# 兰顿蚂蚁第三次工作报告

软件1班 202430551743 严嘉鹏

一、闯关模式

考虑到游戏规则，这里的闯关模式就先设置了3关，每一关根据回溯步数又分为3个不同的难度模式（低、中、高）。从思考上很容易达成，但是，一旦自己开始写的时候又会发现过程过于繁琐，目前又无法优化。

1. 基础功能介绍：

**levelMode.h如下：**

class LevelMode {

public:

void chooseStage(User& user);//选择关卡和难度模式

void stageBoard(int \_stage, int \_diffic, User& user);//某关某难度下电脑内置的蚁行图,计算游戏完成时间

double compareBoard(int \_userOne[], int \_initialOne[], int \_size,bool shouldPrint =true);//打印初始棋盘和用户作答，并比较后计算出相似率（正确率）

int judgeBoard(int \_userOne[], int \_initialOne[], int \_size);//判断用户是否答对

};

1. 特别功能学习：

计算用时(C++有一个库里含有该方法，直接调用)：头文件<ctime>

std::clock\_t start\_time = std::clock();

………….

std::clock\_t end\_time = std::clock();

double elapsed\_time = static\_cast<double>(end\_time - start\_time) / CLOCKS\_PER\_SEC;

(3)遇到问题：compareBoard有两个功能，第一个是打印输出初始棋盘和用户做答以及输出相似率，第二个功能是返回相似率这个值作为游戏记录构造函数的参数。所以有两次调用，要分别输出不同的结果。原先做法：设置一个成员变量callcount用来区分，但是游戏的进行是用switch写的，仔细分析，无法通过callcount++后重置为0来解决该函数在后续操作的重复使用。这里就利用给他加一个默认参数的方法来解决，原型

double compareBoard(int \_userOne[], int \_initialOne[], int \_size,bool shouldPrint =true);最右边加了一个默认参数（形参列表的默认参数只能放在最右边，而且不能在函数那里重定义，相同的默认值也不行，编译器会报错），第一次调用默认为true,第二次调用修改参数为false,添加两条if语句做判断就很好的解决了这个问题。

compareBoard(userOne1, initialOne1, 3);

std::cout << "游戏用时：" << elapsed\_time << "秒" << std::endl;

double accuracy = compareBoard(userOne1, initialOne1, 3,0);

1. 具体代码

**levelMode.cpp如下:(目前除了switch，if,while还想不到方法可以优化这种多分支的代码)**

#include"levelMode.h"

#include"gameHomepage.h"

#include <iostream>

double LevelMode::compareBoard(int \_userOne[], int \_initialOne[], int \_size,bool shouldPrint)

{

if (shouldPrint==true)

{

std::cout << "初始棋盘如下：" << std::endl;

for (int i = 0; i < \_size; i++)

{

for (int j = 0; j < \_size; j++)

{

std::cout << \_initialOne[i \* \_size + j] << " ";

}

std::cout << std::endl;

}

std::cout << "用户作答如下：" << std::endl;

for (int i = 0; i < \_size; i++)

{

for (int j = 0; j < \_size; j++)

{

std::cout << \_userOne[i \* \_size + j] << " ";

}

std::cout << std::endl;

}

}

double counter = 0;

for (int i = 0; i < \_size \* \_size; i++)

{

if (\_userOne[i] == \_initialOne[i])

{

counter++;

}

}

double accuracyRate = counter / (\_size \* \_size);

if (shouldPrint==true)

{

std::cout << "相似率为：" << accuracyRate << std::endl;

}

return accuracyRate;

}

int LevelMode::judgeBoard(int \_userOne[], int \_initialOne[], int \_size)

{

for (int i = 0; i < \_size \* \_size; i++)

{

if (\_userOne[i] != \_initialOne[i])

{

std::cout << "回答错误！" << std::endl;

return 0;

}

}

std::cout << "恭喜您，回答正确！" << std::endl;

return 1;

}

void LevelMode::stageBoard(int \_stage, int \_diffic, User& user)

{

switch (\_stage)

{

case 1:

{

switch (\_diffic)

{

case 1:

{

int initialOne1[4] = { 0,0,0,0 };

int finalOne1[4] = { 0,1,1,1 };

int userOne1[4] = { 0 };

std::cout << "蚁行图如下" << std::endl;

for (int i = 0; i < 2; i++)

{

for (int j = 0; j < 2; j++)

{

std::cout << finalOne1[i \* 2 + j] << " ";

}

std::cout << std::endl;

}

std::cout << "已知：2\*2棋盘，已走5步，蚂蚁位置（1，0），蚂蚁朝向UP" << std::endl;

std::cout << "请用户开始作答，输入原始棋盘" << std::endl;

std::clock\_t start\_time = std::clock();

for (int i = 0; i < 2; i++)

