同濟大學

同濟大學

TONGJI UNIVERSITY

虚拟发动机性能监控模块开发

最后一次高程大作业报告

班级：国豪工科嘉定3班

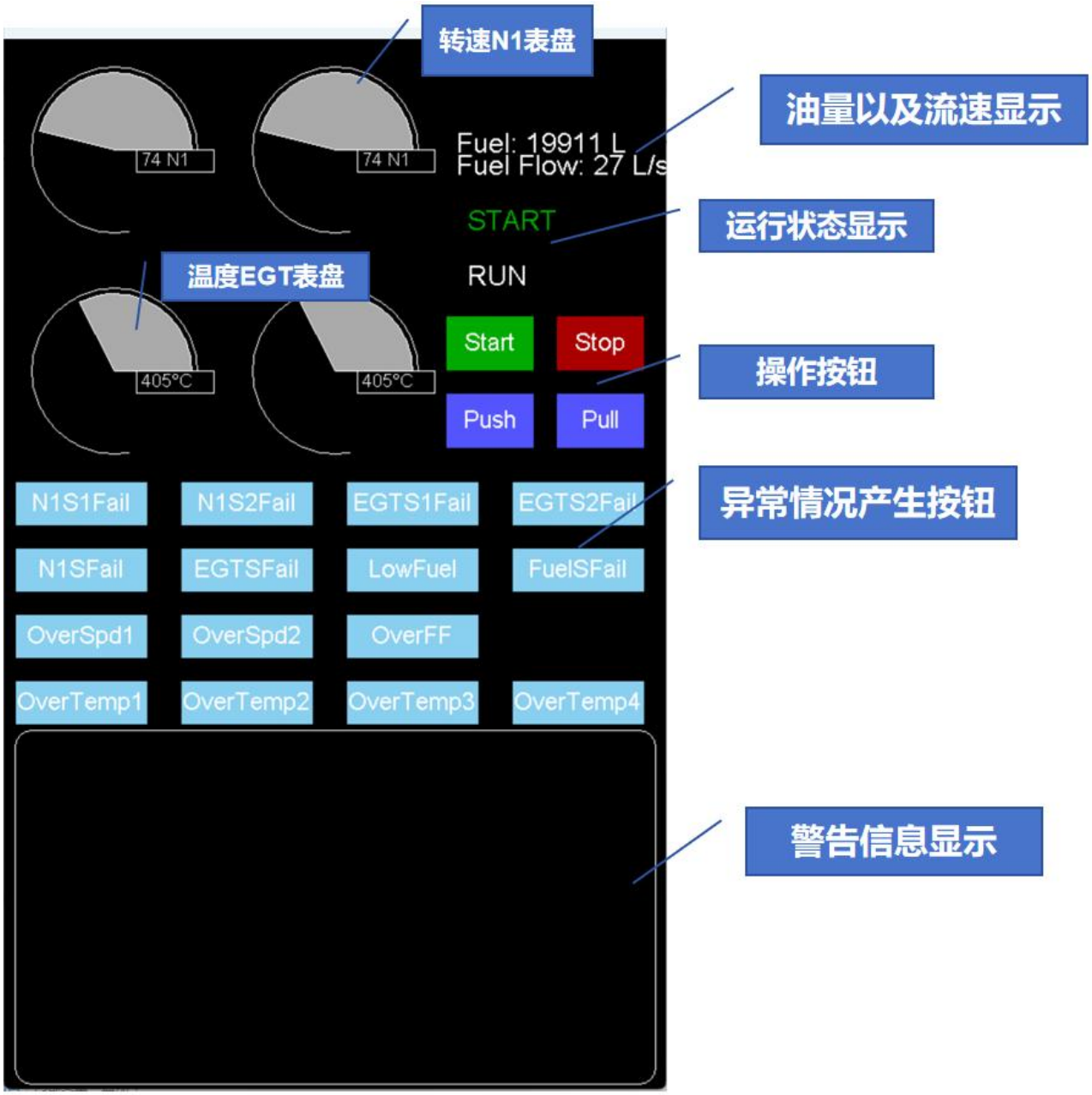
姓名：倪雨舒

学号： 2353105

完成日期： 2024年12月5日星期四

项目基本情况介绍

本次大作业利用easyX以C++面向对象设计的思想，实现了对发动机运行全过程的模拟以及数据的监测与记录，同时包含了对若干异常状况的模拟与处理。界面与对应功能如下：



备注：每个按钮被设置为按下后，在5秒之内无法再次按下，警告信息只显示5秒，然后就不再显示，部分按键之间存在关联，即按下后与其关联的按键无法按下；表盘的颜色以及警告信息的颜色与异常的严重情况相关，后续报告中都会提到。

