期末作业报告

2452900 吴俊霖

项目基本情况介绍

本项目为《虚拟发动机模拟测试界面》，是基于 Qt 框架开发的 C++ 模拟程序，旨在实现飞机发动机指示和机组警告系统（EICAS）的核心功能。程序分为数据模拟生成、实时数据显示、异常检测与告警、日志记录四大模块，完整实现了作业要求中的所有基础项与进阶项功能。

项目采用 Visual Studio 2022 + Qt 6.x 开发环境，使用面向对象设计，结构清晰、模块独立，便于维护与扩展。最终交付物包含源码、可执行程序、日志文件及完整文档，符合工程化开发规范。

整体设计思路

1. 顶层架构设计

程序采用**分层架构**，核心模块包括：

- **全局状态定义（EngineState.h）**：作为唯一全局头文件，定义所有模块共享的枚举、结构体与常量，包括 `EnginePhase`、`AlertLevel`、`SensorData`、`AnomalyState` 以及 `RATED_RPM`、`TIME_STEP` 等常量。
- **数据层（DataGenerator）**：负责按 5ms 时间步长生成传感器数据，管理发动机阶段（Idle/Starting/Stable/Stopping），并支持推力控制。
- **逻辑层（EngineSimulator）**：作为主控模块，连接 UI 与数据层，处理按钮事件、定时器更新、异常检测与仪表盘刷新。
- **日志层（Logger）**：独立负责创建日志目录、写入 CSV 数据与告警信息，支持 5 秒内同类型告警去重。
- **UI 控件层（GaugeWidget）**：自定义仪表盘控件，支持 N1、EGT、燃油余量三种模式的扇形显示，并根据告警级别动态更新颜色。
- **UI 界面层（.ui 文件）**：使用 Qt Designer 设计主窗口布局，包含仪表盘占位符、控制按钮与状态指示灯。

2. 模块间协作关系

所有模块通过 `#include "EngineState.h"` 共享统一状态定义，避免重复声明与类型不一致。数据流如下：

```
    [UI] ↔ [EngineSimulator] ↔ [DataGenerator]
     ↓            ↓
[GaugeWidget]  [Logger]
```

- `EngineSimulator` 作为协调者，接收用户输入，驱动 `DataGenerator` 更新 `SensorData`，并调用 `Logger::logDataAndAlerts()` 记录。
- `GaugeWidget` 通过 `updateValue()` 和 `updateLevel(AlertLevel)` 接收数值与状态，实现颜色与指针动画。

3. 设计图示意

```
+-----------------------------+
| EngineState.h (全局头文件)  |
| - EnginePhase / AlertLevel  |
| - SensorData / AnomalyState |
| - 常量定义                  |
+-----------------------------+
              ↑
+-----------------------------+
| DataGenerator               | ← 5ms 更新 SensorData
+-----------------------------+
              ↑
+-----------------------------+
| EngineSimulator             | ← 定时器 + 按钮响应
+-----------------------------+
     ↑              ↑
[GaugeWidget]    [Logger]
  (仪表盘)       (日志+告警)
```

主要功能实现

1. 全局状态定义（EngineState.h）

作为项目唯一全局头文件，被所有 `.cpp` 和 `.h` 包含：

```cpp
// EngineState.h
#ifndef ENGINESTATE_H
#define ENGINESTATE_H
#include <QString>

enum class EnginePhase { Idle, Starting, Stable, Stopping };
enum class AlertLevel { normal, whiteWarning, amberWarning, redWarning };

struct SensorData {
    double n1LeftAverage; double n1RightAverage;
    double egtLeftAverage; double egtRightAverage;
    double fuelLevel; double fuelFlow;
    EnginePhase phase; EnginePhase lastPhase;
    double elapsedTime; bool EGTOverSpeed1;
};

struct AnomalyState {
    bool N1LS1Fail; bool N1RS1Fail; // ... 14 种异常标志位
    int N1OverSpeedLevel; int EGTOverSpeedLevel;
    bool LowFuel; bool FFOverSpeed; bool FuelSFail;
};

// 常量
constexpr double RATED_RPM = 40000.0;
constexpr double MAX_FUEL = 20000.0;
```

```
constexpr double TIME_STEP = 0.005;
// ... 按钮样式定义
#endif
```

