面向对象程序设计

课程设计报告

题 目 ： 乐器演奏模拟系统

班 级 ： 19G231

本科生姓名： 张淋迩

本科生学号： 20231003496

本科生专业： 计算机科学与技术（大数据）

所 在 院 系 计算机学院

日 期 ： 2024 年 6 月 11 日

课程论文评语

对课程论文的评语：

课程论文成绩：	评阅人签名：

目 录

一、课程设计题目与要求

1.题目：乐器演奏模拟系统
2.系统功能要求：
（1）程序分为自动和手动两个板块。
（2）手动板块提供多种乐器音色选择，可以演奏多种乐器。
（3）手动板块演奏时显示乐器画面，根据使用者在键盘上按下的按键发出对应的琴声，要求可以同时演奏多个音并且连续演奏没有延迟。
（4）手动板块具有录制记谱的功能，能将使用者所按下播放的随机内容用 txt 格式及记录并保存。
（5）自动板块提供预先写好的 txt 琴谱，在使用者选择后直接播放对应琴谱，要求使用者可以选择演奏的速度。
（6）自动板块可以输入之前手动板块记录的文件名后选择任意速度播放。
（7）程序可以在运行过程中随意切换页面，且可以随时结束程序。

二、需求分析

1.问题描述：
用户需求一个能够模拟真实乐器演奏的系统，以便于学习、演示和娱乐。
2.数据源：
系统需要包括各种乐器的音色库。
3.系统环境：
应支持主流操作系统，如 Windows。
4.运行要求：
（1）系统应具备用户友好的界面，能够实时响应用户的输入，如键盘敲击，并能以高保真音质输出模拟演奏结果。
（2）系统需要有良好的扩展性，支持未来添加新的乐器内容和新的功能。

三、概要设计

根据以上的需求分析，按功能将乐器演奏模拟系统分为七个模块：初始化模块、用户界面模块、播放控制模块、录制模块、音频处理模块、文件操作模块、退出处理模块。其中，初始化模块实现了设置控制台的宽度和高度，确保用户界面能够在一个合适的空间内展示，提供良好的用户体验，定义了全局变量 showpage 用于控制程序流程和界面展示的状态；用户界面模块实现了负责显示不同的页面，接收用户输入，调用相应的功能模块，提供了与其他模块的交互；播放控制模块接收用户通过用户界面模块做出的选择，播放控制模块将启动相应的播放流程，控制音频的实际播放，同步多个音轨的播放，响应用户操作，管理播放速度；录制模板提供实时监听和捕捉用户通过键盘进行的演奏输入并将其保

存为文件；音频处理模板将所输入的案件内容解析成对应的文件名，为播放控制模板与文件操作模块提供交互条件；文件操作模块负责管理文件的创建、读取、写入、修改和删除等，它为系统中的其他模块提供文件处理服务，确保数据的持久化和有效管理；退出处理模块负责在用户决定退出程序时执行一系列清理和结束操作。

其功能结构图如下图所示：

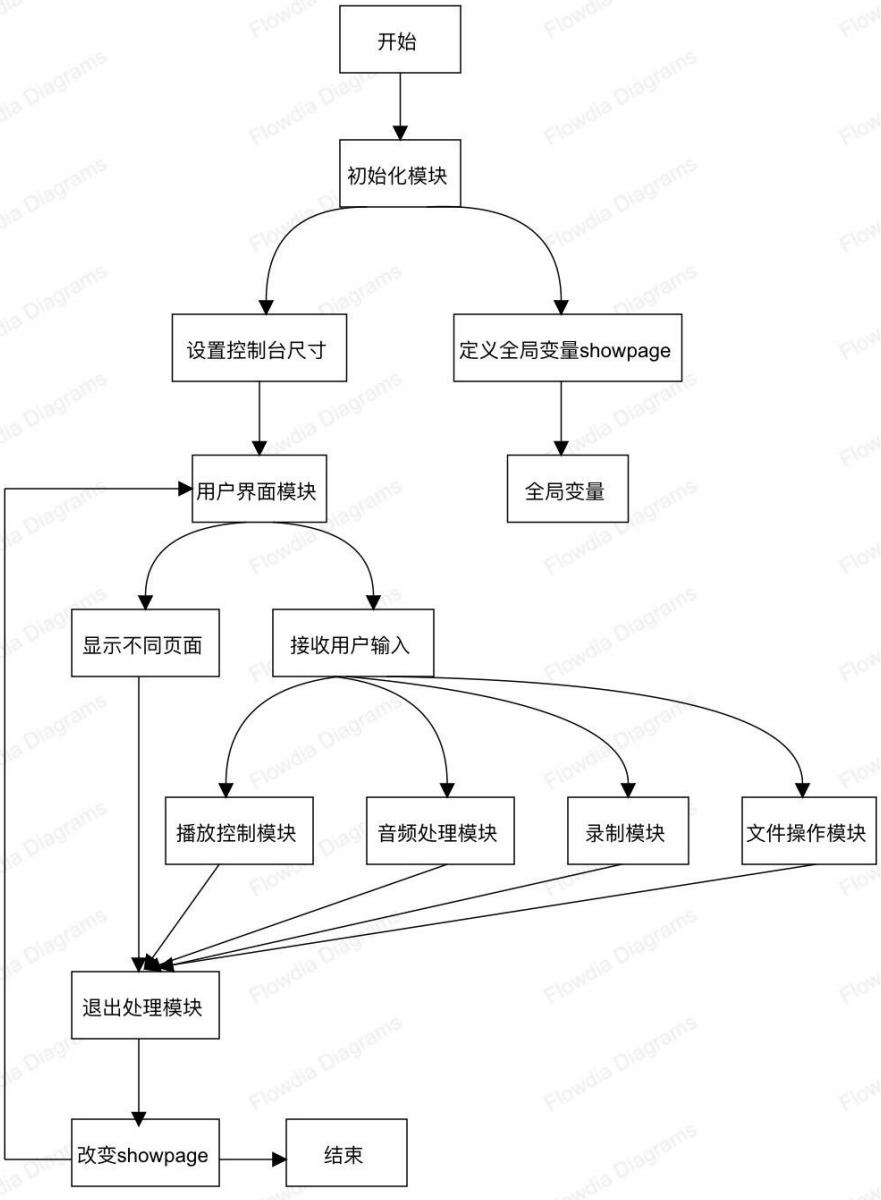

图 1　　乐器演奏模拟系统的功能结构图

从上图可看出，这五个模块以用户界面为中心，通过有机组合形成一个完整的乐器演奏模拟系统，解决了模拟乐器弹奏问题。其数据流程如下图所示：

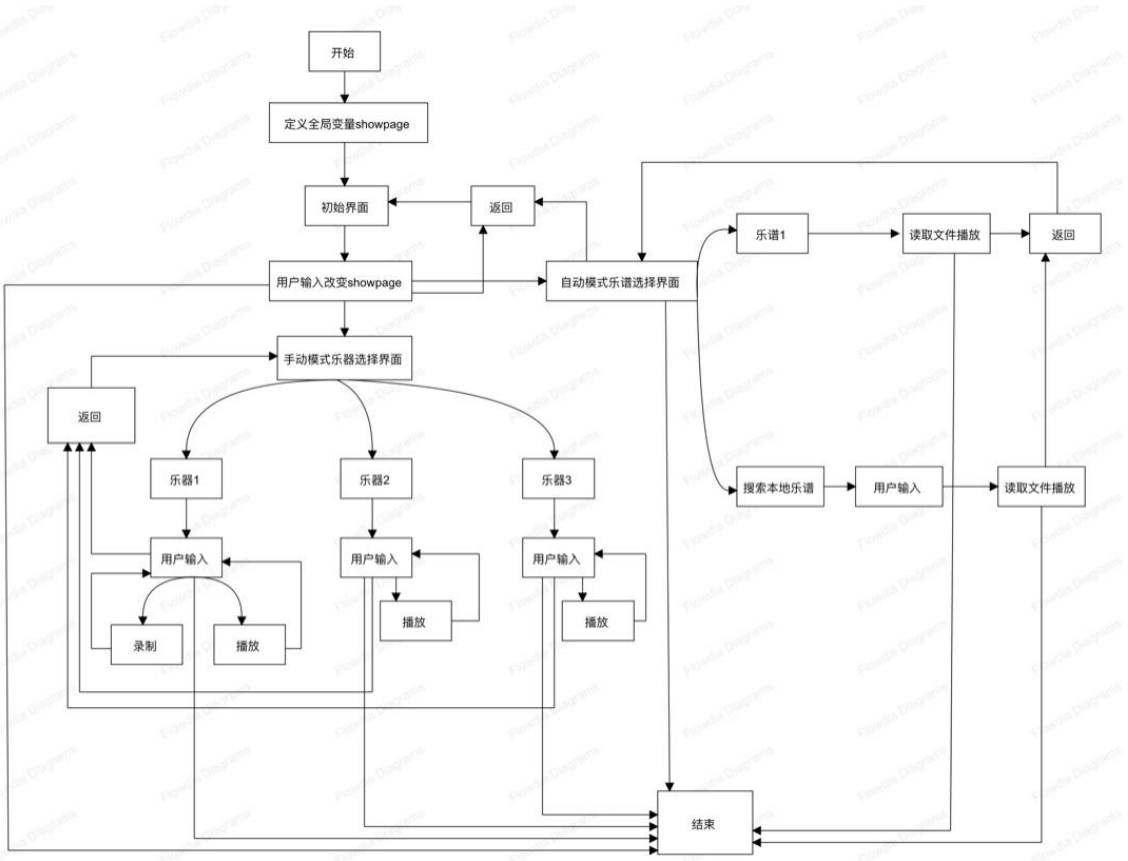

图 2　　乐器演奏模拟系统的数据流程图

乐器演奏模拟系统首先定义全局变量 showpage 并进入初始界面，空格键继续进入模式选择界面，后根据使用者所输入的内容跳转相应的界面或返回初始界面。其中手动模式进入乐器选择界面，根据使用者所输入的内容进入不同乐器界面，再循环根据使用者所输入的内容播放对应音符的音频或返回上一个界面；其中自动模式进入乐谱选择界面，个剧使用者所输入的内容打开不同乐谱的 txt 文件，再根据使用者所输入的乐器与速度播放乐谱对应乐曲。其中任何时刻都可以直接进入结束页面并结束程序。对于使用者随时想推出播放的情况，乐器演奏模拟系统也能进行处理，其特点有：1）响应灵敏，2）随时触发，3）界面交替便捷。