整体设计思路&主要功能实现（思路）

一、理论公式推导

根据题干中提供的信息，我们可以手动总结出关键变量N,T,V,C的相关公式，联立得到的公式如下：

$$N=\begin{cases}10000t,0\le t\le 2;\\ 23000\lg(t-1)+20000,2<t<t_0;\\ 0.95N_{rated}(1\pm 3\%),t\ge t_0\end{cases}$$

$$T=\begin{cases}20,0\le t\le 2;\\ 900\lg(t-1)+T_0,2<t<t_0;\\ T_{stable}(1\pm 3\%),t\ge t_0\end{cases}$$

$$V=\begin{cases}5t,0\le t\le 2;\\ 42\lg(t-1)+10,2<t<t_0;\\ V_{stable}(1\pm 3\%),t\ge t_0\end{cases}$$

$$C=C_0-V(t)\Delta t,\Delta t=0.005s$$

其中t_0为转速达到额定转速时的时间节点，题目中在没有异常情况下规定达到额定转速即为稳定状态，此时相关变量N,T,V,C动态平稳。下面讨论停止阶段的变量函数形式。

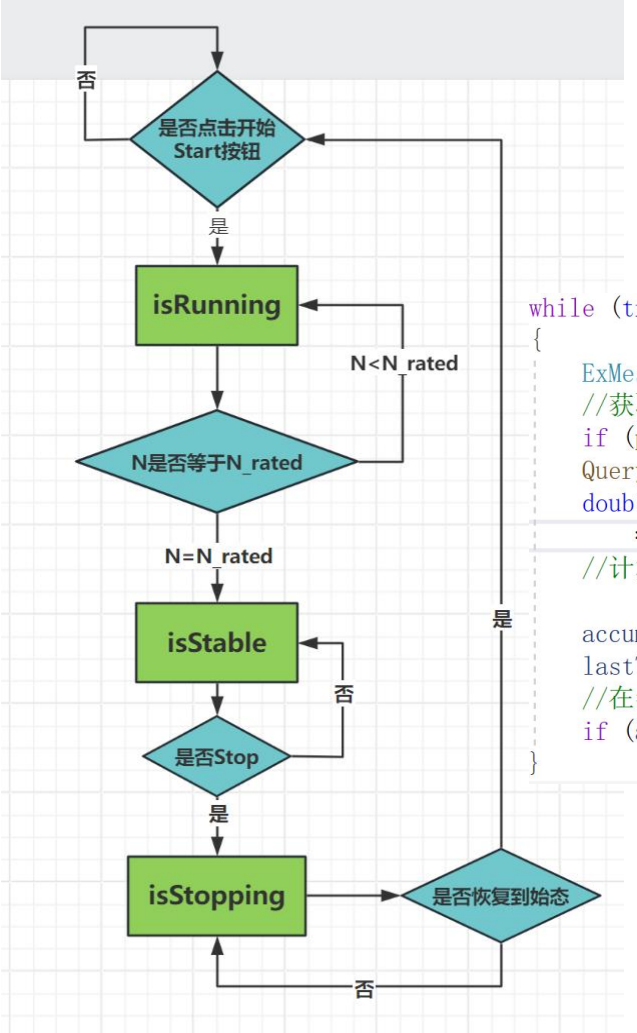
首先明确，停止函数仍旧为一个以10为底的对数函数，我调整的是对数函数之前的系数，以转速N为例，设停止时的时间节点为t_{stop}，设在此之后的转速的函数为：$N(t)=-A\lg(t+b)+N_{stop}$首先我们明确$N(t_{stop})=N_{stop}$，则有$-A\lg(t_{stop}+b)=0$因此$b=1-t_{stop}$，题目要求在10秒内完成停车，我设置为6秒，则又有方程$-A\lg(6+b)+N_{stop}=0$，最终得到$A=\dfrac{N_0}{\lg 7}$相同道理我们最终也可以得到温度函数对应的系数。最终得到的停车函数表达式如下：

$$N(t) = -\frac{N_{stop}}{\lg 7}\lg(t+1-t_{stop}) + N_{stop}$$

$$T(t) = -\frac{T_{stop}-20}{\lg 7}\lg((t+1-t_{stop}) + T_{stop}$$

二、模拟主流程

我将全过程分为3个主要状态，isRunning状态为启动状态，直到转速达到额定值；isStable状态为稳态，若没有收到Stop信号（点击Stop按钮）则一直维持稳态；isStopping为结束态，直到最终转速N，温度T都恢复到原始状态，流程图如下：




```
while (true)
{
    ExMessage msg;
    //获取鼠标信息，各种按钮
    if (peekmessage(&msg, EM_MOUSE)) { ... }
    QueryPerformanceCounter(&currentTime);  // 获取当前时间
    double deltaTime = (currentTime.QuadPart - lastTime.QuadPart)
        * 1000.0 / frequency.QuadPart;
    //计算从上次调用 QueryPerformanceCounter 以来经过的时间（以毫秒为单位）

    accumulatedTime += deltaTime;  // 累积时间
    lastTime = currentTime;        // 更新 lastTime
    //在省略部分的最后将accumulatedTime更新为0
    if (accumulatedTime >= 5.0) { ... }
}
```

题目中还要求刷新频率为5ms，相当于每5ms做一次运算得到相关数据，每5ms根据得到的数据画图，包括表盘以及其他信息。

在思考如何得到5ms的时间间隔之前，我注意到了在计算油量的时候需要利用这个时间间隔，所以我们可以将其转化为在一个无限循环中不断累加时间（acc

umulatedTime）直到达到5ms就执行相关操作，执行结束之后再将其归零，反复上述过程最终可以得到一个真正的以5ms为更新周期的程序。得到的时间间隔正好可以用来计算C剩余油量。程序如上图代码段所示。

