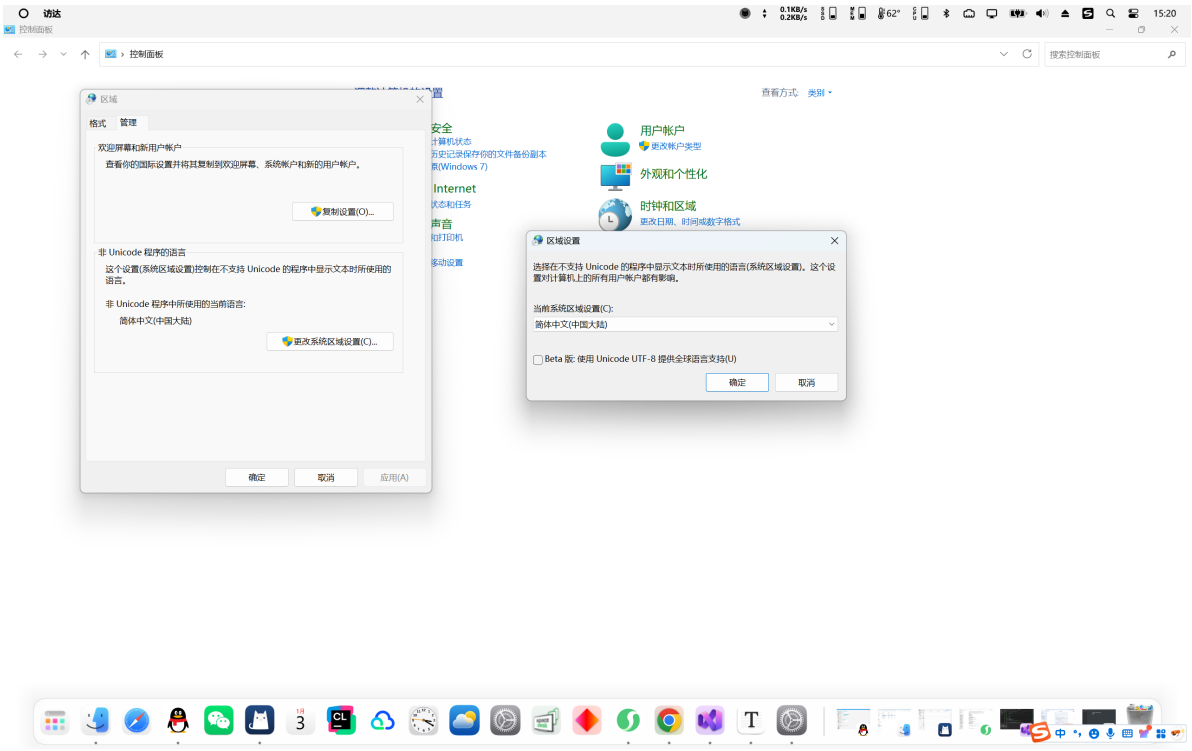
解决游戏中的中文输出问题

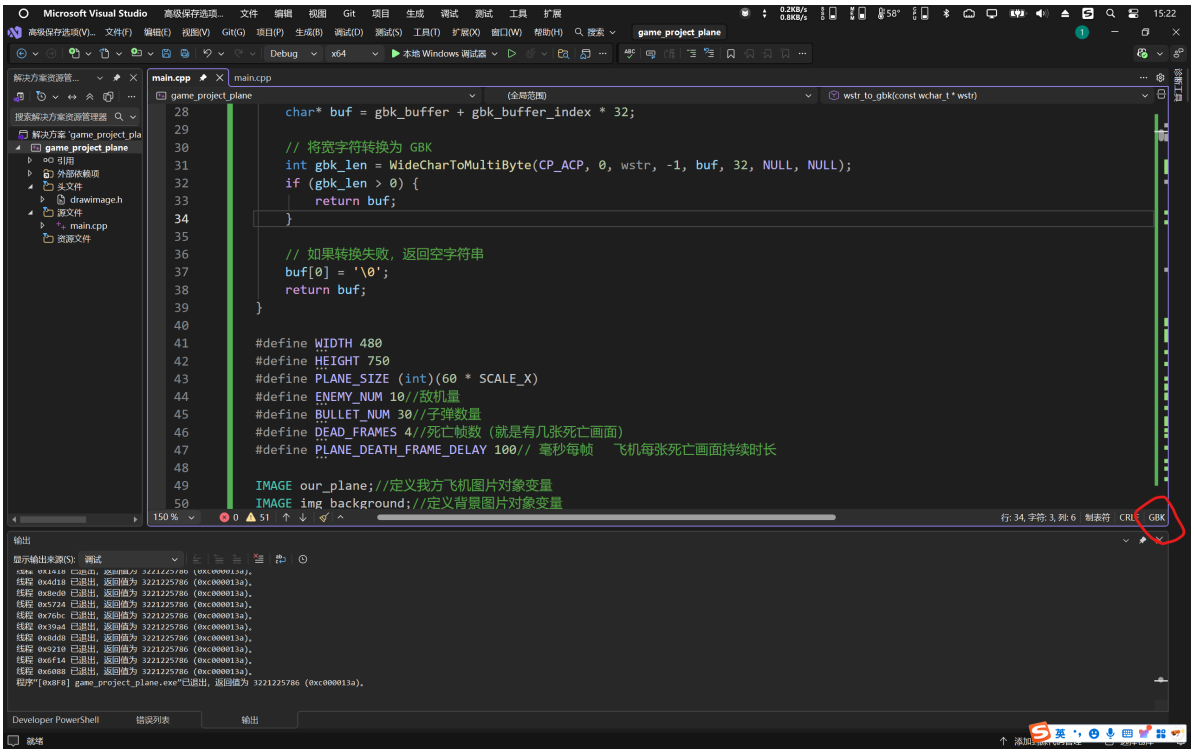
1.设置步骤

关闭系统选项UTF-8



在VS中创建一个新的游戏项目（可以直接将原来的项目复制或者直接更改）

选中右下角的编码选项



左键点击，选择通过编码重新打开，选择第一个“简体中文”

