高级语言程序设计期末大作业

虚拟发动机性能监控模块开发

学　号：　2353900

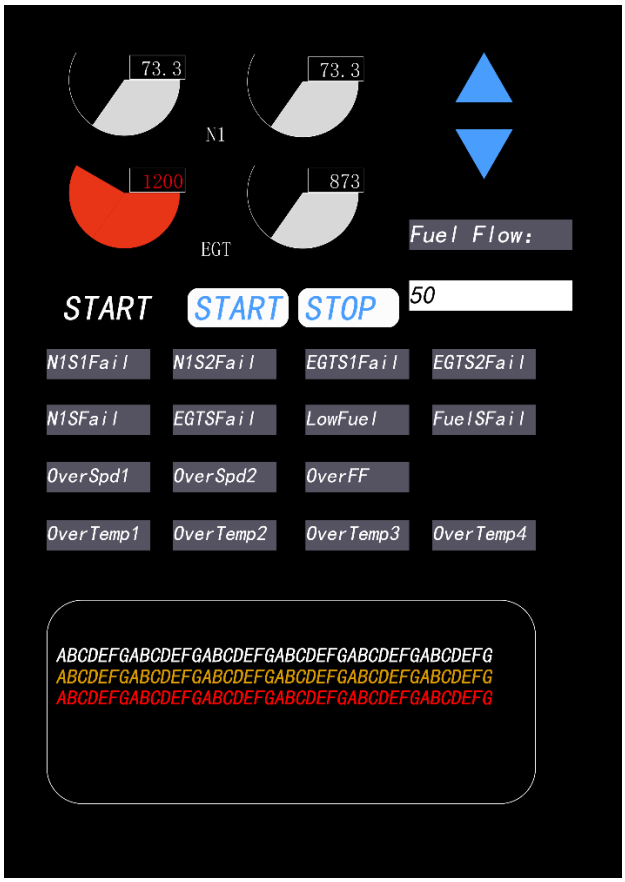
姓　名：　汪嘉晨

学　院：　国豪书院

班　级：　工科嘉定二班

一、设计思路与功能概述

(一)设计思路



题目**要求**主要如下：

①本次作业需要你完成发动机转速动态变化的界面显示。你的程序应当分为两部分：数据模拟生成：按照固定时间间隔（5ms）生成数据，模拟传感器的输入。数据处理显示：接收模拟数据的输入，根据一定的规则显示在界面上。

②程序每次运行时，新建表格文件保存数据。界面每 5ms 进行一次数据刷新，且将数据添加至表格文件中。

③表盘需要包括 N1 发动机转速表、两大按钮、燃油流速、状态显示框（包括启动 start 和稳态 run）。

④实现添加“增加推力”和“减小推力”按钮；设计异常测试情况，程序可以产生 14 种异常情况的数据，且能够检测出这些异常情况。使用其他颜色突出异常值/扇形，并添加文字警示。

因此，本程序共分为 5 个主要的部分：逻辑处理，数据保存，界面绘制，鼠标点击与 main 函数：

1.逻辑处理（数据计算，检测，状态管理）：

（1）发动机数据模拟：

定义了 EngineData 结构体，用于存储发动机的各种数据，包括转速、排气温度、燃油余量和燃油流速等：

```
// 结构体用于存储发动机数据
struct EngineData
{
    double N1[2]; // 左发动机转速（两个传感器）
    double N2[2]; // 右发动机转速（两个传感器）
    double T1[2]; // 左发动机排气温度（两个传感器）
    double T2[2]; // 右发动机排气温度（两个传感器）
    double C;     // 燃油余量
    double V;     // 燃油流速

    // 基准值，允许在稳定状态下调整
    double baseN1;
    double baseN2;
    double baseT1;
    double baseT2;
    double baseV;

    EngineData() :
        baseN1(0.95 * RATED_RPM),
        baseN2(0.95 * RATED_RPM),
        baseT1(735),
        baseT2(735),
        baseV(43),
    {
    }
};
```

generateData 函数可以根据不同的发动机状态（初始化、启动前两秒、启动两秒以后、稳定、停止）生成题目要求的对应的模拟数据，并引入随机噪声来模拟数据的随机波动。generateRandomNoise 函数用于生成随机噪声，增加数据的真实性。

```
// 生成随机噪声，用于模拟数据的随机波动
double generateRandomNoise(double range)
{
    return ((rand() / (double)RAND_MAX) * 2 - 1) * range;
}
```

（2）发动机状态管理：

定义了多个布尔变量：

```
bool isInit = true;
bool isStarted_1 = false; // 启动前两秒
bool isStarted_2 = false; // 启动对数函数
bool isStable = false;
bool isStopping = false;
```

来表示发动机的不同状态。通过这些状态变量和 generateData 函数，控制发动机数据的生成过程。

（3）推力控制：

实现 increaseThrust 和 decreaseThrust 函数，用于增加和减少发动机的推力。在推力增加和减少的过程中，调整发动机的基准值和当前值，模拟推力变化对发动机性能的影响：

```
// 增大推力
void increaseThrust(EngineData& data)
{
    data.V += 1; // 燃油流速增加1单位每秒

    if (isStable)
    {
        // 基准值提升3%到5%
        double incrementFactor = 1.03 + (rand() % 3) * 0.01;
        data.baseN1 *= incrementFactor;
        data.baseN2 *= incrementFactor;
        data.baseT1 *= incrementFactor;
        data.baseT2 *= incrementFactor;
        data.baseV += 1; // 根据需求调整
    }
    for (int i = 0; i < 2; ++i)
    {
        data.N1[i] = max(0.0, data.N1[i] + data.N1[i] * (0.03 + (rand() % 3) * 0.01)); // 转速增加3%到5%
        data.N2[i] = max(0.0, data.N2[i] + data.N2[i] * (0.03 + (rand() % 3) * 0.01));
        data.T1[i] = max(T0, data.T1[i] + data.T1[i] * (0.03 + (rand() % 3) * 0.01)); // 温度增加3%到5%
        data.T2[i] = max(T0, data.T2[i] + data.T2[i] * (0.03 + (rand() % 3) * 0.01));
    }
}
```

（4）故障检测与处理：

定义了 FaultType 枚举类型，列出了各种可能的故障类型：

```
//定义错误类型
enum FaultType
{   //第一种故障（单个）
    N1_1Fail, N1_2Fail, N2_1Fail, N2_2Fail, T1_1Fail, T1_2Fail, T2_1Fail, T2_2Fail,
    N1Fail, N2Fail, T1Fail, T2Fail,//第一种故障（单发）
    N1andN2Fail, T1andT2Fail,//第一种故障（双发）
    LowFuel, FuelSFail, OverFF,//第二种故障（燃油不足/流速）
    OverSpd1, OverSpd2,//第三种故障（转速过高）
    OverTemp1, OverTemp2, OverTemp3, OverTemp4//第四种故障（温度过高）
};
```

通过 checkData 函数，检测发动机数据中的异常情况，并根据不同的故障类型进行处理和记录 logFault 函数，用于记录故障信息，并保存到日志文件中。

2.数据保存：

对于**数据**：实现了 saveDataToFile 函数，将发动机数据保存到文件中。通过 ofstream 对象，将数据以 CSV 格式保存到文件中，便于后续分析。

对于**故障**：在 main 函数中打开了一个日志文件 fails.log，用于记录故障信息，通过 logFault 函数将故障信息写入日志文件。

```
// 将数据保存到表格文件
void saveDataToFile(const EngineData& data, double time, ofstream& file)
{
    file << time << ",";
    for (int i = 0; i < 2; ++i)
    {
        file << data.N1[i] << "," << data.N2[i] <<
                            "," << data.T1[i] << "," << data.T2[i] << ",";
    }
    file << data.C << "," << data.V << "\n";
}
```

3. 界面绘制：

使用 eaxyx 库来绘制界面，包括发动机数据的仪表盘、燃油量显示、推力控制按钮和故障显示等。

checkFaultColor 函数：根据故障类型返回相应的颜色，用于在界面上显示故障信息。drawFaultDisplay 函数，绘制故障显示区域，并根据故障类型设置相应的颜色和文本。

```
// 通过错误列表，判断N1 表盘颜色
COLORREF checkN1DataBoardColor(std::vector<FaultType> faults) {
    if (faults.empty()) {
        return GRAY;
    }
    else {
        for (FaultType fault : faults) {
            if (fault == OverSpd2) {
                return RED;
            }
            else if (fault == OverSpd1) {
                return AMBER;
            }
        }
        return GRAY;
    }
}
/*——故障显示——*/
COLORREF checkFaultColor(const std::vector<FaultType>& faults,
                         FaultType faultType) {
 if (std::find(faults.begin(), faults.end(), faultType) != faults.end()) {
        switch (faultType) {
        case OverSpd2:
        case OverTemp2:
```

```
            case OverTemp4:
            case FuelSFail:
            case N1andN2Fail:
            case T1andT2Fail:
                return RED;
            ....................................
            }
        }
        return DARKGRAY;
    }
```

4.鼠标点击处理：

handleMouseClick 函数用来处理用户的鼠标点击事件。根据不同的点击区域，执行相应的操作，如启动发动机、停止发动机、增加推力、减少推力和模拟故障等。

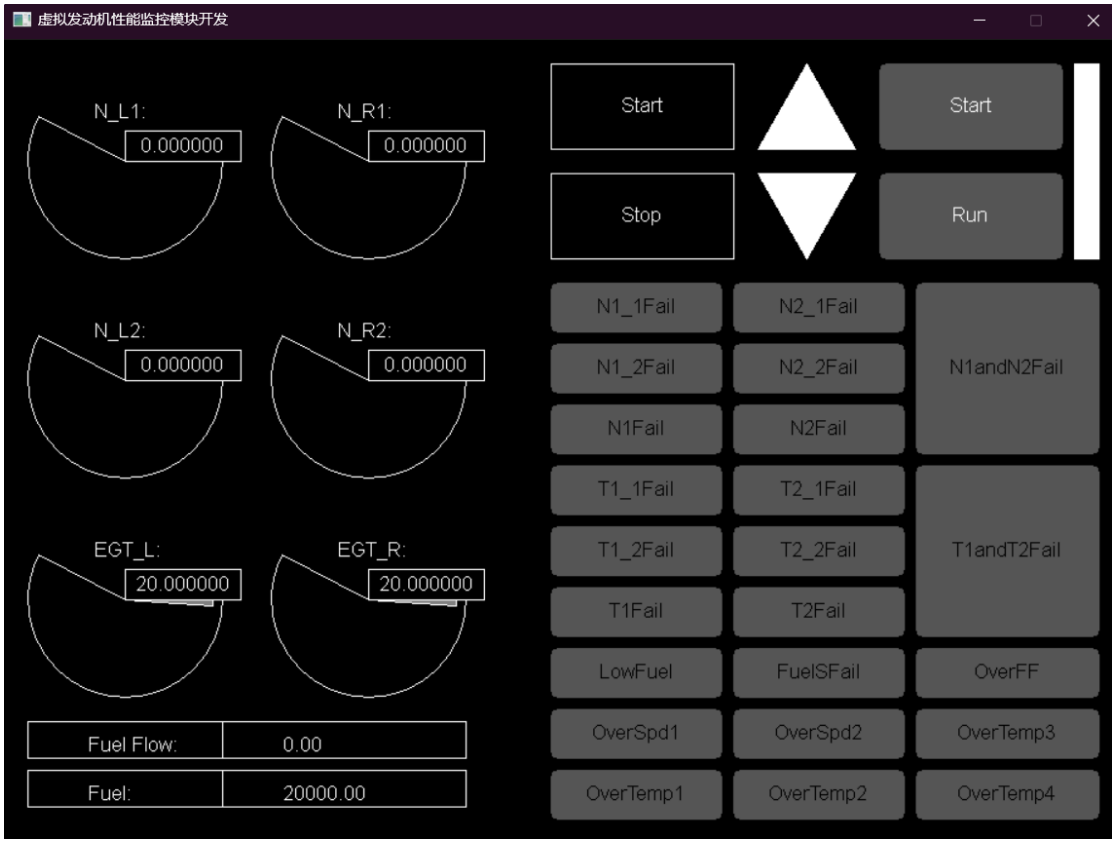
```
int handleMouseClick()
{
    MOUSEMSG m = GetMouseMsg();
    switch (m.uMsg)
    {
    case WM_LBUTTONDOWN: // 如果是左键按下
        // 检查是否点击了 Start 按钮
        if (m.x >= 450 && m.x <= 600 && m.y >= 20 && m.y <= 90)
        {
            return 1; // 返回命令值 1
        }
        // 检查是否点击了 Stop 按钮
        if (m.x >= 450 && m.x <= 600 && m.y >= 110 && m.y <= 180)
        {
            return 2; // 返回命令值 2
        }
        // 检查是否点击了增加推力按钮（上三角）
        if (m.x >= 620 && m.x <= 700 && m.y >= 20 && m.y <= 90)
        {
            return 3; // 返回命令值 3
        }
        // 检查是否点击了减小推力按钮（下三角）
        if (m.x >= 620 && m.x <= 700 && m.y >= 110 && m.y <= 180)
        {
            return 4; // 返回命令值 4
        }

        //检查每一个故障按钮：
        // 11-17.21-27.31-33.41-42，51-54分别对应故障类型一（N）、一（T）、二、三、四
        if (m.x >= 450 && m.x <= 590 && m.y >= 200 && m.y <= 240)
        {
            return 11;
        }
        if (m.x >= 450 && m.x <= 590 && m.y >= 250 && m.y <= 290)
        {
            return 12;
        }
        if (m.x >= 450 && m.x <= 590 && m.y >= 300 && m.y <= 340)
        {
```