三、类的设计

1、按钮类（Button）

为了减少思考量，按钮只包含了必要的成员（函数）示意图如下：

（1）是否有效（isValid）：故障按钮在按下之后会有5s的缓冲时间，在这5s内无法按下按钮，防止同一种故障在5s内重复出现（后来重新理解了：*5s内同一种告警只记录一次，是指在记录文件时每隔5s记录一次故障，但这个功能还是保留了*）

（2）按钮被按下的时间，实时时间（beginTime，nowTime）：对第一个有效值对应，记录按钮从按下开始到模拟过程时间间隔，来判断按钮是否有效。

（3）尺寸、文本内容、颜色：顾名思义，不解释

（4）绘制按钮、判断按钮是否点击：顾名思义，不解释。

```
class Button
{
public:
bool isValid;  // 按钮是否有效
double begineTime;
double nowTime;
int x, y, width, height; // 按钮的位置和尺寸
const char* text;        // 按钮显示的文本
COLORREF button_color;   // 按钮颜色
Button(int pos_x, int pos_y, int tmp_width, int
tmp_height, const char* tmp_text, COLORREF
tmp_color, bool Valid)//构造函数
 void drawButton();
bool isClicked(ExMessage& msg) ;
}
```

2、警报类（Warning）

有关警报有一个要求是只展示5s，我设定警报在展示5s之后消失，由此引入isShow变量判断是否需要展示警报信息，所以警报也需要记录时间（beginTime，nowTime）还有一个要求是每隔5s将警报信息写入文件中，所以要记录一下该条警

报是否被写入文件（islogged）。由于警报不同要求显示的颜色也不相同所以要记录一下警报的等级（level），其余成员如下：

（1）警报显示的位置，写入文件的信息（t_x，t_y，char*name）：顾名思义，不解释。

（2）绘制警报信息（draw_warning）：警报等级在其中被调用。

（3）写入警报信息（record_warning）：将警报内容写入文件，关于时间的判断在函数外进行判断。

```
class warning
{
public:
   int t_x, t_y,level;
   const char* name;
   double begine_time, now_time;
   bool islogged = false;
   int isShow = 1;
   warning(int x, int y, const char* w_name,int w_level,double
w_begin_time=0);
   void draw_warning();
  void record_warning(double currentTime);
  };
```

3、发动机类（Engine）

首先我们思考一个发动机需要哪些部件？根据题干所给出的信息我们知道了一个发动机包含两个转速传感器（N_sensor），两个温度传感器（T_sensor），对于整个发动机系统还需要一个燃油余量传感器和一个燃油流速传感器（C_V_sensor，我把他俩集成了），还有绘制的仪表盘（N_Dial，T_Dial）每个传感器都应该有相关的接口便于调用数据。

以上是比较宏观的类，还有一些相对“微观”的类：

（1）发动机运行时间（engineTime）：记录发动机运行时间，最重要的数据，计算转速，温度的变量，在一次模拟过程中应不断从外部传入并且保持递增，直到模拟结束重新归零。