这时可能会报错或者出现中文乱码，不用管

这时，复制以下的main.cpp代码替换

```
#define _CRT_SECURE_NO_WARNINGS
#include "drawimage.h"
#include <ctime>
#include <easyx.h>
#include <stdio.h>
#include<string.h>
#include<Windows.h>
#pragma comment(lib,"winmm.lib")

#define MAX_ENEMY 60//最多敌机数量
#define MAX_SPEED 5//最大速度（‘1’型敌机速度可变）
#define BULLET_NUM 30//子弹数量
#define DEAD_FRAMES 4//死亡帧数（就是有几张死亡画面）
#define PLANE_DEATH_FRAME_DELAY 100// 毫秒每帧  飞机每张死亡画面持续时长

// 定义尺寸与缩放比例
int WIDTH;
int HEIGHT;
int PLANE_SIZE;
float scale = 1.0; // 缩放比例


// 使用宽字符字面量可以避免源代码编码问题
static char gbk_buffer[256];
static int gbk_buffer_index = 0;

// 辅助函数：将宽字符字符串转换为 GBK 编码（用于修复中文乱码）
inline const char* wstr_to_gbk(const wchar_t* wstr) {
    if (wstr == NULL) {
        return NULL;
    }

    // 循环使用缓冲区（支持多个字符串）
    gbk_buffer_index = (gbk_buffer_index + 1) % 8;
    char* buf = gbk_buffer + gbk_buffer_index * 32;

    // 将宽字符转换为 GBK
    int gbk_len = WideCharToMultiByte(CP_ACP, 0, wstr, -1, buf, 32, NULL, NULL);
    if (gbk_len > 0) {
        return buf;
    }

    // 如果转换失败，返回空字符串
    buf[0] = '\0';
    return buf;
}


IMAGE our_plane;//定义我方飞机图片对象变量
IMAGE img_background;//定义背景图片对象变量
IMAGE img_munue;//定义主菜单图片对象变量
IMAGE img_enemy[2];//定义敌机图片对象变量
IMAGE img_bullet;//定义子弹图片对象变量
```

```
IMAGE img_enemy_dead[2][DEAD_FRAMES]; //定义敌机每帧死亡图片 [type][frame]
IMAGE img_our_plane_dead[DEAD_FRAMES];//定义我方飞机每帧死亡图片
IMAGE tips_img;//定义提示图片对象变量
IMAGE chart_background;//定义排行榜背景图片
IMAGE img_gameover;//定义结束画面图片变量
IMAGE img_skill; // 技能炮弹图片

//功能相关全局变量
int score = 0; //分数变量
int main_menu_state = 1; //主菜单状态变量，1为显示主菜单，0为游戏中
int BGM = 0; //背景音乐是否播放
int tips = 0; //显示提示
int charts = 0; //显示排行榜
static DWORD lastDifficultyTick = 0;//记录上次难度改变的时间
int difficultyChangeInterval = 3000;//难度改变时间间隔
int max_score = 0;//一局中达到的最大分数

//我方飞机相关变量
int x_our_plane, y_our_plane; // 坐标改为在 main 中初始化
int x_move = 0, y_move = 0; //我方飞机移动方向变量
int our_plane_speed; // 移动速度
int our_plane_blood = 3;    //我方飞机血量
bool our_plane_live = false;//我方飞机存活状态

int skill_num = 0;//技能数量
bool unstoppable = false;//我方飞机是否处于无敌状态
DWORD unstoppable_start = 0;//标记我方飞机无敌状态开始的时间的变量
const DWORD unstoppable_long = 1500; // 无敌时间长度
bool our_plane_dying = false;//是否正在死亡
int our_plane_death_frame = 0;//我方飞机当前死亡动画帧
DWORD our_plane_death_frame_tick = 0;//我方飞机死亡帧时间标记

//定义成绩相关结构体
struct grade {
    char name[20];
    int score;
}grade_now;

//定义敌机相关参数
int ENEMY_NUM = 5;//定义屏幕上的敌机数量（默认）
int speed1 = 2;//‘1’型敌机速度（默认）
static DWORD lastGenerateTime = 0;//标记上一次生成敌机的时间
const int generateInterval = 400;//生成敌机的间隔
int perGenerate = 3;//每次生成敌机的数量
static int nextSkillScore = 300; // 下次获得技能的分数阈值

// 技能炮弹结构（场上最多一个）
struct Skill {
    int x;
    int y;
    bool live = false;
    int speed;
} skill = { 0,0,false, (int)(5 * scale) };

struct Enemy_Plane
```

```
{
    int x, y;   //坐标
    bool live = false;  //存活状态
    bool dying = false;//是否死亡中
    int death_frame = 0; // 当前死亡动画帧
    DWORD death_tick = 0;//死亡帧时间标记
    int width, height;  //敌机图片宽高
    int type;    //敌机类型
    int blood;    //敌机血量
} enemy[MAX_ENEMY];//最多50架敌机

//定义子弹相关参数
static DWORD lastShootTime = 0; // 记录上次发射子弹的时间
int shootInterval = 200; // 发射间隔（毫秒）
struct Bullet
{
    int x, y;    //坐标
    bool live = false;  //存活状态
    int speed;    //子弹速度
} bullet[BULLET_NUM];

//声明函数
void init_window();//窗口自适应
void loadImage();//加载图片资源
void our_plane_generate();//我方飞机生成函数
void enemy_generate();//敌机生成函数
void bullet_generate();//子弹生成函数
int check_pause_key();//检测暂停键函数
void score_display();//分数显示函数
void main_menu();//主菜单函数
void gameBGM_play();//游戏背景音乐播放函数
void pause();//暂停函数
void record();//记录成绩函数
void chart_display();//排行榜显示函数
void difficulty_change();//难度变化函数
void gameover();//游戏结束函数
void skill_update();//技能更新函数

//矩形相交检测函数
bool rectIntersect(int x1, int y1, int w1, int h1, int x2, int y2, int w2, int
h2)
{
    return !(x1 + w1 < x2 || x2 + w2 < x1 || y1 + h1 < y2 || y2 + h2 < y1);
}

//主程序部分
int main()
{
    // 设置控制台代码页为 GBK，确保中文正确显示
    SetConsoleOutputCP(936);
    SetConsoleCP(936);

    srand((unsigned)time(NULL)); // 初始化随机种子

    init_window(); // 初始化窗口尺寸
```

```
    initgraph(WIDTH, HEIGHT);

    //帧率设置
    const int fps = 1000 / 60; // 60 FPS
    int startTime, freamTime;//定义开始时间以及程序完成一帧的用时用于后面控制帧率

    //统一加载所需的图片资源****
    loadImage();

    //// 初始化飞机坐标
    //x_our_plane = WIDTH / 2 - PLANE_SIZE / 2;
    //y_our_plane = HEIGHT - PLANE_SIZE;

    //使用双缓冲技术绘制动画
    while (true)
    {
        //获取消息与时间
        startTime = clock();

        BeginBatchDraw();//开始双缓冲绘图
        cleardevice();//清空屏幕

        if (!main_menu_state) {
            drawimage(0, 0, &img_background);//背景图片生成

            if (!check_pause_key() && our_plane_live) {
                our_plane_generate();//调用我方飞机生成函数
                enemy_generate();//调用敌机生成函数
                bullet_generate();//调用子弹生成函数
                score_display();//调用分数显示函数
                gameBGM_play();//调用背景音乐播放函数
                difficulty_change();//调用难度变化函数
                skill_update(); // 调用技能更新函数
            }
            else if (!our_plane_live) {
                gameover();
            }
            else {
                pause();
            }
        }
        else {

            main_menu();
        }

        EndBatchDraw();//结束双缓冲绘图

        //处理帧率
        freamTime = clock() - startTime;
        if (freamTime < fps)
        {
            Sleep(fps - freamTime);
        }
    }
```

```
    getchar();
    return 0;
}

// 初始化窗口尺寸函数
void init_window() {
    int screen_h = GetSystemMetrics(SM_CYSCREEN); // 获取屏幕高度
    HEIGHT = (int)(screen_h * 0.9); // 窗口高度设为屏幕的85%
    scale = (float)HEIGHT / 750.0f; // 计算相对于原设计(750px)的缩放比例
    WIDTH = (int)(480 * scale); // 宽度按比例缩放
    PLANE_SIZE = (int)(60 * scale); // 飞机尺寸按比例缩放
    our_plane_speed = (int)(6 * scale); // 速度按比例缩放
}

void pause() {
    //暂停状态下显示暂停提示
    setbkmode(TRANSPARENT);
    settextcolor(BLACK);
    settextstyle((int)(40 * scale), 0, ("黑体")); // 字体大小随比例缩放
    // 将提示文字居中显示
    const char* pauseText = wstr_to_gbk(L"请按esc键开始与暂停");
    int ptw = textwidth(pauseText);
    int pth = textheight(pauseText);
    outtextxy((WIDTH - ptw) / 2, (HEIGHT - pth) / 2, pauseText);
}

void gameBGM_play()
{
    if (!BGM) {
        mciSendString("open resource/sound/game_music.mp3 alias gameBGM", NULL,
0, NULL);
        mciSendString("play gameBGM repeat", NULL, 0, NULL);
        BGM = 1;
    }
}

void main_menu()
{
    if (!BGM) {
        //播放主菜单背景音乐
        mciSendString("open resource/sound/homeBGM.mp3 alias menuBGM", NULL, 0,
NULL);
        mciSendString("play menuBGM repeat", NULL, 0, NULL);
        BGM = 1;
    }

    if (tips) {
        drawimage(0, 0, &tips_img);
        int current_key_state = GetAsyncKeyState(0x4D) & 0x8000; // 检测 m 键
        if (current_key_state) {
            tips = 0;
        }
    }
    else if (charts) {
```

```
            //显示排行榜
            chart_display();