5.main 函数：

Main 函数主要来综合以上部分，实现用户交互（文末附全部代码）。

(二)功能概述

主界面如下：



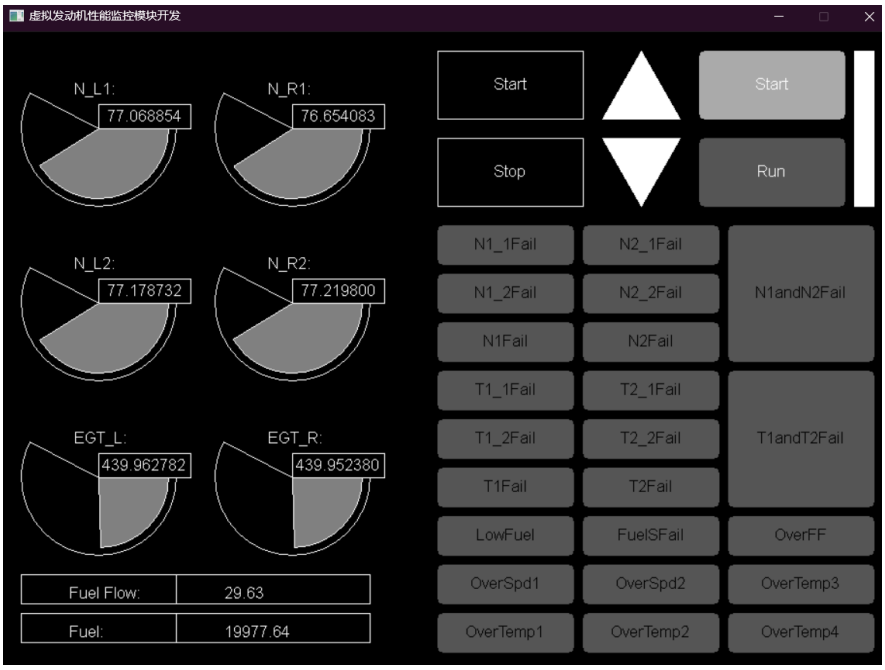
界面左侧仪表盘对应左右两发动机的四个传感器（出现故障传感器变色），左下方显示当前流量与剩余油量；界面右侧上部分别为启动按钮、停止按钮、增加推力按钮、减少推力按钮、状态显示框与油箱对应的油量条，右侧下部显示了共 14 种 7+7+3+2+4=23 个异常情况的显示栏，可以判断目前出现的异常情况，并作为按钮可以触发模拟该种异常情况。

1.启动与停止，加速与减速

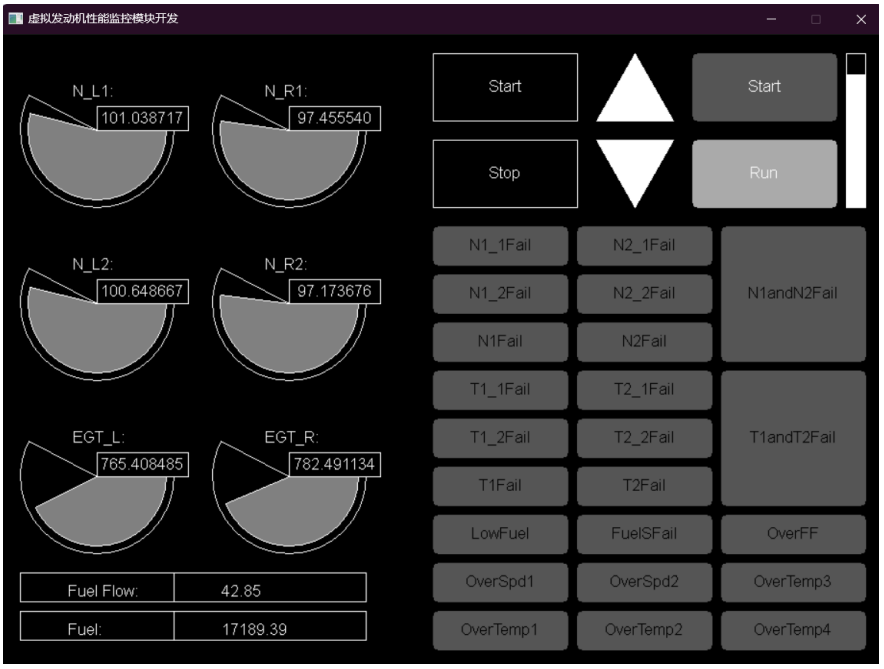
Start 按钮按下后，将按照上述的启动规则进行数据生成。Stop 按钮按下后，将按照上述的停车规则进行数据生成。Stop 按钮优先级高于其他所有按钮（包括加分项的按钮）。（如图）

点击上三角/下三角后，转速比 N,温度 T 在 3%~5%的范围内随机加/减。

状态显示框包括启动 start 和稳态 run，亮暗规则如下：①Start 按钮按下后，程序正常启动，start 窗口亮起。②达稳态后，start 窗口变暗，run 窗口亮起。③此后，若 N1 下降至 95 以下，则 run 窗口变暗。回升后则再次亮起。



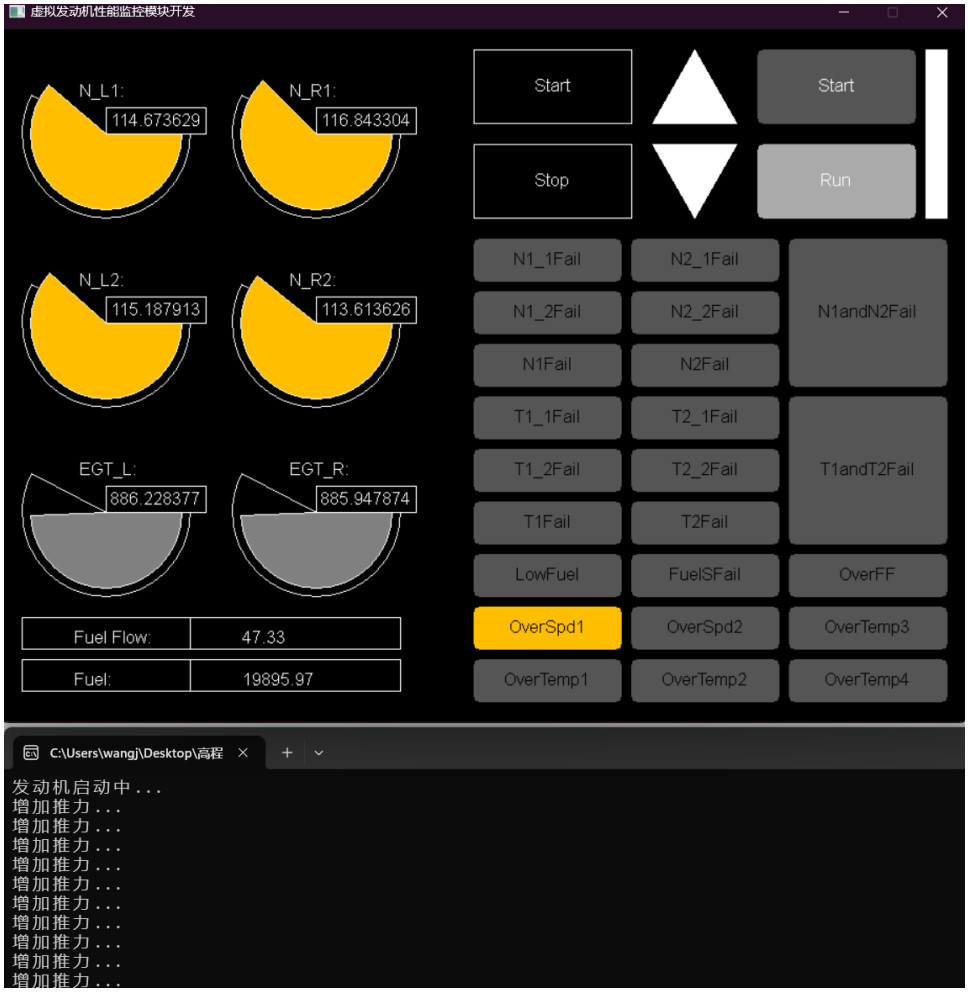
（刚开始）



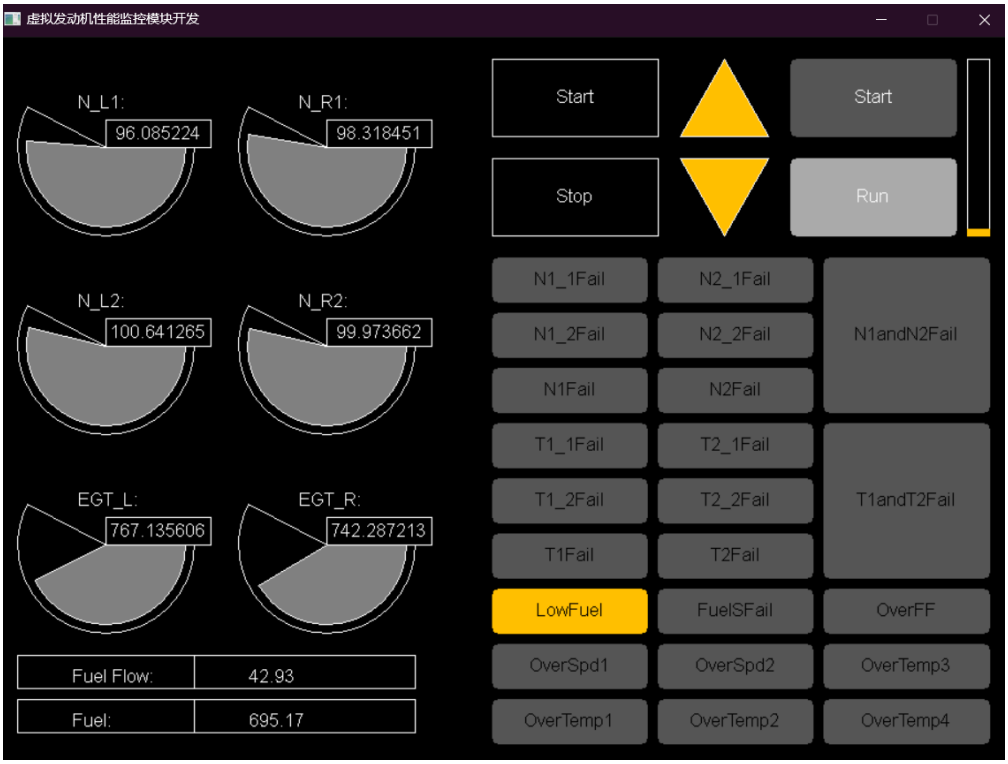
（达稳态）

2. 【有关故障显示】

在本程序中，如果出现故障，那么会亮起对应的灯（白色、琥珀色、红色），同时和题目中要求的警示持续五秒相同，即将显示文字的描述方式变成了亮灯显示，使得界面更加直观。（如图所示，一直提高推力，出现了 OverSpd1 提醒，如果一直 OverSpd，那么会一直显示，直到到达正常值后五秒才停止点亮故障模块）。