2. 自定义仪表盘控件（GaugeWidget.cpp/.h）

- **功能**：支持 N1（0–125%）、EGT（–5–1200℃）、燃油余量（0–20000）三种模式，扇形角度 0°–210°，颜色随 `AlertLevel` 动态变化。
- **关键接口**（由 `EngineSimulator` 调用）：

```
n1LeftGauge->updateValue(currentData.n1LeftAverage);
n1LeftGauge->updateLevel(AlertLevel::amberWarning);
```

3. 数据生成与阶段控制（DataGenerator.cpp）

```
void DataGenerator::updateData() {
    auxData.fuelLevel = data.fuelLevel -= data.fuelFlow * TIME_STEP;
    data.elapsedTime += TIME_STEP;

    switch (data.phase) {
        case EnginePhase::Starting:
            if (data.elapsedTime < 2.0) {
                data.n1LeftAverage = data.n1RightAverage = 10000.0 *
data.elapsedTime * 100.0 / RATED_RPM;
                data.fuelFlow = 5.0 * data.elapsedTime;
            } else {
                data.n1LeftAverage = 23000.0 * std::log10(data.elapsedTime - 1.0)
* 100.0 / RATED_RPM + 50.0;
                data.fuelFlow = 42 * std::log10(data.elapsedTime - 1.0) + 10.0;
            }
            break;
        case EnginePhase::Stable: data = auxData; break;
        case EnginePhase::Stopping: /* 对数衰减 */; break;
    }
    checkPhase();
}
```

4. 告警去重与日志（Logger.cpp）

```
void Logger::triggerAlert(double elapsedTime, AlertLevel level, const QString &
alertMessage, QTextEdit * alertDisplay) {
    if (lastAlertTime.contains(msg)) {
        if (elapsedTime - lastAlertTime[msg] < 5.0) return; // 5秒去重
    }
    lastAlertTime[msg] = elapsedTime;
    if (alertFile.isOpen()) alertStream << fullText << "\n"; // 写入 .log
```

```
    if (alertDisplay) { /* UI 显示告警 */ }
}
```

开发过程中遇到的问题及解决方案

问题 1：Qt 初体验

- **现象**：本次大作业也是对自己的一个挑战，虽然期末周很紧，但还是希望能完成一个完整的，ui美观的工程性项目。学习与使用 Qt 框架过程中，遇到了很多问题。
- **解决方案**：
  - 看了很多 Qt 官方文档和教程，学习 Qt 的信号与槽机制，理解事件驱动编程模型。
  - 使用 Qt Designer 设计 UI 界面，学习布局管理与控件属性设置。
  - 通过调试和查阅资料，解决了自定义控件绘图

问题 2：自定义仪表盘颜色更新不同步

- **现象**：告警触发后，仪表盘颜色未及时更新，或恢复时颜色残留。
- **解决方案**：
  - 在 `GaugeWidget` 中分离 `updateValue()`（数值）与 `updateLevel(AlertLevel)`（颜色）。
  - 在 `EngineSimulator::updateSensor()` 中，根据 `AnomalyState` 显式调用 `gauge->updateLevel()`：

```
if (anomalyState.N1LS1Fail && anomalyState.N1LS2Fail) {
    n1LeftGauge->updateLevel(AlertLevel::amberWarning);
} else if (anomalyState.N1LS1Fail) {
    n1LeftGauge->updateLevel(AlertLevel::whiteWarning);
}
```

