МИНОБРНАУКИ РОССИИ

РГУ НЕФТИ И ГАЗА (НИУ) ИМЕНИ И.М. ГУБКИНА

Факультет **Автоматики и вычислительной техники**
Кафедра **Прикладной математики и компьютерного моделирования**

Оценка комиссии: _____ Рейтинг: ____
Подписи членов комиссии:

_____ Фомочкина А.С.
(подпись) (фамилия, имя, отчество)
_____ Серова Д.А.
(подпись) (фамилия, имя, отчество)
_____
(дата)

# КУРСОВОЙ ПРОЕКТ

по дисциплине Дополнительные главы информатики

на тему Реализация методов объектно-ориентированного программирования

«К ЗАЩИТЕ»

Доцент, к.т.н. Серова Д.А
(должность, ученая степень; фамилия, и.о.)

_____
(подпись)

_____
(дата)

ВЫПОЛНИЛ:
Студент группы АМ-23-09
(номер группы)

Попов Р.А.
(фамилия, имя, отчество)

_____
(подпись)

18.12.2024
(дата)

Москва, 20 24

МИНОБРНАУКИ РОССИИ

РГУ НЕФТИ И ГАЗА (НИУ) ИМЕНИ И.М. ГУБКИНА

Факультет **Автоматики и вычислительной техники**
Кафедра **Прикладной математики и компьютерного моделирования**

## ЗАДАНИЕ НА КУРСОВОЕ ПРОЕКТИРОВАНИЕ

по дисциплине Дополнительные главы информатики

на тему Реализация методов объектно-ориентированного программирования

ДАНО студенту **Попову Роману Алексеевичу** (фамилия, имя, отчество в дательном падеже) группы **АМ-23-09** (номер группы)

**Содержание проекта:**

1. Постановка задачи
2. Инструкция к программной реализации
3. Программный код и пояснения к нему
4. Демонстрация работы программы (скриншоты) с описанием
5. Выводы

**Исходные данные для выполнения проекта:**

1. Данные из рекомендованной литературы.

**Рекомендуемая литература:**

1. Распространение вибраций в неметаллических композитах в составе трубопроводного транспорта/Галанский А. Н. Дубинов Ю. С. Корниенко Д. А.// Neftegaz.RU 2023 номер 8, С.106-110
2. Официальная страница Qt: https://qt.io
3. Образовательный портал РГУ нефти и газа (НИУ) имени И.М. Губкина, курс "Дополнительные главы информатики 2024/2025-осенний", edu.gubkin.ru

**Требования к представлению результатов:**

| | |
|---|---|
| ✓ | Электронная версия |
| | Бумажный вариант и электронный образ документа |

Руководитель: к.т.н. (уч.степень) доцент (должность) _____ (подпись) Серова Д. А. (фамилия, имя, отчество)

Задание принял к исполнению: студент _____ (подпись) Попов Р.А. (фамилия, имя, отчество)

**СОДЕРЖАНИЕ**

## ПОСТАНОВКА ЗАДАЧИ.

## ТЕОРЕТИЧЕСКАЯ ЧАСТЬ.

Каталитическая конверсия оксида углерода первой и второй ступени играет важнейшую роль в различных газохимических процессах. при ее надлежащем исполнении удается существенно повысить качество водородсодержащих газовых смесей на существующих промышленных установках.

В значительной степени это определяет эффективность работы крупнотоннажных агрегатов аммиака, а также планируемый перевод различных отраслей промышленности н транспортных средств на использование водородной технологии. конверсия оксида углерода обеих ступеней функционально связана со структурой применяемых каталитических систем и свойствами входящих в их состав промоторов. в представленной статье дан глубокий анализ фундаментальных и прикладных исследований по данной проблеме. Воспользуемся формулой для нахождения теплового эффекта ΔH реакции конверсии CO от температуры при постоянном давлении описывается:

$$H = 9\Delta 420 + 3{,}16T - 8{,}314\times 10^{-3}T^2 + 5{,}17\times 10^{-6}T^3 - 1{,}131\times 10^{-9}T^4$$

состав конкретной смеси определяется константой равновесия $K_p$ из соотношения

$$\frac{P_{CO_2}P_{H_2}}{P_{CO}P_{H_2O}} = K_p,$$

Зависимость константы равновесия от температуры

$$\lg K_p = -2{,}4198 + 0{,}0003855\ T + \frac{2180{,}6}{T}.$$

Процесс конверсии CO в реакторе среднетемпературной конверсии (СТК): при температуре 320-420℃ и давлении 100-667 КПа, соотношение пар:газ 0,65-0,75:1 угарный газ взаимодействует с водяным паром, образуя водород и углекислый газ. Реакция водяного газа: $CO+H_2O \rightarrow CO_2 + H_2 (\Delta H < 0, \text{экзотермическое})$ Эта реакция основная для процесса среднетемпературной конверсии.

## ПРОГРАММНАЯ ЧАСТЬ.

Требуется создать интерактивную модель каталитической конверсии углерода.

Модель будет содержать следующие элементы:

1) Задание имени пользователя через диалоговое окно, так же будет проверка на корректность введенного имени, оно не должно быть пустым.

2) Выбор режима. Пользователь сможет выбрать режим для работы: “Базовый”-вывод окна, описывающий что будет происходить в анимации; “Продвинутый”-вывод окна, в котором можно посчитать при заданным пользователем температуре и давлении теплового эффекта и константы равновесия, после чего увидит анимацию конверсии оксида углерода

3) Пользователь сможет задать параметры: температура и давление. От температуры зависит скорость движения молекул, а от давления – размер.

4) Создание визуализации процесса каталитической конверсии углерода. При визуализации будет показан процесс поступления молекул в резервуар где с ними в реакцию вступит катализатор.

5)Управление мышью. Пользователь может включить кран, тем самым осуществить подачу молекул в резервуар. После подачи молекул, он

может нажать на катализатор, чтобы осуществить процесс конверсии оксида углерода.

6) По завершении работы можно будет сохранить все полученные расчеты в текстовом файле (*.txt).

## ИНСТРУКЦИЯ К ПРОГРАММНОЙ РЕАЛЕЗАЦИИ.

При запуске программы появляется виджет ввода имени, на котором пользователю надо авторизироваться. Присутствуют кнопки “ok”и“cancel”, для подтверждения имени и выхода из программы соответственно.

При нажатии на кнопку “ok” появляется диалоговое окно, в котором пользователю предлагается выбрать “Базовый” или “Продвинутый” режим работы программы, а также “cancel” для выхода из программы. Далее, после нажатия на одну из кнопок выбора режима нас встречает диалоговое окно. При выборе “Базовый” на экран выводится сообщение о том, что будет происходить в анимации. При выборе “Продвинутый” выводится окно в котором предлагается ввести температуру и давление процесса. При их последующем вводе, пользователь получает необходимые значения, после чего может сохранить посчитанные программой результаты в файл, а также анимацию.

Графика включает в себя кран, трубу, ведущую в резервуар, сам резервуар и катализатор, расположенный с правой стороны. При нажатии мышкой на кран в резервуар начинают поступать молекулы, в хаотичном порядке заполняющие резервуар. Пользователь сам решает сколько молекул надо туда запустить, чтобы прекратить подачу нужно повторно нажать на кран. Далее необходимо активировать катализатор, который включается нажатием на него левой кнопкой мыши. Молекулы выпадают в осадок на дно резервуара, а пар, который при этом выделяется, поднимается вверх, конец анимации.

Также пользователь имеет возможность узнать химический состав катализатора, наведя на него курсор.

## ПРОГРАММНЫЙ КОД И ПОЯСНЕНИЯ К НЕМУ.

Для реализации программы используется пять классов: ModeSelectionDialog, MainWindow, ReactionParametersDialog, UserInputDialog,CatalystItem.

В UserInputDialog создается виджет для ввода имени. В заголовочном файле прописаны все метода и переменные класса:

```
#include <QObject>
#include <QString>
#include <QSet>

class UserInputDialog : public QObject {
    Q_OBJECT

public:
    explicit UserInputDialog(QObject *parent = nullptr);
        QString getUserName(const QSet<QString> &existingNames,
    QWidget *parent);
QString getUserName(const QSet<QString> &existingNames, QWidget
*parent);:
```

Метод, который:Принимает existingNames (набор уникальных строк, представляющих уже существующие имена) и указатель на родительский виджет (parent). Возвращает имя пользователя как строку QString.

```
private:
    QString validateUserName(const QString &name, const
    QSet<QString> &existingNames);
```

QString validateUserName(const QString &name, const QSet<QString> &existingNames);

Приватный метод, который: Проверяет введенное имя name на корректность (например, уникальность). Использует existingNames для проверки, не занято ли имя.

signals:

void userCanceled(); // Сигнал для уведомления о том, что пользователь отменил ввод

};

В cpp я прописываю логику этого окна:

```
#include "userinputdialog.h"
#include <QInputDialog>
#include <QMessageBox>
#include <QRegularExpression>

UserInputDialog::UserInputDialog(QObject *parent)
    : QObject(parent) {}

    QString UserInputDialog::getUserName(const QSet<QString> &existingNames, QWidget *parent) {
     QString name;
    bool ok;
```

QString UserInputDialog::getUserName(...): Метод получения имени пользователя. Он принимает:existingNames — множество уже занятых имен (QSet<QString>).parent — указатель на родительский виджет.

Возвращает имя пользователя в виде строки QString.QString name;: Переменная для хранения введенного имени.bool ok;: Переменная для проверки, нажал ли пользователь “ОК” или “Отмена” в диалоге ввода.

```
do {
name = QInputDialog::getText(parent, "Ввод имени пользователя",
                             "Введите ваше имя:",
                              QLineEdit::Normal, "", &ok);
```

do {: Начало цикла do-while, который будет повторяться до успешного ввода имени.QInputDialog::getText(...): Открывает диалог для ввода текста с заголовком “Ввод имени пользователя” и подсказкой “Введите ваше имя:”. Пользователь может ввести имя или нажать “Отмена”.&ok: Сюда будет записан результат — true (если пользователь нажал “ОК”) или false (если “Отмена”).

```
 if (!ok) {
        emit userCanceled();
              return QString(); // Возвращаем пустую строку, если
пользователь отменил ввод
}
```

if (!ok):

Проверяет, нажал ли пользователь “Отмена”.emit userCanceled();: Испускает сигнал userCanceled(), уведомляющий, что пользователь отменил ввод. return QString();: Возвращает пустую строку, указывая, что имя не было введено.

```
QString validationError = validateUserName(name, existingNames);
 if (!validationError.isEmpty()) {
  QMessageBox::warning(parent, "Ошибка", validationError);
} else {
  break;
}
```

QString validationError = ...: Проверяет введенное имя с помощью функции validateUserName, которая возвращает текст ошибки (или

пустую строку, если ошибок нет).if (!validationError.isEmpty()): Если есть ошибка (строка ошибки не пустая), выполняется код внутри if.QMessageBox::warning(...): Показывает предупреждение с заголовком “Ошибка” и текстом ошибки.break;: Прерывает цикл, если ошибок нет.

```
        } while (true);

        return name;
}

        QString UserInputDialog::validateUserName(const QString &name,
        const QSet<QString> &existingNames) {
         if (name.isEmpty()) {
         return "Имя не может быть пустым!";
         }
        if (name.contains(QRegularExpression("[^a-zA-Za-яА-Я0-9_ ]"))) {
         return "Имя содержит недопустимые символы!";
        }
        if (existingNames.contains(name)) {
        return "Это имя уже используется!";
         }
         return QString(); // Нет ошибок
}
```

QString UserInputDialog::validateUserName(...): Метод для проверки имени. Возвращает строку ошибки или пустую строку, если имя корректно. if (name.isEmpty()): Проверяет, пустое ли имя. Если да, возвращает сообщение об ошибке. if (name.contains(QRegularExpression("[^a-zA-Za-яА-Я0-9_ ]"))): Проверяет, содержит ли имя недопустимые символы (все, кроме букв, цифр, пробелов и подчеркиваний). if (existingNames.contains(name)):

Проверяет, занято ли имя. return QString();: Если ошибок нет, возвращает пустую строку.