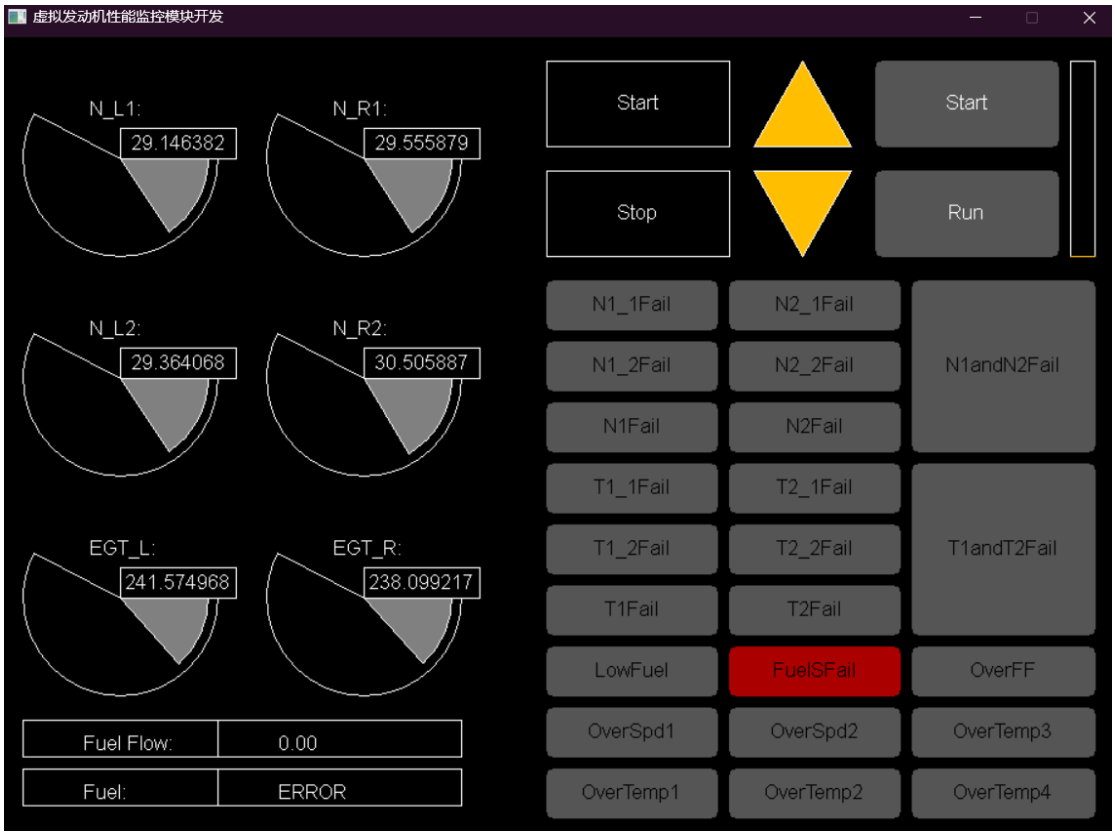


下图显示低油量警告，右上角的油量条变黄：

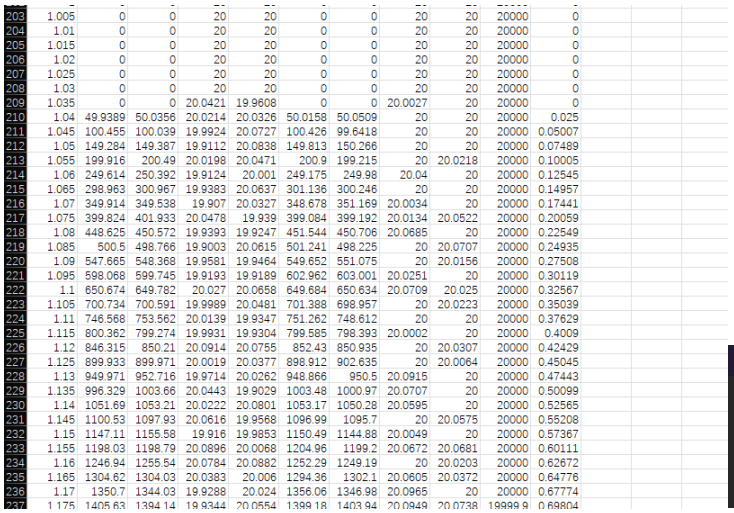


3. 异常测试功能

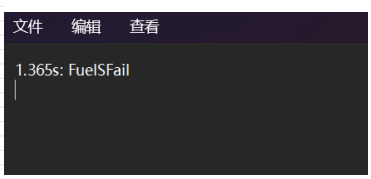
在本程序中，可以自行点击右侧的异常框（没错，二合一!），程序可以产生 14 种异常情况的数据，且能够检测出这些异常情况 （如点击油量告终，程序正常执行，油量显示为 ERROR 代表没油了!）



4. 在关闭程序以后，可以浏览最近一次生成的表格（逐 5ms 各个参数的记录）与日志（故障记录）：



203	1.005	0	0	20	20	0	0	20	20	20000	0
204	1.01	0	0	20	20	0	0	20	20	20000	0
205	1.015	0	0	20	20	0	0	20	20	20000	0
206	1.02	0	0	20	20	0	0	20	20	20000	0
207	1.025	0	0	20	20	0	0	20	20	20000	0
208	1.03	0	0	20	20	0	0	20	20	20000	0
209	1.035	0	0	20.0421	19.9608	0	0	20.0027	20	20000	0
210	1.04	49.9389	50.0356	20.0214	20.0326	50.0158	50.0509	20	20	20000	0.025
211	1.045	100.455	100.039	19.9924	20.0727	100.426	99.6418	20	20	20000	0.05007
212	1.05	149.284	149.387	19.9112	20.0838	149.813	150.266	20	20	20000	0.07489
213	1.055	199.916	200.49	20.0198	20.0471	200.9	199.215	20	20.0218	20000	0.10005
214	1.06	249.614	250.392	19.9124	20.001	249.175	249.98	20.04	20	20000	0.12545
215	1.065	298.963	300.967	19.9383	20.0637	301.136	300.246	20	20	20000	0.14957
216	1.07	349.914	349.538	19.907	20.0327	348.678	351.169	20.0034	20	20000	0.17441
217	1.075	399.824	401.933	20.0478	19.939	399.084	399.192	20.0134	20.0522	20000	0.20059
218	1.08	448.625	450.572	19.9393	19.9247	451.544	450.706	20.0685	20	20000	0.22549
219	1.085	500.5	498.766	19.9003	20.0615	501.241	498.225	20	20.0707	20000	0.24935
220	1.09	547.665	548.368	19.9581	19.9464	549.652	551.075	20	20.0156	20000	0.27508
221	1.095	598.068	599.745	19.9193	19.9189	602.962	603.001	20.0251	20	20000	0.30119
222	1.1	650.674	649.782	20.027	20.0658	649.684	650.634	20.0709	20.025	20000	0.32567
223	1.105	700.734	700.591	19.9989	20.0481	701.388	698.957	20	20.0223	20000	0.35039
224	1.11	746.568	753.562	20.0139	19.9347	751.262	748.612	20	20	20000	0.37629
225	1.115	800.362	799.274	19.9931	19.9304	799.585	798.393	20.0002	20	20000	0.4009
226	1.12	846.315	850.21	20.0914	20.0755	852.43	850.935	20	20.0307	20000	0.42429
227	1.125	899.933	899.971	20.0019	20.0377	898.912	902.635	20	20.0064	20000	0.45045
228	1.13	949.971	952.716	19.9714	20.0262	948.866	950.5	20.0915	20	20000	0.47443
229	1.135	996.329	1003.66	20.0443	19.9029	1003.48	1000.97	20.0707	20	20000	0.50099
230	1.14	1051.69	1053.21	20.0222	20.0801	1053.17	1050.28	20.0595	20	20000	0.52565
231	1.145	1100.53	1097.93	20.0616	19.9568	1096.99	1095.7	20	20.0575	20000	0.55208
232	1.15	1147.11	1155.58	19.916	19.9853	1150.49	1144.88	20.0049	20	20000	0.57367
233	1.155	1198.03	1198.79	20.0896	20.0068	1204.96	1199.2	20.0672	20.0681	20000	0.60111
234	1.16	1246.94	1255.54	20.0784	20.0882	1252.29	1249.19	20	20.0203	20000	0.62672
235	1.165	1304.62	1304.03	20.0383	20.006	1294.36	1302.1	20.0605	20.0372	20000	0.64776
236	1.17	1350.7	1344.03	19.9288	20.024	1356.06	1346.98	20.0965	20	20000	0.67774
237	1.175	1405.63	1394.14	19.9344	20.0554	1399.18	1403.94	20.0949	20.0738	19999.9	0.69804



（第一行给出了每列的定义）

二、问题与解决方法

问题 1：随机数如何生成？

答：可以使用 random 库的现成函数

```
// 生成随机噪声，用于模拟数据的随机波动
double generateRandomNoise(double range)
{
    return ((rand() / (double)RAND_MAX) * 2 - 1) * range;
}
```

无论是增大减小推力的增减 3-5%，还是小范围的波动都可以套用该函数。

问题 2：如何处理、判断各种各样的故障类型，并利用故障类型去生成数值？

答：定义 enum FaultType 可更清晰和结构化地管理和处理各种可能的故障类型：

①可读性： 使用枚举类型可以使代码更具可读性。例如，logFault(N1_1Fail)比 logFault(0)更容易理解。

②可维护性： 枚举类型可以集中管理所有的故障类型，方便将来添加或修改故障类型。例如，如果需要添加新的故障类型，只需在枚举中添加一个新的枚举值。

③类型安全： 枚举类型提供了类型安全性，避免了将不相关的值传递给函数。例如，logFault(N1_1Fail)确保传递的是一个有效的故障类型，而不是一个无效的整数值。

这样的话，如果监测到故障，就可以把记录加入到 faults 向量中，并能够记录故障发生的时间（例如下方代码）

```
auto logFault = [&](FaultType fault) {
    if (currentTime - faultLogTimeMap[fault] >= 5.0) {
        faults.push_back(fault);
        faultTimeMap[fault] = currentTime;
        logFile << currentTime << "s: " << fault << endl;
        faultLogTimeMap[fault] = currentTime;
    }
};
```

显示故障也比较方便：

```
if (!sensorN1_0 && !sensorN1_1 && !sensorN2_0 && !sensorN2_1)
{
    isStopping = true;
    isStarted_1 = false;
    isStarted_2 = false;
```

```
        isStable = false;
        logFault(N1andN2Fail);
    }
```

不过需要注意的是每个周期（0.05s）都需要去刷新故障向量，以免影响其他时刻。

问题 3：如何做到日志保存与数据的表格保存？

答：

日志保存：在 main 函数中打开文档

```
    ofstream logFile("fails.log");
    if (!logFile.is_open())
    {
        cout << "无法打开日志文件！" << endl;
        return -1;
    }
```

后面根据第二问的回答可以判断故障并记录

表格数据保存：打开文档并填写表头

```
    ofstream dataFile("engine_data.csv");
    if (!dataFile.is_open())
    {
        cout << "无法打开数据文件！" << endl;
        return -1;
    }
      dataFile << "Time,N1_1,N1_2,N2_1,N2_2,T1_1,T1_2,T2_1,T2_2,C,V\n";
```

将数据保存到文件：

```
    void saveDataToFile(const EngineData& data, double time, ofstream& f
ile)
    {
        file << time << ",";
        for (int i = 0; i < 2; ++i)
        {
            file << data.N1[i] << "," << data.N2[i] << "," << data.T1[i]
 << "," << data.T2[i] << ",";
        }
        file << data.C << "," << data.V << "\n";
    }
```

注意定期保存数据：

```
    saveDataToFile(engineData, totalTime, dataFile);
```

在程序结束时，你需要关闭日志文件和数据文件即可。

~~不过我不知道为什么日志文件保存时有的内容是以我自己设定的代码展示的~~

问题 4：如何实现相同故障数据每 5 秒只记录一次？

答：

使用时间戳记录故障发生时间

```
std::map<FaultType, double> faultTimeMap;
std::map<FaultType, double> faultLogTimeMap;
```

在更新故障列表时，判断相差有没有五秒，超过五秒移除异常知道下一个 5ms 继续判断是否异常：

```
auto it = faultTimeMap.begin();
while (it != faultTimeMap.end())
{
    if (currentTime - it->second > 5.0)
    {
        // 故障已超过5 秒，移除
        it = faultTimeMap.erase(it);
    }
    else
    {
        ++it;
    }
}
// 清空 faults 列表，重新添加未过期的故障
faults.clear();
for (const auto& pair : faultTimeMap)
{
    faults.push_back(pair.first);
}
```

问题 5：卡顿怎么办？设定的点亮五秒，然后就被狠狠的延长了——

答：双缓冲的实现步骤

1. 创建两个缓冲区：一个绘制图形（后台缓冲区），一个显示图形（前台缓冲区）

2. 在后台缓冲区中绘制图形：所有的绘制操作都在后台缓冲区中进行。

3. 将后台缓冲区的内容复制到前台缓冲区：绘制完成后，将后台缓冲区的内容复制到前台缓冲区进行显示。

不够卡顿可能是因为程序是~~屎山~~吧，各种各样库的使用确实降低了程序运行的速度。

三、智慧编程

智慧编程真的帮了我大忙!前面列举的问题绝大多数都是交给 copilot 老师解决的，github 的 copilot 也在升级更新中，o1-preview 用着感觉真的好厉害!

同时，一些重复的~~屎山~~代码也交给了 copilot，这样达到了举一反三事半功倍的效果~~，敲击 tab 敲到爽~~。

四、心得体会

再回顾下题目的引子吧：

《最后一程（One Last Program)》

第一次看到这道题时

并没有什么特别的感觉

因为独属于我的高程体验

我早已遇见

初次遇见你的那天

齿轮开始转动

无法停止那将要失去什么的预感

虽说已经有很多次了

让我们再来一道题吧

Can you give me one last program?

不想遗忘之事

不愿遗忘之事

I love programming than you'll ever know

可曾记得 追寻着 AC 的

那些灯火通明的夜晚

断断续续写的代码，总用时 30+h，前期摆烂后期突击，呈现了 1800 余行代码，高程期末大作业，虚拟发动机性能监控模块开发就这么（大致）完成了!

自此，高程这门课也总算是结束了!