            int current_key_state = GetAsyncKeyState(0x4D) & 0x8000; // 检测 m 键
            if (current_key_state) {
                charts = 0;
            }
        }
        else {
            drawimage(0, 0, &img_munue);
            // 检测 s 键，开始游戏
            if (GetAsyncKeyState(0x53) & 0x8000) {
                //停止播放音乐
                mciSendString("stop menuBGM repeat", NULL, 0, NULL);
                mciSendString("close menuBGM", NULL, 0, NULL);
                BGM = 0;
                main_menu_state = 0; // 退出主菜单，开始游戏
                score = 0; // 分数重置
                max_score = 0;
                //我方飞机重置
                x_our_plane = WIDTH / 2 - PLANE_SIZE / 2;
                y_our_plane = HEIGHT - PLANE_SIZE;
                our_plane_dying = false; // 重新初始化变量
                our_plane_live = true;
                our_plane_blood = 3; // 血量重置
                shootInterval = 200; //重置子弹发射间隔
                skill_num = 0;//重置技能数
                nextSkillScore = 300; // 重置技能阈值
                //重置敌机
                ENEMY_NUM = 5;//重置敌机数量
                speed1 = 2;//重置‘1’型敌机速度
                perGenerate = 3;//重置敌机单次生成数量
                for (int i = 0; i < MAX_ENEMY; i++) {//重置敌机状态
                    enemy[i].live = false;
                    enemy[i].dying = false;
                }
                // 重置子弹
                for (int i = 0; i < BULLET_NUM; i++) {
                    bullet[i].live = false;
                }
            }
            // 检测 r 键，显示提示
            else if (GetAsyncKeyState(0x52) & 0x8000) {
                tips = 1;
            }
            //检测 c 键，显示排行榜
            else if (GetAsyncKeyState(0x43) & 0x8000) {
                charts = 1;
            }
        }
    }

    void chart_display()
    {
        struct Entry {
            char name[20];
```

```
        int score;
    }score_list[100];

    //从储存的文件中读取数据
    FILE* fp = fopen("resource\\score.txt", "r");
    int length = 0;
    if (fp != NULL) {
        while (fscanf(fp, "%19s %d", score_list[length].name,
&score_list[length].score) == 2) {
            length++;
        }
        fclose(fp);
    }

    // 按分数从高到低排序（冒泡排序）
    for (int i = 0; i < length - 1; i++) {
        for (int j = 0; j < length - i - 1; j++) {
            // 比较分数
            if (score_list[j].score < score_list[j + 1].score) {
                Entry temp = score_list[j];
                score_list[j] = score_list[j + 1];
                score_list[j + 1] = temp;
            }
            // 分数相同则按姓名升序
            else if (score_list[j].score == score_list[j + 1].score) {
                if (strcmp(score_list[j].name, score_list[j + 1].name) > 0) {
                    Entry temp = score_list[j];
                    score_list[j] = score_list[j + 1];
                    score_list[j + 1] = temp;
                }
            }
        }
    }

    if (length > 10) {
        length = 10; // 只显示前10名
    }

    // 绘制排行榜
    drawimage(0, 0, &chart_background);//背景图片生成
    int topY = HEIGHT / 4;
    int leftX = (int)(40 * scale);
    int rightX = WIDTH - (int)(40 * scale);

    // 标题（黑色字体）
    setbkmode(TRANSPARENT);
    settextcolor(BLACK);
    settextstyle((int)(36 * scale), 0, ("黑体"));
    const char* title = wstr_to_gbk(L"(排行榜)");
    int tw = textwidth(title);
    outtextxy((WIDTH - tw) / 2, topY - (int)(40 * scale), title);

    // 列头（黑色字体）
    settextstyle((int)(22 * scale), 0, ("黑体"));
    int startY = topY + 10;
```

```
    int lineH = (int)(32 * scale);

    if (length == 0) {
        const char* noRecord = wstr_to_gbk(L"暂无排行榜记录");
        int nw = textwidth(noRecord);
        settextcolor(BLACK);
        outtextxy((WIDTH - nw) / 2, startY + 40, noRecord);
    }
    else {
        for (int i = 0; i < length; i++) {
            char buf[64];
            sprintf(buf, "%2d. %s", (int)i + 1, score_list[i].name);

            // 设置文字颜色为黑色
            settextcolor(BLACK);
            outtextxy(leftX + 10, startY + (int)i * lineH, buf);

            // 分数右对齐显示（黑色字体）
            char scoreBuf[32];
            sprintf(scoreBuf, "%d", score_list[i].score);
            int sw = textwidth(scoreBuf);
            settextcolor(BLACK);
            outtextxy(rightX - sw - 10, startY + (int)i * lineH, scoreBuf);
        }
    }

    // 底部提示（黑色字体）
    settextstyle((int)(20 * scale), 0, ("黑体"));
    const char* back = wstr_to_gbk(L"按 m 键返回主菜单");
    int bw = textwidth(back);
    settextcolor(BLACK);
    outtextxy((WIDTH - bw) / 2, HEIGHT - (int)(100 * scale), back);

    // 处理按键返回
    if (GetAsyncKeyState(0x4D) & 0x8000) {
        charts = 0;
    }
}

void score_display()
{
    //显示分数
    char score_str[20];
    sprintf(score_str, "Score: %d", score);//格式化分数字符串
    setbkmode(TRANSPARENT);//设置文字背景透明
    settextcolor(BLACK);//设置文字颜色为黑色
    settextstyle((int)(20 * scale), 0, ("黑体"));
    outtextxy(WIDTH - (int)(120 * scale), 10, score_str);//绘制分数字符串

    // 显示技能数量
    char skill_str[32];
    sprintf(skill_str, "Skill: %d", skill_num);
    outtextxy(WIDTH - (int)(120 * scale), 30, skill_str);//绘制技能数量字符串
}
```

```
//炮弹技能，V发射，B爆炸
void skill_update()
{
    // 读取当前按键状态
    static SHORT lastV = 0, lastB = 0;
    SHORT curV = GetAsyncKeyState(0x56) & 0x8000; // V
    SHORT curB = GetAsyncKeyState(0x42) & 0x8000; // B

    // 发射：边缘触发(GetAsyncKeyState 在按住键时会一直返回按下状态)
    if (curV && !lastV) {
        if (!skill.live && skill_num > 0) {
            skill.live = true;
            skill.x = x_our_plane + PLANE_SIZE / 2 - (int)(14 * scale);//飞机正中
间
            skill.y = y_our_plane; // 从飞机位置发射
            skill.speed = (int)(5 * scale);
            skill_num--;
        }
    }

    // 炮弹前进并绘制
    if (skill.live) {
        // 按速度移动（向上）
        skill.y -= skill.speed;
        // 绘制炮弹图片
        drawimage(skill.x, skill.y, &img_skill);
        // 飞出边界则删除
        if (skill.y < -50) {
            skill.live = false;
        }
    }

    // 爆炸：边缘触发(GetAsyncKeyState 在按住键时会一直返回按下状态)
    if (curB && !lastB) {
        if (skill.live) {
            // 对所有敌机进行 y 距离判定，距离<=300*缩放比例 的敌机直接进入死亡动画
            for (int i = 0; i < ENEMY_NUM; i++) {
                if (!enemy[i].live) continue;
                if (abs(enemy[i].y - skill.y) <= 300 * scale) {
                    enemy[i].live = false;
                    enemy[i].dying = true;
                    enemy[i].death_frame = 0;
                    enemy[i].death_tick = GetTickCount();
                }
            }
            // 炮弹爆炸后消失
            skill.live = false;
        }
    }
    // 更新按键状态
    lastV = curV;
    lastB = curB;
}

void gameover()
```

```
{
    drawimage(0, 0, &img_gameover);//图片生成
    //显示当前分数
    char score_str[30];
    sprintf(score_str, "Your Score: %d", score);//格式化分数字符串
    settextstyle((int)(30 * scale), 0, ("黑体")); // 设置字体
    int tw1 = textwidth(score_str);//计算字符串宽度