В ModeSelectionDialog прописываю логику выбора режима работы, слоты onBasicmodeselected() и onAdvancedModeSelected() устанваливают режимы работы Базовый и Продвинутый соответсвенно

```
#include <QDialog>
#include <QString>
#include <QPushButton>

class ModeSelectionDialog : public QDialog {
        Q_OBJECT

public:
        explicit ModeSelectionDialog(QWidget *parent = nullptr);
        QString getSelectedMode() const;

private:
         QString selectedMode;
         QPushButton *basicModeButton;
         QPushButton *advancedModeButton;

private slots:
        void onBasicModeSelected();
         void onAdvancedModeSelected();
};
```

В .cpp файле этого класса я прописываю логику работы режимов

```
#include "modeselectiondialog.h"
#include <QVBoxLayout>
#include <QLabel>
#include <QMessageBox>

ModeSelectionDialog::ModeSelectionDialog(QWidget *parent)
        : QDialog(parent), selectedMode("") {

         setWindowTitle("Выбор режима работы");

        auto *layout = new QVBoxLayout(this);
         auto *label = new QLabel("Выберите режим работы:", this);
         layout->addWidget(label);
```

auto *layout = new QVBoxLayout(this);: Создает вертикальный компоновщик для размещения виджетов.auto *label = new QLabel("Выберите режим работы:", this);: Создает текстовую метку с сообщением.layout->addWidget(label);: Добавляет текстовую метку в компоновщик.

```
basicModeButton = new QPushButton("Базовый", this);
        advancedModeButton = new QPushButton("Продвинутый", this);
         auto *cancelButton = new QPushButton("Отмена", this);
```

basicModeButton = new QPushButton("Базовый", this);: Создает кнопку “Базовый” и сохраняет её в basicModeButton.advancedModeButton = new QPushButton("Продвинутый", this);: Создает кнопку “Продвинутый” и сохраняет её в advancedModeButton.auto *cancelButton = new QPushButton("Отмена", this);: Создает кнопку “Отмена”.

```
        layout->addWidget(basicModeButton);
```

```
        layout->addWidget(advancedModeButton);
        layout->addWidget(cancelButton);

        connect(basicModeButton, &QPushButton::clicked, this,
        &ModeSelectionDialog::onBasicModeSelected);
        connect(advancedModeButton, &QPushButton::clicked, this,
        &ModeSelectionDialog::onAdvancedModeSelected);
    connect(cancelButton, &QPushButton::clicked, this, &QDialog::reject);
}
```

connect(basicModeButton, &QPushButton::clicked, this, &ModeSelectionDialog::onBasicModeSelected);:Связывает сигнал clicked кнопки basicModeButton с методом onBasicModeSelected.connect(advancedModeButton, &QPushButton::clicked, this, &ModeSelectionDialog::onAdvancedModeSelected);Связывает сигнал clicked кнопки advancedModeButton с методом onAdvancedModeSelected.connect(cancelButton, &QPushButton::clicked, this, &QDialog::reject);:Связывает сигнал clicked кнопки cancelButton с методом reject, который закрывает диалог без выбора.

```
QString ModeSelectionDialog::getSelectedMode() const {
    return selectedMode;
}
```

QString ModeSelectionDialog::getSelectedMode() const: Возвращает строку selectedMode, содержащую выбранный режим.

```
void ModeSelectionDialog::onBasicModeSelected() {
        selectedMode = "Базовый";
    accept();
}
```

void ModeSelectionDialog::onBasicModeSelected(): Слот, вызываемый при нажатии кнопки “Базовый”.selectedMode = "Базовый";: Устанавливает значение selectedMode как “Базовый”.accept();: Завершает диалог с сигналом успешного выбора.

```
void ModeSelectionDialog::onAdvancedModeSelected() {
    selectedMode = "Продвинутый";
    accept();
}
```

В классе ReactionParametrsDialog задаются параметры конверсии оксида углерода соответственно. Как работает .h файл:

```
#include <QDialog>
#include <QString>

class QLineEdit;

class ReactionParametersDialog : public QDialog {
    Q_OBJECT

public:
    explicit ReactionParametersDialog(QWidget *parent = nullptr);
    ~ReactionParametersDialog();

    double getTemperature() const; // Получение температуры
    double getPressure() const;    // Получение давления

private:
```

```
    QLineEdit *tempInput;          // Поле ввода температуры
    QLineEdit *pressureInput;      // Поле ввода давления
};
```

Рассмотрим .cpp файл

```
#include "reactionparametersdialog.h"
#include <QFormLayout>
#include <QLineEdit>
#include <QDialogButtonBox>
#include <QVBoxLayout>

ReactionParametersDialog::ReactionParametersDialog(QWidget
*parent)
    : QDialog(parent), tempInput(new QLineEdit(this)),
pressureInput(new QLineEdit(this)) {
    setWindowTitle("Параметры реакции");

    // Создаем форму ввода
    QFormLayout *formLayout = new QFormLayout;
    formLayout->addRow("Температура (T):", tempInput);
    formLayout->addRow("Давление:", pressureInput);

    // Кнопки "OK" и "Отмена"
    QDialogButtonBox *buttonBox = new
QDialogButtonBox(QDialogButtonBox::Ok |
QDialogButtonBox::Cancel, this);
    connect(buttonBox, &QDialogButtonBox::accepted, this,
&QDialog::accept);
connect(buttonBox, &QDialogButtonBox::rejected, this,
&QDialog::reject);
```

QDialogButtonBox *buttonBox = new QDialogButtonBox(...): Создает набор кнопок “OK” и “Отмена”.connect(buttonBox, &QDialogButtonBox::accepted, this, &QDialog::accept);: Связывает сигнал “accepted” с методом accept, чтобы закрыть окно с подтверждением.connect(buttonBox, &QDialogButtonBox::rejected, this, &QDialog::reject);: Связывает сигнал “rejected” с методом reject, чтобы закрыть окно с отменой.

```
    // Общий макет
    QVBoxLayout *layout = new QVBoxLayout(this);
    layout->addLayout(formLayout);
    layout->addWidget(buttonBox);

    setLayout(layout);
}

ReactionParametersDialog::~ReactionParametersDialog() = default;

double ReactionParametersDialog::getTemperature() const {
    return tempInput->text().toDouble();
}
```

Метод возвращает введенное значение температуры. Использует text() для получения строки из поля ввода и преобразует ее в double.

```
double ReactionParametersDialog::getPressure() const {
    return pressureInput->text().toDouble();
}
```

Метод возвращает введенное значение давления. Аналогично температуре, использует преобразование строки в число.

В классе mainwindow вызываются поочередно все выше перечисленные виджеты, после чего описывается логика анимации, ее работа.

```
#include <QGraphicsView>
#include <QRandomGenerator>
#include <QBrush>
#include <QPen>
#include <QInputDialog>
#include <QMessageBox>
#include <QComboBox>
#include <QFormLayout>
#include <QLineEdit>
#include <QDialogButtonBox>
#include <cmath>
#include <QPushButton>
#include "modeselectiondialog.h"
#include "reactionparametersdialog.h"
#include <QFile>
#include <QTextStream>
#include <QFileDialog>
#include <QMessageBox>
#include <QMainWindow>
#include <QGraphicsScene>
#include <QGraphicsRectItem>
#include <QGraphicsEllipseItem>
#include <QTimer>
#include <QMouseEvent>
#include <QSet>
#include "userinputdialog.h"
#include <QLineEdit>
```

```
#include <QPushButton>
class MainWindow : public QMainWindow {
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = nullptr);
    ~MainWindow() override;

protected:
    void mousePressEvent(QMouseEvent *event) override;

private slots:
    void updateMolecules();        // Обновление молекул
    void toggleValve();            // Переключение состояния крана
    void activateCatalyst();       // Активация катализатора
    void startReaction();          // Начало реакции
    void placeMoleculeInReservoir(QGraphicsEllipseItem *molecule);
    void applyParameters();
    void resetAnimation();
```

mousePressEvent: Обрабатывает нажатие мыши на графической сцене. Слоты: void updateMolecules(); - обновление молекул, void toggleValve(); - Переключение состояния крана, void activateCatalyst(); -Активация катализатора, void startReaction(); - Начало реакции, void placeMoleculeInReservoir(QGraphicsEllipseItem *molecule); - размещение молекулы в резервуаре, void applyParameters(); - фиксирование введенных парамтеров, void resetAnimation(); - функция для перезапуска анимации.

```
private:
    QTimer *steamTimer = nullptr; // Таймер для анимации пара
```

```
    void animateSteam();
    void requestUserName(); // Запрос имени пользователя
    UserInputDialog *userInputDialog;
    void setupScene();          // Инициализация сцены
    void drawBackground();      // Отрисовка фона     // Запрос имени
пользователя
    void selectMode();          // Выбор режима
    void enterReactionParameters();// Ввод параметров реакции
    void animateSettling();     // Анимация осаждения
    void animateClouds();// Анимация облаков
    void setTemperature(int temp);
    void setPressure(int pres);
    void updateCatalystColor();
    void generateSteam();
    void updateMoleculeSpeed();
```

void animateSteam(); функция для анимации пара, после входа в реакцию конверсии катализатора, void animateClouds(); анимация облаков, функции void setTemperature(int temp); и void setPressure(int pres); используются для установления значений температуры и давления, void updateCatalystColor(); изменяет цвет катализатора в зависимости от температуры реакции, void generateSteam(); функция для генерации пара в рамках резервуара, void updateMoleculeSpeed(); функция, которая устанавливает скорость с какой будут двигаться молекулы в зависимости от температуры

```
    // Поля
    QGraphicsScene *scene;
    QVector<QGraphicsEllipseItem *> clouds;
    QVector<QGraphicsEllipseItem *> molecules;
```

QVector<QGraphicsEllipseItem *> settlingMolecules;

QLineEdit *temperatureInput; // Поле ввода температуры

QLineEdit *pressureInput;  // Поле ввода давления

QPushButton *applyButton;

QList<QGraphicsEllipseItem*> steamParticles;

QPushButton *resetButton;

Отрисовка полной сцены с полями для ввода температуры и давления, а также частиц, таких как: молекулы, облака и пар

QGraphicsRectItem *reservoir;

QGraphicsRectItem *valveBase;

QGraphicsRectItem *valveHandle;

QGraphicsRectItem *valveHandleTop;

QGraphicsRectItem *pipe;

QGraphicsRectItem *catalyst;

Отрисовка резервуара с трубой и краном

QTimer *moleculeTimer;

QTimer *settlingTimer;

QTimer *cloudTimer;

Таймеры для частиц, которые был описаны выше.

QString userName;

QSet<QString> existingNames;

QString selectedMode;

double temperature; // Начальная температура (Градусы)

double pressure;  // Начальное давление (Паскали)

// Состояния

bool valveOpen = false;

bool catalystActive = false;

bool reactionStarted = false;

```
    bool valveDisabled = false;
    bool parametersSet = false;

    // Геттеры и сеттеры
    void setValveOpen(bool open);
    bool isValveOpen() const;

    void setCatalystActive(bool active);
    bool isCatalystActive() const;

    void setReactionStarted(bool started);
    bool isReactionStarted() const;

    void setValveDisabled(bool disabled);
    bool isValveDisabled() const;
};
```

Сеттеры и геттеры управляют состоянием объекта, предоставляя доступ и модификацию значений.

Mainwindow.cpp

```
#include "mainwindow.h"
#include <QtWidgets/qdialog.h>
#include "catalystitem.h"
#include <QLabel>
// Конструктор
MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent), userInputDialog(new
UserInputDialog(this)) {
    setFixedSize(800, 600);
    requestUserName();
    selectMode();
```

```
    setupScene();
}
```

Инициализация главного окна

```
// Деструктор
MainWindow::~MainWindow() {
    delete userInputDialog;
    delete moleculeTimer;
    delete settlingTimer;
    delete cloudTimer;
    delete scene;
}
```

Освобождаются динамически созданные объекты: диалог, таймеры, сцена.

```
// Инициализация сцены
void MainWindow::setupScene() {
    scene = new QGraphicsScene(this);
    auto *view = new QGraphicsView(scene, this);
    view->setFixedSize(800, 600);
    scene->setSceneRect(0, 0, 800, 600);
    setCentralWidget(view);

    drawBackground();
    // Добавляем ввод температуры
    temperatureInput = new QLineEdit(this);
    temperatureInput->setPlaceholderText("Температура (320-420°C)");
    temperatureInput->setValidator(new QDoubleValidator(320, 430, 2, this)); // Ограничение на ввод
```

```
    temperatureInput->setGeometry(10, 40, 200, 30); // Позиция на главном окне

    // Добавляем ввод давления
    pressureInput = new QLineEdit(this);
    pressureInput->setPlaceholderText("Давление 100-667(КПа)");
    pressureInput->setValidator(new QDoubleValidator(100, 667, 2, this));  // Ограничение на ввод
    pressureInput->setGeometry(10, 80, 200, 30);
```

Поля для ввода температуры и давления QLineEdit: Однострочное текстовое поле.setPlaceholderText: Устанавливает текст-подсказку, видимый, когда поле пустое.QDoubleValidator: Ограничивает ввод допустимыми значениями (320–430 для температуры и 100–667 для давления).setGeometry: Задает позицию и размер элемента.

```
    // Кнопка подтверждения
    applyButton = new QPushButton("Применить", this);
    applyButton->setGeometry(10, 120, 150, 30);

    resetButton = new QPushButton("Перезапуск", this);
    resetButton->setGeometry(10, 160, 150, 30);
    connect(resetButton, &QPushButton::clicked, this, &MainWindow::resetAnimation);

    // Подключаем сигнал нажатия кнопки к слоту
    connect(applyButton, &QPushButton::clicked, this, &MainWindow::applyParameters);

// Облака
    for (int i = 0; i < 5; ++i) {
```

```
        int x = QRandomGenerator::global()->bounded(0, 800);
        int y = QRandomGenerator::global()->bounded(20, 100);
        QGraphicsEllipseItem *cloud = scene->addEllipse(x, y, 100, 50,
    QPen(Qt::NoPen), QBrush(Qt::white));
        clouds.append(cloud);
    }
```

Генерируются 5 случайно размещенных эллипсов, представляющих облака. QRandomGenerator: Создает случайные координаты в пределах сцены (по оси X: 0–800, по оси Y: 20–100).addEllipse: Добавляет эллипсы (100×50) с белым заливом и без границы.clouds.append(cloud): Сохраняет указатели на облака в списке для последующей анимации.

```
    // Резервуар
    reservoir = scene->addRect(300, 100, 200, 400, QPen(Qt::black),
    QBrush(Qt::lightGray));

    // Кран
    QBrush valveBrush(Qt::darkGray);
    valveBase = scene->addRect(50, 480, 40, 20, QPen(Qt::black),
    valveBrush);
    valveHandle = scene->addRect(60, 460, 20, 20, QPen(Qt::black),
    valveBrush);
    valveHandleTop = scene->addRect(50, 450, 40, 10,
    QPen(Qt::black), Qt::red);
    pipe = scene->addRect(90, 480, 210, 10, QPen(Qt::black),
    QBrush(Qt::darkGray));

    // Катализатор
    catalyst = new CatalystItem("CO + H2 -> CH3OH", nullptr);
```

```
    catalyst->setRect(500, 200, 40, 200); // Устанавливаем размер
    катализатора
    catalyst->setBrush(QBrush(Qt::blue)); // Цвет катализатора
    scene->addItem(catalyst);catalyst = new CatalystItem("Fe3 O4",
    nullptr);
    catalyst->setRect(500, 200, 40, 200); // Устанавливаем размер
    катализатора
    catalyst->setBrush(QBrush(Qt::blue)); // Цвет катализатора
    scene->addItem(catalyst);
```