四、详细设计

4.1 按键对应音频播放设计

要将使用者按键的名称对应成电脑中存储的 MP3 文件名，流程图如下图所示：

图 3　　乐器演奏模拟系统的按键对应音频播放设计流程图

像鼓和吉他和弦所需要对应的按键与文件名数量较少，直接用含字符的数组与含字符串的数组一一对应。具体代码如下图所示：

```
char key[] = {'0','1','2','3','4','5','6','7','8' };
char letter[][5] = { "0","1","2","3","4","5","6","7","8" };
for (int i = 0; i < 9; i++) {
    if (keyboard_key == key[i]) {
        strcpy(sound_name, letter[i]); break;
    }
    else continue;
}
```

```
char key[] = {'0','1','2','3','4','5','6','7','8','9','q','w','e','r' };
char letter[][5] = { "0","A","Am","B","Bj","Bm","C","D","Dm","Ej","Em","F","Fm","G" };
for (int i = 0; i < 14; i++) {
    if (keyboard_key == key[i]) {
        strcpy(sound_name, letter[i]); break;
    }
    else continue;
}
```

图 4　　乐器演奏模拟系统中鼓和吉他和弦的按键对应音频文件名代码

但如钢琴所需要对应的按键与文件名数量较多，则将钢琴对应的音频文件以其音名加其八度数的格式保存。将全部按键名以字符存入 key 数组，全部音名和全部八度数以字符串存入 key_letter 与 key_num 数组（这里区分字符和字符串是因为输入函数_getc 只能输入单个字符给 keyboard_key,但赋值函数 strcpy 只能传入字符串）；按 12 个一组按键名赋予音名和八度数的组合作为所对应的文件名。具体代码如下图所示：

```
void map_func(char keyboard_key, char* sound_name) {
    char key[] = {'1','!','2','@','3','4','$','5','%','6','^','7',
                  '8','*','9','(','0','q','Q','w','W','e','E','r',
                  't','T','y','Y','u','i','I','o','O','p','P','a',
                  's','S','d','D','f','g','G','h','H','j','J','k',
                  'l','L','z','Z','x','c','C','v','V','b','B','n',
                  'm' }; //所有音名对应的按键(键盘按键有限，仅表示61键)
    char key_letter[][3] = { "C","Cs","D","Ds","E","F","Fs","G","Gs","A","As","B" }; //音名字母部分
    char key_num[][2] = { "1","2","3","4","5","6" };  //音名数字部分

    for (int i = 0; i < sizeof(key); i++) {
        if (keyboard_key == key[i]) {
            int LetterName = (i % 12); //key中的每一个按键对应的字母的名字，字母部分12个一轮回
            int NumName = (i / 12);  //key中的每一个按键对应的数字的名字，数字部分每十二个音+1
            char ss_letter[5], ss_num[2]; //建立两个空数组，存放字母名和数字名
            strcpy_s(ss_letter, key_letter[LetterName]);
            strcpy_s(ss_num, key_num[NumName]); //用strcpy拷贝存放
            strcat_s(ss_letter, ss_num);//用stract合并到ss_letter
            strcpy(sound_name, ss_letter);//最终存放在sound_name
        }
        else {
            continue;
        }
    }
}//根据输入按键得出piano对应音频文件名的函数
```

图 5　　乐器演奏模拟系统中钢琴的按键对应音频文件名代码

MP3 格式音频文件的打开与播放通过 c++自身的 mciSendString 函数进行，其中 temp_command 作为指令字符串，用 sprintf_s 函数将文件名填入 temp_command，再放进 mciSendString 执行打开、播放与关闭功能。具体代码如下图所示：

```
sprintf_s(temp_command, "open piano\\%s.mp3 alias %s", sound_name, sound_name);
mciSendString(temp_command, 0, 0, 0); //打开音名.mp3
sprintf_s(temp_command, "play %s  ", sound_name);
mciSendString(temp_command, 0, 0, 0);    //播放
Sleep(1000);//
sprintf_s(temp_command, "close %s", sound_name);   //关闭
mciSendString(temp_command, 0, 0, 0);
```

图 6　　乐器演奏模拟系统的音频播放代码

4.2 播放控制板块设计

播放控制板块的具体流程图如下图所示：

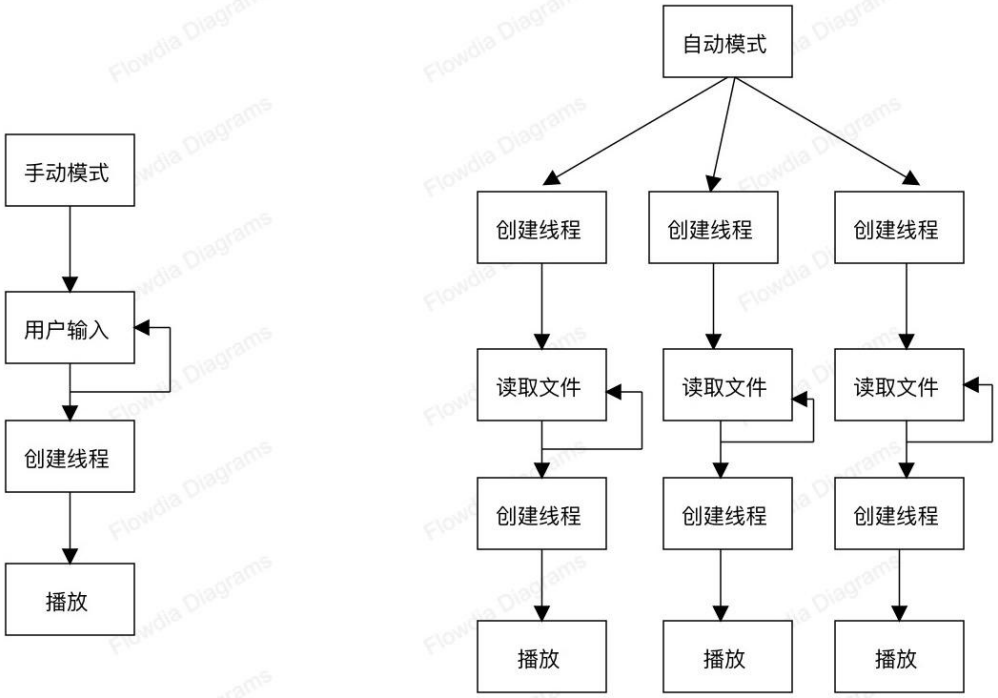

图 7　　乐器演奏模拟系统的播放控制板块流程图

为模拟乐器演奏的及时性，需要在按下按键及时播放对应琴音，此处使用 _getch()函数，不需要按回车键就可以执行。具体代码如下图所示：

```
char keyboard_key = _getch();
```

图 8　　乐器演奏模拟系统的播放即时控制代码

每个音频都具有一定时长，但为模拟乐器演奏的连续性与同时性，不能等一个琴音播放结束后再响应下一个输入，此处使用多线程 thread，为每次按键输入创建一个线程，互不干扰，可以同时播放。具体代码如下图所示：

```
thread newThread1(play_sound_piano, keyboard_key);//创建线程
newThread1.detach();
```

图 9　　乐器演奏模拟系统的单音演奏播放的多线程控制代码

在自动模式读取乐谱播放时，为模拟多乐器合奏，需要同时读取多个乐谱并同时播放，此处用多线程 thread，为每个乐谱创建一个线程，互不干扰，可以同时读取，其中又为每个乐谱中的每个字符创建线程，使其按乐谱正常播放。具体代码如下图所示：

```
void Read_puzi(int k, char i, int bmp) {//假设读出来的是i,乐器是k
    if (i == '|') {}//小节分隔
    else {
        if (k == 1) {
            thread newThread1(play_sound_piano, i);//创建线程
            newThread1.detach();
            Sleep(bmp);//相当于四分音符
        }//piano
        else if (k == 2) {
            thread newThread1(play_sound_drum, i);//创建线程
            newThread1.detach();
            Sleep(bmp);
        }//drum
        else if (k == 3) {
            thread newThread1(play_sound_guitar, i);//创建线程
            newThread1.detach();
            Sleep(bmp);
        }
    }
}//读取谱子文件
```

```
    std::thread t1(piano1, sudu);
    std::thread t2(piano2, sudu);
    std::thread t3(drum1, sudu);
    std::thread t4(drum2, sudu);
    std::thread t5(drum3, sudu);
    //std::thread t6(guitar, sudu);
    t1.join();
    t2.join();
    t3.join();
    t4.join();
    t5.join();
    //t6.join();
```