从上学期的矩阵运算/2048/孔明棋,再到这学期的字符压缩/钟表设计/贪吃蛇游戏与此次作业虚拟发动机性能监控模块开发，

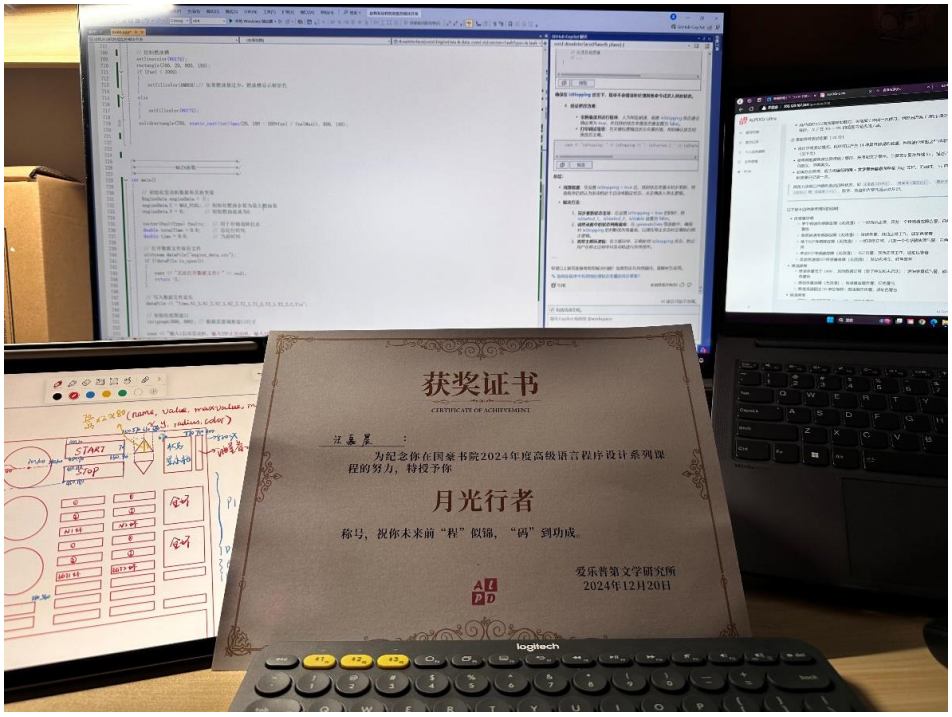
从 Hello World，到 if while 的判断，数组字符串和指针的折磨，再到类和对象的数据结构，重载继承虚函数的高深莫测，我作为一个初学者，已然从 C 风格时代的原始走入了 OOP 时代的先进。