    int th1 = textheight(score_str);//计算字符串高度
    outtextxy((WIDTH - tw1) / 2, (HEIGHT - th1) / 2 - 20, score_str);
    //关闭背景音乐
    if (BGM) {
        mciSendString("stop gameBGM repeat", NULL, 0, NULL);
        mciSendString("close gameBGM", NULL, 0, NULL);
        BGM = 0;
    }

    int current_key_state = GetAsyncKeyState(0x4D) & 0x8000; // 检测 m 键
    if (current_key_state) {
        record();//将成绩输入排行榜文本文件
        main_menu_state = 1;//返回主菜单
    }
}

void record() //记录成绩函数
{
    FILE* fp;
    fp = fopen("resource\\score.txt", "a");//以追加方式打开文件
    if (fp == NULL) {
        // 文件不存在则创建新文件
        fp = fopen("score.txt", "a");
    }
    InputBox(grade_now.name, 20,NULL, wstr_to_gbk(L"请输入玩家姓名"), "unknown");//
弹出输入框获取玩家姓名
    grade_now.score = score;

    if (fp) {
        fprintf(fp, "%s %d\n", grade_now.name, grade_now.score);//将姓名与分数写入文
件
        fclose(fp);
    }
}

void loadImage()
{
    // 加载图片时乘以缩放比例
    loadimage(&img_background, "resource\\images\\background.png", WIDTH,
HEIGHT);//加载背景图片
    loadimage(&img_munue, "resource\\images\\home_menu.png", WIDTH, HEIGHT);//加
载主菜单图片
    loadimage(&img_gameover, "resource\\images\\gameover.png", WIDTH, HEIGHT);//
加载结束画面图片
    loadimage(&tips_img, "resource\\images\\tips.png", WIDTH, HEIGHT);//加载提示图
片
    loadimage(&chart_background, "resource\\images\\chart.jpg", WIDTH,
HEIGHT);//加载排行榜背景图片
```

```
    loadimage(&our_plane, "resource\\images\\hero1.png", PLANE_SIZE,
PLANE_SIZE);//加载我方飞机图片并调整大小

    loadimage(&img_enemy[0], "resource\\images\\enemy1.png", (int)(70 * scale),
(int)(60 * scale));//‘1’型敌机
    loadimage(&img_enemy[1], "resource\\images\\enemy2.png", (int)(99 * scale),
(int)(95 * scale));//‘2’型敌机
    loadimage(&img_bullet, "resource\\images\\bullet1.png", (int)(7 * scale),
(int)(21 * scale));//子弹

    loadimage(&img_skill, "resource\\images\\skill.png", (int)(28 * scale),
(int)(56 * scale));//炮弹技能图片

    loadimage(&img_our_plane_dead[0], "resource\\images\\hero_blowup_n1.png",
PLANE_SIZE, PLANE_SIZE);//我方飞机死亡
    loadimage(&img_our_plane_dead[1], "resource\\images\\hero_blowup_n2.png",
PLANE_SIZE, PLANE_SIZE);
    loadimage(&img_our_plane_dead[2], "resource\\images\\hero_blowup_n3.png",
PLANE_SIZE, PLANE_SIZE);
    loadimage(&img_our_plane_dead[3], "resource\\images\\hero_blowup_n4.png",
PLANE_SIZE, PLANE_SIZE);

    int w1 = (int)(70 * scale), h1 = (int)(60 * scale);
    loadimage(&img_enemy_dead[0][0], "resource\\images\\enemy1_down1.png", w1,
h1);//‘1’型敌机死亡
    loadimage(&img_enemy_dead[0][1], "resource\\images\\enemy1_down2.png", w1,
h1);
    loadimage(&img_enemy_dead[0][2], "resource\\images\\enemy1_down3.png", w1,
h1);
    loadimage(&img_enemy_dead[0][3], "resource\\images\\enemy1_down4.png", w1,
h1);

    int w2 = (int)(99 * scale), h2 = (int)(95 * scale);
    loadimage(&img_enemy_dead[1][0], "resource\\images\\enemy2_down1.png", w2,
h2);//‘2’型敌机死亡
    loadimage(&img_enemy_dead[1][1], "resource\\images\\enemy2_down2.png", w2,
h2);
    loadimage(&img_enemy_dead[1][2], "resource\\images\\enemy2_down3.png", w2,
h2);
    loadimage(&img_enemy_dead[1][3], "resource\\images\\enemy2_down4.png", w2,
h2);
}

void our_plane_generate()//我方飞机生成函数
{
    ExMessage msg = { 0 };//定义消息变量结构体，用于储存获取的键盘监测消息

    if (peekmessage(&msg, EX_KEY))//监测键盘消息，用于控制我方飞机移动
    {
        if (msg.message == WM_KEYDOWN)//按键按下
        {
            switch (msg.vkcode)
            {
            case VK_UP:
                y_move = -1;
```

```
                break;
            case VK_DOWN:
                y_move = 1;
                break;
            case VK_LEFT:
                x_move = -1;
                break;
            case VK_RIGHT:
                x_move = 1;
                break;
            }
        }
        else if (msg.message == WM_KEYUP)//按键抬起
        {
            if ((msg.vkcode == VK_LEFT) || (msg.vkcode == VK_RIGHT))//水平移动按键
抬起
                x_move = 0;
            if ((msg.vkcode == VK_UP) || (msg.vkcode == VK_DOWN))//垂直移动按键抬起
                y_move = 0;
        }
    }

    //判断我方飞机是否超出范围，超出反弹动画
    if (x_our_plane <= WIDTH - PLANE_SIZE && x_our_plane >= 0)//x在范围内，正常移动
    {
        x_our_plane += x_move * our_plane_speed;
    }
    else if (x_our_plane <= 0)//超出左边界
    {
        x_our_plane = 0;
    }
    else//超出右边界
    {
        x_our_plane = WIDTH - PLANE_SIZE;
    }

    if (y_our_plane >= 0 && y_our_plane <= HEIGHT - PLANE_SIZE)//y在范围内，正常移动
    {
        y_our_plane += y_move * our_plane_speed;
    }
    else if (y_our_plane <= 0)//超出上边界
    {
        y_our_plane = 0;
    }
    else//超出下边界
    {
        y_our_plane = HEIGHT - PLANE_SIZE;
    }

    //绘制我方飞机
    if (our_plane_dying) {//如果正在死亡
        DWORD now = GetTickCount();
        if (now - our_plane_death_frame_tick > PLANE_DEATH_FRAME_DELAY) {//以实践
间隔判断是否播放下一张画面
            our_plane_death_frame++;
```

```
            our_plane_death_frame_tick = now;
        }

        if (our_plane_death_frame < DEAD_FRAMES) {//如果还没播完
            drawimage(x_our_plane, y_our_plane,
&img_our_plane_dead[our_plane_death_frame]);
        }
        else {
            our_plane_live = false;//标记我方飞机死亡
        }
    }
    else {//非死亡状态
        if (unstoppable) {//无敌状态
            DWORD now = GetTickCount();
            if (now - unstoppable_start > unstoppable_long) {//无敌状态到时
                unstoppable = false;
            }
            else {//无敌状态没到时
                if ((now / 100) % 2 == 0) {//无敌状态闪烁（间隔100ms）（now是100的奇数
倍还是偶数倍）
                    drawimage(x_our_plane, y_our_plane, &our_plane);
                }
            }
        }

        if (!unstoppable) {//不是无敌状态      （用if而不是else是为了把 无敌状态到时的标记
也考虑进去）
            drawimage(x_our_plane, y_our_plane, &our_plane);//正常绘制我方飞机
        }
    }

    msg.message = 0;
}

void enemy_generate()
{
    // 更新并绘制所有敌机状态（移动、死亡动画、碰撞）
    for (int i = 0; i < ENEMY_NUM && i < MAX_ENEMY; i++) {
        if (enemy[i].dying) {
            DWORD now = GetTickCount();
            if (now - enemy[i].death_tick > PLANE_DEATH_FRAME_DELAY) {//判断是否播
放下一张画面
                enemy[i].death_frame++;
                enemy[i].death_tick = now;
            }

            if (enemy[i].death_frame < DEAD_FRAMES) {//如果还没播完
                int t = enemy[i].type - 1;
                drawimage(enemy[i].x, enemy[i].y, &img_enemy_dead[t]
[enemy[i].death_frame]);
            }
            else {//播放结束，清除此敌机
                enemy[i].dying = false;
                if (enemy[i].type == 1) {
                    score++;//分数+1
```

```
                }
                else if (enemy[i].type == 2) {
                    score += 2;//分数+2
                }

            }
            continue;//处理下一个
        }

        if (enemy[i].live) {
            // 存活的敌机每帧向下移动并绘制
            int speed = (enemy[i].type == 1) ? (int)(speed1 * scale) : (int)(1 *
scale); // 根据比例设置速度