图 10　　乐器演奏模拟系统的读取乐谱播放的多线程控制代码

4.3 乐谱文件模块设计与实现

在手动模式中，需要将弹奏的内容录入记谱，此处将按键名称用文件操作直接写入 txt，其中每个按键代表一个八分音符，通过空音频调整节拍。自动模式中，需要将提前写好的乐谱用文件操作读取并播放，其中用 Sleep 函数控制每个音之后的休眠时间来调整乐谱播放速度。具体流程图如下图所示：

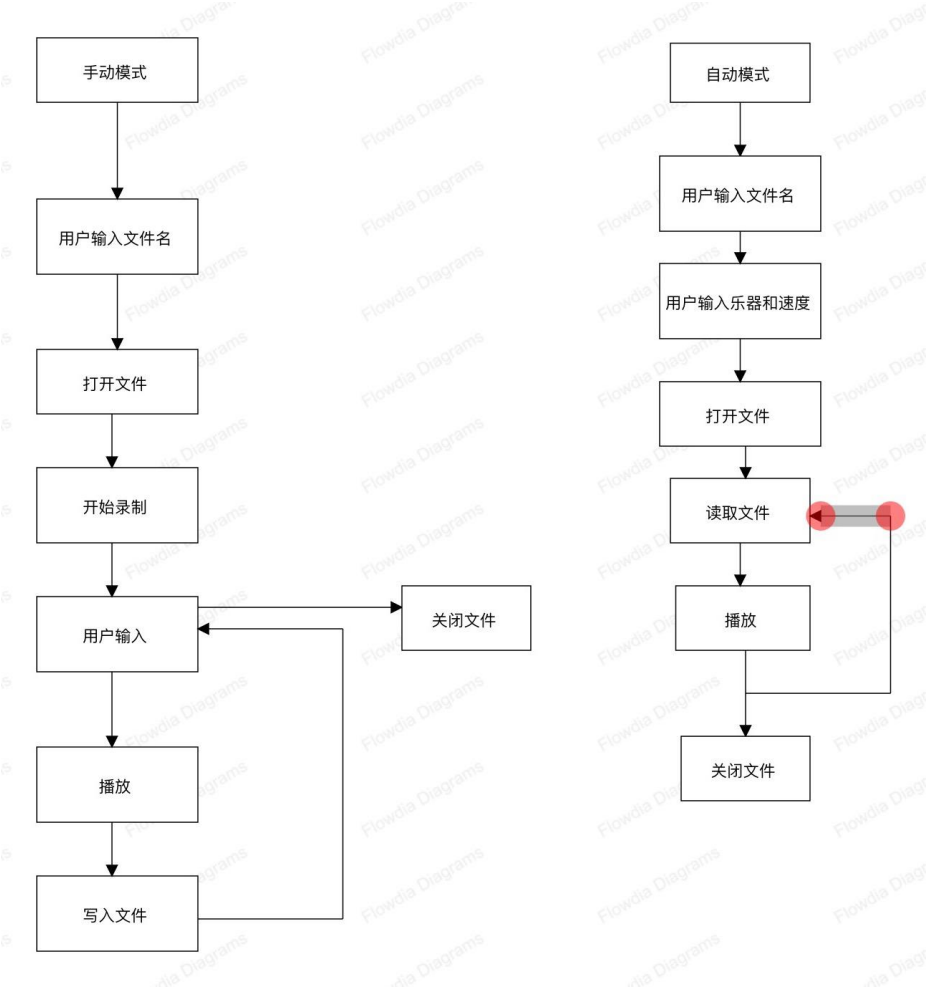

图 11　乐器演奏模拟系统的乐谱文件板块设计流程图

具体代码如下图所示：

```
FILE* writefile = fopen(file, "w");//只写
if (file == NULL) {
    prin("Error!", 0, 0);
}
else {
    while (1) {
        char keyboard_key = _getch();
        thread newThread1(play_sound_piano, keyboard_key);//创建线程
        newThread1.detach();
        fwrite(&keyboard_key, sizeof(keyboard_key), 1, writefile);
        if (kd(VK_SPACE)) {
            break; //再次按空格键结束
        }
    }
}
fclose(writefile);//关闭文件
```

图 12　乐器演奏模拟系统的乐谱文件板块写入文件代码

```
void Open_puzi(int k, const char* puzi_command, int sudu) {//传入文件地址
    FILE* puzi = fopen(puzi_command, "r");

    if (puzi != NULL) {
        char c = fgetc(puzi);
        while (c != EOF) {
            if (kd(VK_BACK)) {
                break;
            }
            else Read_puzi(k, c, 30000 / sudu);
            c = fgetc(puzi);
        }// 打开文件成功，进行文件读写操作
        fclose(puzi); // 关闭文件
    }
    else {
        printf("Error!");
    }
}//读取键盘播放音符
```

图 13　乐器演奏模拟系统的乐谱文件板块读取文件代码

乐谱文件保存目录如下图所示：

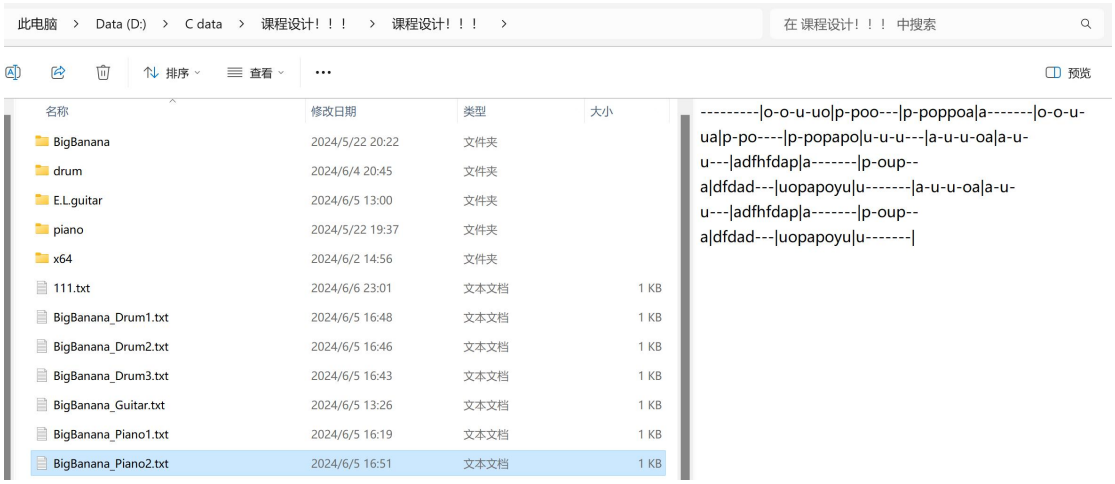

图 14　乐器演奏模拟系统的乐谱文件板块的乐谱文件保存目录

4.4 用户界面显示模块设计与实现

首先确定控制面板大小，以便确定其他画面的坐标与大小，后根据控制面板的长和宽，在面板中间打印用户界面。但从左上角开始用空格输入的话太过于麻烦和不准确，此处用c++自带的 COORD 类来将光标直接移动到指定坐标。具体流程图如下图所示：

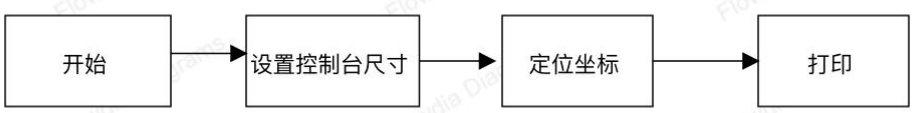

图 15　乐器演奏模拟系统的用户界面显示流程图

具体代码如下图所示：

```
void position(int x, int y){
    COORD pos = { (short)x,(short)y };
    HANDLE hOut = GetStdHandle(STD_OUTPUT_HANDLE);
    SetConsoleCursorPosition(hOut, pos);
    return;
}//定位到指定坐标
```