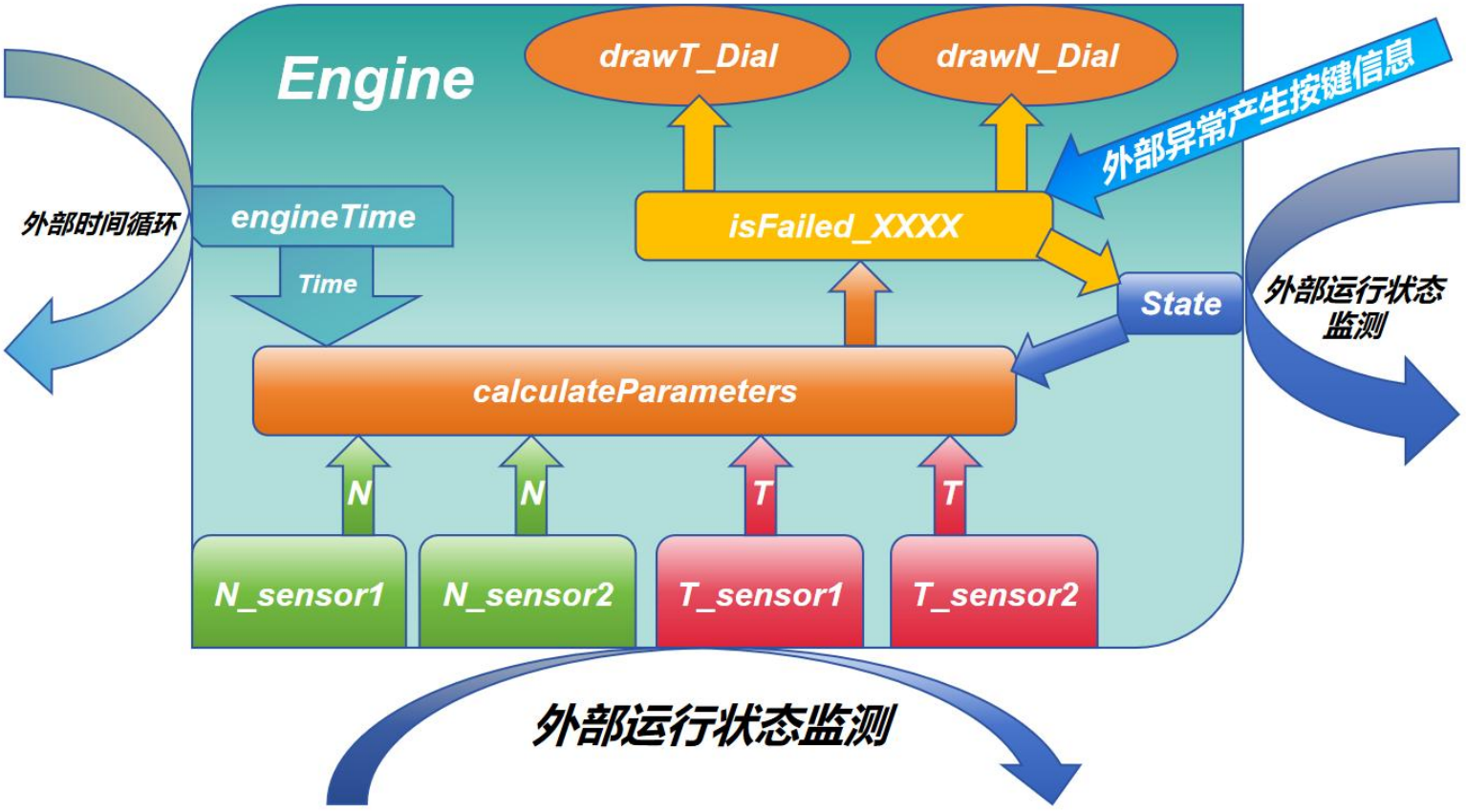
（2）暂存数据（N0,V0,T0，t0）：这些数据临时储存需要记录的信息，在停车状态（isStopping）以及稳态（isStable）格外重要。

（3）发动机状态（state）：上文提到外部信息（按钮信息，例如pull，push，stop按钮）以及内部信息（转速、温度）不断改变发动机会处于不同的状态，处于不同状态数据的计算方法也会不同。

（4）故障标志（isFailed_xxxxx）：故障标志用于故障发生的时候改变发动机中的数据（一般是人为创造故障时）或者改变仪表盘的显示。

（5）仪表盘（draw_X_dial）：直接使用绘制函数代替表盘显示，此外一些故障标志也会在其中调用。

（6）“处理器”（calculateParameters）：相当于中心处理器，根据传感器得到的信息计算相应数据的值并且根据不同状态调整不同的计算公式，是一个函数。最终得到的示意图（部分类内具体成员隐去）如下：



4、模拟类（Simulation）

模拟类中综合了之前提到的基本上的所有类型，其实也相当于整个飞机的整体架构，包括两台发动机（lengine，rengine）对模拟全过程运行模式的记录标志（isRunning，isStable，isStopping，pull，push）这些记录标志与发动机传出的数据紧密联系，便于切换不同的状态。异常检测方面，首先要根据异常产生按键的信息以及实时监测的数据对异常进行检测，根据异常情况的类型显示不同的信息，并且根据句异常的类型处理异常的。成员函数如下：

（1）开启发动机，停止发动机（startEngine，stopEngine）：开启发动机的时候包含的操作主要有：清空之前的记录文件，准备开始记录新的数据，停止发动机之后，要记录此时发动机的转速、温度等信息，便于后续运算处理。