            if (speed < 1) speed = 1;
            enemy[i].y += speed;
            //绘制当前帧的敌机
            if (enemy[i].type == 1) {
                drawimage(enemy[i].x, enemy[i].y, &img_enemy[0]);
            }
            else {
                drawimage(enemy[i].x, enemy[i].y, &img_enemy[1]);
            }

            // 与我方飞机碰撞检测
            if (!our_plane_dying && !unstoppable) {//我方飞机处于可碰撞状态时
                if (rectIntersect(enemy[i].x, enemy[i].y, enemy[i].width,
enemy[i].height,
                    x_our_plane, y_our_plane, PLANE_SIZE, PLANE_SIZE)) {//如果此敌
机碰到我方飞机
                    our_plane_blood--;//血量-1
                    unstoppable = true;//无敌状态
                    unstoppable_start = GetTickCount();//标记无敌开始时间
                    if (our_plane_blood <= 0) {//如果血量过低，准备开始死亡
                        our_plane_dying = true;
                        our_plane_death_frame = 0;
                        our_plane_death_frame_tick = GetTickCount();//标记死亡开始
时间
                    }
                    //被碰敌机也死亡
                    enemy[i].live = false;
                    enemy[i].dying = true;
                    enemy[i].death_frame = 0;
                    enemy[i].death_tick = GetTickCount();//标记敌机死亡开始时间
                }
            }

            // 离开屏幕后标记为未存活，下一帧可被重新生成
            if (enemy[i].y > HEIGHT) {
                enemy[i].live = false;
                if (enemy[i].type == 1) {
                    score--;//分数-1
                }
                else if (enemy[i].type == 2) {
                    score -= 2;//分数-2
                }
```

```
            }
        }
    }

    DWORD now = GetTickCount();
    if (now - lastGenerateTime >= generateInterval) {//隔一段时间生成一次
        int canGenerate = 0;//统计可以再生成的敌机数
        for (int i = 0; i < ENEMY_NUM; i++) {
            if (!enemy[i].live) canGenerate++;
        }

        if (canGenerate > 0) {
            int generate_num = (canGenerate < perGenerate) ? canGenerate :
perGenerate;
            //生成
            for (int s = 0; s < generate_num; s++) {
                int idx = -1;
                for (int i = 0; i < ENEMY_NUM; i++) {//找到那个要生成的敌机的下标
                    if (!enemy[i].live && !enemy[i].dying) {
                        idx = i;
                        break;
                    }
                }
                if (idx == -1) break;//没找到就退出生成

                // 60%‘1’型敌机，40%‘2’型敌机
                if (rand() % 10 > 3) {
                    enemy[idx].type = 1;
                    enemy[idx].blood = 1;
                    enemy[idx].width = (int)(70 * scale);
                    enemy[idx].height = (int)(60 * scale);
                }
                else {
                    enemy[idx].type = 2;
                    enemy[idx].blood = 3;
                    enemy[idx].width = (int)(99 * scale);
                    enemy[idx].height = (int)(95 * scale);
                }
                enemy[idx].live = true;
                enemy[idx].x = rand() % (WIDTH - enemy[idx].width);//x坐标在显示范
围内随机生成
                enemy[idx].y = -enemy[idx].height;
            }
        }
        lastGenerateTime = now;
    }
}

void bullet_generate()
{
    //创建子弹
    DWORD currentTime = GetTickCount(); // 获取当前系统时间

    if (currentTime - lastShootTime > shootInterval) {
        //按下空格键生成子弹
```

```
        if (GetAsyncKeyState(VK_SPACE) & 0x8000) {
            for (int i = 0; i < BULLET_NUM; i++) {
                if (!bullet[i].live) {
                    bullet[i].x = x_our_plane + PLANE_SIZE / 2 - (int)(3 *
scale); // 子弹位置居中
                    bullet[i].y = y_our_plane;
                    bullet[i].live = true; // 标记子弹为存活状态
                    break; // 生成一个子弹后退出循环
                }
            }
            lastShootTime = currentTime; //更新上次发射时间
        }
    }

    //子弹的移动
    for (int i = 0; i < BULLET_NUM; i++) {
        if (bullet[i].live) {
            bullet[i].speed = (int)(8 * scale);
            bullet[i].y -= bullet[i].speed;
            if (bullet[i].y < 0) {
                bullet[i].live = false;//子弹出界，标记为未存活
            }
        }
    }

    for (int i = 0; i < BULLET_NUM; i++) {//遍历每个子弹
        if (!bullet[i].live) continue;//跳过不存活的
        for (int j = 0; j < ENEMY_NUM && j < MAX_ENEMY; j++) {//遍历每个敌机
            if (!enemy[j].live) continue;//跳过不存活的
            int bSize = (int)(10 * scale);
            if (rectIntersect(bullet[i].x, bullet[i].y, bSize, bSize,
                enemy[j].x, enemy[j].y, enemy[j].width, enemy[j].height)) {
                //如果碰撞
                bullet[i].live = 0;
                enemy[j].blood--;
                if (enemy[j].blood <= 0) {//判断敌机是否死亡
                    enemy[j].live = false;
                    enemy[j].dying = true;
                    enemy[j].death_frame = 0;
                    enemy[j].death_tick = GetTickCount();//标记死亡时间
                }
            }
        }
    }

    //绘制子弹
    for (int i = 0; i < BULLET_NUM; i++) {
        if (bullet[i].live) {
            drawimage(bullet[i].x, bullet[i].y, &img_bullet);
        }
    }
}

void difficulty_change() {
    DWORD now = GetTickCount();//更新当前时间
```

```
    if (max_score < score) {
        max_score = score;//更新max_score
    }

    if (score >= 100) {//分数大于等于100开始改变
        int targetEnemyNum = (int)(score / 10);//每10分多1个敌机，上限是60
        if (targetEnemyNum > MAX_ENEMY) {
            targetEnemyNum = MAX_ENEMY;
        }
        int targetSpeed = (int)(2 + score / 200);//每200分‘1’型敌机速度加1，上限是5
        if (targetSpeed > MAX_SPEED) {
            targetSpeed = MAX_SPEED;
        }
        //根据分数设置target难度，每隔一段时间实际值向target靠近，防止难度陡增

        if (now - lastDifficultyTick >= difficultyChangeInterval) {//敌机数量、速度
生成速率的变化
            if (ENEMY_NUM < targetEnemyNum) {
                ENEMY_NUM++;
            }
            if (speed1 < targetSpeed) {
                speed1++;
            }
            perGenerate = int(3 + score / 1000);//每1000分单次敌机生成加1 (1000很大，
所以不用考虑perGenerate突变)
            lastDifficultyTick = now;
        }

        if (score % 50 == 0 && shootInterval > 60) {//子弹速度的变化
            shootInterval = 200 - 20 * score / 50;  //每50分射击间隔缩短20，最少为60
            if (shootInterval < 60) {
                shootInterval = 60;
            }
        }

        // 每达到 nextSkillScore 增加一次技能并提升阈值，避免重复多加
        while (max_score >= nextSkillScore) {
            skill_num = skill_num + 1 + max_score / 2000;//每300分加1点技能，每多
2000分额外多加1点
            nextSkillScore += 300;
        }
    }
}

//暂停函数
int check_pause_key() {
    static int is_paused = 0; // 静态变量记录暂停状态
    static int last_key_state = 0; // 记录上一次按键状态
    int current_key_state = GetAsyncKeyState(VK_ESCAPE) & 0x8000; // 检测 ESC 键

    // 判断是否为"按下"的瞬间（边缘触发）
    if (current_key_state && !last_key_state) {
        is_paused = ~is_paused; // 切换暂停状态
        if (BGM) {
```