{

for (int j = 0; j < 2; j++)

{

std::cin >> userOne1[i \* 2 + j];

}

}

std::clock\_t end\_time = std::clock();

double elapsed\_time = static\_cast<double>(end\_time - start\_time) / CLOCKS\_PER\_SEC;

judgeBoard(userOne1, initialOne1, 2);

compareBoard(userOne1, initialOne1, 2);

std::cout << "游戏用时：" << elapsed\_time << "秒" << std::endl;

double accuracy = compareBoard(userOne1, initialOne1, 2,0);

GameRecord record(user.getUsername(), \_stage, \_diffic, elapsed\_time, accuracy);

user.addRecord(record);

break;

}

case 2:

{

int initialOne1[4] = { 0,0,0,0 };

int finalOne1[4] = { 1,0,1,0 };

int userOne1[4] = { 0 };

std::cout << "蚁行图如下" << std::endl;

for (int i = 0; i < 2; i++)

{

for (int j = 0; j < 2; j++)

{

std::cout << finalOne1[i \* 2 + j] << " ";

}

std::cout << std::endl;

}

std::cout << "已知：2\*2棋盘，已走10步，蚂蚁位置（1，1），蚂蚁朝向LEFT" << std::endl;

std::cout << "请用户开始作答，输入原始棋盘" << std::endl;

std::clock\_t start\_time = std::clock();

for (int i = 0; i < 2; i++)

{

for (int j = 0; j < 2; j++)

{

std::cin >> userOne1[i \* 2 + j];

}

}

std::clock\_t end\_time = std::clock();

double elapsed\_time = static\_cast<double>(end\_time - start\_time) / CLOCKS\_PER\_SEC;

judgeBoard(userOne1, initialOne1, 2);

compareBoard(userOne1, initialOne1, 2);

std::cout << "游戏用时：" << elapsed\_time << "秒" << std::endl;

double accuracy = compareBoard(userOne1, initialOne1, 2,0);

GameRecord record(user.getUsername(), \_stage, \_diffic, elapsed\_time, accuracy);

user.addRecord(record);

break;

}

case 3:

{

int initialOne1[4] = { 0,0,0,0 };

int finalOne1[4] = { 1,1,1,1 };

int userOne1[4] = { 0 };

std::cout << "蚁行图如下" << std::endl;

for (int i = 0; i < 2; i++)

{

for (int j = 0; j < 2; j++)

{

std::cout << finalOne1[i \* 2 + j] << " ";

}

std::cout << std::endl;

}

std::cout << "已知：2\*2棋盘，已走20步，蚂蚁位置（0，0），蚂蚁朝向RIGHT" << std::endl;

std::cout << "请用户开始作答，输入原始棋盘" << std::endl;

std::clock\_t start\_time = std::clock();

for (int i = 0; i < 2; i++)

{

for (int j = 0; j < 2; j++)

{

std::cin >> userOne1[i \* 2 + j];

}

}

std::clock\_t end\_time = std::clock();

double elapsed\_time = static\_cast<double>(end\_time - start\_time) / CLOCKS\_PER\_SEC;

judgeBoard(userOne1, initialOne1, 2);

compareBoard(userOne1, initialOne1, 2);

std::cout << "游戏用时：" << elapsed\_time << "秒" << std::endl;

double accuracy = compareBoard(userOne1, initialOne1, 2,0);

GameRecord record(user.getUsername(), \_stage, \_diffic, elapsed\_time, accuracy);

user.addRecord(record);

break;

}

}

break;

}

case 2:

{

switch (\_diffic)

{

case 1:

{

int initialOne1[9] = { 0,0,1,0,0,1,1,0,1 };

int finalOne1[9] = { 1,0,1,0,0,1,0,1,1 };

int userOne1[9] = { 0 };

std::cout << "蚁行图如下" << std::endl;

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

{

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

{

std::cout << finalOne1[i \* 3 + j] << " ";

}

std::cout << std::endl;

}

std::cout << "已知：3\*3棋盘，已走5步，蚂蚁位置（2，1），蚂蚁朝向DOWN" << std::endl;

std::cout << "请用户开始作答，输入原始棋盘" << std::endl;

std::clock\_t start\_time = std::clock();

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

{

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

{

std::cin >> userOne1[i \* 3 + j];

}

}

std::clock\_t end\_time = std::clock();

double elapsed\_time = static\_cast<double>(end\_time - start\_time) / CLOCKS\_PER\_SEC;

judgeBoard(userOne1, initialOne1, 3);

compareBoard(userOne1, initialOne1, 3);

std::cout << "游戏用时：" << elapsed\_time << "秒" << std::endl;

double accuracy = compareBoard(userOne1, initialOne1, 3,0);