图 16　乐器演奏模拟系统的用户界面显示定位坐标代码

其中，程序边框与乐器界面的打印都是用这种定位坐标的方式直接进行，具体代码如图下所示：

```
void print_kuang() {
    short start_x = 20, start_y = 5;
    short page_x = 80, page_y = 20;
    position(start_x + page_x - 8, start_y + page_y - 3);
    cout << "<ESC>";
    position(start_x + 3, start_y + 2);
    cout << "<BACK>";
    position(start_x, start_y);
    for (short y = 0; y < page_y; y++) {
        if (y == 0 || y == page_y - 1) {
            position(start_x, start_y + y);
            for (short x = 0; x <= page_x; x++) {
                cout << "=";      //若为第一行或最后一行，则打印page_x个=
            }
        }
        else {
            position(start_x, start_y + y);
            cout << '|';
            position(start_x + page_x, start_y + y);
            cout << '|';       //每行开头和末尾打印|
        }
    }
}//打印边框
```

```
void print_pkeys() {
    int ystart = 8;
    position(20 + 1, ystart++);//第八行，start=9
    cout << " ___________________________________________________________________________";
    for (short i = 0; i < 4; i++) {//打印4行
        position(20 + 1, ystart++);
        cout << "|  | | | |  | | | | | |  | | | | |  | | | | | | |  | | | | |  | | |";
    }//此时ystart=12
    position(20 + 1, ystart++);
    cout << "|  |!| |@|  |  |$| |%| |^|  |  |*| |(|  |  |Q| |W| |E|  |  |T| |Y|  |  |I|  |";
    for (short i = 0; i < 1; i++) {
        position(20 + 1, ystart++);
        cout << "|  |S| |D|  |  |G| |H| |J|  |  |L| |Z|  |  |C| |V| |B|  |  |M| | |  | | |  |";
    }
    position(20 + 1, ystart++);
    cout << "|  |_| |_|  |  |_| |_| |_|  |  |_| |_|  |  |_| |_| |_|  |  |_| |_|  |  |_|  |";
    for (short i = 0; i < 1; i++) {
        position(20 + 1, ystart++);
        cout << "|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |";
    }
    position(20 + 1, ystart++);
    cout << "| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | q | w | e | r | t | y | u | i | o |";
    position(20 + 1, ystart++);
    cout << "| C1| D1| E1| F1| G1| A1| B1| C2| D2| E2| F2| G2| A2| B2| C3| D3| E3| F3| G3|";
    position(20 + 1, ystart++);
    cout << "| s | d | f | g | h | j | k | l | z | x | c | v | b | n | m |   |   |   |  |";
    position(20 + 1, ystart++);
    cout << "| C4| D4| E4| F4| G4| A4| B4| C5| D5| E5| F5| G5| A5| B5| C6|   |   |   |  |";
    position(20 + 1, ystart++);
    cout << "|___|___|___|___|___|___|___|___|___|___|___|___|___|___|___|___|___|___|___|";
}
```

```
void print_drum() {
    int ystart = 8;
    int xstart = 37;
    position(xstart, ystart++);//第八行，第39列
    cout << "_________  _________  _________  _________";
    position(xstart, ystart++);//第九行
    cout << "|        | |        | |        | |        |";
    position(xstart, ystart++);
    cout << "|  踏钹  | |  铙钹  | | 小军鼓 | |  脚鼓  |";
    position(xstart, ystart++);
    cout << "|   1    | |   2    | |   3    | |   4    |";
    position(xstart, ystart++);
    cout << "|        | |        | |        | |        |";
    position(xstart, ystart++);
    cout << "|________| |________| |________| |________|";
    position(xstart, ystart++);//中间空一行
    cout << endl;
    position(xstart, ystart++);
    cout << "_________  _________  _________  _________";
    position(xstart, ystart++);
    cout << "|        | |        | |        | |        |";
    position(xstart, ystart++);
    cout << "| 嗵嗵鼓 | |  铃鼓  | |  沙锤  | | 康家鼓 |";
    position(xstart, ystart++);
    cout << "|   5    | |   6    | |   7    | |   8    |";
    position(xstart, ystart++);
    cout << "|        | |        | |        | |        |";
    position(xstart, ystart++);
    cout << "|________| |________| |________| |________|";
}
```

```
void print_guitar() {
    int ystart = 10;
    int xstart = 28;
    position(xstart, ystart++);//第10行，第25列
    cout << " ___  ___  ___  ___  ___  ___  ___  ___  ___  ___  ___  ___  ___ ";
    for (int i = 1; i < 4; i++) {
        position(xstart, ystart++);
        cout << "|   ||   ||   ||   ||   ||   ||   ||   ||   ||   ||   ||   |";
    }
    position(xstart, ystart++);//
    cout << "| 1 || 2 || 3 || 4 || 5 || 6 || 7 || 8 || 9 || q || w || e || r |";
    position(xstart, ystart++);//
    cout << "| A ||Am || B ||Bj ||Bm || C || D ||Dm ||Ej ||Em || F ||Fm || G |";
    for (int i = 1; i < 3; i++) {
        position(xstart, ystart++);
        cout << "|   ||   ||   ||   ||   ||   ||   ||   ||   ||   ||   ||   |";
    }
    position(xstart, ystart++);//
    cout << "|___||___||___||___||___||___||___||___||___||___||___||___| ";
}
```

图 17　乐器演奏模拟系统的用户界面显示中边框与乐器打印代码

其中，提示标语的打印需要居中显示，但每次数具体坐标值来定位坐标也过于麻烦和不准确，此处引用 c++自带的句柄来获取控制台的大小，编写函数使语句能够居中输出。具体代码如下图所示：

```
void prin(string s, int X, int Y)
{
    HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
    CONSOLE_SCREEN_BUFFER_INFO bInfo;
    GetConsoleScreenBufferInfo(hConsole, &bInfo);
    int y = bInfo.dwMaximumWindowSize.Y, x = bInfo.dwMaximumWindowSize.X;
    position((x - s.size()) / 2 + X, y / 2 + Y);
    cout << s;
}//居中输出
```

图 18　乐器演奏模拟系统的用户界面显示中居中显示语句代码

由于在打印界面时会有光标频闪，此处引用 c++自带的句柄写成的不显示光标的函数。具体代码如下图所示：

```
void HideCursor() {
    HANDLE hStdOut = GetStdHandle(STD_OUTPUT_HANDLE);
    CONSOLE_CURSOR_INFO cursorInfo;
    GetConsoleCursorInfo(hStdOut, &cursorInfo);
    cursorInfo.bVisible = FALSE; // 设置光标不可见
    SetConsoleCursorInfo(hStdOut, &cursorInfo);
}//不显示光标
```

图 19　乐器演奏模拟系统的用户界面显示中光标不显示代码

但在需要用户进行输入速度与文件名时，又需要有光标进行输入提示，此处引用 c++自带的句柄写成的显示光标的函数。具体代码如下图所示：

```
void ShowCursor() {
    HANDLE hOutput = GetStdHandle(STD_OUTPUT_HANDLE);
    CONSOLE_CURSOR_INFO cInfo{};
    GetConsoleCursorInfo(hOutput, &cInfo); //获取现有光标信息
    cInfo.bVisible = TRUE;
    SetConsoleCursorInfo(hOutput, &cInfo);  //重新设置光标信息
}
```

图 20　**乐器演奏模拟系统的用户界面显示中光标显示代码**

4.4 返回与退出模块设计与实现

为实现用户界面的切换，此处将 main 主函数作为中转站，通过使用者的不同输入改变函数的返回值，再通过函数的返回值改变全局变量，通过全局变量在主函数中进入到不同用户功能函数。具体流程图如下图所示：

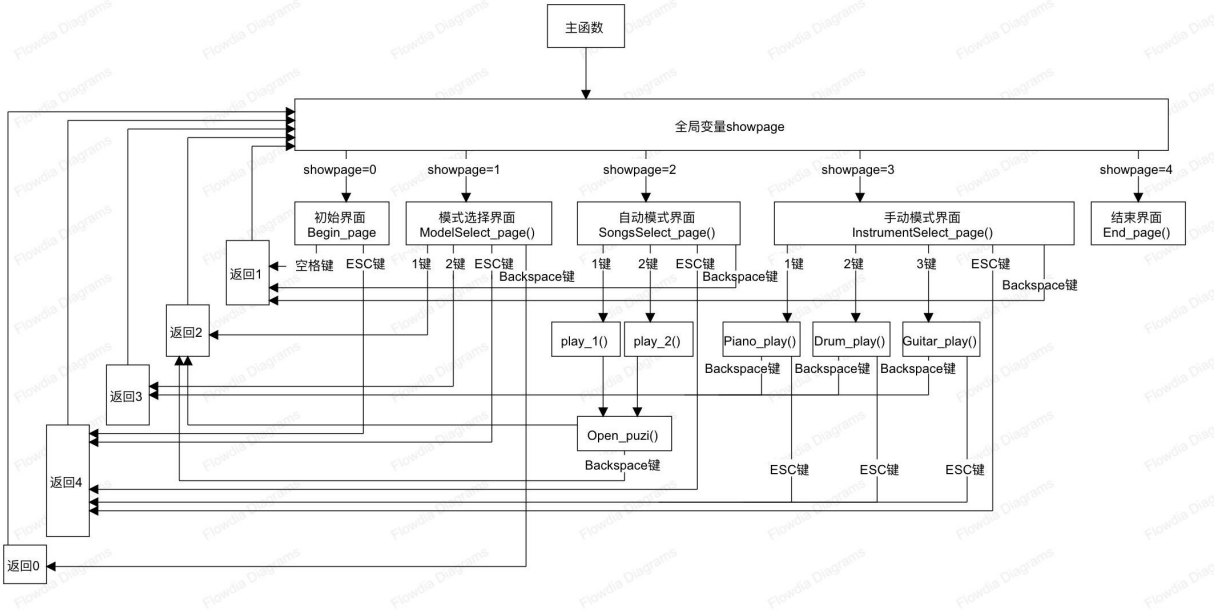

图 21　**乐器演奏模拟系统的用户界面交互流程图**

监测按键输入，随时可以按 Backspace 键返回上一个界面与按 ESC 键退出程序，此处通过选择结构和 c++自带的 kd 及其类型实现。具体代码如下图所示：

```
#define kd(VK_NONAME) ((GetAsyncKeyState(VK_NONAME) & 0x8000) ? 1:0)
```

```
if (kd(VK_ESCAPE)) {
    cls_kuang();
    return 4;//退出键直接跳转结束页面
}
if (kd(VK_BACK)) {
    cls_kuang();
    return 3;//返回上一个界面
}
else if (kd(VK_SPACE)) {
    Record_drum();
    continue;
}
```

图 22　乐器演奏模拟系统的监测按键返回与退出具体代码

其中，为实现能够在手动模式演奏时随时返回，与在自动模式读取曲谱播放时随时返回，此处将按键监测放入循环。具体代码如下图所示：

```
while (1) {
    char keyboard_key = _getch();
    thread newThread1(play_sound_piano, keyboard_key);//创建线程
    newThread1.detach();
    if (kd(VK_ESCAPE)) {
        cls_kuang();
        return 4;//退出键直接跳转结束页面
    }
    else if (kd(VK_BACK)) {
        cls_kuang();
        return 3;//返回上一个界面
    }
    else if (kd(VK_SPACE)) {
        Record_piano();
        continue;
    }//空格键开始录制
}//通过循环不断演奏
```