作为我人生中学到的系统性的第一门编程语言，C++带给我折磨，也给我折磨后的快乐。

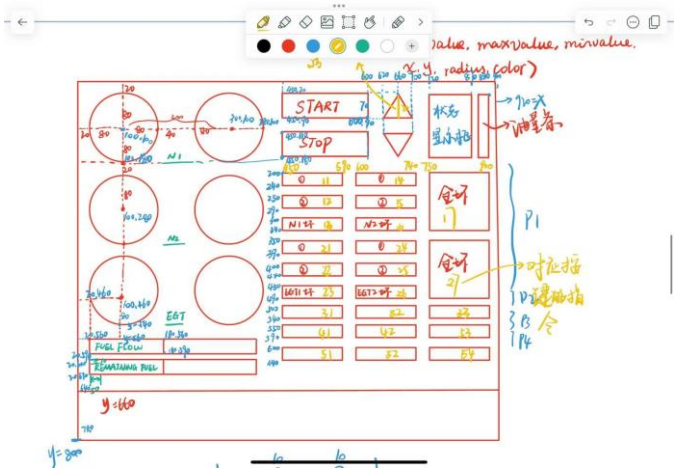
可曾记得 追寻着 AC 的那些灯火通明的夜晚——

感谢陈老师、龚老师与各位助教老师，感谢群内各路大神与细心帮助我的 uu 们！

祝好！



“月光行者”——12.27 01：23



设计花絮

四、源代码

组织结构

```
#include<iostream>
#include<random>
#include <fstream>
#include <cmath>
#include <chrono>
#include <thread>
#include <cstdlib>
#include <ctime>
#include <graphics.h>
#include <conio.h>
#include <vector>
#include <string>
#include <map>
using namespace std;
const double M_PI = 3.14;
/*————————————————*/
/*————————逻辑处理————————*/
/*————————————————*/
// 定义时间间隔：5ms
const int INTERVAL_MS = 5;
// 定义最大燃油量
const int MAX_FUEL = 20000;
// 定义额定转速
const int RATED_RPM = 40000;
// 环境温度（初始温度）
const double T0 = 20.0;
// 定义发动机各种阶段（初始阶段，启动阶段，稳定阶段，停车阶段）
bool isInit = true;
bool isStarted_1 = false; // 启动前两秒
bool isStarted_2 = false; // 启动对数函数
bool isStable = false;
bool isStopping = false;
// 结构体用于存储发动机数据
struct EngineData
{
    double N1[2]; // 左发动机转速（两个传感器）
    double N2[2]; // 右发动机转速（两个传感器）
    double T1[2]; // 左发动机排气温度（两个传感器）
    double T2[2]; // 右发动机排气温度（两个传感器）
    double C;     // 燃油余量
    double V;     // 燃油流速
    // 基准值，允许在稳定状态下调整
    double baseN1;
    double baseN2;
    double baseT1;
    double baseT2;
    double baseV;
    EngineData() :
        baseN1(0.95 * RATED_RPM),
        baseN2(0.95 * RATED_RPM),
        baseT1(735),
        baseT2(735),
        baseV(43)
    {
    }
```

```
};
// 生成随机噪声，用于模拟数据的随机波动
double generateRandomNoise(double range)
{
    return ((rand() / (double)RAND_MAX) * 2 - 1) * range;
}
// 增大推力
void increaseThrust(EngineData& data)
{
    data.V += 1; // 燃油流速增加 1 单位每秒
    if (isStable)
    {
        // 基准值提升 3%到 5%
        double incrementFactor = 1.03 + (rand() % 3) * 0.01;
        data.baseN1 *= incrementFactor;
        data.baseN2 *= incrementFactor;
        data.baseT1 *= incrementFactor;
        data.baseT2 *= incrementFactor;
        data.baseV += 1; // 根据需求调整
    }
    for (int i = 0; i < 2; ++i)
    {
        data.N1[i] = max(0.0, data.N1[i] + data.N1[i] * (0.03 + (rand() % 3) * 0.01
)); // 转速增加 3%到 5%
        data.N2[i] = max(0.0, data.N2[i] + data.N2[i] * (0.03 + (rand() % 3) * 0.01
));
        data.T1[i] = max(T0, data.T1[i] + data.T1[i] * (0.03 + (rand() % 3) * 0.01)
); // 温度增加 3%到 5%
        data.T2[i] = max(T0, data.T2[i] + data.T2[i] * (0.03 + (rand() % 3) * 0.01)
);
    }
}
// 减小推力
void decreaseThrust(EngineData& data)
{
    if (isStable)
    {
        // 基准值降低 3%到 5%
        double decrementFactor = 1.00 - (0.03 + (rand() % 3) * 0.01);
        data.baseN1 *= decrementFactor;
        data.baseN2 *= decrementFactor;
        data.baseT1 *= decrementFactor;
        data.baseT2 *= decrementFactor;
        data.V = max(0.0, data.V - 1); // 燃油流速减少 1 单位每秒，不能小于 0
    }
    for (int i = 0; i < 2; ++i)
    {
        data.N1[i] = max(0.0, data.N1[i] - data.N1[i] * (0.03 + (rand() % 3) * 0.01
)); // 转速减少 3%到 5%
        data.N2[i] = max(0.0, data.N2[i] - data.N2[i] * (0.03 + (rand() % 3) * 0.01
));
        data.T1[i] = max(T0, data.T1[i] - data.T1[i] * (0.03 + (rand() % 3) * 0.01)
); // 温度减少 3%到 5%
        data.T2[i] = max(T0, data.T2[i] - data.T2[i] * (0.03 + (rand() % 3) * 0.01)
);
    }
}
//定义错误类型
enum FaultType
{
    N1_1Fail, N1_2Fail, N2_1Fail, N2_2Fail, T1_1Fail, T1_2Fail, T2_1Fail, T2_2Fail,
//第一种故障（单个）
    N1Fail, N2Fail, T1Fail, T2Fail,//第一种故障（单发）
```

```
    N1andN2Fail, T1andT2Fail,//第一种故障（双发）
    LowFuel, FuelSFail, OverFF,//第二种故障（燃油不足/流速）
    OverSpd1, OverSpd2,//第三种故障（转速过高）
    OverTemp1, OverTemp2, OverTemp3, OverTemp4//第四种故障（温度过高）
};
//异常情况检测与处理
void checkData(EngineData& data, bool& isInit, bool& isStarted_1, bool& isStarted_2
, bool& isStable, bool& isStopping,
    vector<FaultType>& faults, std::map<FaultType, double>& faultTimeMap, double cu
rrentTime, ofstream& logFile, std::map<FaultType, double>& faultLogTimeMap)
{
    /*————传感器异常————*/
    bool sensorN1_0 = true, sensorN1_1 = true, sensorN2_0 = true, sensorN2_1 = true
; //假设都是正常的
    bool sensorT1_0 = true, sensorT1_1 = true, sensorT2_0 = true, sensorT2_1 = true
; //假设都是正常的
    if (data.N1[0] > 1.25 * RATED_RPM)
        sensorN1_1 = false;
    if (data.N1[1] > 1.25 * RATED_RPM)
        sensorN1_1 = false;
    if (data.N2[0] > 1.25 * RATED_RPM)
        sensorN2_0 = false;
    if (data.N2[1] > 1.25 * RATED_RPM)
        sensorN2_1 = false;
    if (data.T1[0] > 1200 || data.T1[0] < -5)
        sensorT1_0 = false;
    if (data.T1[1] > 1200 || data.T1[1] < -5)
        sensorT1_1 = false;
    if (data.T2[0] > 1200 || data.T2[0] < -5)
        sensorT2_0 = false;
    if (data.T2[1] > 1200 || data.T2[1] < -5)
        sensorT2_1 = false;
    auto logFault = [&](FaultType fault) {
        if (currentTime - faultLogTimeMap[fault] >= 5.0) {
            faults.push_back(fault);
            faultTimeMap[fault] = currentTime;
            logFile << currentTime << "s: " << fault << endl;
            faultLogTimeMap[fault] = currentTime;
        }
        };
    // 单个转速传感器故障（无效值）
    // 单发转速传感器故障（无效值）
    // 双发转速传感器故障（无效值）
    if (!sensorN1_0 && !sensorN1_1 && !sensorN2_0 && !sensorN2_1)
    {
        // 红色警告
        isStopping = true;
        isStarted_1 = false;
        isStarted_2 = false;
        isStable = false;
        logFault(N1andN2Fail);
    }
    else if (!sensorN1_0 && !sensorN1_1)
    {
        // 琥珀色警告
        logFault(N1Fail);
    }
    else if (!sensorN2_0 && !sensorN2_1)
    {
        // 琥珀色警告
        logFault(N2Fail);
    }
    else if (!sensorN1_0)
```

```
    {
        //白色警告
        logFault(N1_1Fail);
    }
    else if (!sensorN1_1)
    {
        //白色警告
        logFault(N1_2Fail);
    }
    else if (!sensorN2_0)
    {
        //白色警告
        logFault(N2_1Fail);
    }
    else if (!sensorN2_1)
    {
        //白色警告
        logFault(N2_2Fail);
    }
    // 单个 EGT 传感器故障（无效值）
    // 单发 EGT 传感器故障（无效值）
    // 双发 EGT 传感器故障（无效值）
    if (!sensorT1_0 && !sensorT1_1 && !sensorT2_0 && !sensorT2_1)
    {
        // 红色警告
        isStopping = true;
        isStarted_1 = false;
        isStarted_2 = false;
        isStable = false;
        logFault(T1andT2Fail);
    }
    else if (!sensorT1_0 && !sensorT1_1)
    {
        // 琥珀色警告
        logFault(T1Fail);
    }
    else if (!sensorT2_0 && !sensorT2_1)
    {
        // 琥珀色警告
        logFault(T2Fail);
    }
    else if (!sensorT1_0)
    {
        //白色警告
        logFault(T1_1Fail);
    }
    else if (!sensorT1_1)
    {
        //白色警告
        logFault(T1_2Fail);
    }
    else if (!sensorT2_0)
    {
        //白色警告
        logFault(T2_1Fail);
    }
    else if (!sensorT2_1)
    {
        //白色警告
        logFault(T2_2Fail);
    }
    /*————燃油异常————*/
    if (data.C <= 1000 && data.C > 0)
```

```
    {
        // 燃油余量低告警
        // 琥珀色警告
        logFault(LowFuel);
    }
    else if (data.C <= 0)
    {
        // 传感器故障告警
        // 红色警告
        isStopping = true;
        isStarted_1 = false;
        isStarted_2 = false;
        isStable = false;
        data.C = 0;
        logFault(FuelSFail);
    }
    if (data.V > 50)
    {
        // 燃油指示告警
        // 琥珀色警告
        logFault(OverFF);
    }
    /*————转速异常（平均值代替）————*/
    if (((data.N1[0] + data.N1[0] + data.N2[0] + data.N1[1]) / 4) > 1.05 * RATED_RP
M && ((data.N1[0] + data.N1[0] + data.N2[0] + data.N1[1]) / 4) <= 1.20 * RATED_RPM)
    {
        //琥珀色警告
        logFault(OverSpd1);
    }
    else if (((data.N1[0] + data.N1[0] + data.N2[0] + data.N1[1]) / 4) > 1.20 * RAT
ED_RPM)
    {
        //红色警告
        isStopping = true;
        isStarted_1 = false;
        isStarted_2 = false;
        isStable = false;
        logFault(OverSpd2);
    }
    /*————温度异常（平均值代替）————*/
    //启动过程中
    if (isStarted_1 || isStarted_2)
    {
        if (((data.T1[0] + data.T1[1] + data.T2[0] + data.T2[1]) / 4) > 1000)
        {
            //红色警告
            isStopping = true;
            isStarted_1 = false;
            isStarted_2 = false;
            isStable = false;
            logFault(OverTemp2);
        }
        else if (((data.T1[0] + data.T1[1] + data.T2[0] + data.T2[1]) / 4) > 850 &&
 ((data.T1[0] + data.T1[1] + data.T2[0] + data.T2[1]) / 4) <= 1000)
        {
            //琥珀色警告
            logFault(OverTemp1);
        }
    }
    //稳态中
    else if (isStable)
    {
        if (((data.T1[0] + data.T1[1] + data.T2[0] + data.T2[0]) / 4) > 1100)
```

```
        {
            //红色警告
            isStopping = true;
            isStarted_1 = false;
            isStarted_2 = false;
            isStable = false;
            logFault(OverTemp4);
        }
        else if (((data.T1[0] + data.T1[1] + data.T2[0] + data.T2[0]) / 4) > 950 &&
 ((data.T1[0] + data.T1[1] + data.T2[0] + data.T2[0]) / 4) <= 1100)
        {
            //琥珀色警告
            logFault(OverTemp3);
        }
    }
}
// 根据时间生成发动机数据（可以有小范围的浮动）
void generateData(EngineData& data, bool& isInit, bool& isStarted_1, bool& isStarte
d_2, bool& isStable, bool& isStopping, const vector<FaultType>& faults)
{
    static double phaseTime = 0.0; // 用于跟踪当前阶段的时间
    if (isInit)
    {
        // 初始化阶段，数据为0
        for (int i = 0; i < 2; ++i)
        {
            data.N1[i] = 0;
            data.N2[i] = 0;
            data.T1[i] = T0;
            data.T2[i] = T0;
        }
        phaseTime = 0.0; // 重置阶段时间
        // 定义基准值
        double baseN1 = 0.95 * RATED_RPM; // 稳定阶段转速为额定转速的95%
        double baseN2 = 0.95 * RATED_RPM;
        double baseT1 = 735;
        double baseT2 = 735;
        double baseV = 43;//根据数据设置
    }
    else if (isStarted_1)
    {
        // 启动阶段1，线性增加
        for (int i = 0; i < 2; ++i)
        {

            if (std::find(faults.begin(), faults.end(), N1_1Fail) == faults.end() &
& std::find(faults.begin(), faults.end(), N1Fail) == faults.end())
            {
                data.N1[0] = 10000 * phaseTime + generateRandomNoise(data.N1[0] * 0
.005);
            }
            else
            {
                data.N1[0] = 0;
            }
            if (std::find(faults.begin(), faults.end(), N1_2Fail) == faults.end() &
& std::find(faults.begin(), faults.end(), N1Fail) == faults.end())
            {
                data.N1[1] = 10000 * phaseTime + generateRandomNoise(data.N1[1] * 0
.005);
            }
            else
            {
```

```
                data.N1[1] = 0;
            }
            if (std::find(faults.begin(), faults.end(), N2_1Fail) == faults.end() &
& std::find(faults.begin(), faults.end(), N2Fail) == faults.end())
            {
                data.N2[0] = 10000 * phaseTime + generateRandomNoise(data.N2[0] * 0
.005);
            }
            else
            {
                data.N2[0] = 0;
            }
            if (std::find(faults.begin(), faults.end(), N2_2Fail) == faults.end() &
& std::find(faults.begin(), faults.end(), N2Fail) == faults.end())
            {
                data.N2[1] = 10000 * phaseTime + generateRandomNoise(data.N2[1] * 0
.005);
            }
            else
            {
                data.N2[1] = 0;
            }
            if (std::find(faults.begin(), faults.end(), T1_1Fail) == faults.end() &
& std::find(faults.begin(), faults.end(), T1Fail) == faults.end())
            {
                if (std::find(faults.begin(), faults.end(), OverTemp3) != faults.en
d())
                {
                    data.T1[0] = 1000;
                }
                else if (std::find(faults.begin(), faults.end(), OverTemp1) != faul
ts.end())
                {
                    data.T1[0] = 900;
                }
                else
                {
                    data.T1[0] = T0 + generateRandomNoise(data.T1[0] * 0.005);
                }
            }
            else
            {
                data.T1[0] = T0;
            }
   if (std::find(faults.begin(), faults.end(), T1_2Fail) == faults.end() && std::fi
nd(faults.begin(), faults.end(), T1Fail) == faults.end())
   {
    if (std::find(faults.begin(), faults.end(), OverTemp3) != faults.end())
    {
     data.T1[1] = 1000;
    }
    else if (std::find(faults.begin(), faults.end(), OverTemp1) != faults.end())
    {
     data.T1[1] = 900;
    }
    else
    {
     data.T1[1] = T0 + generateRandomNoise(data.T1[1] * 0.005);
    }
   }
   else
   {
    data.T1[1] = T0;
```

```
   }
            if (std::find(faults.begin(), faults.end(), T2_1Fail) == faults.end() &
& std::find(faults.begin(), faults.end(), T2Fail) == faults.end())
            {
    if (std::find(faults.begin(), faults.end(), OverTemp3) != faults.end())
    {
     data.T2[0] = 1000;
    }
    else if (std::find(faults.begin(), faults.end(), OverTemp1) != faults.end())
    {
     data.T2[0] = 900;
    }
    else
    {
     data.T2[0] = T0 + generateRandomNoise(data.T2[0] * 0.005);
    }
   }
            else
            {
                data.T2[0] = T0;
            }
   if (std::find(faults.begin(), faults.end(), T2_2Fail) == faults.end() && std::fi
nd(faults.begin(), faults.end(), T2Fail) == faults.end())
   {
    if (std::find(faults.begin(), faults.end(), OverTemp3) != faults.end())
    {
     data.T2[1] = 1000;
    }
    else if (std::find(faults.begin(), faults.end(), OverTemp1) != faults.end())
    {
     data.T2[1] = 900;
    }
    else
    {
     data.T2[1] = T0 + generateRandomNoise(data.T2[1] * 0.005);
    }
   }
            //不可能出现小于0的情况
            data.N1[i] = max(0.0, data.N1[i]);
            data.N2[i] = max(0.0, data.N2[i]);
            data.T1[i] = max(T0, data.T1[i]);
            data.T2[i] = max(T0, data.T2[i]);
        }
        data.V = 5 * phaseTime + generateRandomNoise(data.V * 0.005);
        if (phaseTime >= 2.0)
        {
            isStarted_1 = false;
            isStarted_2 = true;
            phaseTime = 0.0; // 重置阶段时间
        }
        phaseTime += INTERVAL_MS / 1000.0; // 增加阶段时间
    }
    else if (isStarted_2)
    {
        // 启动阶段2，对数函数增加
        for (int i = 0; i < 2; ++i)
        {
            //阶段时间重置，所以函数需要更改
            if (std::find(faults.begin(), faults.end(), N1_1Fail) == faults.end() &
& std::find(faults.begin(), faults.end(), N1Fail) == faults.end())
            {
                data.N1[0] = (23000 * log(phaseTime + 1) + 20000) + generateRandomN
oise(data.N1[0] * 0.005);