GameRecord record(user.getUsername(), \_stage, \_diffic, elapsed\_time, accuracy);

user.addRecord(record);

break;

}

case 2:

{

int initialOne1[9] = { 0,0,1,0,0,1,1,0,1 };

int finalOne1[9] = { 1,1,1,0,1,0,0,0,0 };

int userOne1[9] = { 0 };

std::cout << "蚁行图如下" << std::endl;

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

{

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

{

std::cout << finalOne1[i \* 3 + j] << " ";

}

std::cout << std::endl;

}

std::cout << "已知：3\*3棋盘，已走10步，蚂蚁位置（0，2），蚂蚁朝向RIGHT" << std::endl;

std::cout << "请用户开始作答" << std::endl;

std::clock\_t start\_time = std::clock();

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

{

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

{

std::cin >> userOne1[i \* 3 + j];

}

}

std::clock\_t end\_time = std::clock();

double elapsed\_time = static\_cast<double>(end\_time - start\_time) / CLOCKS\_PER\_SEC;

judgeBoard(userOne1, initialOne1, 3);

compareBoard(userOne1, initialOne1, 3);

std::cout << "游戏用时：" << elapsed\_time << "秒" << std::endl;

double accuracy = compareBoard(userOne1, initialOne1, 3,0);

GameRecord record(user.getUsername(), \_stage, \_diffic, elapsed\_time, accuracy);

user.addRecord(record);

break;

}

case 3:

{

int initialOne1[9] = { 0,0,1,0,0,1,1,0,1 };

int finalOne1[9] = { 1,1,1,0,1,0,0,0,0 };

int userOne1[9] = { 0 };

std::cout << "蚁行图如下" << std::endl;

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

{

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

{

std::cout << finalOne1[i \* 3 + j] << " ";

}

std::cout << std::endl;

}

std::cout << "已知：3\*3棋盘，已走20步，蚂蚁位置（0，2），蚂蚁朝向RIGHT" << std::endl;

std::cout << "请用户开始作答" << std::endl;

std::clock\_t start\_time = std::clock();

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

{

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

{

std::cin >> userOne1[i \* 3 + j];

}

}

std::clock\_t end\_time = std::clock();

double elapsed\_time = static\_cast<double>(end\_time - start\_time) / CLOCKS\_PER\_SEC;

judgeBoard(userOne1, initialOne1, 3);

compareBoard(userOne1, initialOne1, 3);

std::cout << "游戏用时：" << elapsed\_time << "秒" << std::endl;

double accuracy = compareBoard(userOne1, initialOne1, 3,0);

GameRecord record(user.getUsername(), \_stage, \_diffic, elapsed\_time, accuracy);

user.addRecord(record);

break;

}

}

break;

}

case 3:

{

switch (\_diffic)

{

case 1:

{

int initialOne1[16] = { 0,0,0,0,0,0,1,0,0,1,1,1,0,0,1,0 };

int finalOne1[16] = { 1,0,0,0,0,0,1,0,0,1,1,1,0,0,1,0 };

int userOne1[16] = { 0 };

std::cout << "蚁行图如下" << std::endl;

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

{

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

{

std::cout << finalOne1[i \* 4 + j] << " ";

}

std::cout << std::endl;

}

std::cout << "已知：4\*4棋盘，已走5步，蚂蚁位置（3,0），蚂蚁朝向DOWN" << std::endl;

std::cout << "请用户开始作答" << std::endl;

std::clock\_t start\_time = std::clock();

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

{

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

{

std::cin >> userOne1[i \* 4 + j];

}

}

std::clock\_t end\_time = std::clock();

double elapsed\_time = static\_cast<double>(end\_time - start\_time) / CLOCKS\_PER\_SEC;

judgeBoard(userOne1, initialOne1, 4);

compareBoard(userOne1, initialOne1, 4);

std::cout << "游戏用时：" << elapsed\_time << "秒" << std::endl;

double accuracy = compareBoard(userOne1, initialOne1, 4,0);

GameRecord record(user.getUsername(), \_stage, \_diffic, elapsed\_time, accuracy);

user.addRecord(record);

break;

}

case 2:

{

int initialOne1[16] = { 0,0,0,0,0,0,1,0,0,1,1,1,0,0,1,0 };

int finalOne1[16] = { 1,0,0,0,0,0,1,0,0,1,1,1,1,0,1,0 };

int userOne1[16] = { 0 };

std::cout << "蚁行图如下" << std::endl;

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

{

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

{

std::cout << finalOne1[i \* 4 + j] << " ";

}

std::cout << std::endl;