（2）记录信息（record_info）：记录信息到文件中。

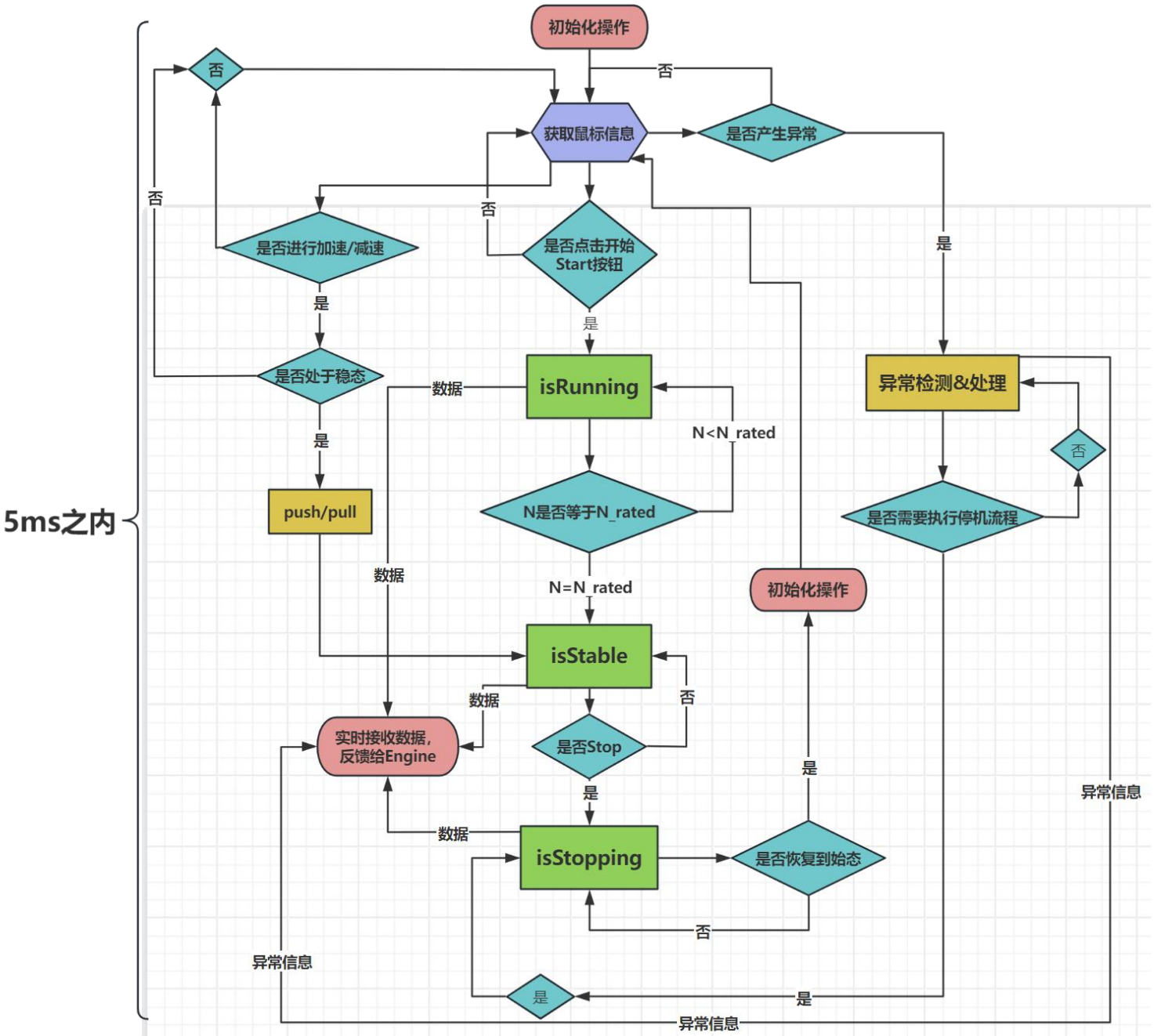
（3）恢复初始状态（set_state）：若要进行多次模拟，那么需要在每次模拟之前将比如：警告、按钮、发动机的状态进行初始化。

（4）异常产生函数（create_XXXXXFail）：根据不同的按键信息产生不同的异常。

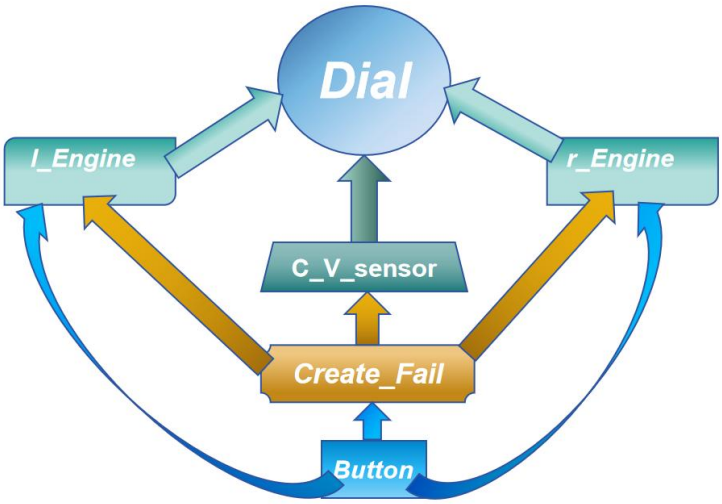
（5）检查&处理异常信息（Check_fail）:实时检测数据，如果产生异常那么就要显示异常信息并且做出处理。