图 23　乐器演奏模拟系统演奏过程中的监测按键返回与退出具体代码

在执行返回退出界面跳转时，要先对控制台现有内容进行清空。用 system(“cls”)清除整个屏幕内容再继续打印边框和内容会造成边框的闪动，于是这里编写清屏函数通过将框内特定行数内容全部写成空格以完成只清空边框内内容。具体代码如下：

```
void cls_kuang() {//第八行到第22行
    position(20 + 1, 8);
    for (short c = 0; c < 80 - 1; c++) {
        printf(" ");
    }
    for (short c = 0; c < 15; c++) {
        position(20 + 1, 8 + c);
        for (short c = 0; c < 80 - 1; c++) {
            printf(" ");
        }
    }
    position(20 + 1, 8);
}//只清除边框内的内容
```

图 24　乐器演奏模拟系统返回退出清屏操作代码

五、测试

5.1 编译、链接和运行

基于上面的分析和设计，使用 C++语言进行了程序编码，并用 Visual C++ 2022 编译器进行程序编译、链接和运行，生成了乐器演奏模拟系统的可执行程序。运行效果如下节所述。

5.2 运行效果

运行乐器演奏模拟系统，根据不同的输入项，可调用乐器演奏模拟系统的不同功能模块，实现了手动演奏与自动播放的功能，其各模块的运行效果如下：

（1）初始界面

乐器演奏模拟系统的初始界面如图所示：

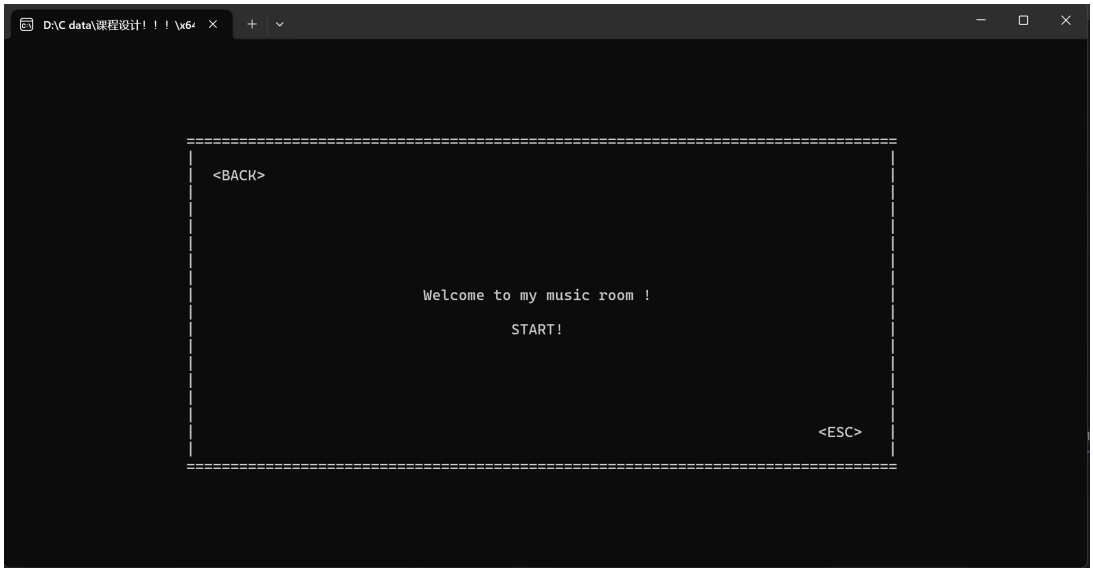

图 25　乐器演奏模拟系统初始界面

在该初始界面里，可输入“空格键”继续；也可输入“ESC 键”直接退出程序。

（2）模式选择模块

在初始界面中输入空格键，进入模式选择模块，如下图所示：

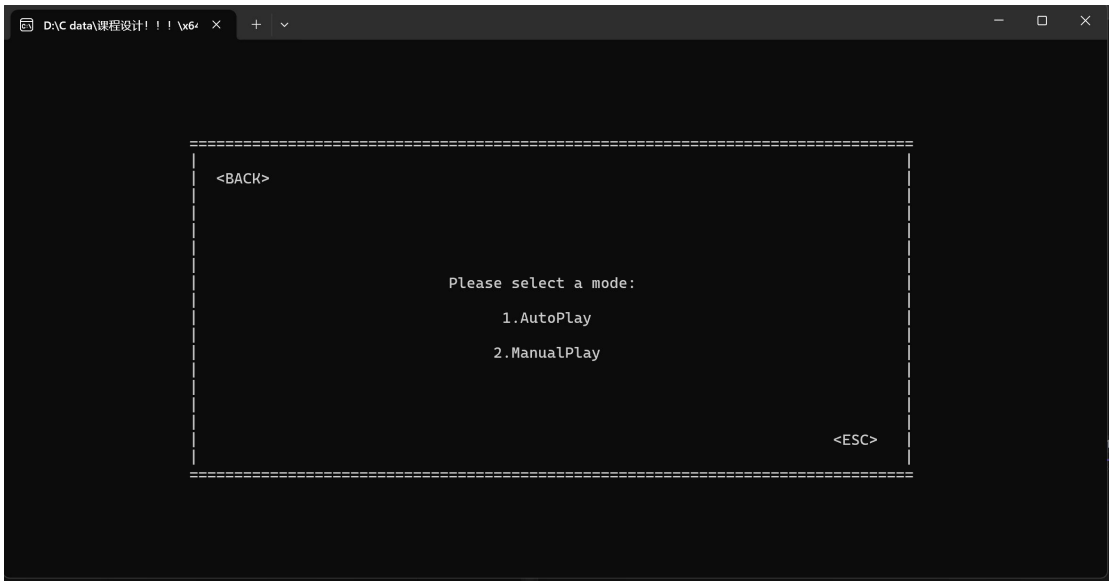

图 26　　乐器演奏模拟系统模式选择模块

在此模块中，可输入“1”进入自动播放模式；可输入“2”进入手动演奏模式；可输入“Backspace 键”返回初始界面；可输入“ESC 键”直接退出程序。

（3）手动演奏乐器选择模块

在模式选择界面中输入“2”，进入手动演奏的乐器选择模块，如下图所示：

图 27　　乐器演奏模拟系统手动演奏的乐器选择模块

在此模块中，可输入“1”进入钢琴演奏；可输入“2”进入鼓声演奏；可输入“3”进入电吉他演奏；可输入“Backspace 键”返回模式选择模块；可输入“ESC 键”直接退出程序。

（4）乐器演奏模块

在手动演奏乐器选择页面中输入“1”进入钢琴演奏模块，如下图所示：

图 28　　乐器演奏模拟系统手动演奏的钢琴演奏模块

在此模块中，可输入界面中提示键盘按键播放对应钢琴声音，可同时按下多个键盘按键演奏对应和弦；可输入“空格键”进入乐谱录制模块；可输入“Backspace 键”返回乐器选择模块；可输入“ESC 键”直接退出程序。

在手动演奏乐器选择页面中输入“2”进入鼓声演奏模块，如下图所示：

图 29　　乐器演奏模拟系统手动演奏的鼓声演奏模块

在此模块中，可输入界面中提示键盘按键播放对应鼓声，可同时按下多个键盘按键演奏；可输入“空格键”进入乐谱录制模块；可输入“Backspace 键”返回乐器选择模块；可输入“ESC 键”直接退出程序。

在手动演奏乐器选择页面中输入“3”进入电吉他演奏模块，如下图所示：

图 30　　乐器演奏模拟系统手动演奏的电吉他演奏模块

在此模块中，可输入界面中提示键盘按键播放对应电吉他和弦声，可同时按下多个键盘按键演奏；可输入“空格键”进入乐谱录制模块；可输入“Backspace 键”返回乐器选择模块；可输入“ESC 键”直接退出程序。