}

std::cout << "已知：4\*4棋盘，已走10步，蚂蚁位置（3，0），蚂蚁朝向LEFT" << std::endl;

std::cout << "请用户开始作答" << std::endl;

std::clock\_t start\_time = std::clock();

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

{

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

{

std::cin >> userOne1[i \* 4 + j];

}

}

std::clock\_t end\_time = std::clock();

double elapsed\_time = static\_cast<double>(end\_time - start\_time) / CLOCKS\_PER\_SEC;

judgeBoard(userOne1, initialOne1, 4);

compareBoard(userOne1, initialOne1, 4);

std::cout << "游戏用时：" << elapsed\_time << "秒" << std::endl;

double accuracy = compareBoard(userOne1, initialOne1, 4,0);

GameRecord record(user.getUsername(), \_stage, \_diffic, elapsed\_time, accuracy);

user.addRecord(record);

break;

}

case 3:

{

int initialOne1[16] = { 0,0,0,0,0,0,1,0,0,1,1,1,0,0,1,0 };

int finalOne1[16] = { 1,0,0,0,0,0,1,0,0,1,1,1,1,0,1,0 };

int userOne1[16] = { 0 };

std::cout << "蚁行图如下" << std::endl;

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

{

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

{

std::cout << finalOne1[i \* 4 + j] << " ";

}

std::cout << std::endl;

}

std::cout << "已知：4\*4棋盘，已走20步，蚂蚁位置（3,0），蚂蚁朝向LEFT" << std::endl;

std::cout << "请用户开始作答" << std::endl;

std::clock\_t start\_time = std::clock();

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

{

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

{

std::cin >> userOne1[i \* 4 + j];

}

}

std::clock\_t end\_time = std::clock();

double elapsed\_time = static\_cast<double>(end\_time - start\_time) / CLOCKS\_PER\_SEC;

judgeBoard(userOne1, initialOne1, 4);

compareBoard(userOne1, initialOne1, 4);

std::cout << "游戏用时：" << elapsed\_time << "秒" << std::endl;

double accuracy = compareBoard(userOne1, initialOne1, 4,0);

GameRecord record(user.getUsername(), \_stage, \_diffic, elapsed\_time, accuracy);

user.addRecord(record);

break;

}

}

break;

}

}

}

void LevelMode::chooseStage(User& user)

{

std::cout << "欢迎来到闯关模式" << std::endl;

std::cout << "请注意：本模式下，请选择您想参与的关卡和难度，然后电脑会打印对应的蚁行图供您作答" << std::endl;

std::cout << "请输入您想参与的关卡（输入阿拉伯数字1、2、3)和难度（1低、2中、3高）" << std::endl;

int stages = 0;

int difficulty = 0;

std::cin >> stages >> difficulty;

switch (stages)

{

case 1:

{

std::cout << "您选择的是第一关" << std::endl;

switch (difficulty)

{

case 1:

{

std::cout << "您选择的是低难度" << std::endl;

stageBoard(1, 1, user);

user.displayRecords();

break;

}

case 2:

{

std::cout << "您选择的是中难度" << std::endl;

stageBoard(1, 2, user);

user.displayRecords();

break;

}

case 3:

{

std::cout << "您选择的是高难度" << std::endl;

stageBoard(1, 3, user);

user.displayRecords();

break;

}

}

break;

}

case 2:

{

std::cout << "您选择的是第二关" << std::endl;

switch (difficulty)

{

case 1:

{

std::cout << "您选择的是低难度" << std::endl;

stageBoard(2, 1, user);

user.displayRecords();

break;

}

case 2:

{

std::cout << "您选择的是中难度" << std::endl;

stageBoard(2, 2, user);

user.displayRecords();

break;

}

case 3:

{

std::cout << "您选择的是高难度" << std::endl;

stageBoard(2, 3, user);

user.displayRecords();

break;

}

}

break;

}

case 3:

{

std::cout << "您选择的是第三关" << std::endl;

switch (difficulty)

{

case 1:

{

std::cout << "您选择的是低难度" << std::endl;

stageBoard(3, 1, user);

user.displayRecords();

break;

}

case 2:

{

std::cout << "您选择的是中难度" << std::endl;

stageBoard(3, 2, user);

user.displayRecords();

break;

}

case 3:

{

std::cout << "您选择的是高难度" << std::endl;

stageBoard(3, 3, user);

user.displayRecords();

break;

}

}

break;

}

default:

{

std::cout << "您输入的关卡不存在，请重新输入" << std::endl;

chooseStage(user);

}

}

std::cout << "输入您的用户名，可一键查看所有闯关记录：" << std::endl;

std::string username1;

std:: cin >> username1;

user.printAllRecordsFromFile(username1);