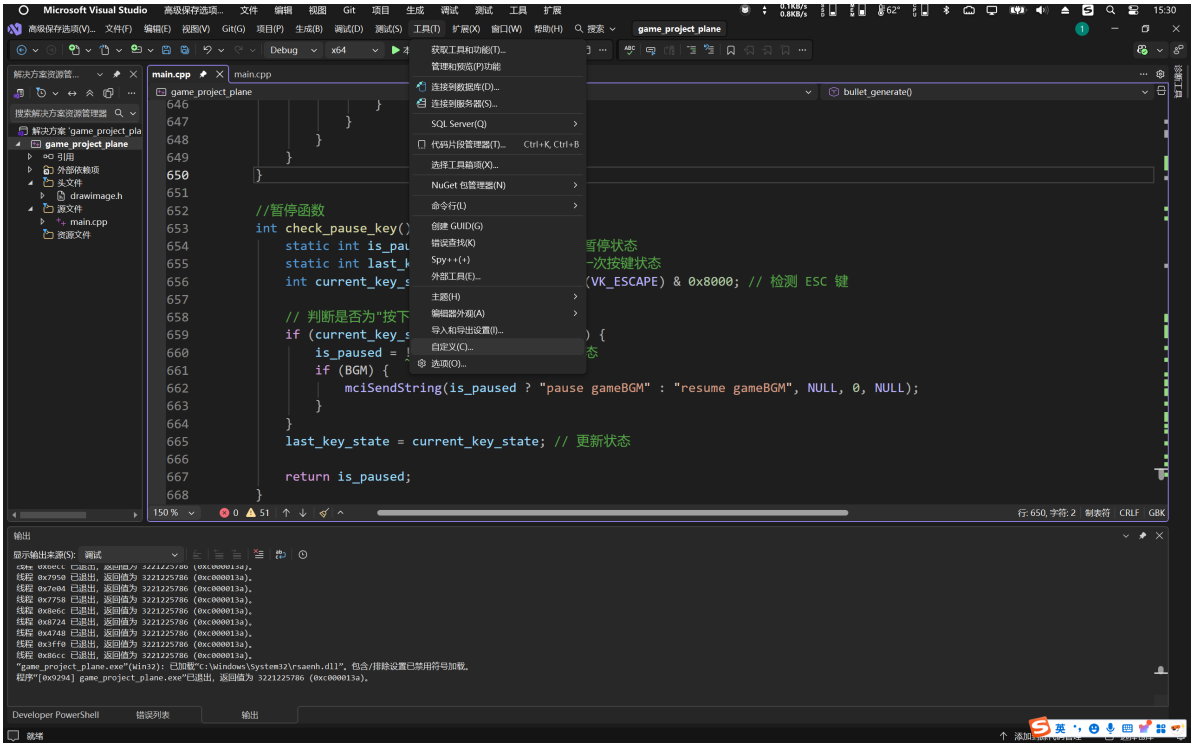
```
            mciSendString("pause gameBGM", NULL, 0, NULL);
            BGM = 0;
        }
        else {
            mciSendString("play gameBGM repeat", NULL, 0, NULL);
            BGM = 1;
        }
    }
    last_key_state = current_key_state; // 更新状态

    return is_paused;
}
```