```

```
            }
            else
            {
                data.N1[0] = 0;
            }
            if (std::find(faults.begin(), faults.end(), N1_2Fail) == faults.end() &
& std::find(faults.begin(), faults.end(), N1Fail) == faults.end())
            {
                data.N1[1] = (23000 * log(phaseTime + 1) + 20000) + generateRandomN
oise(data.N1[1] * 0.005);
            }
            else
            {
                data.N1[1] = 0;
            }
            if (std::find(faults.begin(), faults.end(), N2_1Fail) == faults.end() &
& std::find(faults.begin(), faults.end(), N2Fail) == faults.end())
            {
                data.N2[0] = (23000 * log(phaseTime + 1) + 20000) + generateRandomN
oise(data.N2[0] * 0.005);
            }
            else
            {
                data.N2[0] = 0;
            }
            if (std::find(faults.begin(), faults.end(), N2_2Fail) == faults.end() &
& std::find(faults.begin(), faults.end(), N2Fail) == faults.end())
            {
                data.N2[1] = (23000 * log(phaseTime + 1) + 20000) + generateRandomN
oise(data.N2[1] * 0.005);
            }
            else
            {
                data.N2[1] = 0;
            }
            if (std::find(faults.begin(), faults.end(), T1_1Fail) == faults.end() &
& std::find(faults.begin(), faults.end(), T1Fail) == faults.end())
            {
                if (std::find(faults.begin(), faults.end(), OverTemp4) != faults.en
d())
                {
                    data.T1[0] = 1200;
                }
                else if (std::find(faults.begin(), faults.end(), OverTemp3) != faul
ts.end())
                {
                    data.T1[0] = 1000;
                }
                else if (std::find(faults.begin(), faults.end(), OverTemp2) != faul
ts.end())
                {
                    data.T1[0] = 1100;
                }
                else if (std::find(faults.begin(), faults.end(), OverTemp1) != faul
ts.end())
                {
                    data.T1[0] = 900;
                }
                else
                {
                    data.T1[0] = 900 * log(phaseTime + 1) + T0 + generateRandomNois
e(data.T1[0] * 0.005);
                }
```

```
            }
            else
            {
                data.T1[0] = T0;
            }
   if (std::find(faults.begin(), faults.end(), T1_2Fail) == faults.end() && std::fi
nd(faults.begin(), faults.end(), T1Fail) == faults.end())
   {
    if (std::find(faults.begin(), faults.end(), OverTemp4) != faults.end())
    {
     data.T1[1] = 1200;
    }
    else if (std::find(faults.begin(), faults.end(), OverTemp3) != faults.end())
    {
     data.T1[1] = 1000;
    }
    else if (std::find(faults.begin(), faults.end(), OverTemp2) != faults.end())
    {
     data.T1[1] = 1100;
    }
    else if (std::find(faults.begin(), faults.end(), OverTemp1) != faults.end())
    {
     data.T1[1] = 900;
    }
    else
    {
     data.T1[1] = 900 * log(phaseTime + 1) + T0 + generateRandomNoise(data.T1[1] *
0.005);
    }
   }
   else
   {
    data.T1[1] = T0;
   }
            if (std::find(faults.begin(), faults.end(), T2_1Fail) == faults.end() &
& std::find(faults.begin(), faults.end(), T2Fail) == faults.end())
            {
                if (std::find(faults.begin(), faults.end(), OverTemp4) != faults.en
d())
                {
                    data.T2[0] = 1200;
                }
                else if (std::find(faults.begin(), faults.end(), OverTemp3) != faul
ts.end())
                {
                    data.T2[0] = 1000;
                }
                else if (std::find(faults.begin(), faults.end(), OverTemp2) != faul
ts.end())
                {
                    data.T2[0] = 1100;
                }
                else if (std::find(faults.begin(), faults.end(), OverTemp1) != faul
ts.end())
                {
                    data.T2[0] = 900;
                }
                else
                {
                    data.T2[0] = 900 * log(phaseTime + 1) + T0 + generateRandomNois
e(data.T2[0] * 0.005);
                }
            }
```

```
            else
   {
    data.T2[0] = T0;
   }
            if (std::find(faults.begin(), faults.end(), T2_2Fail) == faults.end() &
& std::find(faults.begin(), faults.end(), T2Fail) == faults.end())
            {
                if (std::find(faults.begin(), faults.end(), OverTemp4) != faults.en
d())
                {
                    data.T2[1] = 1200;
                }
                else if (std::find(faults.begin(), faults.end(), OverTemp3) != faul
ts.end())
                {
                    data.T2[1] = 1000;
                }
                else if (std::find(faults.begin(), faults.end(), OverTemp2) != faul
ts.end())
                {
                    data.T2[1] = 1100;
                }
                else if (std::find(faults.begin(), faults.end(), OverTemp1) != faul
ts.end())
                {
                    data.T2[1] = 900;
                }
                else
                {
                    data.T2[1] = 900 * log(phaseTime + 1) + T0 + generateRandomNois
e(data.T2[1] * 0.005);
                }
   }
            else
            {
                data.T2[1] = T0;
            }
        }
        data.V = 42 * log(phaseTime + 1) + 10 + generateRandomNoise(data.V * 0.005)
;
        // 判断是否达到稳定状态
        if ((data.N1[0] / RATED_RPM >= 0.95 && data.N2[0] / RATED_RPM >= 0.95) ||
            (data.N1[1] / RATED_RPM >= 0.95 && data.N2[1] / RATED_RPM >= 0.95))
        {
            isStarted_2 = false;
            isStable = true;
            phaseTime = 0.0; // 重置阶段时间
        }
        phaseTime += INTERVAL_MS / 1000.0; // 增加阶段时间
    }
    else if (isStable)
    {
        // 稳定阶段，数据在±3%范围内波动
        for (int i = 0; i < 2; ++i)
        {
            if (std::find(faults.begin(), faults.end(), N1_1Fail) == faults.end() &
& std::find(faults.begin(), faults.end(), N1Fail) == faults.end())
            {
                data.N1[0] = data.baseN1 + generateRandomNoise(data.baseN1 * 0.03);
            }
            else
            {
                data.N1[0] = 0;
```

```
            }
            if (std::find(faults.begin(), faults.end(), N1_2Fail) == faults.end() &
& std::find(faults.begin(), faults.end(), N1Fail) == faults.end())
            {
                data.N1[1] = data.baseN1 + generateRandomNoise(data.baseN1 * 0.03);
            }
            else
            {
                data.N1[1] = 0;
            }
            if (std::find(faults.begin(), faults.end(), N2_1Fail) == faults.end() &
& std::find(faults.begin(), faults.end(), N2Fail) == faults.end())
            {
                data.N2[0] = data.baseN2 + generateRandomNoise(data.baseN2 * 0.03);
            }
            else
            {
                data.N2[0] = 0;
            }
            if (std::find(faults.begin(), faults.end(), N2_2Fail) == faults.end() &
& std::find(faults.begin(), faults.end(), N2Fail) == faults.end())
            {
                data.N2[1] = data.baseN2 + generateRandomNoise(data.baseN2 * 0.03);
            }
            else
            {
                data.N2[1] = 0;
            }
            if (std::find(faults.begin(), faults.end(), T1_1Fail) == faults.end() &
& std::find(faults.begin(), faults.end(), T1Fail) == faults.end())
            {
                if (std::find(faults.begin(), faults.end(), OverTemp4) != faults.en
d())
                {
                    data.T1[0] = 1200;
                }
                else if (std::find(faults.begin(), faults.end(), OverTemp3) != faul
ts.end())
                {
                    data.T1[0] = 1000;
                }
                else if (std::find(faults.begin(), faults.end(), OverTemp2) != faul
ts.end())
                {
                    data.T1[0] = 1100;
                }
                else if (std::find(faults.begin(), faults.end(), OverTemp1) != faul
ts.end())
                {
                    data.T1[0] = 900;
                }
                else
                {
                    data.T1[0] = data.baseT1 + generateRandomNoise(data.baseT1 * 0.
03);
                }
            }
            else
            {
                data.T1[0] = T0;
            }
            if (std::find(faults.begin(), faults.end(), T1_2Fail) == faults.end() &
& std::find(faults.begin(), faults.end(), T1Fail) == faults.end())
```

```
            {
                if (std::find(faults.begin(), faults.end(), OverTemp4) != faults.en
d())
                {
                    data.T1[1] = 1200;
                }
                else if (std::find(faults.begin(), faults.end(), OverTemp3) != faul
ts.end())
                {
                    data.T1[1] = 1000;
                }
                else if (std::find(faults.begin(), faults.end(), OverTemp2) != faul
ts.end())
                {
                    data.T1[1] = 1100;
                }
                else if (std::find(faults.begin(), faults.end(), OverTemp1) != faul
ts.end())
                {
                    data.T1[1] = 900;
                }
                else
                {
                    data.T1[1] = data.baseT1 + generateRandomNoise(data.baseT1 * 0.
03);
                }
            }
            else
            {
                data.T1[1] = T0;
            }
   if (std::find(faults.begin(), faults.end(), T2_1Fail) == faults.end() && std::fi
nd(faults.begin(), faults.end(), T2Fail) == faults.end())
   {
    if (std::find(faults.begin(), faults.end(), OverTemp4) != faults.end())
    {
     data.T2[0] = 1200;
    }
    else if (std::find(faults.begin(), faults.end(), OverTemp3) != faults.end())
    {
     data.T2[0] = 1000;
    }
    else if (std::find(faults.begin(), faults.end(), OverTemp2) != faults.end())
    {
     data.T2[0] = 1100;
    }
    else if (std::find(faults.begin(), faults.end(), OverTemp1) != faults.end())
    {
     data.T2[0] = 900;
    }
    else
    {
     data.T2[0] = data.baseT2 + generateRandomNoise(data.baseT2 * 0.03);
    }
   }
   else
   {
    data.T2[0] = T0;
   }
            if (std::find(faults.begin(), faults.end(), T2_2Fail) == faults.end() &
& std::find(faults.begin(), faults.end(), T2Fail) == faults.end())
            {
```

```
                if (std::find(faults.begin(), faults.end(), OverTemp4) != faults.en
d())
                {
                    data.T2[1] = 1200;
                }
                else if (std::find(faults.begin(), faults.end(), OverTemp3) != faul
ts.end())
                {
                    data.T2[1] = 1000;
                }
                else if (std::find(faults.begin(), faults.end(), OverTemp2) != faul
ts.end())
                {
                    data.T2[1] = 1100;
                }
                else if (std::find(faults.begin(), faults.end(), OverTemp1) != faul
ts.end())
                {
                    data.T2[1] = 900;
                }
                else
                {
                    data.T2[1] = data.baseT2 + generateRandomNoise(data.baseT2 * 0.
03);
                }
            }
   else
   {
    data.T2[1] = T0;
   }

        }
        data.V = data.baseV + generateRandomNoise(data.baseV * 0.03);
    }
    else if (isStopping)
    {
        // 定义停止时间
        const double maxStoppingTime = 10.0; // 最大停车时间为10 秒
        const double decayBase = 0.90; // 底数a，确保在10 秒内衰减到接近零
        //底数越大，衰减速度越慢
        // 随机扰动因子
        const double randomFactor = 0.01; // 随机扰动因子
        // 判断是否完全停止
        bool allStopped = true;
        for (int i = 0; i < 2; ++i)
        {
            // 如果任一转速未达到0，则没有完全停止
            if (data.N1[i] > 0 || data.N2[i] > 0 || data.T1[i] > T0 * 1.01 || data.
T2[i] > T0 * 1.01)
            {
                allStopped = false;
                break;
            }
        }
        if (allStopped)
        {
            // 完全停止后重置所有数据
            for (int i = 0; i < 2; ++i)
            {
                data.N1[i] = 0;
                data.N2[i] = 0;
                data.T1[i] = T0;
                data.T2[i] = T0;
```

```
            }
            data.V = 0;
            isStopping = false;
            isInit = true;
            phaseTime = 0.0; // 重置阶段时间
        }
        else
        {
            // 模拟转速对数衰减（没有达到0 时才衰减）
            for (int i = 0; i < 2; ++i)
            {
                if (data.N1[i] > 0)
                {
                    data.N1[i] = max(0.0, data.N1[i] * pow(decayBase, phaseTime) +
(rand() % 100 - 50) * randomFactor);
                }
                if (data.N2[i] > 0)
                {
                    data.N2[i] = max(0.0, data.N2[i] * pow(decayBase, phaseTime) +
(rand() % 100 - 50) * randomFactor);
                }
                // 温度扰动：在 T0 附近波动
                data.T1[i] = max(T0, T0 + (data.T1[i] - T0) * pow(decayBase, phaseT
ime) + (rand() % 100 - 50) * randomFactor);
                data.T2[i] = max(T0, T0 + (data.T2[i] - T0) * pow(decayBase, phaseT
ime) + (rand() % 100 - 50) * randomFactor);
            }
            // 燃油流速直接归零
            data.V = 0;
            phaseTime += INTERVAL_MS / 1000.0; // 增加阶段时间
        }
    }
    // 更新燃油余量
    data.C -= data.V * (INTERVAL_MS / 1000.0);
    if (data.C < 0)
    {
        data.C = 0;
        isStopping = true; // 如果没油则停止
    }
}
/*————————————————*/
/*————————数据保存————————*/
/*————————————————*/
    // 将数据保存到文件
void saveDataToFile(const EngineData& data, double time, ofstream& file)
{
    file << time << ",";
    for (int i = 0; i < 2; ++i)
    {
        file << data.N1[i] << "," << data.N2[i] << "," << data.T1[i] << "," << data
.T2[i] << ",";
    }
    file << data.C << "," << data.V << "\n";
}
/*————————————————*/
/*————————界面绘制————————*/
/*————————————————*/
// 定义颜色
const COLORREF GRAY = RGB(128, 128, 128);
const COLORREF AMBER = RGB(255, 191, 0);
struct DataBoard
{
    LPCTSTR name;                       //表盘名称
```

```
    double value;                       //当前值
    double maxDegree = 210;             //最大值（默认 210 度）
    double minDegree = 0;               //最小值
    double maxValue;                    //最大值
    double minValue;                    //最小值