}

二、游戏记录类（闯关模式）

**（1）说明：**

本周在这个类上耗费了最长的时间。先谈思路，刚开始认为有两种方法来实现，第1种是将用户名作为文件名，以追加的形式不断往里面填充用户、模式选择、正确率、用时。第2种是设计一个游戏记录类专门用来管理游戏记录的内容，再创建一个用户类，通过用户对象，调用每个游戏模式，在游戏结束时保存游戏记录。如此一看，其实二者可以一起用，最后我选择了了用文件输入输出流将用户的用户名、密码、登录时间和各条游戏记录（追加）写入关联用户名的文件，再利用类和对象，把游戏记录的数据作为参数传入游戏记录的构造函数里保存，通过display()显示某个游戏记录的具体信息。

然而，问题就出现在游戏记录的保存上，涉及两个难点：作用域识别难（多类和对象，多函数得到的数据，很难传入构造函数里），用户和对应游戏记录的链接问题（如果模式简单，可以创建对象数组，通过对象来调用游戏模式，从而每一个对象都能有对应的成员变量副本，在一个类下才方便传入数据）。

这里暴露出我的类和对象设计的不合理，我在尝试保存游戏记录时发生了大量的作用域无法识别问题和用户的游戏记录做不到完整的对应，而且这种多个类和对象的代码，我一旦改动一点，各个.h和.cpp都在报错，花费大量时间debug后我放弃了原来的类图，上传代码给AI，要求逻辑不变，重新设计类图。如下：

**原先：**

gameMode类

Main.cpp

gameRecord类

freedomMode.h（自由模式）

gameHomepage.h主页面

levelMode.h(闯关模式)

gameHomepage类

User类

freedomMode.cpp

levelMode.cpp

gameHomepage.cpp

**AI调整：(一个类一种头文件，功能分区管理)**

Main.cpp

void addRecord(const GameRecord& record);

void displayRecords();

std::vector<GameRecord>& getRecords() const { return records; }管理用户游戏记录

std::vector<GameRecord> records;

用户注册与登录

Begingame()

levelMode.h/cpp

freedomMode.h/cpp

User.h/cpp

gameRecord.h

gamehomePage.h/cpp

**（2）游戏记录类：**

在AI修改完类图后，，将几个游戏模式都定义成函数接口，在gamehomePage类里创建对应的游戏模式类的对象，用来调用参与不同的游戏模式，用户和游戏记录的链接就采用往游戏模式函数的参数传入用户类对象（用用户文件名初始化）来解决。关键代码如下：

class GameRecord {

public:

GameRecord(const std::string& \_username, int \_stage, int \_difficulty, double \_elapsedTime, double \_accuracy)

: username(\_username), stage(\_stage), difficulty(\_difficulty), elapsedTime(\_elapsedTime), accuracy(\_accuracy) {

}

std::string username;

int stage;

int difficulty;

double elapsedTime;

double accuracy;

};

#include "gameHomepage.h"

#include <iostream>

GameHomepage::GameHomepage() {}

void GameHomepage::beginGame() {

std::cout << "欢迎来到兰顿蚂蚁游戏" << std::endl;

while (true) {

std::cout << "请输入用户名" << std::endl;

std::string username;

std::cin >> username;

std::cout << "请输入密码" << std::endl;

std::string password;

std::cin >> password;

User user(username);

if (user.userLogin(password)) {

chooseMode(user);

break;

}

else {

std::cout << "用户未注册,请先完成账号注册" << std::endl;

std::cout << "请输入用户名" << std::endl;

std::string username2;

std::cin >> username2;

std::cout << "请输入密码" << std::endl;

std::string password2;

std::cin >> password2;

user = User(username2);

user.userRegister(username2,password2);

}

}

}

void GameHomepage::chooseMode(User& user) {

std::cout << "请选择游戏模式" << std::endl;

std::cout << "1. 自由模式" << std::endl;

std::cout << "2. 闯关模式" << std::endl;

std::cout << "请输入你想参加的模式对应的序号" << std::endl;

int choice;

std::cin >> choice;

switch (choice) {

case 1:

freedommode.freeMode();

break;

case 2:

levelMode.chooseStage(user);

break;

default:

std::cout << "输入无效，请重新选择。" << std::endl;

chooseMode(user);

}

}

**（3）将时间戳转化为当前时间**

在 C++ 里，时间戳一般指的是从某个固定的起始点（通常是 1970 年 1 月 1 日 00:00:00 UTC，也就是 Unix 纪元）到指定时间所经过的秒数或者毫秒数。C++ 标准库提供了多种处理时间和时间戳的方式。