Катализатор добавляется как пользовательский элемент CatalystItem.Устанавливаются: химическая реакция.Цвет заливки: синий.

```
// Таймеры
    moleculeTimer = new QTimer(this);
        connect(moleculeTimer, &QTimer::timeout, this,
    &MainWindow::updateMolecules);
    moleculeTimer->start(50);

    cloudTimer = new QTimer(this);
    connect(cloudTimer, &QTimer::timeout, this,
&MainWindow::animateClouds);
    cloudTimer->start(100);

    settlingTimer = new QTimer(this);
        connect(settlingTimer, &QTimer::timeout, this,
    &MainWindow::animateSettling);

    //  steamTimer = new QTimer(this);
```

```
// connect(steamTimer, &QTimer::timeout, this, &MainWindow::animateSteam);}
```

moleculeTimer: вызывает updateMolecules каждые 50 мс для обновления положения молекул.cloudTimer: вызывает animateClouds каждые 100 мс для анимации облаков.settlingTimer: управляет осаждением катализатора.

```
void MainWindow::resetAnimation() {
    // Очищаем молекулы из сцены
    for (auto *mol : molecules) {
        scene->removeItem(mol);
        delete mol;
    }
    molecules.clear();
```

Удалить все молекулы, хранящиеся в списке molecules, с графической сцены и из памяти.scene->removeItem(mol): Убирает объект mol из сцены.delete mol: Освобождает память, занимаемую объектом.molecules.clear(): Очищает список, чтобы он стал пустым.

```
    // Сбрасываем молекулы в процессе оседания
    for (auto *mol : settlingMolecules) {
        scene->removeItem(mol);
        delete mol;
    }
    settlingMolecules.clear();
```

Этот блок аналогичен предыдущему, но работает с молекулами, которые были добавлены в список settlingMolecules .

```
// Сбрасываем параметры
 parametersSet = false;
```

```
valveOpen = false;
valveDisabled = false;
catalystActive = false;
reactionStarted = false;
```

parametersSet: Указывает, что пользователь еще не задал параметры (температуру и давление).valveOpen: Кран закрыт.valveDisabled: Кран активен, его можно использовать.catalystActive: Катализатор не активен.reactionStarted: Реакция еще не началась.

```
    QMessageBox::information(this, "Перезапуск", "Анимация сброшена. Введите параметры и начните заново.");
}
// Запрос имени пользователя
void MainWindow::requestUserName() {
    connect(userInputDialog, &UserInputDialog::userCanceled, this, &MainWindow::close);
```

Подключение сигнала об отмене ввода. userInputDialog: Объект типа UserInputDialog, используемый для ввода имени.Сигнал userCanceled: Срабатывает, если пользователь отменил ввод. Слот close: Метод close() закрывает главное окно (MainWindow), завершая приложение.

```
    QString name = userInputDialog->getUserName(existingNames, this);
    if (name.isEmpty()) {
        return; // Пользователь отменил ввод
    }
```

userInputDialog->getUserName: Вызывает метод, отображающий диалог для ввода имени пользователя.existingNames: Список уже существующих имен, чтобы исключить дубли.this: Указатель на главное

окно, который можно использовать для настройки модального поведения диалога.

```
    userName = name;
    existingNames.insert(userName);
    QMessageBox::information(this, "Приветствие", "Добро
пожаловать, " + userName + "!");
}
```

userName = name: Сохраняет введенное имя в переменную userName, принадлежащую классу MainWindow. existingNames.insert(userName): Добавляет имя в набор existingNames для предотвращения дублирующего ввода.

```
// Выбор режима
void MainWindow::selectMode() {
    ModeSelectionDialog dialog(this);
```

Создаётся объект ModeSelectionDialog, который предоставляет пользователю интерфейс для выбора режима.this: Указатель на главное окно, задаёт родительский виджет для диалога.

```
 if (dialog.exec() == QDialog::Accepted) {
        selectedMode = dialog.getSelectedMode();
```

dialog.exec(): Отображает диалог в модальном режиме, блокируя другие действия до тех пор, пока пользователь не завершит взаимодействие с диалогом.Возвращает QDialog::Accepted, если пользователь подтвердил выбор (например, нажал кнопку “OK”).Возвращает QDialog::Rejected, если пользователь отменил выбор или закрыл окно.Если пользователь выбрал режим (QDialog::Accepted), выполняется блок кода внутри if.

```
        if (selectedMode == "Базовый") {
```

```
            QMessageBox::information(this, "Режим", "Каталитическая
        конверсия оксида углерода.");
        } else if (selectedMode == "Продвинутый") {
            enterReactionParameters();
        }
    }
```

Для режима “Базовый”:Показывается всплывающее окно с сообщением: *“Каталитическая конверсия оксида углерода.”*Это сигнализирует, что программа будет работать в упрощённом режиме без настройки дополнительных параметров.Для режима “Продвинутый”:Вызывается метод enterReactionParameters, который отвечает за ввод дополнительных параметров реакции.Это позволяет пользователю настроить параметры, такие как температура, давление или состав реагентов.

```
        else {
            close(); // Закрываем приложение, если пользователь отменил
        выбор
        }
    }

    void MainWindow::enterReactionParameters() {
        ReactionParametersDialog dialog(this);
```

Создаётся объект ReactionParametrsDialog, который предоставляет пользователю интерфейс для выбора режима.this: Указатель на главное окно, задаёт родительский виджет для диалога.

```
        if (dialog.exec() == QDialog::Accepted) {
            double T = dialog.getTemperature();
            double P = dialog.getPressure();
```

```
        double deltaH = 9420 + 3.16 * T - 8.314 * pow(10, -3) * pow(T,
    2) + 5.17 * pow(10, -6) * pow(T, 3) - 1.131 * pow(10, -9) * pow(T, 4);
        double logKp = -2.4198 + 0.0003855 * T + 2180.6 / T;

        QString result = QString("Температура (T): %1\n"
                                 "Давление (P): %2\n"
                                 "Тепловой эффект (ΔH): %3\n"
                                 "Константа равновесия (Kp): %4")
                             .arg(T)
                             .arg(P)
                             .arg(deltaH)
                             .arg(pow(10, logKp));
```

Преобразование log Kp в Kp путём возведения числа 10 в степень log Kp .

```
         // Показываем результаты пользователю
        QMessageBox::information(this, "Результаты", result);

        // Сохраняем результаты в файл
        QString fileName = QFileDialog::getSaveFileName(this,
    "Сохранить результаты", "", "Text Files (*.txt);;All Files (*)");
        if (!fileName.isEmpty()) {
            QFile file(fileName);
            if (file.open(QIODevice::WriteOnly | QIODevice::Text)) {
                QTextStream out(&file);
                out << result; // Записываем строку с результатами
                file.close();
                QMessageBox::information(this, "Успех", "Результаты
    успешно сохранены!");
```

```
        } else {
            QMessageBox::warning(this, "Ошибка", "Не удалось открыть файл для записи.");
        }
    }
}
```

```
// Отрисовка фона
void MainWindow::drawBackground() {
    // Небо
    scene->addRect(0, 0, 800, 500, QPen(Qt::NoPen), QBrush{"#4C98CA"});

    // Трава
    scene->addRect(0, 420, 800, 350, QPen(Qt::NoPen), QBrush{"#80B929"});

    // Почва с камнями
    scene->addRect(0, 500, 800, 250, QPen(Qt::NoPen), QBrush{"#7E4811"});
    for (int i = 0; i < 100; ++i) {
        int x = QRandomGenerator::global()->bounded(0, 800);
        int y = QRandomGenerator::global()->bounded(500, 600);
        scene->addEllipse(x, y, 5, 5, QPen(Qt::NoPen), QBrush(Qt::darkGray));
    }
```

Количество камней: 100.Координаты камней:x : Случайное значение от 0 до 800 (по ширине сцены).y : Случайное значение от 500 до 600 (в пределах почвы).Камень:Рисуется в виде эллипса размером 5 x 5 (почти круг).Используется тёмно-серый цвет (Qt::darkGray) для заливки.Границы эллипса отключены (Qt::NoPen).

}

// Геттеры и сеттеры
void MainWindow::setValveOpen(bool open) { valveOpen = open; }
bool MainWindow::isValveOpen() const { return valveOpen; }
Управление состоянием клапана

void MainWindow::setCatalystActive(bool active) { catalystActive = active; }
bool MainWindow::isCatalystActive() const { return catalystActive; }
Управление состоянием катализатора:

void MainWindow::setReactionStarted(bool started) { reactionStarted = started; }
bool MainWindow::isReactionStarted() const { return reactionStarted; }
Управление состоянием реакции

void MainWindow::setValveDisabled(bool disabled) { valveDisabled = disabled; }
bool MainWindow::isValveDisabled() const { return valveDisabled; }
Управление состоянием отключения клапана:

```
void MainWindow::animateClouds() {
    for (auto *cloud : clouds) {
```

Переменная clouds — это список (QList или аналог), содержащий указатели на графические элементы типа QGraphicsEllipseItem, представляющие облака.Цикл обходит каждое облако для выполнения одинаковых действий.

```
// Двигаем облако вправо
        cloud->moveBy(2, 0); // Движение на 2 пикселя вправо

        // Если облако вышло за правую границу, перемещаем его налево
        if (cloud->x() > 800) {
            cloud->setX(-100); // Возвращаем облако на левую сторону
        }
    }
}
```

Обеспечить плавное движение облаков справа налево. Если облако выходит за пределы правой границы сцены, оно возвращается на левую сторону, создавая эффект “вечного” перемещения облаков.

```
void MainWindow::mousePressEvent(QMouseEvent *event) {
    QPointF scenePos = centralWidget()->mapFromGlobal(event->globalPosition().toPoint());
    scenePos = static_cast<QGraphicsView*>(centralWidget())->mapToScene(scenePos.toPoint());
```

event->globalPosition() возвращает глобальные координаты экрана, где произошел клик.mapFromGlobal() преобразует глобальные координаты в координаты внутри виджета центрального окна

(centralWidget()).mapToScene() конвертирует эти координаты в координаты сцены (для работы с элементами QGraphicsScene).

```
    // Проверяем нажатие на любую часть крана
    if (valveBase->contains(scenePos - valveBase->pos()) ||
        valveHandle->contains(scenePos - valveHandle->pos()) ||
        valveHandleTop->contains(scenePos - valveHandleTop->pos()))
{
        toggleValve();
    }
```

Проверить, кликнул ли пользователь на любую часть крана.valveBase, valveHandle, valveHandleTop — это графические элементы крана.Для каждой части крана выполняется проверка:scenePos - valveBase->pos(): преобразует координаты клика в локальные координаты относительно элемента.valveBase->contains(...): возвращает true, если координаты клика попали в область этого элемента.Если клик попал на любую часть крана, вызывается метод toggleValve() для переключения состояния крана.

```
    // Проверяем нажатие на катализатор
    if (catalyst->contains(scenePos - catalyst->pos())) {
        activateCatalyst();
    }
}
```

Логика аналогична проверке крана

```
void MainWindow::updateMolecules() {
    if (!valveOpen || !parametersSet) return; // Не начинаем анимацию
без параметров
```

Анимация молекул запускается только при выполнении двух условий:Кран открыт: флаг valveOpen должен быть установлен в true.Параметры заданы: флаг parametersSet указывает, что параметры (например, температура и давление) введены пользователем.
Если любое из условий не выполнено, метод завершает выполнение.

```
// Добавление новой молекулы, если кран открыт
   int y = QRandomGenerator::global()->bounded(480, 485); // Позиция в трубе
   auto *molecule = scene->addEllipse(90, y, 5, 5, QPen(Qt::black), QBrush(Qt::red));
molecules.append(molecule);
```

Создается новая молекула в виде маленького эллипса (красного круга):Координата x: 90 (начальная позиция в трубе).Координата y: случайное значение в диапазоне [480, 485], чтобы молекулы появлялись с небольшим разбросом в трубе.Размер: 5x5.Цвет: красный Молекула добавляется на сцену с помощью scene->addEllipse и сохраняется в контейнере molecules.

```
// Движение молекул
for (auto *mol : molecules) {
   if (mol->x() < 200) {
      // Движение вправо по трубе
      mol->moveBy(5, 0);
   } else {
      // Если молекула в резервуаре, продолжаем двигать её там
      placeMoleculeInReservoir(mol);
   }
}
```

Каждая молекула из списка molecules проверяется:Если координата X молекулы меньше 200, она движется вправо вдоль трубы:mol->moveBy(5, 0) — смещение молекулы на 5 пикселей вправо.Если молекула достигла конца трубы (X >= 200), она передается в метод placeMoleculeInReservoir.

```
void MainWindow::toggleValve() {
    if (!parametersSet) {
        QMessageBox::warning(this, "Ошибка", "Пожалуйста, примените параметры перед использованием крана.");
        return;
    }
```

