if (diyChoice != 3) {

tempNum[tempPosition] = tempNum[tempPosition] \* 10 + 4; DiyChoice();

}

break; case '5':

case VK\_NUMPAD5:

if (diyChoice != 3) {

tempNum[tempPosition] = tempNum[tempPosition] \* 10 + 5; DiyChoice();

}

break; case '6':

case VK\_NUMPAD6:

if (diyChoice != 3) {

tempNum[tempPosition] = tempNum[tempPosition] \* 10 + 6; DiyChoice();

}

break; case '7':

case VK\_NUMPAD7:

if (diyChoice != 3) {

tempNum[tempPosition] = tempNum[tempPosition] \* 10 + 7; DiyChoice();

}

break; case '8':

case VK\_NUMPAD8:

if (diyChoice != 3) {

tempNum[tempPosition] = tempNum[tempPosition] \* 10 + 8; DiyChoice();

}

break; case '9':

case VK\_NUMPAD9:

if (diyChoice != 3) {

tempNum[tempPosition] = tempNum[tempPosition] \* 10 + 9; DiyChoice();

}

break; case VK\_BACK:

tempNum[tempPosition] /= 10; DiyChoice();

break;

case VK\_RETURN:

// 回车

if (tempPosition < 2) {

diyChoice++; tempPosition++;

}

else if (tempPosition >= 2) { diyIsChosen = true;

}

break;

}

// 检查鼠标点击的位置是否在菜单项中，如果是，更新选中

if (!(hitPos.Y != 4 || hitPos.X < 21 || hitPos.X > 27)) { tempPosition = 0;

}

else if (!(hitPos.Y != 6 || hitPos.X < 21 || hitPos.X > 27)) { tempPosition = 1;

}

else if (!(hitPos.Y != 8 || hitPos.X < 21 || hitPos.X > 27)) { tempPosition = 2;

}

else if (!(hitPos.Y != 11 || hitPos.X < 15 || hitPos.X > 23)) { diyIsChosen = true;

}

if (diyIsChosen) {

if (tempNum[1] <= 6) {

outChat("自定义宽度应大于6"); WaitChatChoice();

DiyChoice(); diyIsChosen = false; continue;

}

if (tempNum[1] > 50) {

outChat("自定宽度应小于51 "); WaitChatChoice();

DiyChoice(); diyIsChosen = false; continue;

}

if (tempNum[2] < 7) {

outChat("自定义高度应大于6"); WaitChatChoice();

DiyChoice(); diyIsChosen = false;

continue;

}

if (tempNum[2] > 25) {

outChat("自定高度应小于26 "); WaitChatChoice();

DiyChoice(); diyIsChosen = false; continue;

}

if (tempNum[0] < 9) {

outChat("自定雷数至少为10 "); WaitChatChoice();

DiyChoice(); diyIsChosen = false; continue;

}

if (tempNum[0] > tempNum[1] \* tempNum[2] - 9) { outChat(" 定 义 的 雷 数 过 多 ！ "); WaitChatChoice();

DiyChoice(); diyIsChosen = false; continue;

}

boomNum = tempNum[0]; mapWidth = tempNum[1]; mapHeight = tempNum[2]; break;

}

if (diyChoice != diyOldChoice) {

// 若新的选择与旧选择不一致，将旧选择的背景清空clearDiyChoiceBackground(diyOldChoice); diyOldChoice = diyChoice;

}

randerDiyChoiceBackground(diyChoice); // 绘制选中项的背景

// 处理每帧的事务

frame++; // 渲染帧数自增

clock\_t elapsed = 25 - (clock() - tic); // 检查上一帧渲染时间，并计算与25的差值

Sleep(elapsed > 0 ? elapsed : 0); // 若差值大于零，则休眠该差值的毫秒数，以确保每帧渲染不超过50

帧

tic = clock(); // 更新上一次记录的时间

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: checkChoice()

Parameter: None(void) Return: None(void) Description:

检查键盘/鼠标输入，并更新选中的菜单项

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ void checkChoice() {

FlushInput(); // 刷新输入缓冲区pos = GetCursorPos(); // 获取鼠标输入key = GetKeyHit(); // 获取键盘输入

hitPos = GetCursorHitPos(); // 获取鼠标单击输入

isChosen = false; // 重置选中状态

// 检查鼠标位置，更新选中项

if (!(pos.Y != 4 || pos.X < 10 || pos.X > 30)) { choice = 0;

}

else if (!(pos.Y != 6 || pos.X < 10 || pos.X > 30)) { choice = 1;

}

else if (!(pos.Y != 8 || pos.X < 10 || pos.X > 30)) { choice = 2;

}

// 检查键盘按键，更新选中项

switch (key) { case VK\_UP:

// 上方向键

if (choice > 0)choice--; break;

case VK\_DOWN:

// 下方向键

if (choice < 3)choice++; break;

case VK\_RETURN:

// 回车，更新选中isChosen = true; break;

}

// 检查鼠标点击的位置是否在菜单项中，如果是，更新选中

if (!(hitPos.Y != 4 || hitPos.X < 10 || hitPos.X > 30)) { isChosen = true;

}

else if (!(hitPos.Y != 6 || hitPos.X < 10 || hitPos.X > 30)) { isChosen = true;

}

else if (!(hitPos.Y != 8 || hitPos.X < 10 || hitPos.X > 30)) { isChosen = true;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: WaitDiyChoice()

Parameter: None(void) Return: None(void) Description:

该菜单用于自定义雷数的设定。

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ void WaitDiyChoice() {

checkDiyChoice(); // 检查输入

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: WaitSubChoice()

Parameter: None(void) Return: None(void) Description:

次级菜单的循环，若不退出将一直渲染下去，用于扫雷难度的选定。

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ void WaitSubChoice() {

bool runSubFlag = true; while (runSubFlag) {

checkSubChoice(); // 检查输入

if (subChoice != subOldChoice) {

// 若新的选择与旧选择不一致，将旧选择的背景清空clearSubChoiceBackground(subOldChoice); subOldChoice = subChoice;

}

randerSubChoiceBackground(subChoice); // 绘制选中项的背景

// 如果选中菜单，执行操作

if (subIsChosen) {

switch (subChoice) { case 0:

case 1:

case 2:

case 3:

}

// 开始游戏

boomNum = 10;

mapWidth = 10;

mapHeight = 10; runSubFlag = false; break;

boomNum = 40;

mapWidth = 16;

mapHeight = 16; runSubFlag = false; break;

boomNum = 99;

mapWidth = 30;

mapHeight = 16; runSubFlag = false; break;

DiyChoice(); WaitDiyChoice(); runSubFlag = false; break;

}

// 处理每帧的事务

frame++; // 渲染帧数自增

clock\_t elapsed = 25 - (clock() - tic); // 检查上一帧渲染时间，并计算与25的差值

Sleep(elapsed > 0 ? elapsed : 0); // 若差值大于零，则休眠该差值的毫秒数，以确保每帧渲染不超过50

帧

tic = clock(); // 更新上一次记录的时间

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: WaitModeChoice()

Parameter: None(void) Return: None(void) Description:

mode菜单的循环，若不退出将一直渲染下去，用于扫雷模式的选定。

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ void WaitModeChoice() {

bool runModeFlag = true; while (runModeFlag) {

checkModeChoice(); // 检查输入

if (modeChoice != modeOldChoice) {

// 若新的选择与旧选择不一致，将旧选择的背景清空clearModeChoiceBackground(modeOldChoice); modeOldChoice = modeChoice;

}

randerModeChoiceBackground(modeChoice); // 绘制选中项的背景

// 如果选中菜单，执行操作

if (modeIsChosen) {

switch (modeChoice) { case 0:

case 1:

}

}

// 开始游戏ModeChoice(); WaitSubChoice(); modeFinallyFlag = false; runModeFlag = false; break;

ModeChoice(); WaitSubChoice(); modeFinallyFlag = true; runModeFlag = false; break;