// 记录新的登录时间（使用统一的关键字）

std::time\_t now = std::time(nullptr);

std::tm localTime;

if (localtime\_s(&localTime, &now) == 0) {

char newTimeStr[100];

std::strftime(newTimeStr, sizeof(newTimeStr), "%Y-%m-%d %H:%M:%S", &localTime);

std::ofstream outFile(filename, std::ios::app);

if (outFile.is\_open()) {

// 修改为统一的关键字前缀

outFile << "最近的登录时间: " << newTimeStr << std::endl;

outFile.close();

}

else {

std::cerr << "无法记录登录时间。" << std::endl;

}

}

else {

std::cerr << "获取本地时间失败。" << std::endl;

}

// 读取文件寻找最近的登录时间

std::ifstream readFile(username + ".txt");

std::string lastLoginTime;

if (readFile.is\_open()) {

std::string line;

// 逆序查找最后一个包含"最近的登录时间:"的行

std::vector<std::string> allLines;

while (std::getline(readFile, line)) {

allLines.push\_back(line);

}

// 反向遍历查找最后一个匹配项

for (auto it = allLines.rbegin(); it != allLines.rend(); ++it) {

if (it->find("最近的登录时间:") != std::string::npos) {

lastLoginTime = \*it;

break; // 找到最后一个后退出循环

}

}

readFile.close();

}

else {

std::cerr << "无法打开文件。" << std::endl;

}

// 输出最近的登录时间

if (!lastLoginTime.empty()) {

std::cout << lastLoginTime << std::endl;

}

else {

std::cout << "没有找到之前的登录时间记录。" << std::endl;