Метод отвечает за управление состоянием крана. Он включает или выключает кран, а также выполняет соответствующие действия, такие как изменение цвета крана, удаление молекул из трубы и блокировка крана после выключения. Перед переключением крана проверяется, были ли установлены параметры (parametersSet).Если параметры не заданы, выводится предупреждение, и метод завершает выполнение

```
 if (valveDisabled) {
        QMessageBox::warning(this, "Кран заблокирован", "Кран уже был выключен и не может быть включён снова!");
        return;
    }
```

Если кран был заблокирован ранее (valveDisabled == true), то его повторное включение невозможно.Выводится сообщение о блокировке, и метод завершает выполнение.

```
    valveOpen = !valveOpen;
```

```
QBrush brush = valveOpen ? QBrush(Qt::green) : QBrush(Qt::darkGray);

valveBase->setBrush(brush);
valveHandle->setBrush(brush);
valveHandleTop->setBrush(brush);
```

Состояние крана (valveOpen) меняется на противоположное.Если кран был закрыт, он открывается.Если кран был открыт, он закрывается.Цвет крана изменяется:Зелёный (Qt::green) для открытого крана.Тёмно-серый (Qt::darkGray) для закрытого крана.Цвет применяется к элементам графического объекта крана: основанию, ручке и верхней части.

```
if (!valveOpen) {
    // Удаляем молекулы из трубы
    auto it = molecules.begin();
    while (it != molecules.end()) {
        QGraphicsEllipseItem *mol = *it;
        if (mol->x() < 300) { // Проверяем, находится ли молекула в трубе
            scene->removeItem(mol);
            it = molecules.erase(it);
            delete mol;
        } else {
            ++it;
        }
    }

    // Блокируем кран после выключения
    valveDisabled = true;
}
```

}

Если кран закрывается (!valveOpen):Удаление молекул из трубы:Перебираются все молекулы из списка molecules.Если координата X молекулы меньше 300 (молекула находится в трубе):Молекула удаляется со сцены с помощью scene->removeItem.Указатель на молекулу удаляется из списка molecules.Указатель освобождается (delete mol).Блокировка крана:Устанавливается флаг valveDisabled = true, чтобы предотвратить повторное включение крана.

void MainWindow::activateCatalyst() {

Метод MainWindow::activateCatalyst() отвечает за активацию катализатора, выполнение предварительных проверок, изменение его визуального состояния и запуск процесса реакции.

```
    if (valveOpen) {
        QMessageBox::warning(this, "Ошибка", "Закройте кран перед активацией катализатора.");
        return;
    }
```

Если кран открыт (valveOpen == true), активация катализатора запрещена.Пользователь получает предупреждение с просьбой закрыть кран.Метод завершается без выполнения оставшегося кода.

```
    if (!parametersSet) {
        QMessageBox::warning(this, "Ошибка", "Пожалуйста, примените параметры перед активацией катализатора.");
        return;
    }
```

Если параметры реакции не были установлены (parametersSet == false), активация катализатора невозможна.Пользователь получает предупреждение о необходимости установки параметров.Метод завершает выполнение.

```
    catalystActive = true;
    catalyst->setBrush(Qt::red);
```

Флаг catalystActive устанавливается в true, сигнализируя, что катализатор активен.Цвет катализатора изменяется на красный (Qt::red), чтобы визуально показать его активацию.

```
    startReaction(); // Начинаем реакцию и осаждение молекул
    generateSteam(); // Генерируем молекулы, которые осаждаются
}
void MainWindow::placeMoleculeInReservoir(QGraphicsEllipseItem
*molecule) {
    static QVector<QPoint> freeCells;
```

Вектор в котором лежат свободные ячейки, в которых могут находиться молекулы(их координаты).

```
// Вычисление размера молекулы на основе давления
    int moleculeSize = qBound(5, static_cast<int>(pressure) / 10, 20); //
Размер молекулы (диаметр), ограничение от 5 до 20
 int cellSize = moleculeSize; // Размер ячейки равен размеру
```

молекулы

pressure — текущее значение давления, влияющее на размер молекулы.qBound ограничивает диаметр молекулы между минимальным (5) и максимальным (20) значениями.cellSize — размер ячейки для молекулы равен её размеру, чтобы она полностью помещалась в одну ячейку.

```
// Границы резервуара
int reservoirLeft = 300;
int reservoirTop = 100;
int reservoirWidth = 200;
int reservoirHeight = 400;

// Количество строк и столбцов
int rows = reservoirHeight / cellSize;
int cols = reservoirWidth / cellSize;

// Инициализация свободных ячеек
if (freeCells.isEmpty()) {
    for (int y = 0; y < rows; ++y) {
        for (int x = 0; x < cols; ++x) {
            freeCells.append(QPoint(reservoirLeft + x * cellSize,
                                    reservoirTop + y * cellSize));
        }
    }
}
```

Если вектор пуст, происходит заполнение его координатами всех возможных ячеек.Координаты рассчитываются на основе позиции резервуара (reservoirLeft, reservoirTop) и размера ячеек.

```
    // Проверяем, остались ли свободные ячейки
    if (freeCells.isEmpty()) {
     return; // Все места заняты, не размещаем новую молекулу
      }

      // Берём случайную свободную ячейку
```

```
    int randomIndex = QRandomGenerator::global()->bounded(freeCells.size());
    QPoint position = freeCells[randomIndex];

    // Удаляем выбранную ячейку из списка
    freeCells.removeAt(randomIndex);
```

Случайно выбирается индекс из списка доступных ячеек.Выбранная ячейка удаляется из списка, чтобы избежать повторного использования.

```
    // Устанавливаем молекулу в центр ячейки
    molecule->setRect(0, 0, moleculeSize, moleculeSize); // Устанавливаем размер молекулы
    molecule->setPos(position.x(), position.y());
}
```

Молекуле задаётся размер (setRect) и позиция (setPos).Размер: определяется ранее вычисленным moleculeSize.Позиция: центр молекулы совпадает с центром выбранной ячейки.

```
void MainWindow::startReaction() {
    if (!catalystActive || reactionStarted) return;
```

Реакция начинается только если:Катализатор активирован **(**catalystActive**)**.Реакция ещё не запущена **(**!reactionStarted**)**.Это предотвращает повторный запуск реакции или её старт без активации катализатора.

```
    reactionStarted = true;

    // Удаляем молекулы, находящиеся в трубе
    auto it = molecules.begin(); // Обычный итератор
    while (it != molecules.end()) {
```

```
        QGraphicsEllipseItem *mol = *it;
        if (mol->x() < 300) { // Если молекула находится в трубе
            scene->removeItem(mol);   // Удаляем элемент из сцены
            it = molecules.erase(it); // Удаляем элемент из QVector и
    обновляем итератор
            delete mol;              // Освобождаем память
        } else {
            ++it; // Переход к следующему элементу
        }
    }
```

Итератор используется для перебора молекул в molecules.Если молекула находится в трубе (ее координата x < 300):Она удаляется со сцены (scene->removeItem).Удаляется из вектора molecules (erase).Освобождается память (delete mol).Молекулы, не попавшие под условие, пропускаются.

```
    // Оставшиеся молекулы добавляем в список для оседания
    for (auto *mol : molecules) {
        settlingMolecules.append(mol);
      }

      molecules.clear();
```

Молекулы, не удалённые на предыдущем этапе, перемещаются в список settlingMolecules для последующего осаждения.После этого molecules очищается, чтобы предотвратить дублирование.

```
    // Запускаем таймер оседания
    settlingTimer->start(50); // Обновление каждые 50 мс
}
```

```
void MainWindow::setTemperature(int temp) {
    temperature = temp;

    // Изменяем скорость молекул
    int newInterval = 10000 / temperature; // Пример зависимости
    moleculeTimer->setInterval(newInterval);
```

Зависимость интервала от температуры:Интервал обновления таймера уменьшается с ростом температуры.Чем выше температура, тем быстрее молекулы двигаются (аналог термодинамической модели).Если температура слишком мала, деление может привести к чрезмерно большому значению интервала, что потребует проверки:

```
    // Меняем цвет катализатора
    QColor catalystColor = QColor::fromHsv(qMin(temperature / 4.0, 255.0), 255, 255);
    catalyst->setBrush(catalystColor);
}

void MainWindow::setPressure(int pres) {
    pressure = pres;

    // Обновляем молекулы в резервуаре
    int moleculeSize = qBound(5, static_cast<int>(pressure) / 10, 20); // Новый размер молекулы (ограничен от 5 до 20)
    for (auto *molecule : molecules) {
        molecule->setRect(molecule->x(), molecule->y(), moleculeSize, moleculeSize);
    }
```

Размер молекулы пропорционален давлению (pressure / 10) с ограничениями:Минимальный размер: 5.Максимальный размер:

20.qBound гарантирует, что значение остаётся в пределах допустимого диапазона.Применяется к каждой существующей молекуле через цикл.

```
    // Добавляем или удаляем молекулы, чтобы их количество соответствовало давлению
    int moleculeCount = pressure / 10; // Пример зависимости
    while (molecules.size() < moleculeCount) {
        auto *molecule = scene->addEllipse(90, QRandomGenerator::global()->bounded(480, 485), moleculeSize, moleculeSize, QPen(Qt::black), QBrush(Qt::red));
        molecules.append(molecule);
    }
```

Если текущее количество молекул меньше необходимого (molecules.size() < moleculeCount):Создаётся новая молекула с указанным размером и случайной координатой y в пределах трубы.Молекула добавляется в сцену и список molecules.

```
    while (molecules.size() > moleculeCount) {
        QGraphicsEllipseItem *molecule = molecules.takeLast();
        scene->removeItem(molecule);
        delete molecule;
    }
}
```

Если текущее количество молекул превышает требуемое (molecules.size() > moleculeCount):Последняя молекула удаляется из списка и сцены.Память освобождается вызовом delete.

```
void MainWindow::animateSettling() {
```

Метод MainWindow::animateSettling управляет анимацией оседания молекул в резервуаре. Он обрабатывает каждую

молекулу в списке settlingMolecules, двигает их вниз и фиксирует на дне резервуара.

```
    if (settlingMolecules.isEmpty()) {
        settlingTimer->stop();
        return;
    }

    // Границы резервуара
    const int reservoirBottom = 500; // Нижняя граница резервуара
    const int reservoirLeft = 300;  // Левая граница резервуара
    const int reservoirRight = 500; // Правая граница резервуара
    const int moleculeSize = 10;    // Размер молекулы (диаметр)

    for (int i = 0; i < settlingMolecules.size(); ++i) {
        QGraphicsEllipseItem *molecule = settlingMolecules[i];

        // Убираем молекулы, которые вышли за пределы резервуара
        if (molecule->x() < reservoirLeft || molecule->x() > reservoirRight
|| molecule->y() > reservoirBottom) {
            scene->removeItem(molecule);
            settlingMolecules.removeAt(i);
            delete molecule;
            --i;
            continue;
        }
```

Если молекула выходит за границы резервуара, она:Удаляется из сцены.Удаляется из списка settlingMolecules.Уничтожается (delete).Индекс уменьшается на 1 (--i), чтобы не пропустить следующую молекулу из-за сдвига индексов.

```
            // Проверяем, достигла ли молекула дна
            double molBottomY = molecule->y() + moleculeSize;

            if (molBottomY < reservoirBottom) {
                // Если молекула выше дна, опускаем её вниз
                molecule->moveBy(0, 5);
            } else {
                // Молекула достигла дна, фиксируем её положение
                molecule->setY(reservoirBottom - moleculeSize);
                molecule->setBrush(Qt::blue); // Меняем цвет, чтобы
обозначить осаждение
            }
        }
```

Молекула перемещается вниз по 5 пикселей за цикл.Если она достигает дна (molBottomY >= reservoirBottom):Её положение фиксируется точно над дном (reservoirBottom - moleculeSize).Цвет изменяется на синий (Qt::blue), чтобы обозначить завершение оседания.

```
        // Если все молекулы осели, остановим таймер
        if (settlingMolecules.isEmpty()) {
            settlingTimer->stop();
        }
    }
    void MainWindow::applyParameters() {
        bool tempOk, presOk;
        double temp = temperatureInput->text().toDouble(&tempOk);
        double pres = pressureInput->text().toDouble(&presOk);

        if (!tempOk || !presOk) {
```

```
        QMessageBox::warning(this, "Ошибка ввода", "Пожалуйста, введите корректные значения температуры и давления.");
        return;
    }

    if (temp < 320 || temp > 430) {
        QMessageBox::warning(this, "Ошибка", "Температура должна быть в диапазоне 320–430 °C.");
        return;
    }

    if (pres < 100 || pres > 667) {
        QMessageBox::warning(this, "Ошибка", "Давление должно быть в диапазоне 100–667 кПа.");
        return;
    }
```

Проверяется, входит ли температура в диапазон 320–430 °C.Проверяется, входит ли давление в диапазон 100–667 кПа.Если значения выходят за пределы допустимых диапазонов, отображается предупреждение, и метод завершает выполнение.

```
    // Обновляем параметры
    temperature = temp;
    pressure = pres;
    parametersSet = true;

    // Обновляем состояние катализатора (например, цвет)
    updateCatalystColor();
    QMessageBox::information(this, "Параметры",
```

```
                    QString("Температура: %1 °C\nДавление: %2
кПа").arg(temperature).arg(pressure));
}

void MainWindow::updateCatalystColor() {
   // Изменяем цвет катализатора в зависимости от температуры
   QColor catalystColor = QColor::fromHsv(qMin(temperature / 4.0,
255.0), 255, 255);
   if (catalyst) {
      catalyst->setBrush(catalystColor);
   }
}
void MainWindow::generateSteam() {
   const int steamWidth = 10;
   const int steamHeight =25;

   // Параметры пара
   const int reservoirTop = 150;
   const int reservoirLeft = 300;
   const int reservoirWidth = 200;
```