// 处理每帧的事务

frame++; // 渲染帧数自增

clock\_t elapsed = 25 - (clock() - tic); // 检查上一帧渲染时间，并计算与25的差值

Sleep(elapsed > 0 ? elapsed : 0); // 若差值大于零，则休眠该差值的毫秒数，以确保每帧渲染不超过50

帧

tic = clock(); // 更新上一次记录的时间

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: ResetSize(int width,int height) Parameter: width(int), height(int)

Return: None(void) Description:

该函数用于根据地图的大小重设命令行窗口的大小，用于正常模块

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ void ResetSize(int width, int height) {

windowWidth = 4 \* width + 4; windowHeight = 2 \* height + 5; InitMenu();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: ProResetSize(int width,int height) Parameter: width(int), height(int)

Return: None(void) Description:

该函数用于根据地图的大小重设命令行窗口的大小，用于扭曲空间模块

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void ProResetSize(int width, int height) { windowWidth = (3 \* width + 2) / 4 \* 4 + 4;

windowWidth = (windowWidth < 40 ? 40 : windowWidth); windowHeight = 2 \* (height) + 6;

InitMenu();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: WaitChoice()

Parameter: None(void) Return: None(void) Description:

主循环，若不退出将一直渲染下去

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ void WaitChoice() {

bool runFlag = true; while (runFlag) {

checkChoice(); // 检查输入

if (choice != oldChoice) {

// 若新的选择与旧选择不一致，将旧选择的背景清空

clearChoiceBackground(oldChoice); oldChoice = choice;

}

randerChoiceBackground(choice); // 绘制选中项的背景

// 如果选中菜单，执行操作

if (isChosen) {

switch (choice) { case 0:

// 开始游戏FormModeChoice(); WaitModeChoice();

if (modeFinallyFlag) {

ProResetSize(mapWidth, mapHeight); GameEnginePro::InitGame(mapWidth, mapHeight, boomNum); GameEnginePro::Play();

GameEnginePro::DestroyGame();

}

else {

}

ResetSize(mapWidth, mapHeight); GameEngine::InitGame(mapWidth, mapHeight, boomNum); GameEngine::Play();

GameEngine::DestroyGame();

case 1:

case 2:

}

}

boomNum = 0;

mapWidth = 0;

mapHeight = 0;

windowHeight = 25;

windowWidth = 44; InitMenu(); renderMenu(); tempNum[0] = 0;

tempNum[1] = 0;

tempNum[2] = 0; modeIsChosen = false; break;

// 绘制说明renderIntro(); renderMenu(); break;

// 将Flag置否以退出循环

runFlag = false; break;

// 处理每帧的事务

frame++; // 渲染帧数自增

clock\_t elapsed = 25 - (clock() - tic); // 检查上一帧渲染时间，并计算与25的差值

Sleep(elapsed > 0 ? elapsed : 0); // 若差值大于零，则休眠该差值的毫秒数，以确保每帧渲染不超过50

帧

tic = clock(); // 更新上一次记录的时间

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Graphic.cpp

Author: Root Date: 2019/12/09 Description:

定义了渲染框架，用Graphic命名空间防止同名污染

！\*！注意：除非你能完全读懂内部都在做什么操作，否则不建议修改这个文件！\*！

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ #include "Graphic.h"

namespace Graphic {

HANDLE hStdOut = NULL; // 标准输出句柄UCHAR\* pCanvas = nullptr; // 画版数组指针UCHAR\* updateQueue = nullptr; // 更新队列数组指针SHORT nowPos[2] = { 0 }; // 当前位置坐标数组

int screenWidth = 0, screenHeight = 0; // 屏幕宽高

char\* outputSequence = nullptr; // 输出屏幕字符串序列指针

size\_t sequenceLength = 0; // 字符串序列的当前长度

size\_t sequenceCapacity = 0; // 字符串序列的容量

size\_t needUpdate = 0; // 需要更新的数量

const DWORD MOD\_FRONT = 0, MOD\_BACK = 1, MOD\_BOTH = 2; // 几个常量

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: MovePos(SHORT x, SHORT y) Parameter: SHORT x, SHORT y