}

三、实验心得与规划

（1）在解决逻辑问题方面，询问deepseek往往更有效

（2）做项目最终要的是先把类图搭建好。

（3）单人开发小项目暴露出了很多问题：对类和对象理解不够深，对文件输入输出流和文件存储格式不够了解。

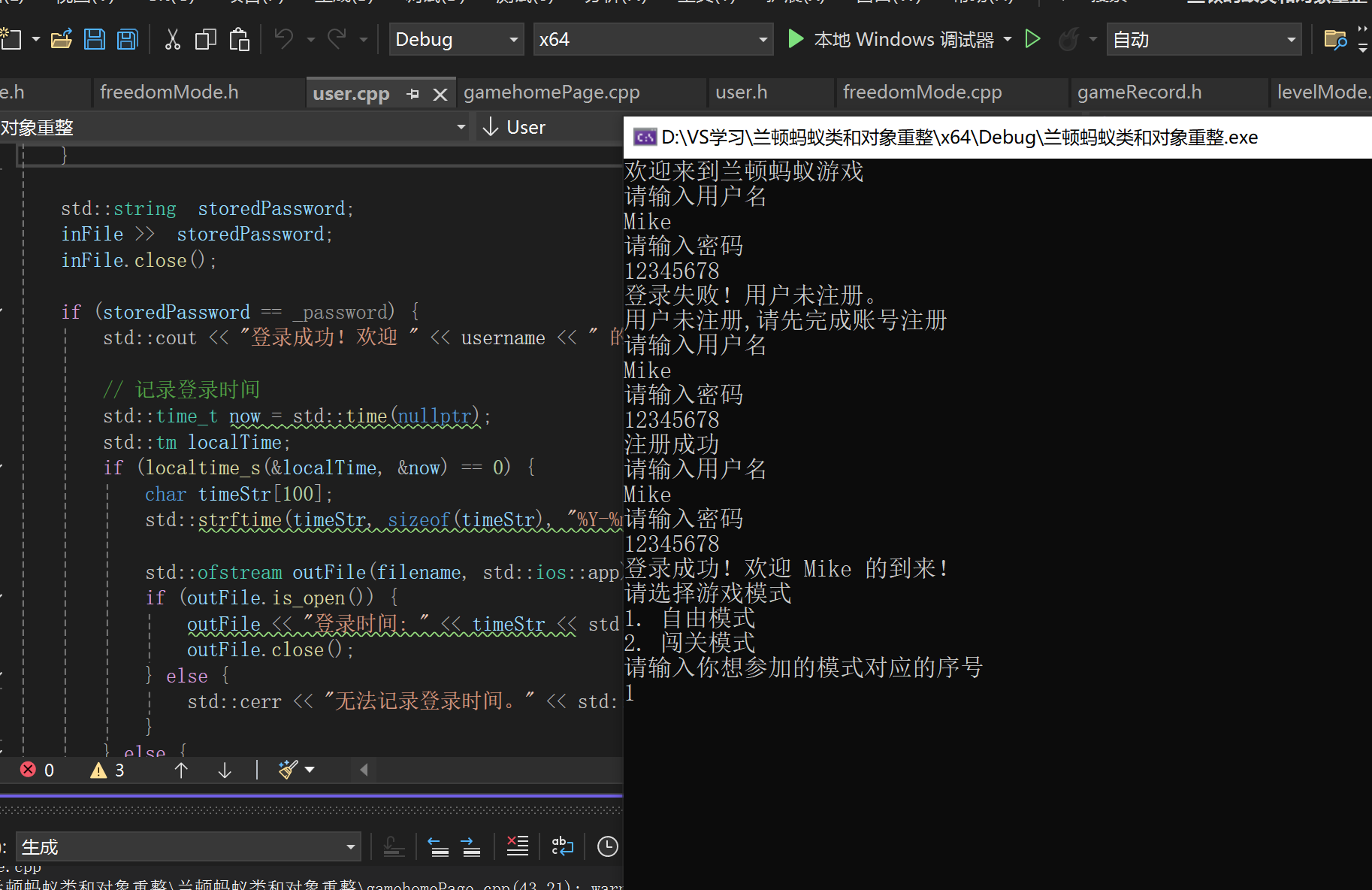
（4）下周的安排：

先解决几个问题：

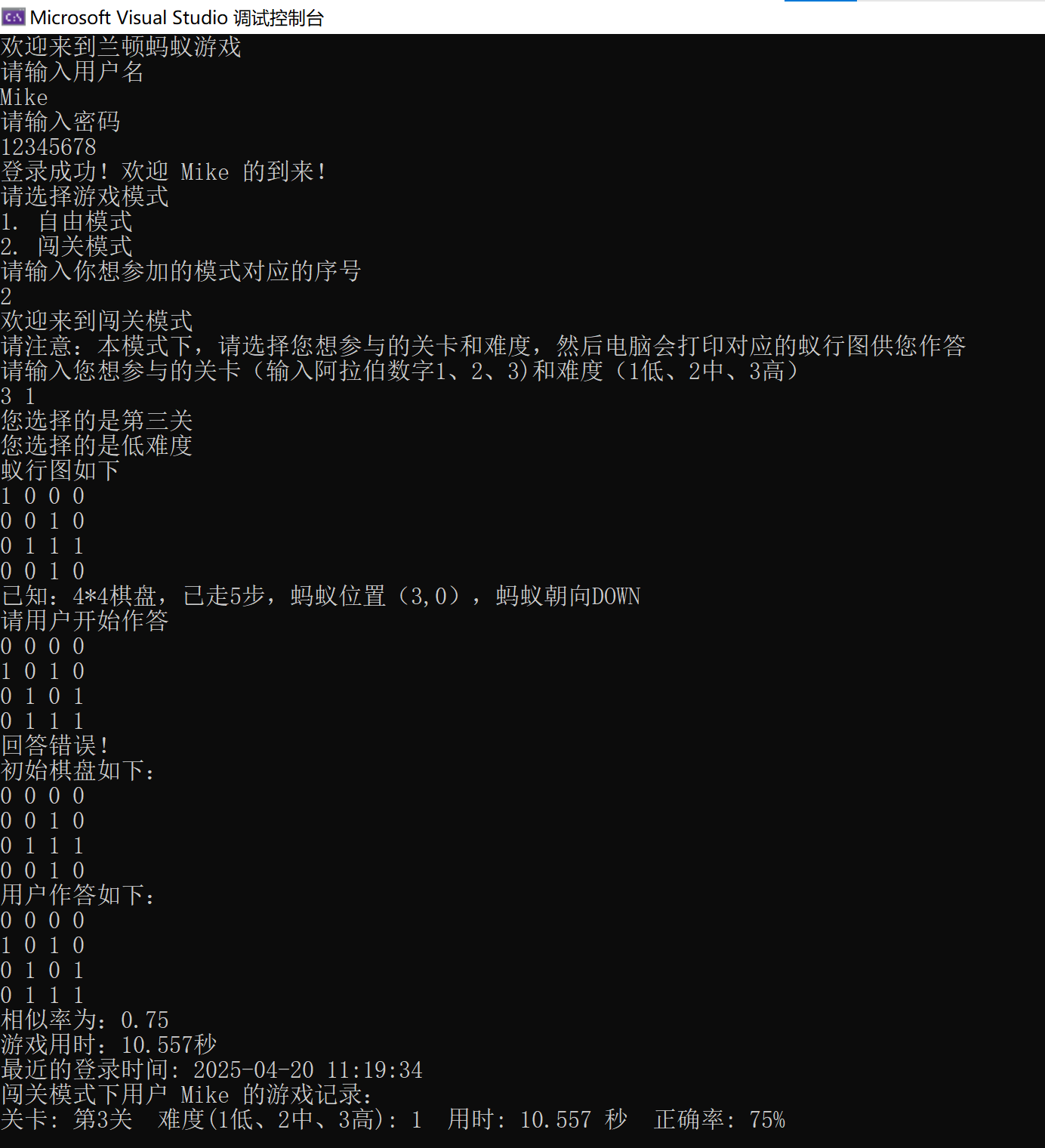
* 用户忘记密码怎么办？
* 如何删除用户和游戏记录？
* 如何在用户输入时和输出文件内容时隐藏密码？
* 页面模式的跳转
* 添加循环解决一人只能玩一次的问题
* 文件输入内容得排序逻辑和默认读取顺序