    double x;                           //圆心 x 坐标
    double y;                           //圆心 y 坐标
    double radius;                      //半径
    COLORREF color;                     //颜色（白色，琥珀色或红色）
    DataBoard(LPCTSTR name, double value, double maxValue, double minValue, double
x, double y, double radius, COLORREF color) :
        name(name), value(value), maxValue(maxValue), minValue(minValue), x(x), y(y
), radius(radius), color(color)
    {
    }
};
double calcDegree(double value, double max, double min, double maxDegree, double mi
nDegree) {
    double Degree = (maxDegree - minDegree) * value / (max - min);
    return Degree;//返回角度 0`210 度
}
void drawDataBoard(DataBoard& DataBoard) {
    double x = DataBoard.x;
    double y = DataBoard.y;
    double radius = DataBoard.radius;
    // 绘制表盘外圈
    setcolor(WHITE);
    pie(x - radius, y - radius, x + radius, y + radius, DataBoard.maxDegree, DataBo
ard.minDegree);
    // 绘制表盘名称
    outtextxy(x - 25, y - 50, DataBoard.name);
    // 绘制数值显示区域
    std::wstring value;
    value = std::to_wstring(DataBoard.value);
    RECT valueRect = { x, y - 25, x + radius + 15, y };
    rectangle(int(valueRect.left), int(valueRect.top), int(valueRect.right), int(va
lueRect.bottom));
    drawtext(value.c_str(), &valueRect, DT_CENTER | DT_VCENTER | DT_SINGLELINE);
    // 绘制填充区域
    setfillcolor(DataBoard.color);
    // 计算当前值对应的角度（从 0°顺时针开始）
    double fillAngle = calcDegree(DataBoard.value, DataBoard.maxValue, DataBoard.mi
nValue, DataBoard.maxDegree, DataBoard.minDegree);
    // 从 minDegree (0°) 到当前计算的角度顺时针填充
    if (fillAngle > 0.01 && fillAngle < 350)
    {
        fillpie(x - radius * 0.9, y - radius * 0.9, x + radius * 0.9, y + radius *
0.9,
            (DataBoard.minDegree - fillAngle) * 2, DataBoard.minDegree);
    }
    // 恢复绘图属性
    setcolor(WHITE);
    setlinestyle(PS_SOLID, 1);
}
// 通过错误列表，判断 N1 表盘颜色
COLORREF checkN1DataBoardColor(std::vector<FaultType> faults) {
    if (faults.empty()) {
        return GRAY;
    }
    else {
        for (FaultType fault : faults) {
            if (fault == OverSpd2) {
                return RED;
```

```
            }
            else if (fault == OverSpd1) {
                return AMBER;
            }
        }
        return GRAY;
    }
}
// 通过错误列表，判断 N2 表盘颜色
COLORREF checkN2DataBoardColor(std::vector<FaultType> faults) {
    if (faults.empty()) {
        return GRAY;
    }
    else {
        for (FaultType fault : faults) {
            if (fault == OverSpd2) {
                return RED;
            }
            else if (fault == OverSpd1) {
                return AMBER;
            }
        }
        return GRAY;
    }
}
// 通过错误列表，判断 EGT 表盘颜色
COLORREF checkTDataBoardColor(std::vector<FaultType> faults) {
    if (faults.empty()) {
        return GRAY;
    }
    else {
        for (FaultType fault : faults) {
            if (fault == OverTemp2) {
                return RED;
            }
            else if (fault == OverTemp4) {
                return RED;
            }
            else if (fault == OverTemp1) {
                return AMBER;
            }
            else if (fault == OverTemp3) {
                return AMBER;
            }
        }
    }
    return GRAY;
}
void drawInterface(const EngineData& data, const std::vector<FaultType>& faults)
{
    // 清除绘图窗口
    cleardevice();
    // 设置字体和颜色
    settextstyle(20, 0, _T("Arial"));
    settextcolor(WHITE);
    COLORREF color;
    /*————仪表盘————*/
    // 绘制左发动机表盘
    color = checkN1DataBoardColor(faults);
    DataBoard leftEngineN1(_T("N_L1:"), data.N1[0] * 100 / RATED_RPM, 125 * 100, 0.
0, 100, 100, 80, color);
    drawDataBoard(leftEngineN1);
    color = checkN1DataBoardColor(faults);
```

```
    DataBoard leftEngineN2(_T("N_L2:"), data.N1[1] * 100 / RATED_RPM, 125 * 100, 0.
0, 100, 280, 80, color);
    drawDataBoard(leftEngineN2);
    color = checkTDataBoardColor(faults);
    DataBoard leftEngineT(_T("EGT_L:"), (data.T1[0] + data.T1[1]) / 2, 1200.0 * 100
, -5.0, 100, 460, 80, color);
    drawDataBoard(leftEngineT);
    // 绘制右发动机表盘
    color = checkN2DataBoardColor(faults);
    DataBoard rightEngineN1(_T("N_R1:"), data.N2[0] * 100 / RATED_RPM, 125 * 100, 0
.0, 300, 100, 80, color);
    drawDataBoard(rightEngineN1);
    color = checkN2DataBoardColor(faults);
    DataBoard rightEngineN2(_T("N_R2:"), data.N2[1] * 100 / RATED_RPM, 125 * 100, 0
.0, 300, 280, 80, color);
    drawDataBoard(rightEngineN2);
    color = checkTDataBoardColor(faults);
    DataBoard rightEngineT(_T("EGT_R:"), (data.T2[0] + data.T2[1]) / 2, 1200.0 * 10
0, -5.0, 300, 460, 80, color);
    drawDataBoard(rightEngineT);
    setfillcolor(WHITE);
    /*————燃油与油耗————*/
    // 绘制燃油槽内的燃油量
    double fuel = data.C;
    double fuelMax = 20000.0;
    //流速
    setlinecolor(WHITE);
    rectangle(20, 560, 180, 590);
    outtextxy(70, 570, _T("Fuel Flow:"));
    rectangle(180, 560, 380, 590);
    wchar_t fuelFlowStr[20];
    swprintf(fuelFlowStr, 20, L"%.2f", data.V);
    outtextxy(230, 570, fuelFlowStr);
    //燃油
    rectangle(20, 600, 180, 630);
    outtextxy(70, 610, _T("Fuel:"));
    rectangle(180, 600, 380, 630);
    if (fuel == 0)
    {
        outtextxy(230, 610, _T("ERROR"));
    }
    else
    {
        wchar_t fuelStr[20];
        swprintf(fuelStr, 20, L"%.2f", fuel);
        outtextxy(230, 610, fuelStr);
    }
    // 绘制燃油槽
    setlinecolor(WHITE);
    rectangle(880, 20, 900, 180);
    if (fuel < 1000)
    {
        setfillcolor(AMBER);// 如果燃油量过少，燃油槽显示琥珀色
    }
    else
    {
        setfillcolor(WHITE);
    }
    solidrectangle(880, static_cast<int>(max(20, 180 - 160 * fuel / fuelMax)), 900,
 180);
    /*————启动停车与状态显示————*/
    //绘制启动按钮
    rectangle(450, 20, 600, 90);
```

```
    RECT startRect = { 450, 20, 600, 90 };
    drawtext(_T("Start"), &startRect, DT_CENTER | DT_VCENTER | DT_SINGLELINE);
    //绘制停车按钮
    rectangle(450, 110, 600, 180);
    RECT stopRect = { 450, 110, 600, 180 };
    drawtext(_T("Stop"), &stopRect, DT_CENTER | DT_VCENTER | DT_SINGLELINE);
    // 绘制增加推力按钮（上三角）
    POINT increaseThrustPoints[] = { {660, 20}, {620, 90}, {700, 90} };
    fillpolygon(increaseThrustPoints, 3);
    // 绘制减小推力按钮（下三角）
    POINT decreaseThrustPoints[] = { {660, 180}, {620, 110}, {700, 110} };
    fillpolygon(decreaseThrustPoints, 3);
    //状态显示框
    // 绘制 Start 状态显示框
    if (isStarted_1 || isStarted_2)
    {
        setfillcolor(LIGHTGRAY);
    }
    else
    {
        setfillcolor(DARKGRAY);
    }
    solidroundrect(720, 20, 870, 90, 10, 10);
    RECT startStatusRect = { 720, 20, 870, 90 };
    setbkmode(TRANSPARENT);
    drawtext(_T("Start"), &startStatusRect, DT_CENTER | DT_VCENTER | DT_SINGLELINE)
;
    // 计算 N 的百分比值
    double averageN = ((data.N1[0] + data.N1[1] + data.N2[0] + data.N2[1])) / (4.0
* RATED_RPM) * 100.0;
    // 绘制 Run 状态显示框
    if (isStable && averageN >= 95.0)
    {
        setfillcolor(LIGHTGRAY);
    }
    else
    {
        setfillcolor(DARKGRAY);
    }
    solidroundrect(720, 110, 870, 180, 10, 10);
    RECT runStatusRect = { 720, 110, 870, 180 };
    setbkmode(TRANSPARENT);
    drawtext(_T("Run"), &runStatusRect, DT_CENTER | DT_VCENTER | DT_SINGLELINE);
}
/*———故障显示———*/
COLORREF checkFaultColor(const std::vector<FaultType>& faults, FaultType faultType)
 {
    if (std::find(faults.begin(), faults.end(), faultType) != faults.end()) {
        switch (faultType) {
        case OverSpd2:
        case OverTemp2:
        case OverTemp4:
        case FuelSFail:
        case N1andN2Fail:
        case T1andT2Fail:
            return RED;
        case OverSpd1:
        case OverTemp1:
        case OverTemp3:
        case LowFuel:
        case OverFF:
        case N1Fail:
        case N2Fail:
```

```
        case T1Fail:
        case T2Fail:
            return AMBER;
        case N1_1Fail:
        case N1_2Fail:
        case N2_1Fail:
        case N2_2Fail:
        case T1_1Fail:
        case T1_2Fail:
        case T2_1Fail:
        case T2_2Fail:
            return WHITE;
        default:
            return DARKGRAY;
        }
    }
    return DARKGRAY;
}
void drawFaultDisplay(const std::vector<FaultType>& faults)
{
    //自定义
    LOGFONT f;
    gettextstyle(&f);       //获取当前字体设置
    f.lfHeight = 20;       //设置字体高度为 48
    _tcscpy_s(f.lfFaceName, _T("Arial")); //设置字体
    f.lfQuality = ANTIALIASED_QUALITY;  //设置输出效果为抗锯齿
    settextstyle(&f);
    settextcolor(BLACK);
    //——第一类故障——
    //N1_1
    setfillcolor(checkFaultColor(faults, N1_1Fail));
    solidroundrect(450, 200, 590, 240, 10, 10);
    RECT N1_1FailRect = { 450, 200, 590, 240 };
    setbkmode(TRANSPARENT);
    drawtext(_T("N1_1Fail"), &N1_1FailRect, DT_CENTER | DT_VCENTER | DT_SINGLELINE)
;
    //N1_2
    setfillcolor(checkFaultColor(faults, N1_2Fail));
    solidroundrect(450, 250, 590, 290, 10, 10);
    RECT N1_2FailRect = { 450, 250, 590, 290 };
    setbkmode(TRANSPARENT);
    drawtext(_T("N1_2Fail"), &N1_2FailRect, DT_CENTER | DT_VCENTER | DT_SINGLELINE)
;
    //N1
    setfillcolor(checkFaultColor(faults, N1Fail));
    solidroundrect(450, 300, 590, 340, 10, 10);
    RECT N1FailRect = { 450, 300, 590, 340 };
    setbkmode(TRANSPARENT);
    drawtext(_T("N1Fail"), &N1FailRect, DT_CENTER | DT_VCENTER | DT_SINGLELINE);
    //N2_1
    setfillcolor(checkFaultColor(faults, N2_1Fail));
    solidroundrect(600, 200, 740, 240, 10, 10);
    RECT N2_1FailRect = { 600, 200, 740, 240 };
    setbkmode(TRANSPARENT);
    drawtext(_T("N2_1Fail"), &N2_1FailRect, DT_CENTER | DT_VCENTER | DT_SINGLELINE)
;
    //N2_2
    setfillcolor(checkFaultColor(faults, N2_2Fail));
    solidroundrect(600, 250, 740, 290, 10, 10);
    RECT N2_2FailRect = { 600, 250, 740, 290 };
    setbkmode(TRANSPARENT);
    drawtext(_T("N2_2Fail"), &N2_2FailRect, DT_CENTER | DT_VCENTER | DT_SINGLELINE)
;
```

```
    //N2
    setfillcolor(checkFaultColor(faults, N2Fail));
    solidroundrect(600, 300, 740, 340, 10, 10);
    RECT N2FailRect = { 600, 300, 740, 340 };
    setbkmode(TRANSPARENT);
    drawtext(_T("N2Fail"), &N2FailRect, DT_CENTER | DT_VCENTER | DT_SINGLELINE);
    //N1andN2Fail
    setfillcolor(checkFaultColor(faults, N1andN2Fail));
    solidroundrect(750, 200, 900, 340, 10, 10);
    RECT N1andN2FailRect = { 750, 200, 900, 340 };
    setbkmode(TRANSPARENT);
    drawtext(_T("N1andN2Fail"), &N1andN2FailRect, DT_CENTER | DT_VCENTER | DT_SINGL
ELINE);
    //——第二类故障——
    //T1_1
    setfillcolor(checkFaultColor(faults, T1_1Fail));
    solidroundrect(450, 350, 590, 390, 10, 10);
    RECT T1_1FailRect = { 450, 350, 590, 390 };
    setbkmode(TRANSPARENT);
    drawtext(_T("T1_1Fail"), &T1_1FailRect, DT_CENTER | DT_VCENTER | DT_SINGLELINE)
;
    //T1_2
    setfillcolor(checkFaultColor(faults, T1_2Fail));
    solidroundrect(450, 400, 590, 440, 10, 10);
    RECT T1_2FailRect = { 450, 400, 590, 440 };
    setbkmode(TRANSPARENT);
    drawtext(_T("T1_2Fail"), &T1_2FailRect, DT_CENTER | DT_VCENTER | DT_SINGLELINE)
;
    //T1
    setfillcolor(checkFaultColor(faults, T1Fail));
    solidroundrect(450, 450, 590, 490, 10, 10);
    RECT T1FailRect = { 450, 450, 590, 490 };
    setbkmode(TRANSPARENT);
    drawtext(_T("T1Fail"), &T1FailRect, DT_CENTER | DT_VCENTER | DT_SINGLELINE);
    //T2_1
    setfillcolor(checkFaultColor(faults, T2_1Fail));
    solidroundrect(600, 350, 740, 390, 10, 10);
    RECT T2_1FailRect = { 600, 350, 740, 390 };
    setbkmode(TRANSPARENT);
    drawtext(_T("T2_1Fail"), &T2_1FailRect, DT_CENTER | DT_VCENTER | DT_SINGLELINE)
;
    //T2_2
    setfillcolor(checkFaultColor(faults, T2_2Fail));
    solidroundrect(600, 400, 740, 440, 10, 10);
    RECT T2_2FailRect = { 600, 400, 740, 440 };
    setbkmode(TRANSPARENT);
    drawtext(_T("T2_2Fail"), &T2_2FailRect, DT_CENTER | DT_VCENTER | DT_SINGLELINE)
;
    //T2
    setfillcolor(checkFaultColor(faults, T2Fail));
    solidroundrect(600, 450, 740, 490, 10, 10);
    RECT T2FailRect = { 600, 450, 740, 490 };
    setbkmode(TRANSPARENT);
    drawtext(_T("T2Fail"), &T2FailRect, DT_CENTER | DT_VCENTER | DT_SINGLELINE);
    //T1andT2Fail
    setfillcolor(checkFaultColor(faults, T1andT2Fail));
    solidroundrect(750, 350, 900, 490, 10, 10);
    RECT T1andT2FailRect = { 750, 350, 900, 490 };
    setbkmode(TRANSPARENT);
    drawtext(_T("T1andT2Fail"), &T1andT2FailRect, DT_CENTER | DT_VCENTER | DT_SINGL
ELINE);
    //第二类故障———
    //LowFuel
```

```
    setfillcolor(checkFaultColor(faults, LowFuel));
    solidroundrect(450, 500, 590, 540, 10, 10);
    RECT LowFuelRect = { 450, 500, 590, 540 };
    setbkmode(TRANSPARENT);
    drawtext(_T("LowFuel"), &LowFuelRect, DT_CENTER | DT_VCENTER | DT_SINGLELINE);
    //FuelSFail
    setfillcolor(checkFaultColor(faults, FuelSFail));
    solidroundrect(600, 500, 740, 540, 10, 10);
    RECT FuelSFailRect = { 600, 500, 740, 540 };
    setbkmode(TRANSPARENT);
    drawtext(_T("FuelSFail"), &FuelSFailRect, DT_CENTER | DT_VCENTER | DT_SINGLELIN
E);
    //OverFF
    setfillcolor(checkFaultColor(faults, OverFF));
    solidroundrect(750, 500, 900, 540, 10, 10);
    RECT OverFFRect = { 750, 500, 900, 540 };
    setbkmode(TRANSPARENT);
    drawtext(_T("OverFF"), &OverFFRect, DT_CENTER | DT_VCENTER | DT_SINGLELINE);
    //第三类故障——
    //OverSpd1
    setfillcolor(checkFaultColor(faults, OverSpd1));
    solidroundrect(450, 550, 590, 590, 10, 10);
    RECT OverSpd1Rect = { 450, 550, 590, 590 };
    setbkmode(TRANSPARENT);
    drawtext(_T("OverSpd1"), &OverSpd1Rect, DT_CENTER | DT_VCENTER | DT_SINGLELINE)
;
    //OverSpd2
    setfillcolor(checkFaultColor(faults, OverSpd2));
    solidroundrect(600, 550, 740, 590, 10, 10);
    RECT OverSpd2Rect = { 600, 550, 740, 590 };
    setbkmode(TRANSPARENT);
    drawtext(_T("OverSpd2"), &OverSpd2Rect, DT_CENTER | DT_VCENTER | DT_SINGLELINE)
;
    //第四类故障——
    //OverTemp1
    setfillcolor(checkFaultColor(faults, OverTemp1));
    solidroundrect(450, 600, 590, 640, 10, 10);
    RECT OverTemp1Rect = { 450, 600, 590, 640 };
    setbkmode(TRANSPARENT);
    drawtext(_T("OverTemp1"), &OverTemp1Rect, DT_CENTER | DT_VCENTER | DT_SINGLELIN
E);
    //OverTemp2
    setfillcolor(checkFaultColor(faults, OverTemp2));
    solidroundrect(600, 600, 740, 640, 10, 10);
    RECT OverTemp2Rect = { 600, 600, 740, 640 };