Задаются параметры пара

```
   for (int i = 0; i < 10; ++i) { // Создаём несколько элементов пара
за раз
      int x = QRandomGenerator::global()->bounded(reservoirLeft,
reservoirLeft + reservoirWidth - steamWidth);
      auto *steam = scene->addEllipse(x, reservoirTop, steamWidth,
steamHeight, QPen(Qt::NoPen), QBrush(Qt::white));
      steamParticles.append(steam);
   }
```

Цикл генерирует 10 частиц пара за вызов.Для каждой частицы:Случайное положение по оси X в пределах резервуара.Верхняя граница резервуара (reservoirTop) задаёт начальную позицию по оси Y.Создаётся эллипс белого цвета и добавляется на scene.Указатель на созданный объект сохраняется в список steamParticles для последующей анимации.

```
    if (!steamTimer) {
        steamTimer = new QTimer(this);
        connect(steamTimer, &QTimer::timeout, this,
&MainWindow::animateSteam);
        steamTimer->start(200);
    }
}
void MainWindow::updateMoleculeSpeed() {
    if (temperature > 0) {
        int newInterval = qMax(1000 / static_cast<int>(temperature), 10);
// Приводим к int
```

Деление 1000 / temperature определяет интервал между обновлениями.Чем выше температура, тем меньше интервал, следовательно, движение быстрее.Используется qMax, чтобы минимальный интервал был не менее 10 мс, что предотвращает чрезмерно частые вызовы таймера.

```
        moleculeTimer->setInterval(newInterval);
    }
}
void MainWindow::animateSteam() {
    for (int i = 0; i < steamParticles.size(); ++i) {
        QGraphicsEllipseItem *steam = steamParticles[i];
```

```
            steam->moveBy(0, -1); // Двигаем пар вверх
```

Каждая частица представлена как QGraphicsEllipseItem и хранится в списке steamParticles.

```
            if (steam->y() + steam->rect().height() < 0) {
                // Убираем частицы, которые вышли за пределы экрана
                scene->removeItem(steam);
                steamParticles.removeAt(i);
                delete steam;
                --i;
            }
        }
```

Проверяется, вышла ли верхняя граница частицы (y + height) за пределы экрана (< 0).Удаляются из:Сцены (scene->removeItem),Списка (steamParticles.removeAt(i)),Памяти (delete steam).Индекс i уменьшается, чтобы компенсировать удаление из списка.

```
        // Останавливаем таймер, если все частицы пара исчезли
        if (steamParticles.isEmpty()) {
            steamTimer->stop();
            delete steamTimer;
            steamTimer = nullptr;
        }
    }
```

Таймер останавливается, когда все частицы удалены.Удаляется сам объект steamTimer, чтобы избежать утечек памяти.

## ДЕМОНСТРАЦИЯ РАБОТЫ ПРОГРАММЫ С ОПИСАНИЕМ(СКРИНШОТЫ).

При запуске программы открывается окно (рис. 1):

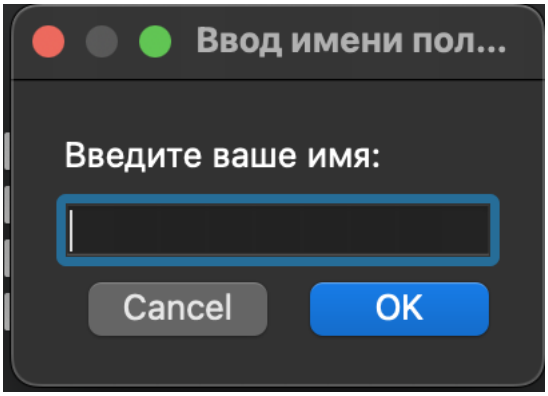



Рис. 1

При нажатии кнопки «ok» (рис. 2):

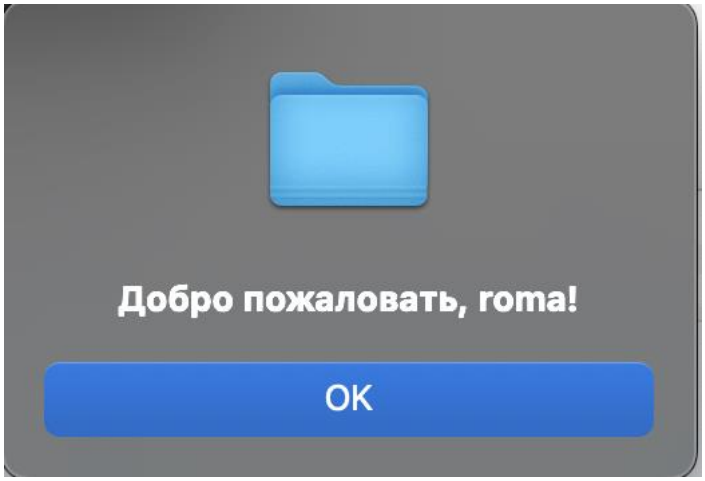



Рис. 2

При выборе режима работы (рис. 3):

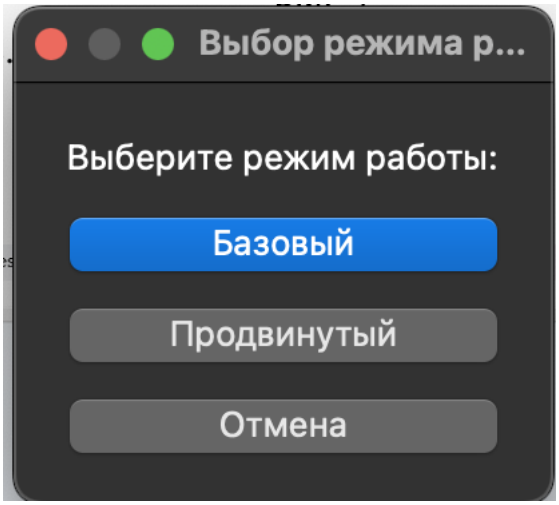
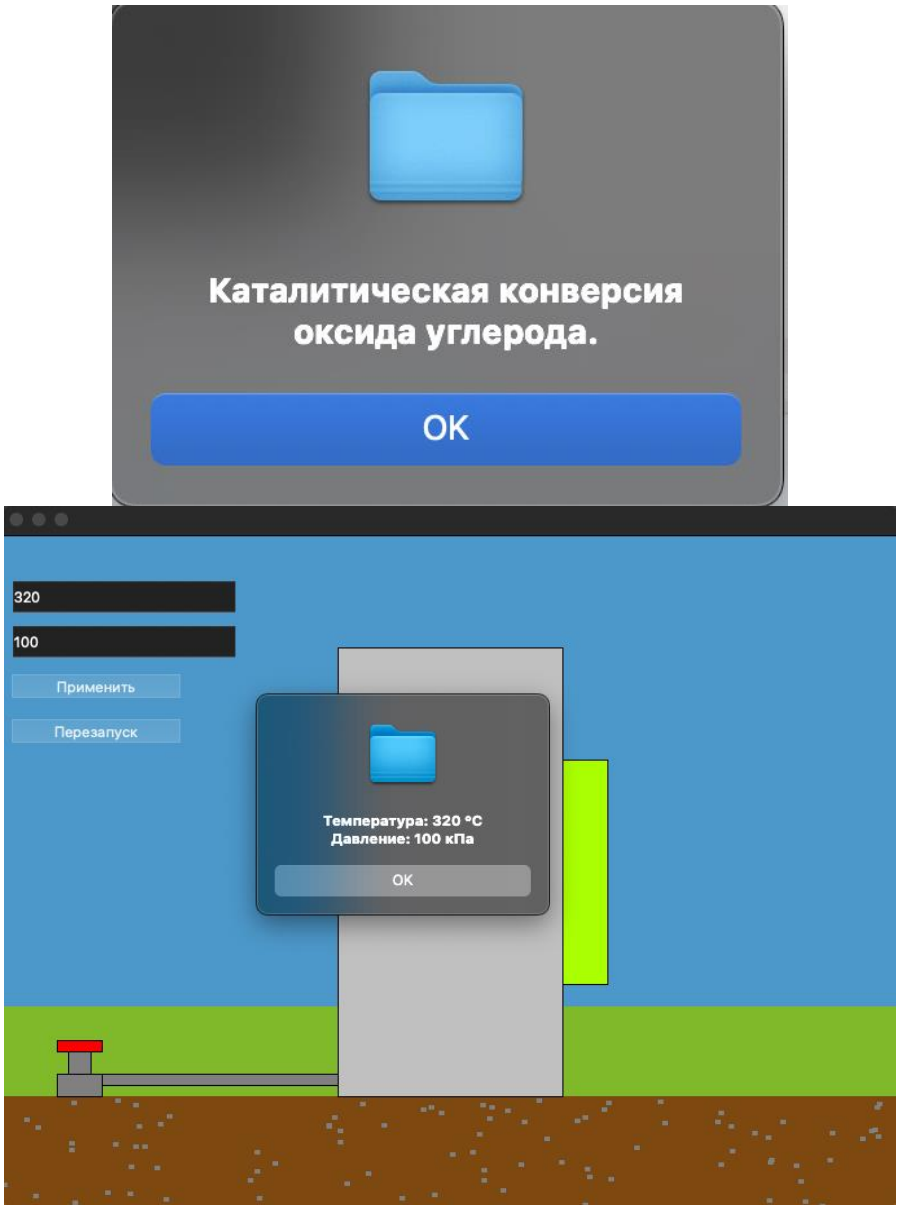



Рис. 3

При выборе «Базовый» (рис. 4):



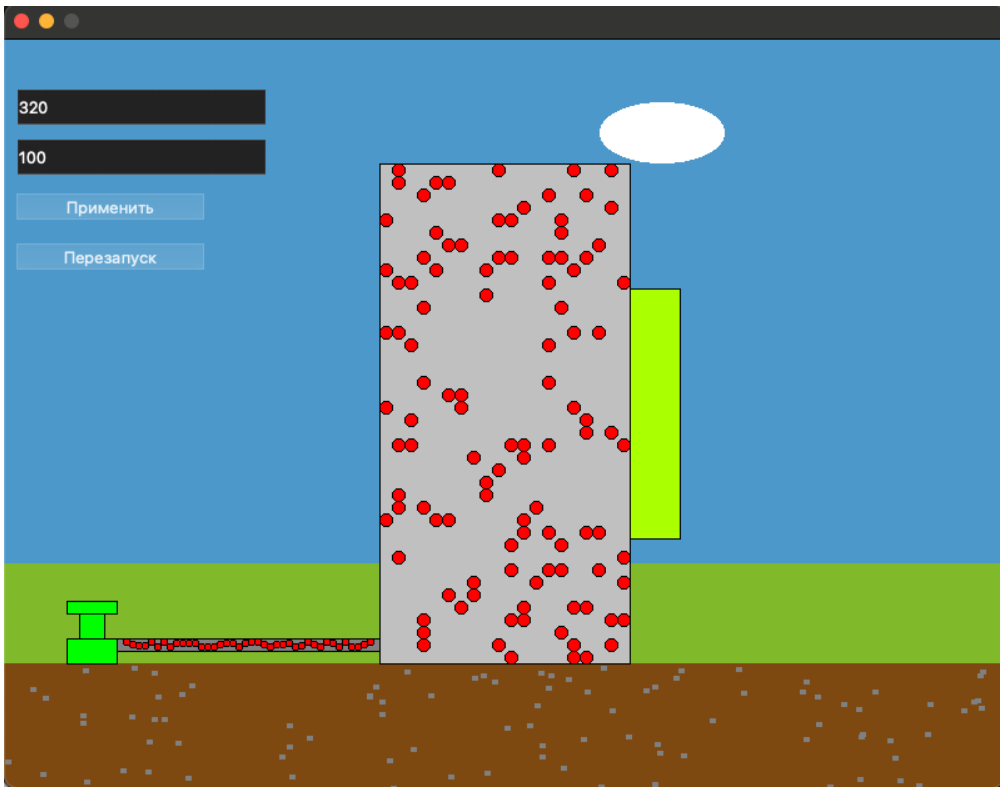



Рис. 4

При выборе «Продвинутый» (рис. 5):