（6）部分显示信息的函数：顾名思义，不解释。

（7）模拟主函数（run_simulation）：如下图流程所示：



发动机的整体架构如下图所示：



主要功能实现（代码）

根据上述类的描述，我们基本上已经得到了相应功能是如何实现的：利用整体的大循环实现时间的更新，利用不同封装类对发动机进行实例化，设置接口传递数据，利用理论推导得到的公式进行数据处理与运算。以下给出部分代码：

```
void calculateParameters
(double t_, double t0 = 0,
double randFactorN = 1.0, double randFactorT = 1.0, double randV = 1.0
)
{
    if (state == 1)
    { // 启动逻辑
        if (t_ >= 0 && t_ <= 2){ ... }
        else
        {
            N_sensor1.N_speed = 23000 * log10(t_ - 1) + 20000;
            N_sensor2.N_speed = 23000 * log10(t_ - 1) + 20000;
            T_sensor1.T_temp = 900 * log10(t_ - 1) + 20;
            T_sensor2.T_temp = 900 * log10(t_ - 1) + 20;
            if (isFailed_OverTemp1_tmp){ ... }
            if (isFailed_OverTemp2_tmp){ ... }
            N = (N_sensor1.N_speed + N_sensor2.N_speed) / 2;
            T = (T_sensor1.T_temp + T_sensor2.T_temp) / 2;
            V = 42 * log10(t_ - 1) + 10;
            if (N >= 38000){ ... }
        }
    }
    else if (state == 2 )//||state==6)
    {
        // 停止逻辑
        V = 0;
        double b = 1 - t0;
        double A = N0 / log10(7);
        double B = (T0 - 20) / log10(7);
        N_sensor1.N_speed = (int(N) > 0) ? -A * log10(t_ + b) + N0 : 0;
        N_sensor2.N_speed = (int(N) > 0) ? -A * log10(t_ + b) + N0 : 0;
        T_sensor1.T_temp = (int(T) > 20) ? -B * log10(t_ + b) + T0 : 20;
        T_sensor2.T_temp = (int(T) > 20) ? -B * log10(t_ + b) + T0 : 20;
        N = (N_sensor1.N_speed + N_sensor2.N_speed) / 2;
        T = (T_sensor1.T_temp + T_sensor2.T_temp) / 2;
    }
    else if (state == 3)
    {
        if (isFailed_OverSpd1_tmp){ ... }
        if (isFailed_OverSpd2_tmp){ ... }
        if (isFailed_OverTemp3_tmp){ ... }
        if (isFailed_OverTemp4_tmp){ ... }
        N_sensor1.N_speed = N0 * randFactorN;
        N_sensor2.N_speed = N0 * randFactorN;
        T_sensor1.T_temp = T0 * randFactorT;
        T_sensor2.T_temp = T0 * randFactorT;
        N = (N_sensor1.N_speed + N_sensor2.N_speed) / 2;
        T = (T_sensor1.T_temp + T_sensor2.T_temp) / 2;
        if (isFailed_OverFF){ ... }
        V = randV;
    }
    else if (state == 4){ ... }
    else if (state == 5)
    {
        V -= 1; N0 = N;
        T0 = T; V0 = V;
    }
}
```

遇到的问题与解决方案

1、Q：如何实现按钮、警告的定时显示?

A：在类中引入时间变量，与运行时间同步，并且记录按下/产生警告时间，记录时间差，达到5秒就不再显示

2、Q：如何实现每5秒记录一次警告信息

A：引入isShow、islogged变量，isShow初始设置为1，islogged初始值为false，如果islogged为false，那么就记录警告，并将islogged更新为true，isShow更新为2，如果时间间隔大于5秒，那么将islogged恢复为false，isShow更新为3，由此反复。

3、Q：如何实现每5ms刷新一次?

A：利用 QueryPerformanceFrequency相关函数并且通过累加的方式来更新时间，防止时间累计出错。

4、Q：如何实现异常情况的同步处理?

A：在Simulation类，Engine类，T_sensor、N_sensor类都设置相应的异常状态变量，并保证他们同步改变，达到点对点的数据同步改变。

5、Q：如何保证异常之间不冲突?

A：利用按钮的有效值，对可能发生冲突的按钮设置为互异有效，阻止用户操作，达到消除冲突的作用。并且通过检测模拟的阶段来控制某些按钮的有效情况。

6、Q：如何保存异常，使对于部分异常，如何使其一直存在?

A：利用多个判断异常情况的标志类型，实现在计算参数过程中对参数的改变，在计算过程对异常标志进行保存和修改。如下图，引入is_Failed_XXXX_tmp对计算过程修正。

```
if (state == 1)
{ // 启动逻辑
    if (t_ >= 0 && t_ <= 2) { ... }
    else
    {
        N_sensor1.N_speed = 23000 * log10(t_ - 1) + 20000
        N_sensor2.N_speed = 23000 * log10(t_ - 1) + 20000
        T_sensor1.T_temp = 900 * log10(t_ - 1) + 20;
        T_sensor2.T_temp = 900 * log10(t_ - 1) + 20;
        if (isFailed_OverTemp1_tmp)
        {
            T_sensor1.T_temp = 900;
            T_sensor2.T_temp = 900;
        }
        if (isFailed_OverTemp2_tmp)
        {
            T_sensor1.T_temp = 1005;
            T_sensor2.T_temp = 1005;
        }
        N = (N_sensor1.N_speed + N_sensor2.N_speed) / 2;
        T = (T_sensor1.T_temp + T_sensor2.T_temp) / 2;
        V = 42 * log10(t_ - 1) + 10;
        if (N >= 38000) { ... }
    }
}
else if (state == 2 ) { ... }
else if (state == 3)
{
    if (isFailed_OverSpd1_tmp)
    {
        N0 = 42000;
        isFailed_OverSpd1_tmp = false;
    }
    if (isFailed_OverSpd2_tmp) { ... }
    if (isFailed_OverTemp3_tmp) { ... }
    if (isFailed_OverTemp4_tmp) { ... }
    N_sensor1.N_speed = N0 * randFactorN;
    N_sensor2.N_speed = N0 * randFactorN;
    T_sensor1.T_temp = T0 * randFactorT;
    T_sensor2.T_temp = T0 * randFactorT;
```