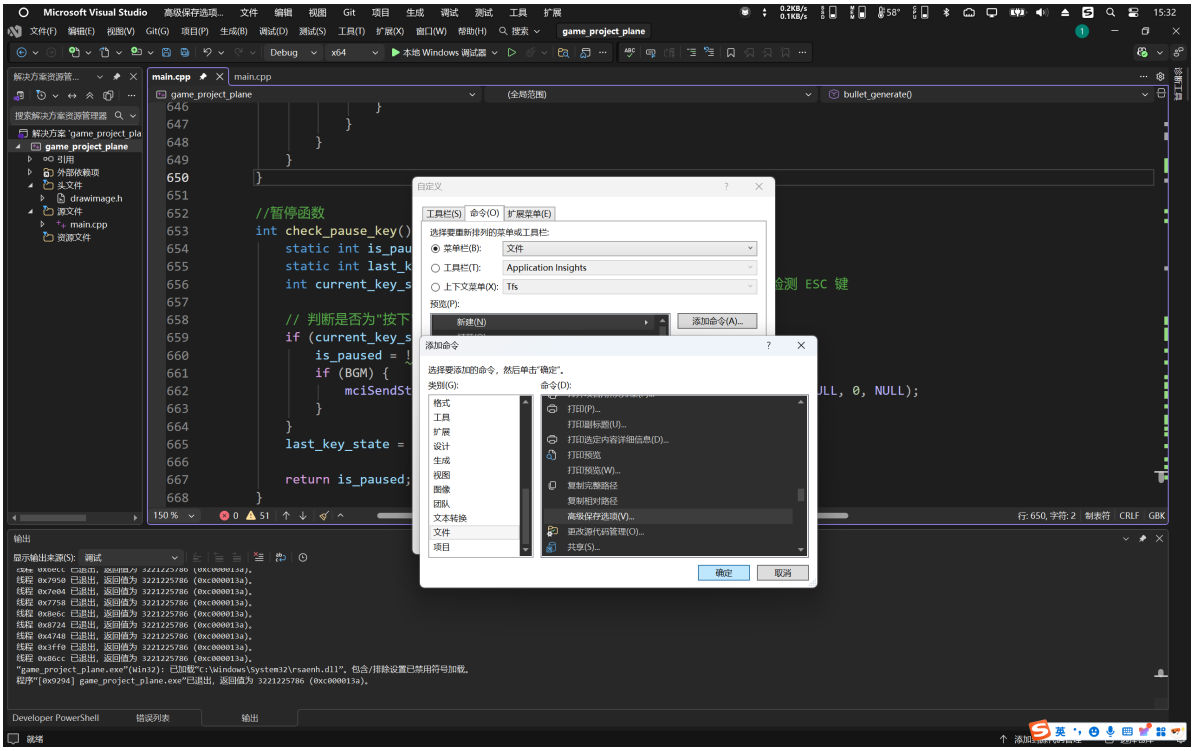
这就是原来的代码，带是因为源代码的文本编码方式改变，所以需要重新输入，如果相关的其他历史代码出现中文输出问题也可以这么解决，但是注意在改变编码方式之前先复制或者备份源代码，改变后再重新粘贴