（5）乐谱录制模块

在乐器演奏页面中输入“空格键”进入乐谱录制模块，如下图所示：

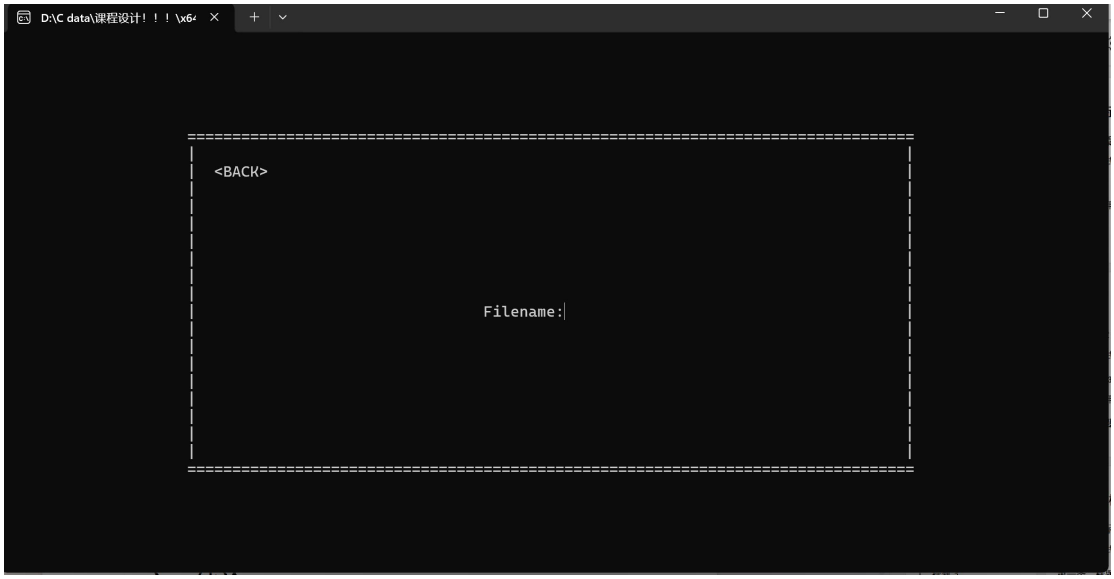

图 31　　乐器演奏模拟系统手动演奏的乐谱录制模块的写入文件名界面

在此页面中，程序会在光标处提醒使用者所保存乐谱的文件名，当写入文件名按下“回车键”，程序会显示开始录制界面，如下图所示：

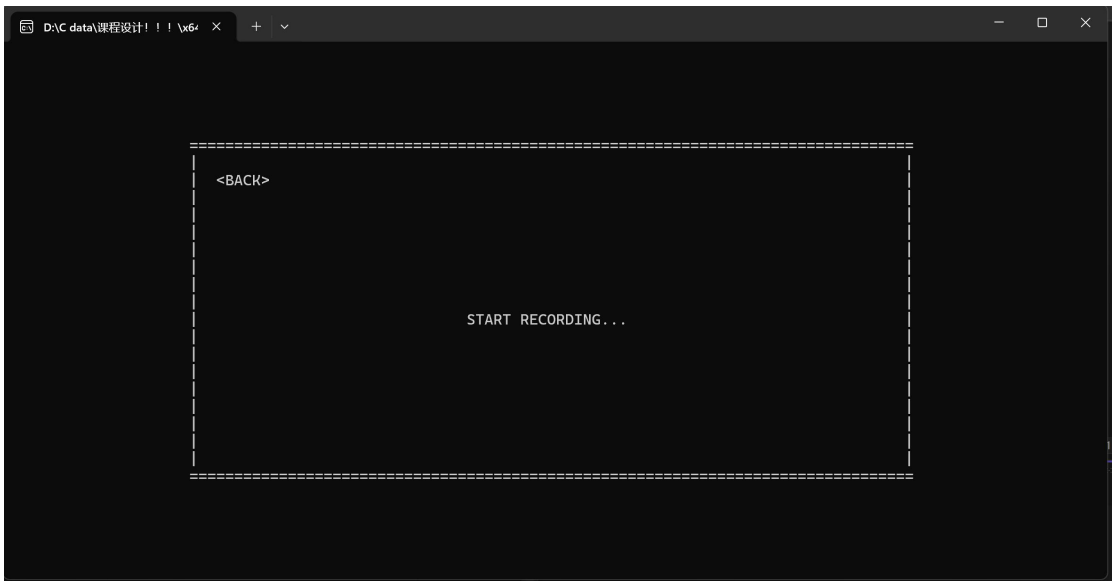

图 32　　乐器演奏模拟系统手动演奏的乐谱录制模块的开始录制标志界面

在此页面后会立马返回乐器演奏界面，当再次按下“空格键”，程序会显示结束录制界面，如下图所示：

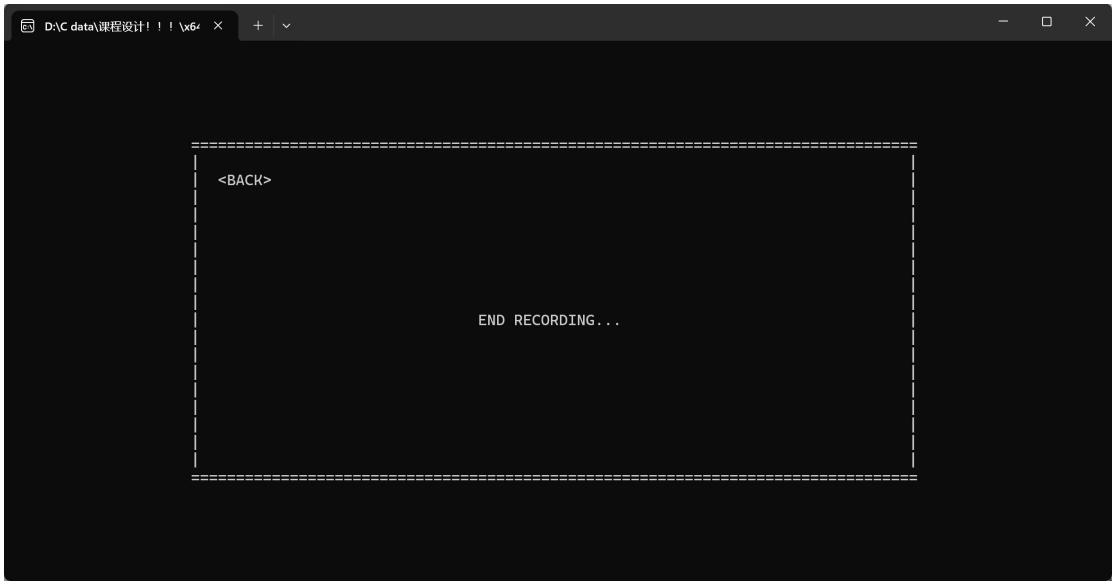

图 33　　乐器演奏模拟系统手动演奏的乐谱录制模块的结束录制标志界面

在此页面后会立马返回乐器演奏界面。

（6）自动播放乐谱选择模块

在模式选择界面中输入“1”，进入自动播放的乐谱选择模块，如下图所示：

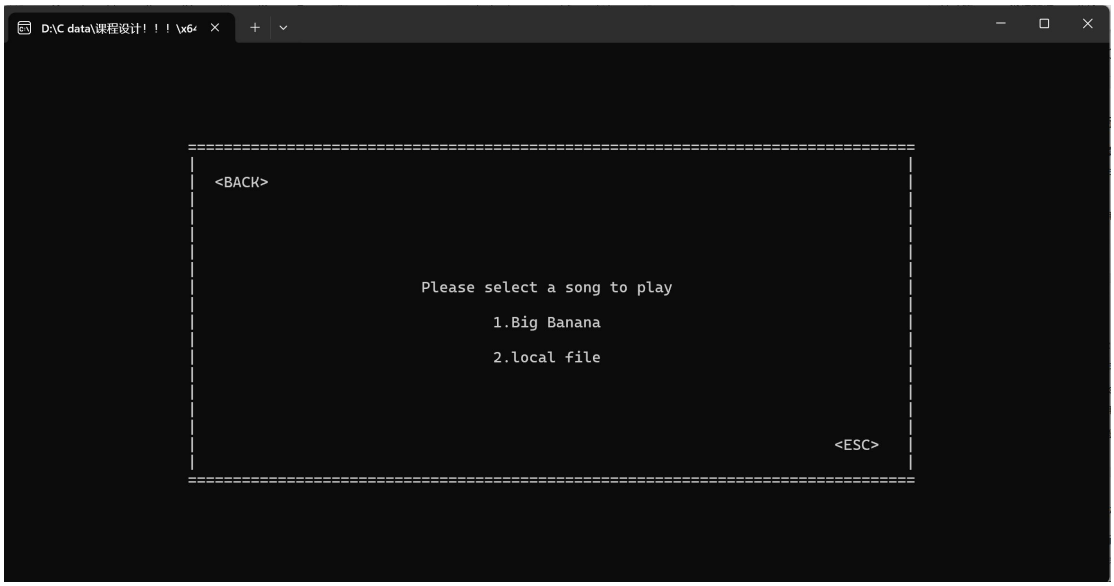

图 34　　乐器演奏模拟系统自动播放的乐谱选择界面

在此模块中，可输入“1”打开播放提前写好的多音轨的乐队乐谱《Big Banana》；可输入“2”打开单音轨播放本地文件；可输入“Backspace 键”返回模式选择模块；可输入“ESC 键”直接退出程序。