四、实验运行结果（仍旧利用已注册的两个用户Jack和Mike）

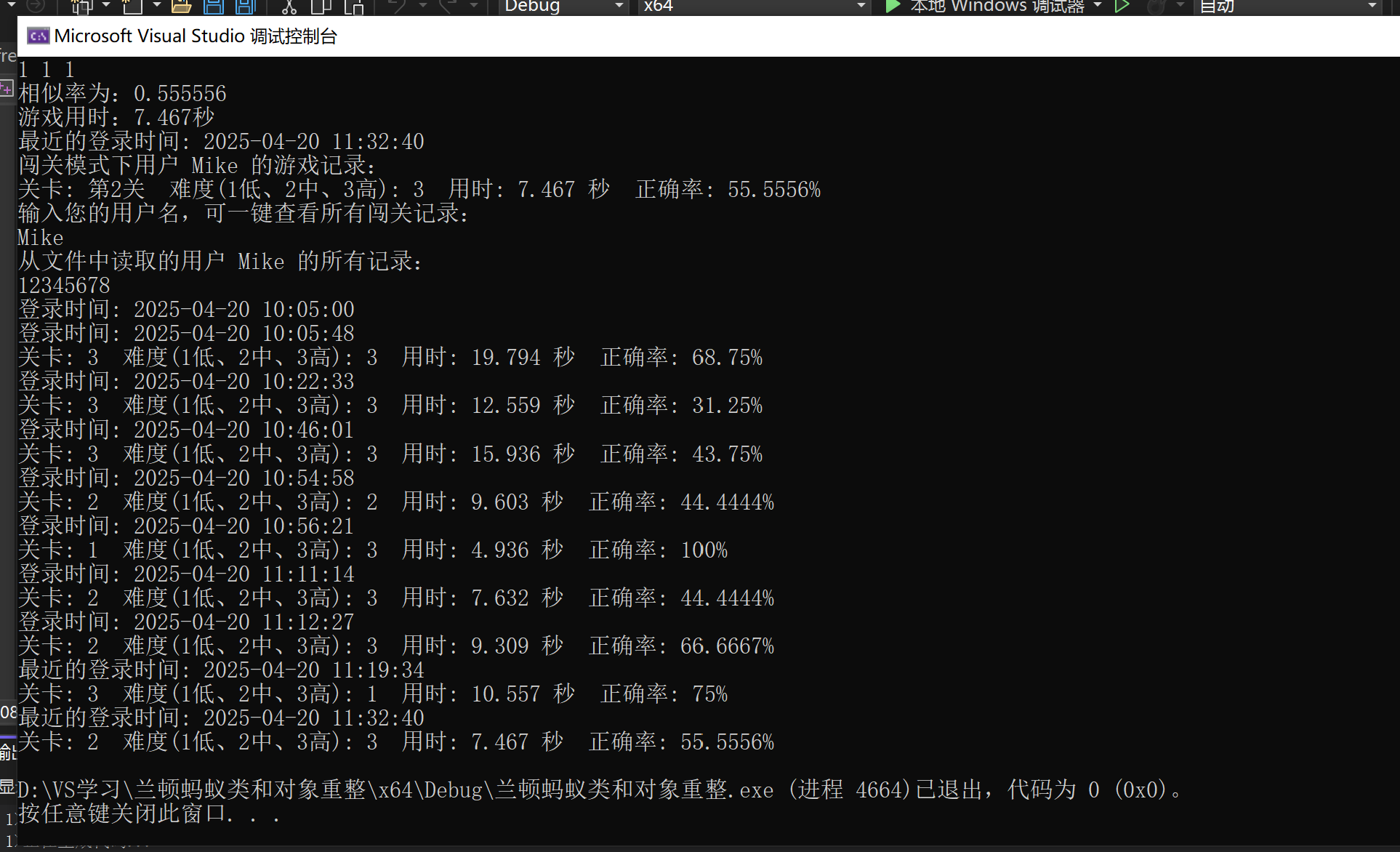
用户注册与登录



Mike参与了闯关模式并且观察自己的游戏记录



Mike参加了不同关卡后的所有游戏记录（这里显示出个人密码是个bug，尚未解决）



Jack参加了自由模式，初步了解了游戏规则