Return: None(void) Description:

将当前坐标更新为入参

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ void MovePos(SHORT x, SHORT y) {

nowPos[0] = x; nowPos[1] = y;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: movePos(SHORT x, SHORT y) Parameter: SHORT x, SHORT y

Return: None(void) Description:

该函数不作导出，在内部将控制台的光标移动至(x, y)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ void movePos(SHORT x, SHORT y) {

SetConsoleCursorPosition(hStdOut, { x, y });

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: pushUpdateQueue(const int screenIndex) Parameter: int screenIndex

Return: None(void) Description:

将要更新的屏幕坐标值推入需要更新的队列

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ void pushUpdateQueue(const int screenIndex) {

// 若没有超更新队列的容量，则推入队列

if (needUpdate < 512) {

// 确认更新队列当前位置下标的常量

const size\_t updateIndex = needUpdate \* 9;

// 分别设置队列的各个值

updateQueue[updateIndex] = pCanvas[screenIndex]; updateQueue[updateIndex + 1] = (UCHAR)nowPos[1] + 1; updateQueue[updateIndex + 2] = (UCHAR)nowPos[0] + 1; updateQueue[updateIndex + 3] = pCanvas[screenIndex + 1]; updateQueue[updateIndex + 4] = pCanvas[screenIndex + 2]; updateQueue[updateIndex + 5] = pCanvas[screenIndex + 3]; updateQueue[updateIndex + 6] = pCanvas[screenIndex + 4]; updateQueue[updateIndex + 7] = pCanvas[screenIndex + 5]; updateQueue[updateIndex + 8] = pCanvas[screenIndex + 6];

}

needUpdate++;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: ModChar(const char newChar) Parameter: char newChar

Return: None(void) Description:

将当前位置(nowPos数组所指示的位置)的字符改为新的newChar

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ void ModChar(const char newChar) {

const int screenIndex = ((int)nowPos[1] \* screenWidth + (int)nowPos[0]) \* 7; pCanvas[screenIndex] = newChar;

pushUpdateQueue(screenIndex);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function: ModColor(DWORD mode, const BYTE textR, const BYTE textG, const BYTE textB, const BYTE backR, const BYTE backG, const BYTE backB)

Parameter: DWORD mode, BYTE textR, textG, textB, backR, backG, backB Return: None(void)

Description:

将当前位置(nowPos数组所指示的位置)的字符的颜色/背景(具体修改哪个由mode指定)改为新的颜色值

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void ModColor(DWORD mode, const BYTE textR, const BYTE textG, const BYTE textB, const BYTE backR, const BYTE backG, const BYTE backB) {

const int screenIndex = ((int)nowPos[1] \* screenWidth + (int)nowPos[0]) \* 7; // 确定当前坐标的数组中的下标

switch (mode) { case MOD\_BOTH:

// 更新前景色与背景色，但不break，switch语句将继续执行

pCanvas[screenIndex + 1] = textR; pCanvas[screenIndex + 2] = textG; pCanvas[screenIndex + 3] = textB;

case MOD\_BACK:

// 更新背景色pCanvas[screenIndex + 4] = backR; pCanvas[screenIndex + 5] = backG; pCanvas[screenIndex + 6] = backB; break;

case MOD\_FRONT:

// 更新前景色pCanvas[screenIndex + 1] = textR; pCanvas[screenIndex + 2] = textG; pCanvas[screenIndex + 3] = textB;

}

pushUpdateQueue(screenIndex);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: PutChar(const char outChar) Parameter: char outChar

Return: None(void) Description:

将当前位置(nowPos数组所指示的位置)放置一个outChar

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ void PutChar(const char outChar) {

const int screenIndex = ((int)nowPos[1] \* screenWidth + (int)nowPos[0]) \* 7; // 确定当前坐标的数组中的下标

pCanvas[screenIndex] = outChar; // 设置字符

pCanvas[screenIndex + 1] = 255; // 设置前景色

pCanvas[screenIndex + 2] = 255;

pCanvas[screenIndex + 3] = 255; pCanvas[screenIndex + 4] = 0; // 设置背景色pCanvas[screenIndex + 5] = 0;

pCanvas[screenIndex + 6] = 0;

pushUpdateQueue(screenIndex); // 推入更新队列

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function: PutCharWithColor(const char outChar, const BYTE textR, const BYTE textG, const BYTE textB, const BYTE backR, const BYTE backG, const BYTE backB)