（7）乐谱一播放模块

在模自动播放的乐谱选择界面中输入“1”，进入乐谱一的多音轨播放模块，如下图所示：

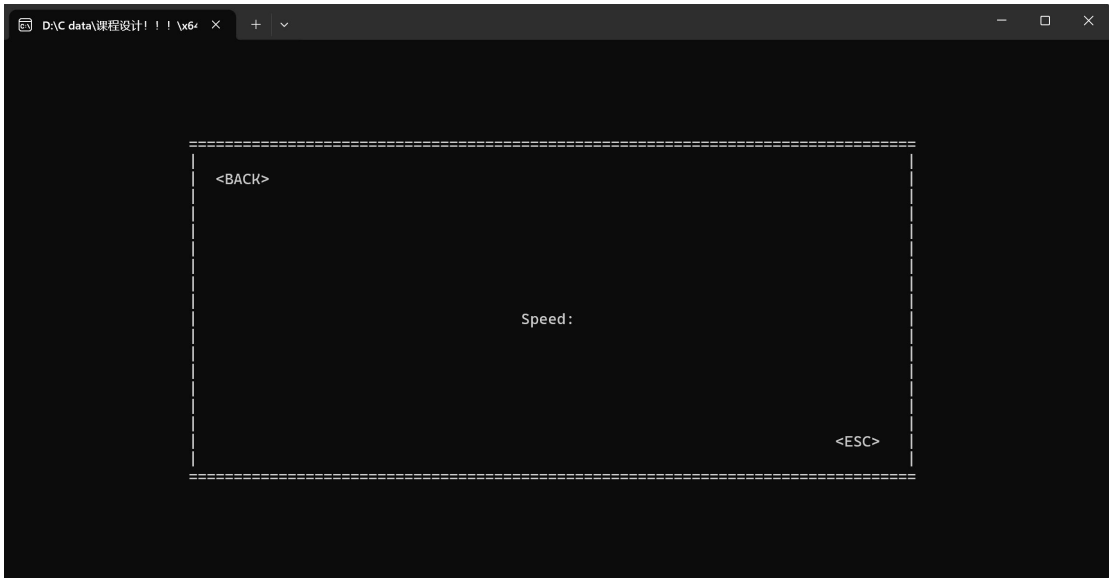

图 35　　乐器演奏模拟系统自动播放乐谱一的速度输入界面

在此页面中，程序会在光标处提醒使用者输入播放乐谱所需要的速度值，当写入速度值按下“回车键”，程序会显示开始演奏标志界面，如下图所示：

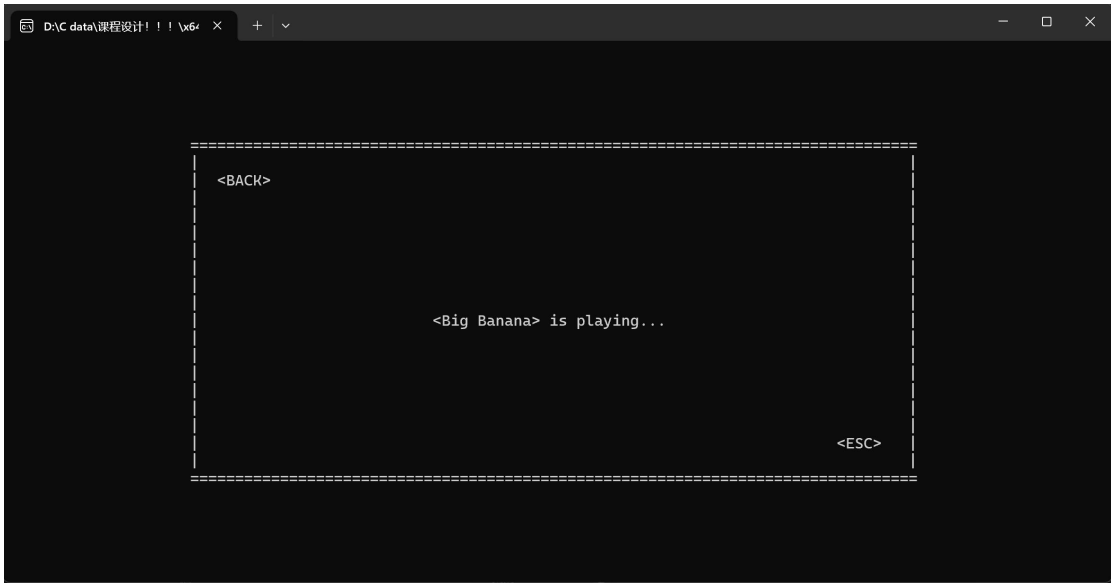

图 36　乐器演奏模拟系统自动播放乐谱一的开始演奏标志界面

在此页面后会立马开始播放进入乐谱界面，如下图所示：

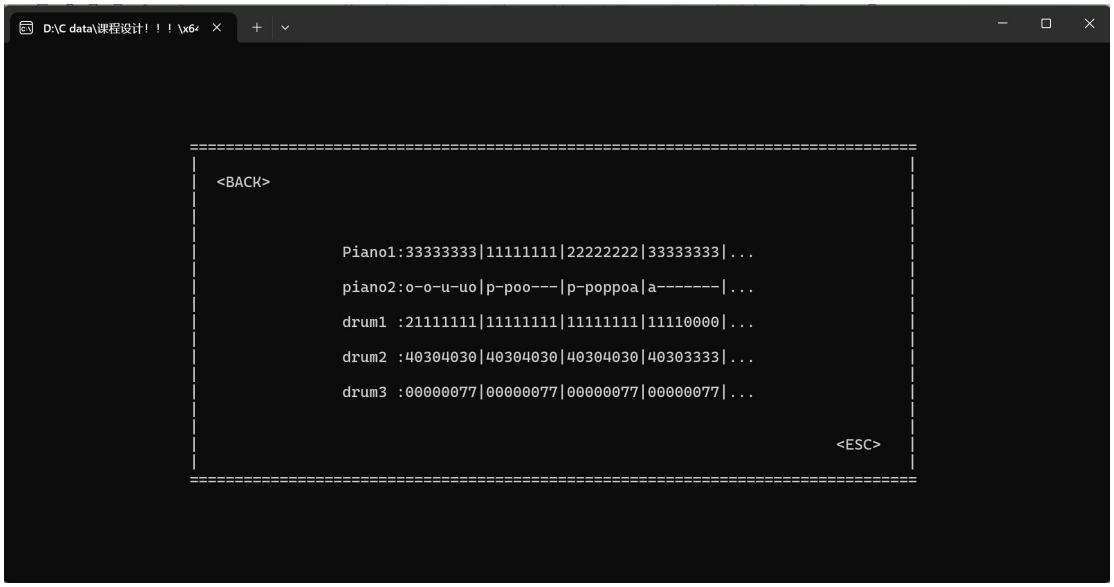

图 37　乐器演奏模拟系统自动播放乐谱一的乐谱显示界面

在此模块中，可随时输入“Backspace 键”返回乐谱选择模块；可随时输入“ESC 键”直接退出程序。

（8）本地文件播放模块

在模自动播放的乐谱选择界面中输入“2”，进入本地文件的单音轨播放模块，如下图所示：

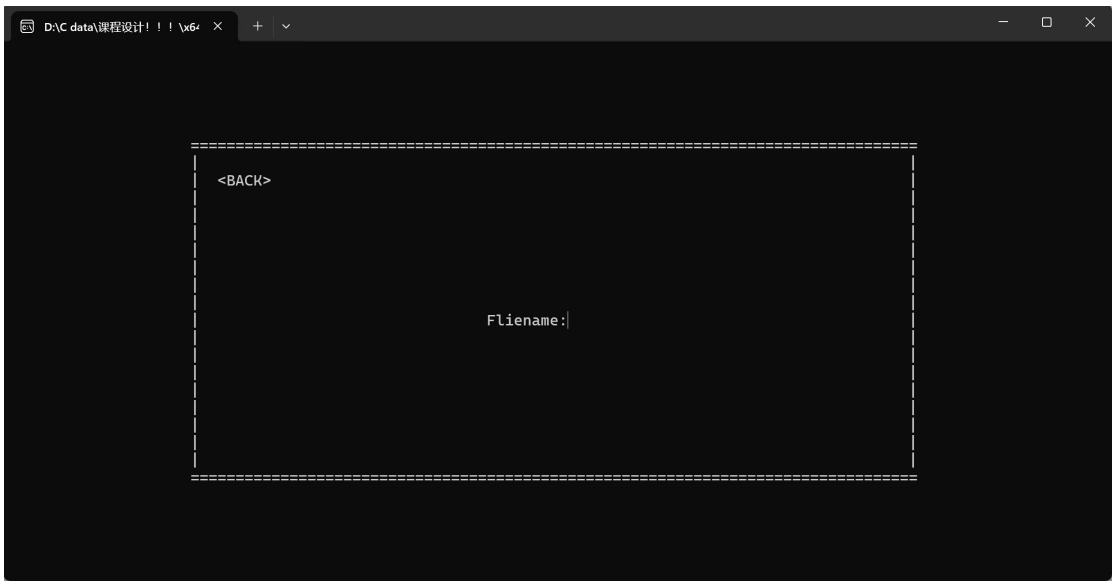

图 38　乐器演奏模拟系统自动播放本地文件的文件名输入界面

在此页面中，程序会在光标处提醒使用者输入播放乐谱的文件名，当写入文件名按下“回车键”，程序会提示输入播放乐器，如下图所示：

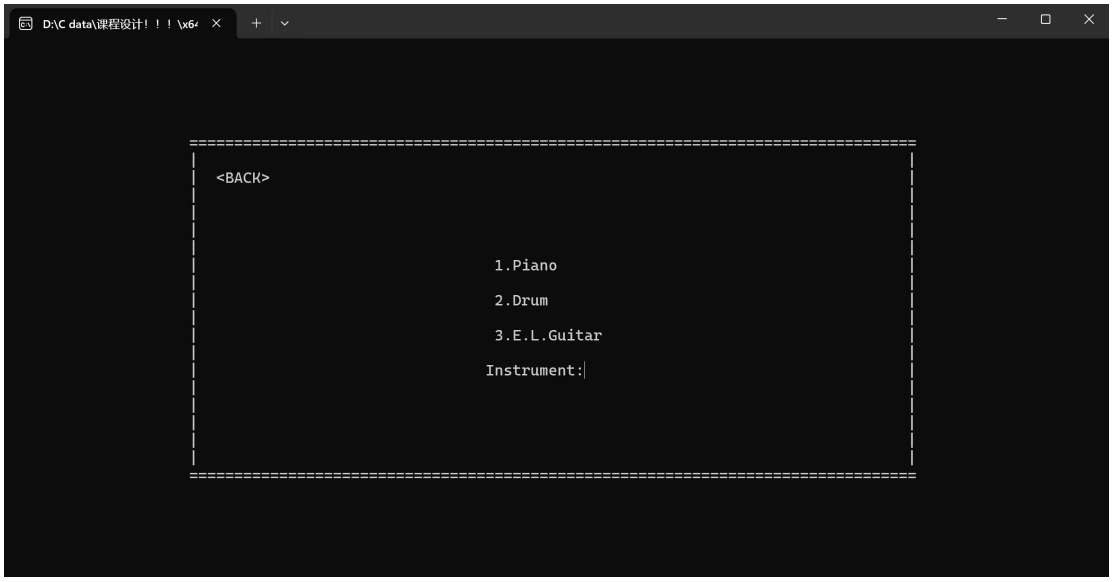

图 39　乐器演奏模拟系统自动播放本地文件的选择乐器输入界面

在此页面中，程序会在光标处提醒使用者输入播放乐谱使用的乐器，当写入乐器序号按下“回车键”，程序会提示输入播放速度，如下图所示：

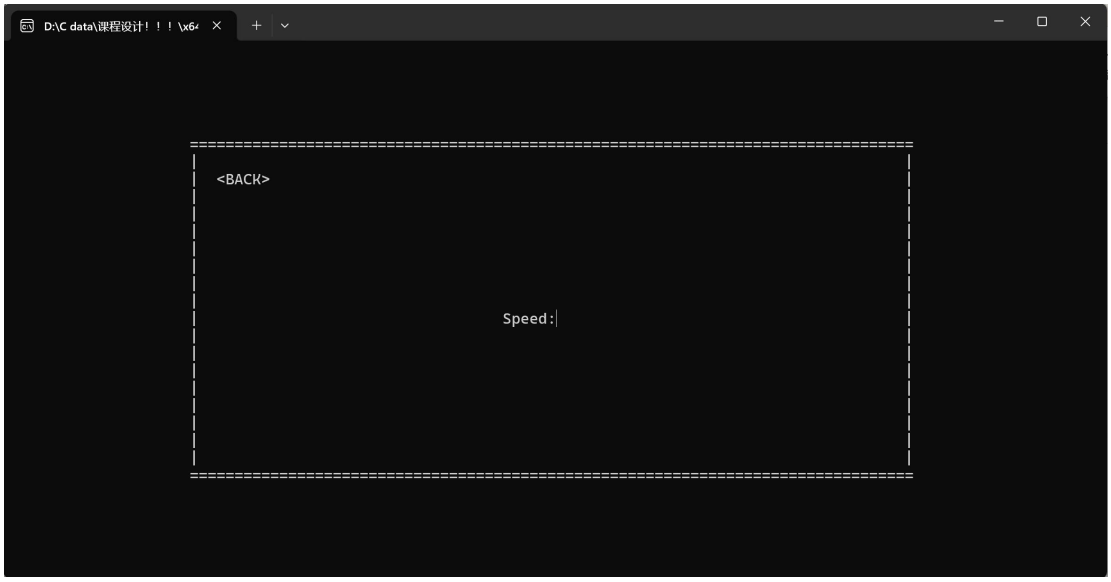

图 40　**乐器演奏模拟系统自动播放本地文件的速度输入界面**

在此页面中，程序会在光标处提醒使用者输入播放乐谱的速度，当写入速度值按下“回车键”，程序会开始播放并显示正在播放提示，如下图所示：

图 41　**乐器演奏模拟系统自动播放本地文件的正在播放提示界面**

在此模块中，《》内显示所打开的本地文件名，可随时输入“Backspace 键”返回乐谱选择模块；可随时输入“ESC 键”直接退出程序。

（9）退出模块

在其他模块中按下“ESC 键”后，进入结束界面，停留 2s 后退出程序，如下图所示：

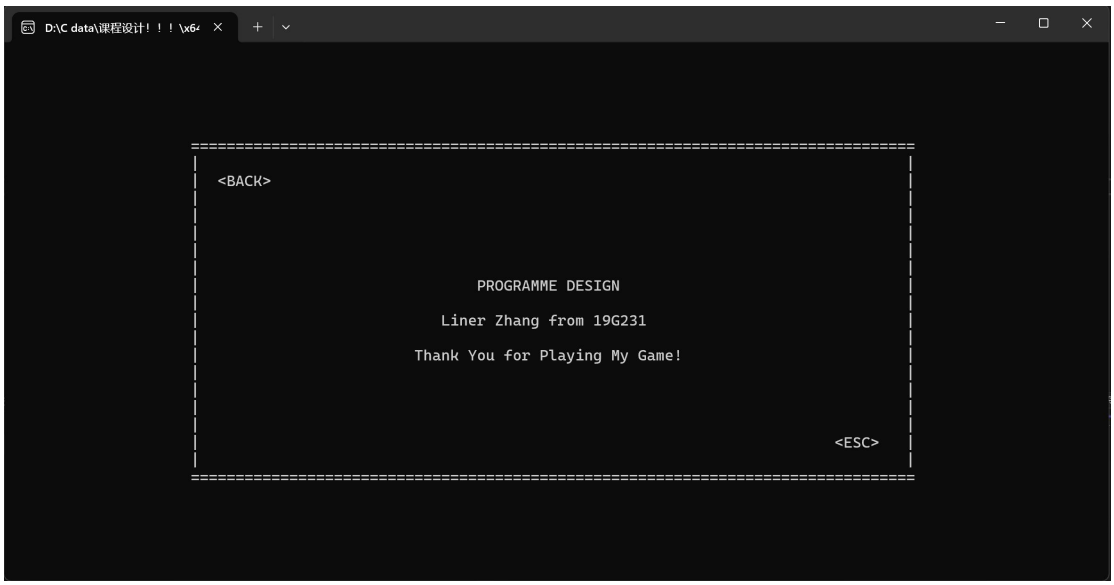

图 42　**乐器演奏模拟系统的结束界面**

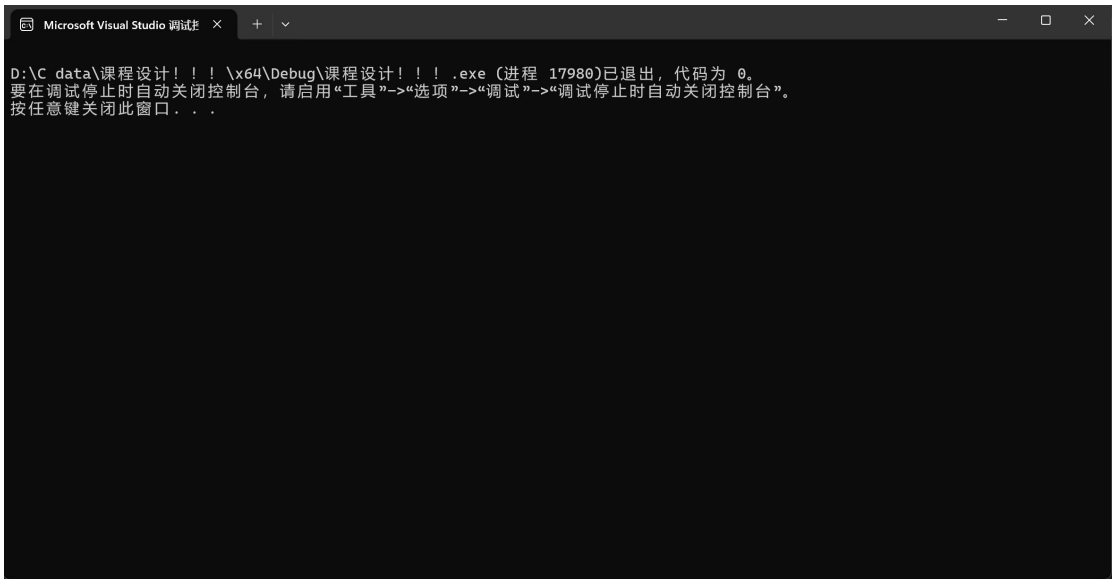