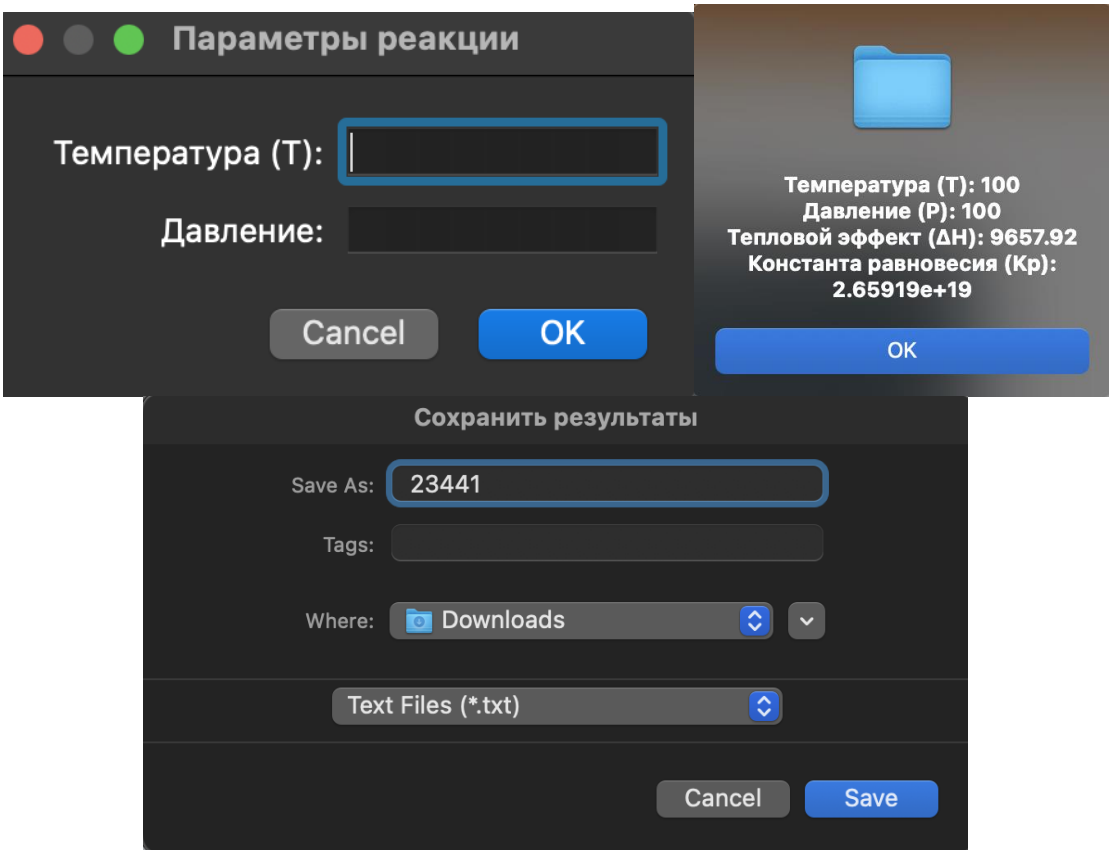



Рис. 5

При нажатии на кран (рис. 6):

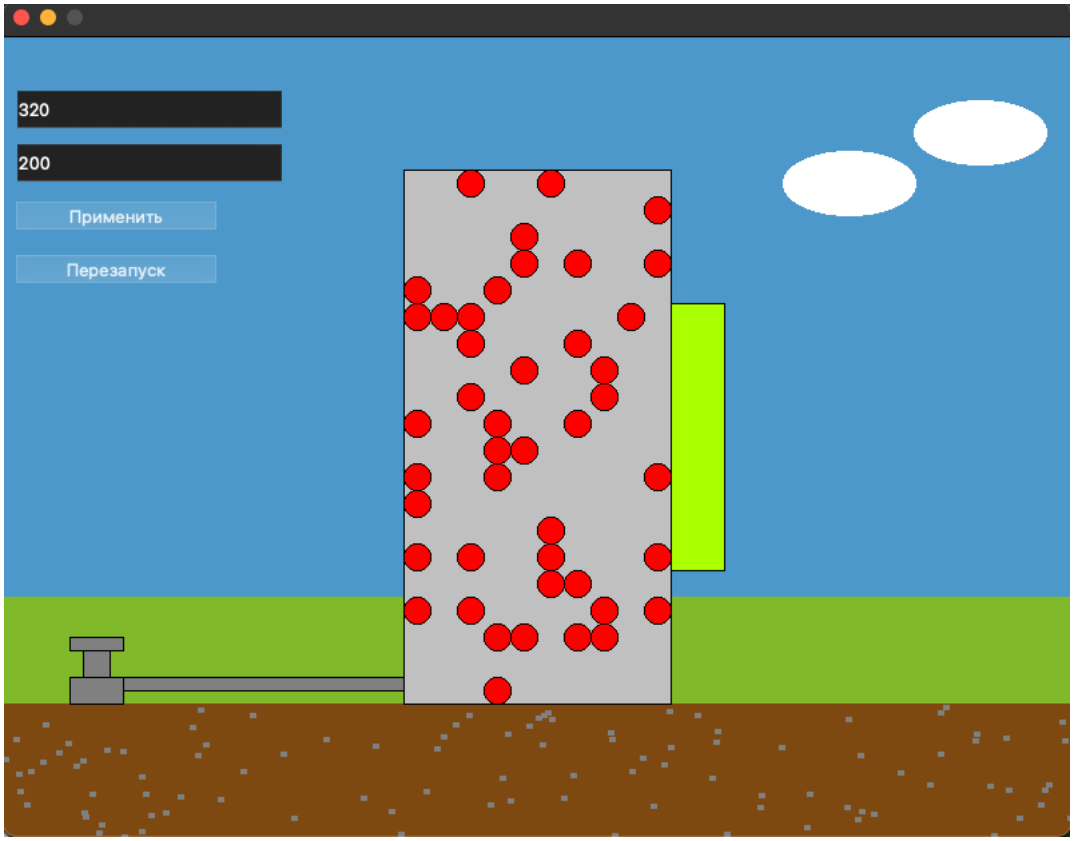



Рис. 6

При нажатии на катализатор(рис. 7):

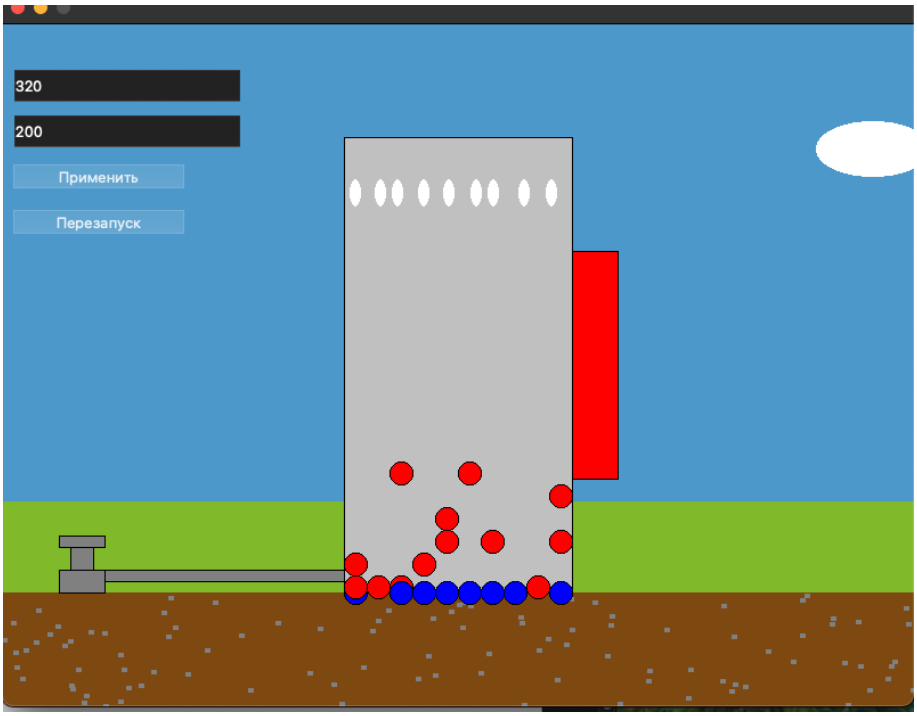



Рис. 7

## ВЫВОДЫ.

С помощью графического представления данной программы становится понятно, будет ли эффективен образец при заданных параметрах. В процессе работы была продемонстрирована возможность применения широкого спектра функциональных возможностей фреймворка Qt версии 6.8.0. Были использованы элементы библиотеки Qt Widgets для создания пользовательского интерфейса.

Успешная реализация проекта подтверждает эффективность использования Qt для разработки кроссплатформенных приложений. Несмотря на то, что Qt 6.8.0 является устаревшей версией, данная работа показала ее способность решать поставленные задачи и обеспечивать достаточный уровень функциональности. Работа достигла поставленных целей и продемонстрировала глубокое понимание принципов программирования на Qt 4.8.6.

## СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ


1. Научные основы приготовления катализаторов. Творческое наследие и дальнейшее развитие работ профессора И.П. Кириллова. Монография / Под ред. А.П. Ильина. - Иваново: ГОУ ВПО Иван. госуд. химико-технол. ун-т. 2008. - 156 с.
2. Обысов А.В., Дульнев А.В. Промышленный катализ в газохимии. Монография / Под ред. д.т.н. С.В. Афанасьева. - Самара: Изд-во Сам. научн. центра РАН. 2018. - 160 с.

3. Цалко Е.В., Михеев А.А. Каталитические процессы в производстве аммиака. Стадия конверсии оксида углерода (II) // Вестник Кузбасского государственного. техн. ун-та. - 2010. - № 5. - С. 126-128.
4. Научные основы приготовления катализаторов. Творческое наследие и дальнейшее развитие работ профессора И.П. Кириллова. Монография / Под ред. А.П. Ильина. - Иваново: ГОУ ВПО Иван. госуд. химико-технол. ун-т. 2008. - 156 с.

## ПРИЛОЖЕНИЯ

Catalystitem.h

```
#ifndef CATALYSTITEM_H
#define CATALYSTITEM_H

#include <QGraphicsRectItem>
#include <QGraphicsTextItem>

class CatalystItem : public QGraphicsRectItem {
public:
explicit CatalystItem(const QString &formula, QGraphicsItem *parent =
nullptr);

protected:
void hoverEnterEvent(QGraphicsSceneHoverEvent *event) override;
void hoverLeaveEvent(QGraphicsSceneHoverEvent *event) override;

private:
QGraphicsTextItem *formulaText;
};
```

```
#endif // CATALYSTITEM_H
```

Catalystitem.cpp

```
#include "catalystitem.h"
#include <QGraphicsSceneHoverEvent>

CatalystItem::CatalystItem(const QString &formula, QGraphicsItem *parent)
    : QGraphicsRectItem(parent), formulaText(new QGraphicsTextItem(formula, this)) {
    setAcceptHoverEvents(true); // Разрешить обработку событий наведения
    formulaText->setDefaultTextColor(Qt::black); // Цвет текста
    formulaText->setVisible(false); // Скрываем текст по умолчанию

    // Размещаем текст рядом с катализатором
    formulaText->setPos(rect().width() + 5, 0);
}

void CatalystItem::hoverEnterEvent(QGraphicsSceneHoverEvent *event) {
    formulaText->setVisible(true); // Показываем текст при наведении
    QGraphicsRectItem::hoverEnterEvent(event); // Вызываем базовую реализацию
}

void CatalystItem::hoverLeaveEvent(QGraphicsSceneHoverEvent *event) {
    formulaText->setVisible(false); // Скрываем текст при уходе мыши
```

```
    QGraphicsRectItem::hoverLeaveEvent(event); // Вызываем базовую реализацию
}
```

Userinputdialog.h

```
#include <QObject>
#include <QString>
#include <QSet>

class UserInputDialog : public QObject {
    Q_OBJECT

public:
    explicit UserInputDialog(QObject *parent = nullptr);
    QString getUserName(const QSet<QString> &existingNames, QWidget *parent);

private:
    QString validateUserName(const QString &name, const QSet<QString> &existingNames);

signals:
    void userCanceled(); // Сигнал для уведомления о том, что пользователь отменил ввод
```

```
};
```

Userinputdialog.cpp

```
#include "userinputdialog.h"
#include <QInputDialog>
#include <QMessageBox>
#include <QRegularExpression>

UserInputDialog::UserInputDialog(QObject *parent)
    : QObject(parent) {}

QString UserInputDialog::getUserName(const QSet<QString> &existingNames, QWidget *parent) {
    QString name;
    bool ok;

    do {
        name = QInputDialog::getText(parent, "Ввод имени пользователя",
                                     "Введите ваше имя:",
                                     QLineEdit::Normal, "", &ok);

        if (!ok) {
            emit userCanceled();
```

```
            return QString(); // Возвращаем пустую строку, если пользователь отменил ввод
        }

        QString validationError = validateUserName(name, existingNames);
        if (!validationError.isEmpty()) {
            QMessageBox::warning(parent, "Ошибка", validationError);
        } else {
            break;
        }
    } while (true);

    return name;
}

QString UserInputDialog::validateUserName(const QString &name, const QSet<QString> &existingNames) {
    if (name.isEmpty()) {
        return "Имя не может быть пустым!";
    }
    if (name.contains(QRegularExpression("[^a-zA-Za-яА-Я0-9_ ]"))) {
        return "Имя содержит недопустимые символы!";
    }
```

```
    if (existingNames.contains(name)) {
        return "Это имя уже используется!";
    }
    return QString(); // Нет ошибок
}
```

Reactionparametrsdialog.h

```
#ifndef REACTIONPARAMETERSDIALOG_H
#define REACTIONPARAMETERSDIALOG_H

#include <QDialog>
#include <QString>

class QLineEdit;

class ReactionParametersDialog : public QDialog {
    Q_OBJECT

public:
    explicit ReactionParametersDialog(QWidget *parent = nullptr);
    ~ReactionParametersDialog();

    double getTemperature() const; // Получение температуры
    double getPressure() const;    // Получение давления
```

```
private:
    QLineEdit *tempInput;          // Поле ввода температуры
    QLineEdit *pressureInput;      // Поле ввода давления
};

#endif // REACTIONPARAMETERSDIALOG_H
```