Parameter: char outChar, BYTE textR, textG, textB, backR, backG, backB Return: None(void)

Description:

将当前位置(nowPos数组所指示的位置)放置一个outChar，并设置为指定的颜色

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void PutCharWithColor(const char outChar, const BYTE textR, const BYTE textG, const BYTE textB, const BYTE backR, const BYTE backG, const BYTE backB) {

const int screenIndex = ((int)nowPos[1] \* screenWidth + (int)nowPos[0]) \* 7; // 确定当前坐标的数组中的下标

pCanvas[screenIndex] = outChar; // 设置字符

pCanvas[screenIndex + 1] = textR; // 设置前景色pCanvas[screenIndex + 2] = textG; pCanvas[screenIndex + 3] = textB; pCanvas[screenIndex + 4] = backR; // 设置背景色

pCanvas[screenIndex + 5] = backG; pCanvas[screenIndex + 6] = backB;

pushUpdateQueue(screenIndex); // 推入更新队列

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: PutString(const char\* outString) Parameter: char\* outString

Return: None(void) Description:

将当前位置(nowPos数组所指示的位置)放置一个字符串outString

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ void PutString(const char\* outString) {

// 检查字符串是否超过屏幕宽度，如果超过则退出操作

const size\_t stringLength = strlen(outString);

if (stringLength + (size\_t)nowPos[0] > (size\_t)screenWidth) { return;

}

// 若不操作，则逐字符的设置字符串

for (size\_t i = 0; i < stringLength; i++) { PutChar(outString[i]); nowPos[0]++;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function: PutStringWithColor(const char\* outString, const BYTE textR, const BYTE textG, const BYTE textB, const BYTE backR, const BYTE backG, const BYTE backB)

Parameter: char\* outString, BYTE textR, textG, textB, backR, backG, backB Return: None(void)

Description:

将当前位置(nowPos数组所指示的位置)放置一个字符串outString，并设置颜色，功能基本同上

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void PutStringWithColor(const char\* outString, const BYTE textR, const BYTE textG, const BYTE textB, const BYTE backR, const BYTE backG, const BYTE backB) {

const size\_t stringLength = strlen(outString);

if (stringLength + (size\_t)nowPos[0] > (size\_t)screenWidth) { return;

}

for (size\_t i = 0; i < stringLength; i++) {

PutCharWithColor(outString[i], textR, textG, textB, backR, backG, backB); nowPos[0]++;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function: DrawRect(const SHORT width, const SHORT height) Parameter: SHORT width, SHORT height

Return: None(void) Description:

将当前位置(nowPos数组所指示的位置)绘制一个长为width，宽为height的矩形

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void DrawRect(const SHORT width, const SHORT height) { const SHORT x = nowPos[0], y = nowPos[1];

// 检查长宽是否超过屏幕

if (x + width > (SHORT)screenWidth || nowPos[1] + height > (SHORT)screenHeight) { return;

}

for (SHORT i = x; i < x + width; i++) {

for (SHORT j = y; j < y + height; j++) {

// 先设置位置，然后放置一个带颜色的空白

MovePos(i, j);

PutCharWithColor(' ', 255, 255, 255, 255, 255, 255);

}

}

nowPos[0] = x; nowPos[1] = y;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function: DrawRectWithColor(const SHORT width, const SHORT height, const BYTE textR, const BYTE textG, const BYTE textB, const BYTE backR, const BYTE backG, const BYTE backB)

Parameter: SHORT width, height, BYTE textR, textG, textB, backR, backG, backB Return: None(void)

Description:

将当前位置(nowPos数组所指示的位置)绘制一个长为width，宽为height的矩形，并带颜色，操作基本相同

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void DrawRectWithColor(const SHORT width, const SHORT height, const BYTE textR, const BYTE textG, const BYTE textB, const BYTE backR, const BYTE backG, const BYTE backB) {

const SHORT x = nowPos[0], y = nowPos[1];

if (x + width > (SHORT)screenWidth || nowPos[1] + height > (SHORT)screenHeight) { return;

}

for (SHORT i = x; i < x + width; i++) {

for (SHORT j = y; j < y + height; j++) { MovePos(i, j);

PutCharWithColor(' ', textR, textG, textB, backR, backG, backB);

}

}

nowPos[0] = x; nowPos[1] = y;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: ClearScreen()

Parameter: None(void) Return: None(void) Description:

强制清屏操作

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ void ClearScreen() {

memset(pCanvas, 0, screenWidth \* screenHeight \* 7 \* sizeof(UCHAR)); // 清空画板数组

memset(updateQueue, 0, 512 \* 9 \* sizeof(UCHAR)); // 清空队列数组

needUpdate = 999; // 将更新值设置

超出容量的数量以应用强制更新

Update(); // 更新屏

幕

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: InitGraphic(int width, int height) Parameter: int width, height

Return: None(void) Description:

初始化图形操作，将屏幕长宽设置为width和height

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ void InitGraphic(int width, int height) {

screenWidth = width \* 2; screenHeight = height; // 设置长宽

pCanvas = new UCHAR[width \* height \* 14]; // 设置画板数组，动态申请

updateQueue = new UCHAR[512 \* 9]; // 设置更新队列数组，动态

申请

outputSequence = new char[(size\_t)width \* (size\_t)height \* 128]; // 设置输出字符串序列数组，动态申请

sequenceCapacity = (int)width \* (int)height \* 128; // 设置字符串序列的容量

system("chcp 936"); // 更新控制台代码页

// 将控制台设置为这个长宽

char cmd\_buffer[32] = "mode con: cols=0000 lines=0000"; cmd\_buffer[15] = '0' + (width / 1000 % 10);

cmd\_buffer[16] = '0' + (width / 100 % 10);

cmd\_buffer[17] = '0' + (width / 10 % 10); cmd\_buffer[18] = '0' + width % 10;

cmd\_buffer[26] = '0' + (height / 1000 % 10);

cmd\_buffer[27] = '0' + (height / 100 % 10);

cmd\_buffer[28] = '0' + (height / 10 % 10); cmd\_buffer[29] = '0' + height % 10;

system(cmd\_buffer);

// 隐藏光标

hStdOut = GetStdHandle(STD\_OUTPUT\_HANDLE); // 获取标准输出句柄

CONSOLE\_CURSOR\_INFO cci;

cci.bVisible = 0;

cci.dwSize = 1; SetConsoleCursorInfo(hStdOut, &cci);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: DestroyGraphic()

Parameter: None(void) Return: None(void) Description:

销毁操作，将动态初始化的数组回收

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ void DestroyGraphic() {

delete[] pCanvas; delete[] updateQueue;

delete[] outputSequence;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: constructUpdateSequence() Parameter: None(void)

Return: None(void) Description:

构建更新操作的字符串序列，注意：这里面的内容需要Win10系统且已打上2017年秋季创意者更新，理论上目前电脑都已 经打过这个补丁

若操作无效，请咨询助教领取另一份Graphic文件使用旧的版本 另本函数没有办法注释清楚，若需要理解，请访问下列网站

https://docs.microsoft.com/en-us/windows/console/console-virtual-terminal-sequences