智慧编程工具的应用

我通过chat找到了高精度的时间控制，并且我在开发到一半的时候就让chat按照目前的逻辑写一下将数据写入文件的部分，我没有采用chat的代码，而是在chat给出代码的相应位置，给出了我的实现，这个大作业的代码过长，其实利用chat直接复制粘贴询问的效果并不好，所以几乎绝大部分是设计都是我一个人完成的，

此外，我还发现chat对数学推导过程不是很擅长，在刚开始写停机逻辑的时候我想让chat写一下函数，但是结果很逆天，他的函数就是不对（找不到聊天记录了），而且他对10秒内减小为0不是很理解，无法得出自定义对数函数的底数是多少，所以最后都是自己推算的（不能偷懒啊orz）（以下是文件输入的部分代码）

```
void record_info()
{
    // 计算转速与额定转速比率、记录数据
    double lN_ratio1 = 100.0 * lengine.N_sensor1.N_speed / 40000;
    double lN_ratio2 = 100.0 * lengine.N_sensor2.N_speed / 40000;
    double rN_ratio1 = 100.0 * rengine.N_sensor1.N_speed / 40000;
    double rN_ratio2 = 100.0 * rengine.N_sensor2.N_speed / 40000;
    double lengine_T_sensor1_T = lengine.T_sensor1.T_temp;
    double lengine_T_sensor2_T = lengine.T_sensor2.T_temp;
    double rengine_T_sensor1_T = rengine.T_sensor1.T_temp;
    double rengine_T_sensor2_T = rengine.T_sensor2.T_temp;
    double V_value  = C_V_sensor0.V;
    string lN_info1 = to_string(int(lN_ratio1));
    string lN_info2 = to_string(int(lN_ratio2));
    string rN_info1 = to_string(int(rN_ratio1));
    string rN_info2 = to_string(int(rN_ratio2));
    string lT_info1 = to_string(int(lengine_T_sensor1_T));
    string lT_info2 = to_string(int(lengine_T_sensor2_T));
    string rT_info1 = to_string(int(rengine_T_sensor1_T));
    string rT_info2 = to_string(int(rengine_T_sensor2_T));
    string V_info = to_string(V_value);
    if (!lengine.N_sensor1.is_working) { ... }
    if (!lengine.N_sensor2.is_working) { ... }
    if (!rengine.N_sensor1.is_working) { ... }
    if (!rengine.N_sensor2.is_working) { ... }
    if (!lengine.T_sensor1.is_working) { ... }
    if (!lengine.T_sensor2.is_working) { ... }
    if (!rengine.T_sensor1.is_working) { ... }
    if (!rengine.T_sensor2.is_working) { ... }
    leftEngineFile
        << fixed << setprecision(4) << engineTime << ",          "
        << lN_info1 << ",                    "
        << lN_info2 << ",                    "
        << lT_info1 <<",                     "
        << lT_info2 << ",                    "
        << fixed << setprecision(3) <<V_info<< ",          "
        <<int(C)<<"\n";
    rightEngineFile
        << fixed << setprecision(4) << engineTime << ",          "
        << rN_info1 << ",                    "
        << rN_info2 << ",                    "
        << rT_info1 << ",                     "
        << rT_info2 << ",                    "
        << fixed << setprecision(3) << V_info << ",          "
        << int(C) << "\n";

}
```

Chat最主要的功能，我觉得是验证我的思路是否正确，而不是直接给出思路，如下图（上面是他的错误公式，下面是他表示认可我的思路）：

(2) 函数总结

1. **转速函数（停车阶段）：**

$$N(t) = 38000 \cdot \log_{0.1}(22.05 - t)$$

2. **温度函数（停车阶段）：**

$$T(t) = 1040 \cdot \log_{0.1}(22.05 - t)$$

好的，在我重新阅读题干之后，我再明确这个题目的流程，首先按下Start按钮，发动机进入启动阶段，同时根据启动阶段的生成数据的规则生成数据，同时根据生成的数据通过easyX绘制表盘，注意每五毫秒，生成一段数据，数据内容包括转速，温度，油箱余量，随后进入平稳阶段，请注意，在上述的全过程中，如果此时点击Stop按钮，那么飞机进入停车阶段，因此停车阶段的函数要根据目前的转速、温度，生成对应的对数函数在10秒内使飞机停止

你对题目的流程理解得非常正确，以下是详细的逻辑与每个阶段的设计说明，以确保实现的清晰性和连贯性：

---

1. 流程概述

1. **启动阶段：**
   - **触发条件：** 按下 **Start 按钮**。
   - **执行内容：**
     - 根据启动阶段规则生成数据：转速、温度、油箱余量、燃油流速。
     - 每 **5毫秒** 生成并记录一组数据，绘制表盘。
     - 当转速达到 **额定转速的95%（38000转/分）** 时，进入 **平稳阶段**。
2. **平稳阶段：**
   - **触发条件：** 启动阶段完成。