    setbkmode(TRANSPARENT);
    drawtext(_T("OverTemp2"), &OverTemp2Rect, DT_CENTER | DT_VCENTER | DT_SINGLELIN
E);
    //OverTemp3
    setfillcolor(checkFaultColor(faults, OverTemp3));
    solidroundrect(750, 550, 900, 590, 10, 10);
    RECT OverTemp3Rect = { 750, 550, 900, 590 };
    setbkmode(TRANSPARENT);
    drawtext(_T("OverTemp3"), &OverTemp3Rect, DT_CENTER | DT_VCENTER | DT_SINGLELIN
E);
    //OverTemp4
    setfillcolor(checkFaultColor(faults, OverTemp4));
    solidroundrect(750, 600, 900, 640, 10, 10);
    RECT OverTemp4Rect = { 750, 600, 900, 640 };
    setbkmode(TRANSPARENT);
    drawtext(_T("OverTemp4"), &OverTemp4Rect, DT_CENTER | DT_VCENTER | DT_SINGLELIN
E);
}
```

```
// 将 FaultType 转换为字符串
string faultToString(FaultType fault)
{
    switch (fault)
    {
    case N1_1Fail: return "N1_1Fail";
    case N1_2Fail: return "N1_2Fail";
    case N2_1Fail: return "N2_1Fail";
    case N2_2Fail: return "N2_2Fail";
    case T1_1Fail: return "T1_1Fail";
    case T1_2Fail: return "T1_2Fail";
    case T2_1Fail: return "T2_1Fail";
    case T2_2Fail: return "T2_2Fail";
    case N1Fail: return "N1Fail";
    case N2Fail: return "N2Fail";
    case T1Fail: return "T1Fail";
    case T2Fail: return "T2Fail";
    case N1andN2Fail: return "N1andN2Fail";
    case T1andT2Fail: return "T1andT2Fail";
    case LowFuel: return "LowFuel";
    case FuelSFail: return "FuelSFail";
    case OverFF: return "OverFF";
    case OverSpd1: return "OverSpd1";
    case OverSpd2: return "OverSpd2";
    case OverTemp1: return "OverTemp1";
    case OverTemp2: return "OverTemp2";
    case OverTemp3: return "OverTemp3";
    case OverTemp4: return "OverTemp4";
    default: return "UnknownFault";
    }
}
/*————————————————*/
/*————鼠标点击处理————*/
/*————————————————*/
int handleMouseClick()
{
    MOUSEMSG m = GetMouseMsg();
    switch (m.uMsg)
    {
    case WM_LBUTTONDOWN: // 如果是左键按下
        // 检查是否点击了 Start 按钮
        if (m.x >= 450 && m.x <= 600 && m.y >= 20 && m.y <= 90)
        {
            return 1; // 返回命令值 1
        }
        // 检查是否点击了 Stop 按钮
        if (m.x >= 450 && m.x <= 600 && m.y >= 110 && m.y <= 180)
        {
            return 2; // 返回命令值 2
        }
        // 检查是否点击了增加推力按钮（上三角）
        if (m.x >= 620 && m.x <= 700 && m.y >= 20 && m.y <= 90)
        {
            return 3; // 返回命令值 3
        }
        // 检查是否点击了减小推力按钮（下三角）
        if (m.x >= 620 && m.x <= 700 && m.y >= 110 && m.y <= 180)
        {
            return 4; // 返回命令值 4
        }
        //检查每一个故障按钮:
        // 11-17.21-27.31-33.41-42，51-54 分别对应故障类型一（N）、一（T）、二、三、四
        if (m.x >= 450 && m.x <= 590 && m.y >= 200 && m.y <= 240)
```

```
        {
            return 11;
        }
        if (m.x >= 450 && m.x <= 590 && m.y >= 250 && m.y <= 290)
        {
            return 12;
        }
        if (m.x >= 450 && m.x <= 590 && m.y >= 300 && m.y <= 340)
        {
            return 13;
        }
        if (m.x >= 600 && m.x <= 740 && m.y >= 200 && m.y <= 240)
        {
            return 14;
        }
        if (m.x >= 600 && m.x <= 740 && m.y >= 250 && m.y <= 290)
        {
            return 15;
        }
        if (m.x >= 600 && m.x <= 740 && m.y >= 300 && m.y <= 340)
        {
            return 16;
        }
        if (m.x >= 750 && m.x <= 900 && m.y >= 200 && m.y <= 340)
        {
            return 17;
        }
        if (m.x >= 450 && m.x <= 590 && m.y >= 350 && m.y <= 390)
        {
            return 21;
        }
        if (m.x >= 450 && m.x <= 590 && m.y >= 400 && m.y <= 440)
        {
            return 22;
        }
        if (m.x >= 450 && m.x <= 590 && m.y >= 450 && m.y <= 490)
        {
            return 23;
        }
        if (m.x >= 600 && m.x <= 740 && m.y >= 350 && m.y <= 390)
        {
            return 24;
        }
        if (m.x >= 600 && m.x <= 740 && m.y >= 400 && m.y <= 440)
        {
            return 25;
        }
        if (m.x >= 600 && m.x <= 740 && m.y >= 450 && m.y <= 490)
        {
            return 26;
        }
        if (m.x >= 750 && m.x <= 900 && m.y >= 350 && m.y <= 490)
        {
            return 27;
        }
        if (m.x >= 450 && m.x <= 590 && m.y >= 500 && m.y <= 540)
        {
            return 31;
        }
        if (m.x >= 600 && m.x <= 740 && m.y >= 500 && m.y <= 540)
        {
            return 32;
        }
```

```
        if (m.x >= 750 && m.x <= 900 && m.y >= 500 && m.y <= 540)
        {
            return 33;
        }
        if (m.x >= 450 && m.x <= 590 && m.y >= 550 && m.y <= 590)
        {
            return 41;
        }
        if (m.x >= 600 && m.x <= 740 && m.y >= 550 && m.y <= 590)
        {
            return 42;
        }
        if (m.x >= 450 && m.x <= 590 && m.y >= 600 && m.y <= 640)
        {
            return 51;
        }
        if (m.x >= 600 && m.x <= 740 && m.y >= 600 && m.y <= 640)
        {
            return 52;
        }
        if (m.x >= 750 && m.x <= 900 && m.y >= 550 && m.y <= 590)
        {
            return 53;
        }
        if (m.x >= 750 && m.x <= 900 && m.y >= 600 && m.y <= 640)
        {
            return 54;
        }
    }
    return -1; // 如果没有点击有效区域，返回无效命令值
}
/*————————————————*/
/*————MAIN 函数————*/
/*————————————————*/
int main()
{
    // 初始化发动机数据和其他变量
    EngineData engineData = {};
    engineData.C = MAX_FUEL; // 初始化燃油余量为最大燃油量
    engineData.V = 0;        // 初始燃油流速为0
    // 故障时间记录数据结构
    std::map<FaultType, double> faultTimeMap;
    // 用于记录每种故障上次记录日志的时间
    std::map<FaultType, double> faultLogTimeMap;
    vector<FaultType> faults;   // 用于存储故障信息
    double totalTime = 0.0;     // 总运行时间
    double time = 0.0;          // 当前时间
    // 打开数据文件保存文件
    ofstream dataFile("engine_data.csv");
    if (!dataFile.is_open())
    {
        cout << "无法打开数据文件！" << endl;
        return -1;
    }
    // 新增：打开日志文件
    ofstream logFile("fails.log");
    if (!logFile.is_open())
    {
        cout << "无法打开日志文件！" << endl;
        return -1;
    }
    // 写入数据文件表头
    dataFile << "Time,N1_1,N1_2,N2_1,N2_2,T1_1,T1_2,T2_1,T2_2,C,V\n";
```

```
    // 初始化绘图窗口
    initgraph(920, 660); // 根据需要调整窗口尺寸
    // 增加一个标志位，记录是否已经提示过用户
    bool hasPrompted = false;
    // 开始批量绘制
    BeginBatchDraw();
    while (true)
    {
        // 检查是否有鼠标点击
        if (MouseHit())
        {
            int command = handleMouseClick();
            if (command != -1)
            {
                if (command == 1)
                {
                    isInit = false;
                    isStarted_1 = true;
                    cout << "发动机启动中..." << endl;
                    hasPrompted = false; // 重置提示标志位
                }
                else if (command == 2)
                {
                    isStopping = true;
                    isStarted_1 = false;
                    isStarted_2 = false;
                    isStable = false;
                    cout << "发动机停止中..." << endl;
                }
                else if (command == 3)
                {
                    increaseThrust(engineData);
                    cout << "增加推力..." << endl;
                }
                else if (command == 4)
                {
                    decreaseThrust(engineData);
                    cout << "减小推力..." << endl;
                }
                else if (command >= 11 && command <= 54)
                {
                    // 处理故障模拟按键
                    FaultType fault;
                    switch (command)
                    {
                    case 11:
                        engineData.N1[0] = 0;
                        fault = N1_1Fail; break;
                    case 12:
                        engineData.N1[1] = 0;
                        fault = N1_2Fail; break;
                    case 13:
                        engineData.N1[0] = 0;
                        engineData.N1[1] = 0;
                        fault = N1Fail; break;
                    case 14:
                        engineData.N2[0] = 0;
                        fault = N2_1Fail; break;
                    case 15:
                        engineData.N2[1] = 0;
                        fault = N2_2Fail; break;
                    case 16:
                        engineData.N2[0] = 0;
```

```
                    engineData.N2[1] = 0;
                    fault = N2Fail; break;
                case 17:
                    engineData.N1[0] = 0;
                    engineData.N1[1] = 0;
                    engineData.N2[0] = 0;
                    engineData.N2[1] = 0;
                    fault = N1andN2Fail; break;
                case 21:
                    engineData.T1[0] = 0;
                    fault = T1_1Fail; break;
                case 22:
                    engineData.T1[1] = 0;
                    fault = T1_2Fail; break;
                case 23:
                    engineData.T1[0] = 0;
                    engineData.T1[1] = 0;
                    fault = T1Fail; break;
                case 24:
                    engineData.T2[0] = 0;
                    fault = T2_1Fail; break;
                case 25:
                    engineData.T2[1] = 0;
                    fault = T2_2Fail; break;
                case 26:
                    engineData.T2[0] = 0;
                    engineData.T2[1] = 0;
                    fault = T2Fail; break;
                case 27:
                    engineData.T1[0] = 0;
                    engineData.T1[1] = 0;
                    engineData.T2[0] = 0;
                    engineData.T2[1] = 0;
                    fault = T1andT2Fail; break;
                case 31:
                    engineData.C = 750;
                    fault = LowFuel; break;
                case 32:
                    engineData.C = 0;
                    fault = FuelSFail; break;
                case 33:
                    engineData.V = 55;
                    fault = OverFF; break;
                case 41:
                    engineData.N1[0] = 1.1 * RATED_RPM;
                    engineData.N1[1] = 1.1 * RATED_RPM;
                    engineData.N2[0] = 1.1 * RATED_RPM;
                    engineData.N2[1] = 1.1 * RATED_RPM;
                    fault = OverSpd1;
                    break;
                case 42:
                    engineData.N1[0] = 1.3 * RATED_RPM;
                    engineData.N1[1] = 1.3 * RATED_RPM;
                    engineData.N2[0] = 1.3 * RATED_RPM;
                    engineData.N2[1] = 1.3 * RATED_RPM;
                    fault = OverSpd2;
                    break;
                case 51:
                    engineData.T1[0] = 900.0;
                    engineData.T1[1] = 900.0;
                    engineData.T2[0] = 900.0;
                    engineData.T2[1] = 900.0;
                    fault = OverTemp1;
```

```
                    break;
                case 52:
                    engineData.T1[0] = 1100.0;
                    engineData.T1[1] = 1100.0;
                    engineData.T2[0] = 1100.0;
                    engineData.T2[1] = 1100.0;
                    fault = OverTemp2; break;
                case 53:
                    engineData.T1[0] = 1000.0;
                    engineData.T1[1] = 1000.0;
                    engineData.T2[0] = 1000.0;
                    engineData.T2[1] = 1000.0;
                    fault = OverTemp3;
                    break;
                case 54:
                    engineData.T1[0] = 1200.0;
                    engineData.T1[1] = 1200.0;
                    engineData.T2[0] = 1200.0;
                    engineData.T2[1] = 1200.0;
                    fault = OverTemp4;
                    break;
                }
                // 记录故障
                faults.push_back(fault);
                faultTimeMap[fault] = totalTime;
                logFile << totalTime << "s: " << faultToString(fault) << endl;
                faultLogTimeMap[fault] = totalTime;
                // 绘制界面，确保数据更新
                drawInterface(engineData, faults);
                drawFaultDisplay(faults);
                // 如果是严重故障，停止发动机
                if (fault == N1andN2Fail || fault == T1andT2Fail || fault == Fu
elSFail || fault == OverSpd2 || fault == OverTemp2 || fault == OverTemp4)
                {
                    isStopping = true;
                    isStarted_1 = false;
                    isStarted_2 = false;
                    isStable = false;
                    cout << "发动机停止中..." << endl;
                }
            }
            else if (command == 0)
            {
                // 退出程序
                break;
            }
        }
    }
    // 生成数据
    generateData(engineData, isInit, isStarted_1, isStarted_2, isStable, isStop
ping,faults);
    // 获取当前时间
    double currentTime = totalTime;
    // 检查故障
    checkData(engineData, isInit, isStarted_1, isStarted_2, isStable, isStoppin
g, faults, faultTimeMap, currentTime, logFile, faultLogTimeMap);
    // 更新故障列表，保持告警显示 5 秒
    auto it = faultTimeMap.begin();
    while (it != faultTimeMap.end())
    {
        if (currentTime - it->second > 5.0)
        {
            // 故障已超过 5 秒，移除
```

```
                it = faultTimeMap.erase(it);
            }
            else
            {
                ++it;
            }
        }
        // 清空 faults 列表，重新添加未过期的故障
        faults.clear();
        for (const auto& pair : faultTimeMap)
        {
            faults.push_back(pair.first);
        }
        // 保存数据到文件
        saveDataToFile(engineData, totalTime, dataFile);
        // 绘制界面
        drawInterface(engineData, faults);
        drawFaultDisplay(faults);
        // 刷新绘图
        FlushBatchDraw();
        // 延时
        this_thread::sleep_for(chrono::milliseconds(INTERVAL_MS));
        // 更新时间
        time += INTERVAL_MS / 1000.0;
        totalTime += INTERVAL_MS / 1000.0;
        // 判断发动机是否已停止并返回初始状态
        if (isInit && !isStarted_1 && !isStarted_2 && !isStable && !isStopping)
        {
            if (!hasPrompted)
            {
                hasPrompted = true; // 标记已经提示过用户
            }
        }
    }
    // 结束批量绘制
    EndBatchDraw();
    // 关闭数据文件
    dataFile.close();
    // 关闭日志文件
    logFile.close();
    // 关闭绘图窗口
    closegraph();
    return 0;
}
```