问题 3：停车阶段数据衰减不稳定

- **现象**：自定义对数衰减函数在 `t=0` 时出现 NaN 或负值。
- **解决方案**：
  - 引入 `auxData` 保存停车瞬间状态作为衰减起点。
  - 使用安全对数函数（底数 0.05，加偏移量防负）：

```
data.n1LeftAverage = auxData.n1LeftAverage * std::log10(0.05 + 15.0 *
stopTime / auxData.n1LeftAverage) / std::log10(0.05);
```

智慧编程工具在此次开发中的作用

1. Visual Studio 2022 + Qt VS Tools

- **智能提示与重构**：自动识别 `EngineState.h` 中的枚举与结构体，提供成员补全。
- **调试可视化**：在断点处直接查看 `SensorData` 结构体内容，快速验证数据生成逻辑。

- **Qt Designer 集成**：拖拽式布局，`layoutN1Left` 等占位符自动绑定到 `GaugeWidget`。

2. 自定义控件开发

- **GaugeWidget 调试**：通过临时 `qDebug()` 输出角度与颜色值，验证扇形绘制逻辑。
- **样式表管理**：将按钮样式定义为 `const QString`（如 `CHECKED_BTN_STYLE_AMBER`），支持一键切换主题。

3. qwenchats AI 助手

- **代码优化建议**：如在实现告警去重逻辑时，AI 提供了使用 `QHash` 存储上次告警时间的思路。
- **调试思路指导**：如遇到停车阶段数据异常时，AI建议检查对数函数的定义域，避免 NaN 产生。

心得体会

- 本次开发让我深刻认识到分层架构设计的重要性。实际上，以前我都是直接把全部代码写在一个文件里，导致代码臃肿、难以维护。这次通过将全局状态定义集中在 `EngineState.h`，使得各模块间的接口清晰，避免了重复定义和类型不一致的问题。同时数据生成、逻辑处理、日志记录和 UI 显示各司其职，职责分明，极大提升了代码的可维护性和扩展性。

- 自定义 `GaugeWidget` 的过程也让我体会到 Qt 绘图系统的强大。通过重写 `paintEvent()`，结合 `QPainter` 的扇形绘制与颜色插值，实现了专业级的航空仪表效果。更关键的是，将"数值"与"状态"分离的设计（`updateValue` / `updateLevel`），使得控件高度解耦，可复用于不同场景。

- **非常有成就感**：本次作业完成了所有功能要求，最终交付了一个稳定、易用的模拟程序。这让我体会到软件工程的乐趣——从需求分析、架构设计、编码实现到测试交付，每一步都充满挑战与成就感。当我从0开始学习 Qt，跳出舒适区，纠正自己不好的编程习惯，最终完成这个项目时，我感到无比自豪。

- 最后，这次作业不仅锻炼了 C++ 与 Qt 技能，更培养了我**从系统角度思考问题**的能力——一个优秀的程序，不仅功能正确，更要结构清晰、易于扩展、鲁棒可靠。

源代码的组织结构

项目采用清晰的模块化结构，所有源码位于 `EngineSimulator/` 目录下：

```
EngineSimulator/
├── EngineState.h            # 全局头文件（唯一共享定义）
├── main.cpp                 # 程序入口
├── EngineSimulator.cpp/h    # 主窗口逻辑
├── Logger.cpp/h             # 日志系统
├── DataGenerator.cpp/h      # 数据生成器
├── GaugeWidget.cpp/h        # 自定义仪表盘控件（核心UI组件）
├── EngineSimulator.ui       # Qt Designer 界面文件
├── EngineSimulator.qrc      # 资源文件
└── *.vcxproj                # Visual Studio 项目文件
```

- **关键设计**：

    - `EngineState.h` 被所有 `.cpp` 和 `.h` 包含，是项目“唯一真相源”。
    - `GaugeWidget` 作为独立控件，封装了所有绘图逻辑，主窗口无需关心实现细节。
- **可维护性**：新增仪表类型只需扩展 `GaugeWidget::Type` 枚举，无需修改主逻辑。