所以AI不是万能的，关键还得自己想！

源代码的组织结构

一些零散的类：按钮、警告、传感器

```
class Button
{
public:
    bool isValid;  // 按钮是否有效
    double begineTime;
    double nowTime;
    int x, y, width, height; // 按钮的位置和尺寸
    const char* text;        // 按钮显示的文本
    COLORREF button_color;   // 按钮颜色
    Button(int pos_x, int pos_y, int tmp_width, int tmp_height, const char* tmp_text, COLORREF tmp_color, bool Valid){ ... }
    // 绘制按钮
    void drawButton(){ ... }
    // 判断按钮是否被点击
    bool isClicked(ExMessage& msg){ ... }
};
```

```
class N_sensor
{
public:
    double N_speed;
    double N0;
    bool is_working;
};
class T_sensor { ... };
class C_V_sensor { ... };
class Warning
{
public:
    int t_x, t_y,level;
    const char* name;
    double begine_time, now_time;
    bool islogged = false;
    int isShow = 1;
    Warning(int x, int y, const char* w_name,int w_level,double w_begin_time=0) { ... }
    void draw_warning() { ... }
    void record_warning(double currentTime) { ... }
};
```

集成类：发动机

```
class Engine {
private:
    double t; // 全局时间
public:
    N_sensor N_sensor1, N_sensor2;
    T_sensor T_sensor1, T_sensor2;
    double N0, T0, V0;// 停止、平稳时的值
    double N, T, C, V;// 转速、温度、燃料
    double t0;
    double engineTime;
    bool isRunning, isStopping, isStable; // 平稳状态标志，其实没用上，懒得删了
    //异常情况标志
    bool isFailed_N1S1, isFailed_N1S2, isFailed_EGTS1, isFailed_EGTS2
        , isFailed_N1S, isFailed_EGTS,
        isFailed_OverSpd1, isFailed_OverSpd1_tmp,//用于改变N0的值
        isFailed_OverSpd2, isFailed_OverSpd2_tmp,//用于改变N0的值
        isFailed_OverFF,
        isFailed_OverTemp1, isFailed_OverTemp1_tmp,
        isFailed_OverTemp2, isFailed_OverTemp2_tmp,
        isFailed_OverTemp3, isFailed_OverTemp3_tmp,
        isFailed_OverTemp4, isFailed_OverTemp4_tmp; // 故障标志
    int state;
    Engine() : t(0), engineTime(0), N(0), T(20), C(20000), N0(0), T0(20), isRunning(false), isStopping(false), isStable(false) {}
    void calculateParameters
    (double t_, double t0 = 0,
     double randFactorN = 1.0, double randFactorT = 1.0, double randV = 1.0
    ) { ... }
    // 绘制单个表盘函数
    void drawN_Dial(int X, int Y) { ... }
    void drawT_Dial(int X, int Y) { ... }
};
```

模拟类：主模拟过程

```
class Simulation
{
public:
    double t;                   // 全局时间
    double engineTime;          // 当前启动过程时间
    Engine lengine;
    Engine rengine;
    C_V_sensor C_V_sensor0;
    bool isRunning, isStopping, isStable, push, pull; // 平稳状态标志，是否加速
    double C = 20000;
    bool isRecording;  // 控制文件记录
    bool isN1S1Fail, isN1S2Fail, isEGTS1Fail, isEGTS2Fail,
        isN1SFail, isEGTSFail,
        isLowFuelFail, isFuelSFail,isOverFFFail
        , isOverSpd1Fail, isOverSpd2Fail,
        isOverTemp1Fail, isOverTemp2Fail, isOverTemp3Fail, isOverTemp4Fail;
    vector<Warning> warnings;
    void startEngine(){ ... }
    void stopEngine(){ ... }
    void create_N1S1Fail();
    void create_N1S2Fail();
    void create_EGTS1Fail();//多次创建EGTS1故障有可能导致EGTS2故障发生
    void create_EGTS2Fail();
    void create_N1SFail();//N1S故障使得所有的N传感器都失灵，因此上述故障都会产生
    void create_EGTSFail();//
    void create_LowFuelFail();
    void create_FuelSFail();
    void create_OverFFFail();
    void create_OverSpd1();
    void create_OverSpd2();
    void create_OverTemp1();
    void create_OverTemp2();
    void create_OverTemp3();
    void create_OverTemp4();
    void Show_fail();
    void Check_fail();
    void start_Warning(int Case);
    void draw_buttons(vector<Button>& buttons);
    void record_info(){ ... }
    void set_State(){ ... }
    void runSimulation(){ ... }
    void show_state(bool isRunning, bool isStopping, bool isStable){ ... }
};
```

心得体会

做最后的大作业的时候能感觉到这个大作业是在逐渐引导我们去思考，去把问题思考周全的，模拟的过肯定有很多种情况，如何合理安排逻辑最终得到一个满意的结果是我们在上手写之前需要解决的。

此外由于临近期末，这个大作业还不是很完善，可以优化的地方有很多，类的封装性可以更好，代码可以更加简洁一些。高程课的陈老师和助教都是我来到同济以来遇到过的最负责的老师，可以看到老师和助教们为每个题都花费了很多心血，这个课也是我目前觉得收获最多的课。最后也衷心感谢各位老师助教，真的很感谢你们的付出!!!