现在，不要运行，不要关闭VS

找到高级保存选项（此选项不会默认显示）

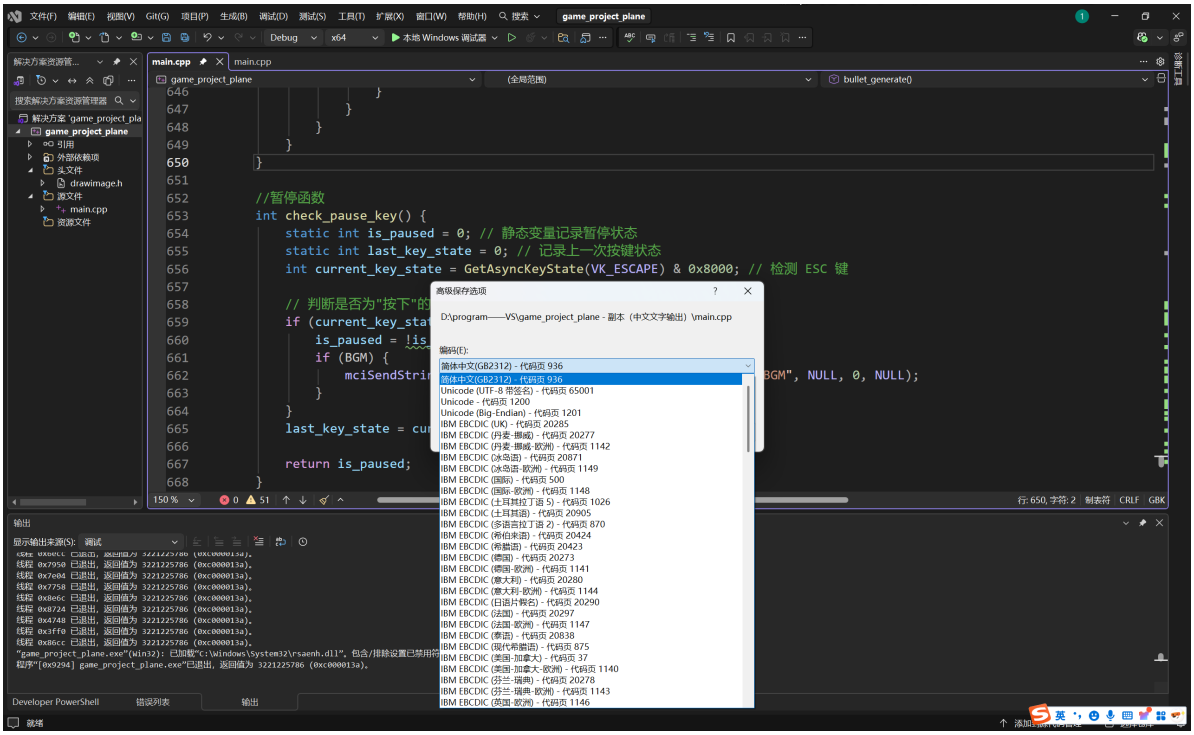
工具-》自定义-》菜单栏（文件）-》添加命令-》文件中找到高级保存选项-》添加





高级保存选项重新保存

任然选择第一个选项（简体中文）



2.原理简介（大白话）

中文输出问题

创建了一个字符转换函数：将宽字符字符串转换为 GBK 编码（用于修复中文乱码）

设置原因 （感谢AI）

取消勾选“UTF-8 Beta”选项之前，您的源代码文件（`main.cpp`）**极有可能是以 UTF-8 编码保存的。**

当您取消 UTF-8 支持后，Windows 的系统活动代码页（Active Code Page）恢复为 **936 (GBK)。**Visual Studio 仍然**从磁盘上读取了那个 UTF-8 编码的源文件。**

这时，IDE 或编译器可能会尝试用当前系统代码页（GBK）去解读这个 UTF-8 文件。由于中文字符在 UTF-8 中是 3 字节，在 GBK 中是 2 字节，这种"误读"有时能"碰巧"解析出部分内容（尤其是ASCII部分），从而**偶然编译成功一次。**但更可能的是，IDE 在加载文件时，**在内存中将其从 UTF-8 转换成了 GBK 格式**，使得编译得以通过。

首次编译**并运行程序后**，Visual Studio 可能会在后台执行一些操作（如格式化、自动保存、或仅仅是内部缓冲区的刷新），**将当前内存中"被转换为GBK"的代码内容，写回（保存）到了磁盘上的** `main.cpp` **文件中。**至此，源代码文件**从 UTF-8 编码被永久地、静默地转换并覆盖为了 GBK 编码。**

3.其他问题

我暂时没有

4.鸣谢

感谢老同学（github：膨胀pengzhang）为修复中文输出做出的贡献

感谢Gemini对于窗口自适应做出的建议和修改方案

感谢元宝对于文件代码编码问题做出的贡献