Reactionparametrsdialog.cpp

```
#include "reactionparametersdialog.h"
#include <QFormLayout>
#include <QLineEdit>
#include <QDialogButtonBox>
#include <QVBoxLayout>

ReactionParametersDialog::ReactionParametersDialog(QWidget *parent)
    : QDialog(parent), tempInput(new QLineEdit(this)), pressureInput(new QLineEdit(this)) {
    setWindowTitle("Параметры реакции");

    // Создаем форму ввода
    QFormLayout *formLayout = new QFormLayout;
    formLayout->addRow("Температура (T):", tempInput);
    formLayout->addRow("Давление:", pressureInput);
```

```
    // Кнопки "ОК" и "Отмена"
    QDialogButtonBox *buttonBox = new QDialogButtonBox(QDialogButtonBox::Ok | QDialogButtonBox::Cancel, this);
    connect(buttonBox, &QDialogButtonBox::accepted, this, &QDialog::accept);
    connect(buttonBox, &QDialogButtonBox::rejected, this, &QDialog::reject);

    // Общий макет
    QVBoxLayout *layout = new QVBoxLayout(this);
    layout->addLayout(formLayout);
    layout->addWidget(buttonBox);

    setLayout(layout);
}

ReactionParametersDialog::~ReactionParametersDialog() = default;

double ReactionParametersDialog::getTemperature() const {
    return tempInput->text().toDouble();
}
```

```
double ReactionParametersDialog::getPressure() const {
    return pressureInput->text().toDouble();
}
```

Modeselectiondialog.h

```
#ifndef MODESELECTIONDIALOG_H
#define MODESELECTIONDIALOG_H

#include <QDialog>
#include <QString>
#include <QPushButton>

class ModeSelectionDialog : public QDialog {
    Q_OBJECT

public:
    explicit ModeSelectionDialog(QWidget *parent = nullptr);
    QString getSelectedMode() const;

private:
    QString selectedMode;
    QPushButton *basicModeButton;
    QPushButton *advancedModeButton;
```

```
private slots:
    void onBasicModeSelected();
    void onAdvancedModeSelected();
};

#endif // MODESELECTIONDIALOG_H
```

Modeselectiondialog.cpp

```
#include "modeselectiondialog.h"
#include <QVBoxLayout>
#include <QLabel>
#include <QMessageBox>

ModeSelectionDialog::ModeSelectionDialog(QWidget *parent)
    : QDialog(parent), selectedMode("") {

    setWindowTitle("Выбор режима работы");

    auto *layout = new QVBoxLayout(this);
    auto *label = new QLabel("Выберите режим работы:", this);
    layout->addWidget(label);

    basicModeButton = new QPushButton("Базовый", this);
    advancedModeButton = new QPushButton("Продвинутый", this);
```

```
    auto *cancelButton = new QPushButton("Отмена", this);

    layout->addWidget(basicModeButton);
    layout->addWidget(advancedModeButton);
    layout->addWidget(cancelButton);

    connect(basicModeButton, &QPushButton::clicked, this, &ModeSelectionDialog::onBasicModeSelected);
    connect(advancedModeButton, &QPushButton::clicked, this, &ModeSelectionDialog::onAdvancedModeSelected);
    connect(cancelButton, &QPushButton::clicked, this, &QDialog::reject);
}

QString ModeSelectionDialog::getSelectedMode() const {
    return selectedMode;
}

void ModeSelectionDialog::onBasicModeSelected() {
    selectedMode = "Базовый";
    accept();
}

void ModeSelectionDialog::onAdvancedModeSelected() {
```

```
    selectedMode = "Продвинутый";
    accept();
}
```

Mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H
#include <QGraphicsView>
#include <QRandomGenerator>
#include <QBrush>
#include <QPen>
#include <QInputDialog>
#include <QMessageBox>
#include <QComboBox>
#include <QFormLayout>
#include <QLineEdit>
#include <QDialogButtonBox>
#include <cmath>
#include <QPushButton>
#include "modeselectiondialog.h"
#include "reactionparametersdialog.h"
#include <QFile>
#include <QTextStream>
#include <QFileDialog>
```

```
#include <QMessageBox>
#include <QMainWindow>
#include <QGraphicsScene>
#include <QGraphicsRectItem>
#include <QGraphicsEllipseItem>
#include <QTimer>
#include <QMouseEvent>
#include <QSet>
#include "userinputdialog.h"
#include <QLineEdit>
#include <QPushButton>
class MainWindow : public QMainWindow {
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = nullptr);
    ~MainWindow() override;

protected:
    void mousePressEvent(QMouseEvent *event) override;

private slots:
    void updateMolecules();        // Обновление молекул
```

```
    void toggleValve();            // Переключение состояния крана
    void activateCatalyst();       // Активация катализатора
    void startReaction();          // Начало реакции
    void placeMoleculeInReservoir(QGraphicsEllipseItem *molecule);
    void applyParameters();
    void resetAnimation();// Размещение молекулы

private:
    QTimer *steamTimer = nullptr;




    void animateSteam();
     // Таймер для анимации пара
    void requestUserName(); // Запрос имени пользователя
    UserInputDialog *userInputDialog;
    void setupScene();             // Инициализация сцены
    void drawBackground();         // Отрисовка фона        // Запрос имени
пользователя
    void selectMode();             // Выбор режима
    void enterReactionParameters();// Ввод параметров реакции
    void animateSettling();        // Анимация осаждения
    void animateClouds();
```

```
void setTemperature(int temp);
void setPressure(int pres);
void updateCatalystColor();
void generateSteam();
void updateMoleculeSpeed();     // Анимация облаков

// Поля
QGraphicsScene *scene;
QVector<QGraphicsEllipseItem *> clouds;
QVector<QGraphicsEllipseItem *> molecules;
QVector<QGraphicsEllipseItem *> settlingMolecules;
QLineEdit *temperatureInput; // Поле ввода температуры
QLineEdit *pressureInput;    // Поле ввода давления
QPushButton *applyButton;
QList<QGraphicsEllipseItem*> steamParticles;
QPushButton *resetButton;

QGraphicsRectItem *reservoir;
QGraphicsRectItem *valveBase;
QGraphicsRectItem *valveHandle;
QGraphicsRectItem *valveHandleTop;
QGraphicsRectItem *pipe;
QGraphicsRectItem *catalyst;
```

```
QTimer *moleculeTimer;
QTimer *settlingTimer;
QTimer *cloudTimer;

QString userName;
QSet<QString> existingNames;
QString selectedMode;


double temperature; // Начальная температура (Кельвин)
double pressure;      // Начальное давление (атмосферы)
// Состояния
bool valveOpen = false;
bool catalystActive = false;
bool reactionStarted = false;
bool valveDisabled = false;
bool parametersSet = false;

// Геттеры и сеттеры
void setValveOpen(bool open);
bool isValveOpen() const;
```

```
    void setCatalystActive(bool active);
    bool isCatalystActive() const;

    void setReactionStarted(bool started);
    bool isReactionStarted() const;

    void setValveDisabled(bool disabled);
    bool isValveDisabled() const;
};

#endif // MAINWINDOW_H
```

Mainwindow.cpp

```
#include "mainwindow.h"
#include <QtWidgets/qdialog.h>
#include "catalystitem.h"
#include <QLabel>
// Конструктор
MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent), userInputDialog(new UserInputDialog(this)) {
    setFixedSize(800, 600);
    requestUserName();
    selectMode();
    setupScene();
```

```
}

// Деструктор
MainWindow::~MainWindow() {
    delete userInputDialog;
    delete moleculeTimer;
    delete settlingTimer;
    delete cloudTimer;
    delete scene;
}


// Инициализация сцены
void MainWindow::setupScene() {
    scene = new QGraphicsScene(this);
    auto *view = new QGraphicsView(scene, this);
    view->setFixedSize(800, 600);
    scene->setSceneRect(0, 0, 800, 600);
    setCentralWidget(view);

    drawBackground();
    // Добавляем ввод температуры
    temperatureInput = new QLineEdit(this);
```

```
    temperatureInput->setPlaceholderText("Температура (320-420°C)");
    temperatureInput->setValidator(new QDoubleValidator(320, 430, 2, this)); // Ограничение на ввод
    temperatureInput->setGeometry(10, 40, 200, 30); // Позиция на главном окне

    // Добавляем ввод давления
    pressureInput = new QLineEdit(this);
    pressureInput->setPlaceholderText("Давление 100-667(КПа)");
    pressureInput->setValidator(new QDoubleValidator(100, 667, 2, this));  // Ограничение на ввод
    pressureInput->setGeometry(10, 80, 200, 30);

    // Кнопка подтверждения
    applyButton = new QPushButton("Применить", this);
    applyButton->setGeometry(10, 120, 150, 30);

    resetButton = new QPushButton("Перезапуск", this);
    resetButton->setGeometry(10, 160, 150, 30);
    connect(resetButton, &QPushButton::clicked, this, &MainWindow::resetAnimation);

    // Подключаем сигнал нажатия кнопки к слоту
```

```
    connect(applyButton, &QPushButton::clicked, this, &MainWindow::applyParameters);
    // Облака
    for (int i = 0; i < 5; ++i) {
        int x = QRandomGenerator::global()->bounded(0, 800);
        int y = QRandomGenerator::global()->bounded(20, 100);
        QGraphicsEllipseItem *cloud = scene->addEllipse(x, y, 100, 50, QPen(Qt::NoPen), QBrush(Qt::white));
        clouds.append(cloud);
    }

    // Резервуар
    reservoir = scene->addRect(300, 100, 200, 400, QPen(Qt::black), QBrush(Qt::lightGray));

    // Кран
    QBrush valveBrush(Qt::darkGray);
    valveBase = scene->addRect(50, 480, 40, 20, QPen(Qt::black), valveBrush);
    valveHandle = scene->addRect(60, 460, 20, 20, QPen(Qt::black), valveBrush);
    valveHandleTop = scene->addRect(50, 450, 40, 10, QPen(Qt::black), Qt::red);
```

```
    pipe = scene->addRect(90, 480, 210, 10, QPen(Qt::black), QBrush(Qt::darkGray));

    // Катализатор
    catalyst = new CatalystItem("CO + H2 -> CH3OH", nullptr);
    catalyst->setRect(500, 200, 40, 200); // Устанавливаем размер катализатора
    catalyst->setBrush(QBrush(Qt::blue)); // Цвет катализатора
    scene->addItem(catalyst);catalyst = new CatalystItem("Fe3 O4", nullptr);
    catalyst->setRect(500, 200, 40, 200); // Устанавливаем размер катализатора
    catalyst->setBrush(QBrush(Qt::blue)); // Цвет катализатора
    scene->addItem(catalyst);

    // Таймеры
    moleculeTimer = new QTimer(this);
    connect(moleculeTimer, &QTimer::timeout, this, &MainWindow::updateMolecules);
    moleculeTimer->start(50);

    cloudTimer = new QTimer(this);
    connect(cloudTimer, &QTimer::timeout, this, &MainWindow::animateClouds);
    cloudTimer->start(100);
```

```
    settlingTimer = new QTimer(this);
    connect(settlingTimer, &QTimer::timeout, this, &MainWindow::animateSettling);

    //  steamTimer = new QTimer(this);
    // connect(steamTimer, &QTimer::timeout, this, &MainWindow::animateSteam);
}
void MainWindow::resetAnimation() {
    // Очищаем молекулы из сцены
    for (auto *mol : molecules) {
        scene->removeItem(mol);
        delete mol;
    }
    molecules.clear();

    // Сбрасываем молекулы в процессе оседания
    for (auto *mol : settlingMolecules) {
        scene->removeItem(mol);
        delete mol;
    }
    settlingMolecules.clear();
```

```
    // Сбрасываем параметры
    parametersSet = false;
    valveOpen = false;
    valveDisabled = false;
    catalystActive = false;
    reactionStarted = false;

    QMessageBox::information(this, "Перезапуск", "Анимация сброшена. Введите параметры и начните заново.");
}
// Запрос имени пользователя
void MainWindow::requestUserName() {
    connect(userInputDialog, &UserInputDialog::userCanceled, this, &MainWindow::close);

    QString name = userInputDialog->getUserName(existingNames, this);
    if (name.isEmpty()) {
        return; // Пользователь отменил ввод
    }

    userName = name;
    existingNames.insert(userName);
```

```
    QMessageBox::information(this, "Приветствие", "Добро пожаловать, " + userName + "!");
}

// Выбор режима
void MainWindow::selectMode() {
    ModeSelectionDialog dialog(this);
    if (dialog.exec() == QDialog::Accepted) {
        selectedMode = dialog.getSelectedMode();

        if (selectedMode == "Базовый") {
            QMessageBox::information(this, "Режим", "Каталитическая конверсия оксида углерода.");
        } else if (selectedMode == "Продвинутый") {
            enterReactionParameters();
        }
    } else {
        close(); // Закрываем приложение, если пользователь отменил выбор
    }
}

void MainWindow::enterReactionParameters() {
    ReactionParametersDialog dialog(this);
```

```
    if (dialog.exec() == QDialog::Accepted) {
        double T = dialog.getTemperature();
        double P = dialog.getPressure();

        double deltaH = 9420 + 3.16 * T - 8.314 * pow(10, -3) * pow(T, 2) + 5.17 * pow(10, -6) * pow(T, 3) - 1.131 * pow(10, -9) * pow(T, 4);
        double logKp = -2.4198 + 0.0003855 * T + 2180.6 / T;

        QString result = QString("Температура (T): %1\n"
                                 "Давление (P): %2\n"
                                 "Тепловой эффект (ΔH): %3\n"
                                 "Константа равновесия (Kp): %4")
                             .arg(T)
                             .arg(P)
                             .arg(deltaH)
                             .arg(pow(10, logKp));

        // Показываем результаты пользователю
        QMessageBox::information(this, "Результаты", result);

        // Сохраняем результаты в файл
        QString fileName = QFileDialog::getSaveFileName(this, "Сохранить результаты", "", "Text Files (*.txt);;All Files (*)");
```