图 43　**乐器演奏模拟系统的退出界面**

六、结论

6.1 学习经验

这次课程设计的制作过程，更是一次不断学习的过程。

首先，制作过程中锻炼了我的自主学习能力，通过我的需求和我的理想功能，自主上相关网站或向老师寻求解决方案，再进一步自主学习相关库和函数，最终应用到课程设计的程序中来。

其次，制作过程中不断发现错误又不断纠正错误的过程中让我更熟练的掌握调试功能，学会自己根据报错内容找到错误原因。

最重要的是，制作过程中建立自己了的分模块实现的课程设计思想，通过模块交互和界面转换，我学会让代码写的更有条理更加简洁。

6.1 不足

这次课程设计制作的程序中仍然存在一些没有克服的问题。

首先，在通过 kd 按键进入乐器演奏界面后，其 kd 按键还会继续被作为输入按键而播放对应的乐器音频，或许可以通过将 kd 按键更换成没有匹配乐器音频的按键让他就算被当作输入也没有声音播放来解决。

其次，在播放多音轨的乐谱时，同时打开读取多份乐谱文件并多线程播放，系统打开文件需要一定时间，加之线程开启的过多会造成系统的卡顿，开头语句时间被压缩，形成杂乱的播放，这里或许可以通过多进程来改进。

最后，在读取本地乐谱文件时，单次只能读取打开播放一份本地乐谱，单音轨播放，对于多乐器多音轨的编曲只能手动在代码中写入多线程，无法在用户界面完成，对多音轨乐曲的编写十分不友好，这里或许可以完善程序的编曲功能模块，通过让用户多次输入乐谱文件名，后台自动创建线程完成。

七、参考文献

[1]【c++游戏小技巧 2：kd(类型) - CSDN App】http://t.csdnimg.cn/QVj7z
[2]【c++游戏小技巧 3：Sleep(停顿) 与 gotoxy(0,0) （无闪清屏） - CSDN App】http://t.csdnimg.cn/tOPMO
[3]【【C++】多线程(thread)使用详解 - CSDN App】http://t.csdnimg.cn/1ATWE
[4]【一文详解 C++多线程 - CSDN App】http://t.csdnimg.cn/hcYTz
[5]【知识点 1——光标位置控制（涉及句柄、COORD 结构体、GetStdHandle 函数、SetConsoleCursorPosition 函数等） - CSDN App】http://t.csdnimg.cn/Sy4tr
[6]【用 C/C++写一个简易的钢琴小程序 - CSDN App】http://t.csdnimg.cn/WeQXQ
[7]【C/C++ 控制台高级操作(非常详细) - CSDN App】http://t.csdnimg.cn/XOgLC