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ void constructUpdateSequence() {

sequenceLength = 0; UCHAR oldColor[8] = { 0 }; UCHAR oldY = 255;

memset(oldColor, 33, 8);

for (size\_t i = 0; i < needUpdate; i++) { const size\_t queueIndex = i \* 9;

if (memcmp(oldColor, updateQueue + queueIndex + 3, 6) == 0) { if (oldY != updateQueue[queueIndex + 1]) {

char sequence[16] = "\x1b[000;000H ";

sequence[10] = updateQueue[queueIndex]; if (sequence[10] == '\0') sequence[10] = ' ';

sequence[2 + 2] = '0' + updateQueue[queueIndex + 1] % 10;

sequence[2 + 1] = '0' + updateQueue[queueIndex + 1] / 10 % 10;

sequence[2] = '0' + updateQueue[queueIndex + 1] / 100;

sequence[6 + 2] = '0' + updateQueue[queueIndex + 2] % 10;

sequence[6 + 1] = '0' + updateQueue[queueIndex + 2] / 10 % 10;

sequence[6] = '0' + updateQueue[queueIndex + 2] / 100;

memcpy\_s(outputSequence + sequenceLength, sequenceCapacity, sequence, 11); sequenceLength += 11;

}

else {

}

}

else {

oldY = updateQueue[queueIndex + 1];

outputSequence[sequenceLength] = updateQueue[queueIndex]; if (outputSequence[sequenceLength] == '\0') {

outputSequence[sequenceLength] = ' ';

}

sequenceLength++;

char sequence[64] = "\x1b[000;000H\x1b[38;2;000;000;000m\x1b[48;2;000;000;000m ";//\x1b[0m

sequence[48] = updateQueue[queueIndex]; if (sequence[48] == '\0')sequence[48] = ' ';

sequence[2 + 2] = '0' + updateQueue[queueIndex + 1] % 10;

sequence[2 + 1] = '0' + updateQueue[queueIndex + 1] / 10 % 10;

sequence[2] = '0' + updateQueue[queueIndex + 1] / 100;

sequence[6 + 2] = '0' + updateQueue[queueIndex + 2] % 10;

sequence[6 + 1] = '0' + updateQueue[queueIndex + 2] / 10 % 10;

sequence[6] = '0' + updateQueue[queueIndex + 2] / 100;

sequence[17 + 2] = '0' + updateQueue[queueIndex + 3] % 10;

sequence[17 + 1] = '0' + updateQueue[queueIndex + 3] / 10 % 10;

sequence[17] = '0' + updateQueue[queueIndex + 3] / 100;

sequence[21 + 2] = '0' + updateQueue[queueIndex + 4] % 10;

sequence[21 + 1] = '0' + updateQueue[queueIndex + 4] / 10 % 10;

sequence[21] = '0' + updateQueue[queueIndex + 4] / 100;

sequence[25 + 2] = '0' + updateQueue[queueIndex + 5] % 10;

sequence[25 + 1] = '0' + updateQueue[queueIndex + 5] / 10 % 10;

sequence[25] = '0' + updateQueue[queueIndex + 5] / 100;

sequence[36 + 2] = '0' + updateQueue[queueIndex + 6] % 10;

sequence[36 + 1] = '0' + updateQueue[queueIndex + 6] / 10 % 10;

sequence[36] = '0' + updateQueue[queueIndex + 6] / 100;

sequence[40 + 2] = '0' + updateQueue[queueIndex + 7] % 10;

sequence[40 + 1] = '0' + updateQueue[queueIndex + 7] / 10 % 10;

sequence[40] = '0' + updateQueue[queueIndex + 7] / 100;

sequence[44 + 2] = '0' + updateQueue[queueIndex + 8] % 10;

sequence[44 + 1] = '0' + updateQueue[queueIndex + 8] / 10 % 10;

sequence[44] = '0' + updateQueue[queueIndex + 8] / 100;

memcpy\_s(outputSequence + sequenceLength, sequenceCapacity, sequence, 49); sequenceLength += 49;

memcpy\_s(oldColor, 8, updateQueue + queueIndex + 3, 6); oldY = updateQueue[queueIndex + 1];

}

}

outputSequence[sequenceLength] = '\0';

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: constructScreenSequence() Parameter: None(void)

Return: None(void) Description:

更新全屏幕的字符串，无法注释，若需理解请参考上述网站

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ void constructScreenSequence() {

sequenceLength = 0; UCHAR oldColor[8] = { 0 }; memset(oldColor, 33, 8);

for (size\_t i = 0; i < (size\_t)screenHeight; i++) { char posSequence[16] = "\x1b[001;001H"; posSequence[2 + 2] = '0' + (i + 1) % 10;

posSequence[2 + 1] = '0' + (i + 1) / 10 % 10;

posSequence[2] = '0' + (char)(i + 1) / 100;

memcpy\_s(outputSequence + sequenceLength, sequenceCapacity, posSequence, 10); sequenceLength += 10;

for (size\_t j = 0; j < (size\_t)screenWidth; j++) {

const size\_t screenIndex = (i \* screenWidth + j) \* 7;

if (j > (size\_t)screenWidth / 2 && pCanvas[screenIndex] == 0) { break;

}

if (memcmp(oldColor, pCanvas + screenIndex + 1, 6) == 0) { outputSequence[sequenceLength] = pCanvas[screenIndex]; if (outputSequence[sequenceLength] == '\0') {

outputSequence[sequenceLength] = ' ';

}

else {

}

sequenceLength++;

char sequence[64] = "\x1b[38;2;000;000;000m\x1b[48;2;000;000;000m ";//\x1b[0m sequence[38] = pCanvas[screenIndex];

if (sequence[38] == '\0')sequence[38] = ' ';

sequence[7 + 2] = '0' + pCanvas[screenIndex + 1] % 10;

sequence[7 + 1] = '0' + pCanvas[screenIndex + 1] / 10 % 10;

sequence[7] = '0' + pCanvas[screenIndex + 1] / 100;

sequence[11 + 2] = '0' + pCanvas[screenIndex + 2] % 10;

sequence[11 + 1] = '0' + pCanvas[screenIndex + 2] / 10 % 10;

sequence[11] = '0' + pCanvas[screenIndex + 2] / 100;

sequence[15 + 2] = '0' + pCanvas[screenIndex + 3] % 10;

sequence[15 + 1] = '0' + pCanvas[screenIndex + 3] / 10 % 10;

sequence[15] = '0' + pCanvas[screenIndex + 3] / 100;

sequence[26 + 2] = '0' + pCanvas[screenIndex + 4] % 10;

sequence[26 + 1] = '0' + pCanvas[screenIndex + 4] / 10 % 10;

sequence[26] = '0' + pCanvas[screenIndex + 4] / 100;

sequence[30 + 2] = '0' + pCanvas[screenIndex + 5] % 10;

sequence[30 + 1] = '0' + pCanvas[screenIndex + 5] / 10 % 10;

sequence[30] = '0' + pCanvas[screenIndex + 5] / 100;

sequence[34 + 2] = '0' + pCanvas[screenIndex + 6] % 10;

sequence[34 + 1] = '0' + pCanvas[screenIndex + 6] / 10 % 10;

sequence[34] = '0' + pCanvas[screenIndex + 6] / 100;

memcpy\_s(outputSequence + sequenceLength, sequenceCapacity, sequence, 39); sequenceLength += 39;

memcpy\_s(oldColor, 8, pCanvas + screenIndex + 1, 6);

}

}

}

outputSequence[sequenceLength] = '\0';

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: Update()

Parameter: None(void) Return: None(void) Description:

将所有操作更新至屏幕，也就是说在这里做的所有操作都不是实时的，只有调用这个函数后屏幕才会真正的更新

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ void Update() {

movePos(0, 0); // 将光标移动至0,0，硬移动

if (needUpdate < 512) {

constructUpdateSequence(); // 如果要更新的数量小于512个，那么则构建更新序列

}

else {

}

constructScreenSequence();// 否则构建全屏幕更新序列

needUpdate = 0; // 重置更新数量

// 将所有操作写入到控制台窗口屏幕

DWORD dWrite = 0;

WriteConsoleA(hStdOut, outputSequence, sequenceLength, &dWrite, NULL);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Main.cpp

Author: Root Date: 2019/12/09 Description:

定义了整个程序的入点，引入Menu头文件，初始化菜单，并进入等待选择的循环

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ #include "GameMenu.h"

int main()

{

GameMenu::InitMenu(); GameMenu::WaitChoice();

}