```
        if (!fileName.isEmpty()) {
            QFile file(fileName);
            if (file.open(QIODevice::WriteOnly | QIODevice::Text)) {
                QTextStream out(&file);
                out << result; // Записываем строку с результатами
                file.close();
                QMessageBox::information(this, "Успех", "Результаты успешно сохранены!");
            } else {
                QMessageBox::warning(this, "Ошибка", "Не удалось открыть файл для записи.");
            }
        }
    }
}



// Отрисовка фона
void MainWindow::drawBackground() {
    // Небо
    scene->addRect(0, 0, 800, 500, QPen(Qt::NoPen), QBrush{"#4C98CA"});
```

```
    // Трава
    scene->addRect(0, 420, 800, 350, QPen(Qt::NoPen), QBrush{"#80B929"});

    // Почва с камнями
    scene->addRect(0, 500, 800, 250, QPen(Qt::NoPen), QBrush{"#7E4811"});
    for (int i = 0; i < 100; ++i) {
        int x = QRandomGenerator::global()->bounded(0, 800);
        int y = QRandomGenerator::global()->bounded(500, 600);
        scene->addEllipse(x, y, 5, 5, QPen(Qt::NoPen), QBrush(Qt::darkGray));
    }
}

// Геттеры и сеттеры
void MainWindow::setValveOpen(bool open) { valveOpen = open; }
bool MainWindow::isValveOpen() const { return valveOpen; }

void MainWindow::setCatalystActive(bool active) { catalystActive = active; }
bool MainWindow::isCatalystActive() const { return catalystActive; }

void MainWindow::setReactionStarted(bool started) { reactionStarted = started; }
```

```
bool MainWindow::isReactionStarted() const { return reactionStarted; }


void MainWindow::setValveDisabled(bool disabled) { valveDisabled = disabled; }
bool MainWindow::isValveDisabled() const { return valveDisabled; }



void MainWindow::animateClouds() {
    for (auto *cloud : clouds) {
        // Двигаем облако вправо
        cloud->moveBy(2, 0); // Движение на 2 пикселя вправо

        // Если облако вышло за правую границу, перемещаем его налево
        if (cloud->x() > 800) {
            cloud->setX(-100); // Возвращаем облако на левую сторону
        }
    }
}
void MainWindow::mousePressEvent(QMouseEvent *event) {
    QPointF scenePos = centralWidget()->mapFromGlobal(event->globalPosition().toPoint());
    scenePos = static_cast<QGraphicsView*>(centralWidget())->mapToScene(scenePos.toPoint());
```

```
    // Проверяем нажатие на любую часть крана
    if (valveBase->contains(scenePos - valveBase->pos()) ||
        valveHandle->contains(scenePos - valveHandle->pos()) ||
        valveHandleTop->contains(scenePos - valveHandleTop->pos())) {
        toggleValve();
    }

    // Проверяем нажатие на катализатор
    if (catalyst->contains(scenePos - catalyst->pos())) {
        activateCatalyst();
    }
}

void MainWindow::updateMolecules() {
    if (!valveOpen || !parametersSet) return; // Не начинаем анимацию без параметров

    // Добавление новой молекулы, если кран открыт
    int y = QRandomGenerator::global()->bounded(480, 485); // Позиция в трубе
    auto *molecule = scene->addEllipse(90, y, 5, 5, QPen(Qt::black), QBrush(Qt::red));
```

```
    molecules.append(molecule);


    // Движение молекул
    for (auto *mol : molecules) {
        if (mol->x() < 200) {
            // Движение вправо по трубе
            mol->moveBy(5, 0);
        } else {
            // Если молекула в резервуаре, продолжаем двигать её там
            placeMoleculeInReservoir(mol);
        }
    }
}
void MainWindow::toggleValve() {
    if (!parametersSet) {
        QMessageBox::warning(this, "Ошибка", "Пожалуйста, примените параметры перед использованием крана.");
        return;
    }


    if (valveDisabled) {
        QMessageBox::warning(this, "Кран заблокирован", "Кран уже был выключен и не может быть включён снова!");
```

```
    return;
}

valveOpen = !valveOpen;
QBrush brush = valveOpen ? QBrush(Qt::green) : QBrush(Qt::darkGray);

valveBase->setBrush(brush);
valveHandle->setBrush(brush);
valveHandleTop->setBrush(brush);

if (!valveOpen) {
    // Удаляем молекулы из трубы
    auto it = molecules.begin();
    while (it != molecules.end()) {
        QGraphicsEllipseItem *mol = *it;
        if (mol->x() < 300) { // Проверяем, находится ли молекула в трубе
            scene->removeItem(mol);
            it = molecules.erase(it);
            delete mol;
        } else {
            ++it;
        }
    }
```

```
        // Блокируем кран после выключения
        valveDisabled = true;
    }
}

void MainWindow::activateCatalyst() {
    if (valveOpen) {
        QMessageBox::warning(this, "Ошибка", "Закройте кран перед активацией катализатора.");
        return;
    }

    if (!parametersSet) {
        QMessageBox::warning(this, "Ошибка", "Пожалуйста, примените параметры перед активацией катализатора.");
        return;
    }

    catalystActive = true;
    catalyst->setBrush(Qt::red);
    startReaction(); // Начинаем реакцию и осаждение молекул
    generateSteam(); // Генерируем молекулы, которые осаждаются
```

```
}

void MainWindow::placeMoleculeInReservoir(QGraphicsEllipseItem *molecule) {
    static QVector<QPoint> freeCells;

    // Вычисление размера молекулы на основе давления
    int moleculeSize = qBound(5, static_cast<int>(pressure) / 10, 20); // Размер молекулы (диаметр), ограничение от 5 до 20
    int cellSize = moleculeSize; // Размер ячейки равен размеру молекулы

    // Границы резервуара
    int reservoirLeft = 300;
    int reservoirTop = 100;
    int reservoirWidth = 200;
    int reservoirHeight = 400;

    // Количество строк и столбцов
    int rows = reservoirHeight / cellSize;
    int cols = reservoirWidth / cellSize;

    // Инициализация свободных ячеек
    if (freeCells.isEmpty()) {
        for (int y = 0; y < rows; ++y) {
```

```
            for (int x = 0; x < cols; ++x) {
                freeCells.append(QPoint(reservoirLeft + x * cellSize,
                                        reservoirTop + y * cellSize));
            }
        }
    }

    // Проверяем, остались ли свободные ячейки
    if (freeCells.isEmpty()) {
        return; // Все места заняты, не размещаем новую молекулу
    }

    // Берём случайную свободную ячейку
    int randomIndex = QRandomGenerator::global()->bounded(freeCells.size());
    QPoint position = freeCells[randomIndex];

    // Удаляем выбранную ячейку из списка
    freeCells.removeAt(randomIndex);

    // Устанавливаем молекулу в центр ячейки
    molecule->setRect(0, 0, moleculeSize, moleculeSize); // Устанавливаем размер молекулы
```

```
    molecule->setPos(position.x(), position.y());
}
void MainWindow::startReaction() {
    if (!catalystActive || reactionStarted) return;

    reactionStarted = true;

    // Удаляем молекулы, находящиеся в трубе
    auto it = molecules.begin(); // Обычный итератор
    while (it != molecules.end()) {
        QGraphicsEllipseItem *mol = *it;
        if (mol->x() < 300) { // Если молекула находится в трубе
            scene->removeItem(mol);   // Удаляем элемент из сцены
            it = molecules.erase(it); // Удаляем элемент из QVector и обновляем
итератор
            delete mol;               // Освобождаем память
        } else {
            ++it; // Переход к следующему элементу
        }
    }

    // Оставшиеся молекулы добавляем в список для оседания
    for (auto *mol : molecules) {
```

```
        settlingMolecules.append(mol);
    }

    molecules.clear();

    // Запускаем таймер оседания
    settlingTimer->start(50); // Обновление каждые 50 мс
}

void MainWindow::setTemperature(int temp) {
    temperature = temp;

    // Изменяем скорость молекул
    int newInterval = 10000 / temperature; // Пример зависимости
    moleculeTimer->setInterval(newInterval);

    // Меняем цвет катализатора
    QColor catalystColor = QColor::fromHsv(qMin(temperature / 4.0, 255.0),
255, 255);
    catalyst->setBrush(catalystColor);
}

void MainWindow::setPressure(int pres) {
```

```
    pressure = pres;


    // Обновляем молекулы в резервуаре
    int moleculeSize = qBound(5, static_cast<int>(pressure) / 10, 20); // Новый размер молекулы (ограничен от 5 до 20)
    for (auto *molecule : molecules) {
        molecule->setRect(molecule->x(), molecule->y(), moleculeSize, moleculeSize);
    }


    // Добавляем или удаляем молекулы, чтобы их количество соответствовало давлению
    int moleculeCount = pressure / 10; // Пример зависимости
    while (molecules.size() < moleculeCount) {
        auto *molecule = scene->addEllipse(90, QRandomGenerator::global()->bounded(480, 485), moleculeSize, moleculeSize, QPen(Qt::black), QBrush(Qt::red));
        molecules.append(molecule);
    }


    while (molecules.size() > moleculeCount) {
        QGraphicsEllipseItem *molecule = molecules.takeLast();
        scene->removeItem(molecule);
        delete molecule;
```

```
    }
}
void MainWindow::animateSettling() {
    if (settlingMolecules.isEmpty()) {
        settlingTimer->stop();
        return;
    }

    // Границы резервуара
    const int reservoirBottom = 500; // Нижняя граница резервуара
    const int reservoirLeft = 300;  // Левая граница резервуара
    const int reservoirRight = 500; // Правая граница резервуара
    const int moleculeSize = 10;    // Размер молекулы (диаметр)

    for (int i = 0; i < settlingMolecules.size(); ++i) {
        QGraphicsEllipseItem *molecule = settlingMolecules[i];

        // Убираем молекулы, которые вышли за пределы резервуара
        if (molecule->x() < reservoirLeft || molecule->x() > reservoirRight ||
molecule->y() > reservoirBottom) {
            scene->removeItem(molecule);
            settlingMolecules.removeAt(i);
            delete molecule;
```

```
            --i;
            continue;
        }

        // Проверяем, достигла ли молекула дна
        double molBottomY = molecule->y() + moleculeSize;

        if (molBottomY < reservoirBottom) {
            // Если молекула выше дна, опускаем её вниз
            molecule->moveBy(0, 5);
        } else {
            // Молекула достигла дна, фиксируем её положение
            molecule->setY(reservoirBottom - moleculeSize);
            molecule->setBrush(Qt::blue); // Меняем цвет, чтобы обозначить осаждение
        }
    }

    // Если все молекулы осели, остановим таймер
    if (settlingMolecules.isEmpty()) {
        settlingTimer->stop();
    }
}
```

```
void MainWindow::applyParameters() {
    bool tempOk, presOk;
    double temp = temperatureInput->text().toDouble(&tempOk);
    double pres = pressureInput->text().toDouble(&presOk);

    if (!tempOk || !presOk) {
        QMessageBox::warning(this, "Ошибка ввода", "Пожалуйста, введите корректные значения температуры и давления.");
        return;
    }

    if (temp < 320 || temp > 430) {
        QMessageBox::warning(this, "Ошибка", "Температура должна быть в диапазоне 320–430 °C.");
        return;
    }

    if (pres < 100 || pres > 667) {
        QMessageBox::warning(this, "Ошибка", "Давление должно быть в диапазоне 100–667 кПа.");
        return;
    }
```

```
    // Обновляем параметры
    temperature = temp;
    pressure = pres;
    parametersSet = true;

    // Обновляем состояние катализатора (например, цвет)
    updateCatalystColor();
    QMessageBox::information(this, "Параметры",
                             QString("Температура: %1 °C\nДавление: %2 кПа").arg(temperature).arg(pressure));
}

void MainWindow::updateCatalystColor() {
    // Изменяем цвет катализатора в зависимости от температуры
    QColor catalystColor = QColor::fromHsv(qMin(temperature / 4.0, 255.0), 255, 255);
    if (catalyst) {
        catalyst->setBrush(catalystColor);
    }
}
void MainWindow::generateSteam() {
    const int steamWidth = 10;
    const int steamHeight =25;
```

```
    // Параметры пара
    const int reservoirTop = 150;
    const int reservoirLeft = 300;
    const int reservoirWidth = 200;

    for (int i = 0; i < 10; ++i) { // Создаём несколько элементов пара за раз
        int x = QRandomGenerator::global()->bounded(reservoirLeft, reservoirLeft + reservoirWidth - steamWidth);
        auto *steam = scene->addEllipse(x, reservoirTop, steamWidth, steamHeight, QPen(Qt::NoPen), QBrush(Qt::white));
        steamParticles.append(steam);
    }

    if (!steamTimer) {
        steamTimer = new QTimer(this);
        connect(steamTimer, &QTimer::timeout, this, &MainWindow::animateSteam);
        steamTimer->start(200);
    }
}
void MainWindow::updateMoleculeSpeed() {
    if (temperature > 0) {
```

```
        int newInterval = qMax(1000 / static_cast<int>(temperature), 10); // Приводим к int
        moleculeTimer->setInterval(newInterval);
    }
}
void MainWindow::animateSteam() {
    for (int i = 0; i < steamParticles.size(); ++i) {
        QGraphicsEllipseItem *steam = steamParticles[i];
        steam->moveBy(0, -1); // Двигаем пар вверх

        if (steam->y() + steam->rect().height() < 0) {
            // Убираем частицы, которые вышли за пределы экрана
            scene->removeItem(steam);
            steamParticles.removeAt(i);
            delete steam;
            --i;
        }
    }

    // Останавливаем таймер, если все частицы пара исчезли
    if (steamParticles.isEmpty()) {
        steamTimer->stop();
        delete steamTimer;
```

```
        steamTimer = nullptr;
    }
}
```