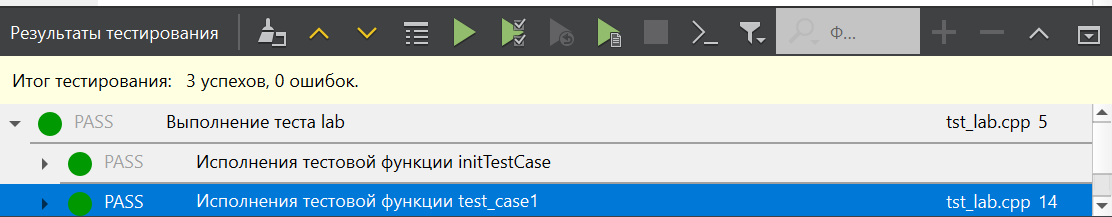
**Отчет по практической части учебной практики**

## **2.1 Решение задач**

**2.1.1 Unit-тестирование**

Дополнение ранее разработанных классов системой Unit-тестов.

Были рассмотрены концепции модульного тестирования, включая написание тестов, методы проверки корректности кода и интеграция тестов в ранее разработанные программы. Это обеспечило понимание важности тестирования в процессе разработки ПО.

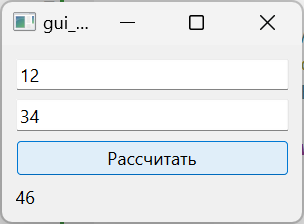


*Рисунок1- Результаты тестирования*

## **2.1.2 GUI-приложение**

Разработка простейшего GUI-приложения в QT (Диалоговое окно с виджетами, обеспечивающими ввод информации, вызов расчетной части приложения и вывод результатов).

Программа представляет собой простое графическое интерфейсное приложение, которое позволяет пользователю вводить два числа, нажимать кнопку для выполнения расчетной операции (в данном случае сложения) и отображать результат в виде текста.



*Рисунок 2- Интерфейс GUI-приложения*

## **2.1.3 Таблицы**

Разработка приложения, отражающего табличные данные с возможностью корректировки (на основе шаблона Модель-Вид), вывода и ввода данных.

1. Инициализация:

* В main.cpp создается объект QApplication и MainWindow, после чего вызывается метод show(), чтобы отобразить главное окно.
* В MainWindow инициализируется таблица (QTableView) и модель данных (QStandardItemModel).
* Устанавливается модель данных для таблицы.
* Создаются кнопка Add Row для добавления новой строки и действия меню Save, Load, Save as Excel, Load Excel

2. Добавление строки:

* Кнопка Add Row вызывает слот addNewRow(), который отображает три диалога ввода (QInputDialog) для получения значений каждой колонки.
* Введенные пользователем значения добавляются в модель в виде новой строки.

3. Сохранение в CSV:

* Действие Save вызывает слот saveToFile().
* Открывается диалог для выбора пути и имени файла.
* Данные из модели считываются и записываются в файл в формате CSV.

4. Загрузка из CSV:

* Действие Load вызывает слот loadFromFile().
* Открывается диалог для выбора файла.
* Данные из выбранного файла читаются и загружаются в модель.

5. Сохранение в Excel:

* Действие Save as Excel вызывает слот saveToExcel().
* Открывается диалог для выбора пути и имени файла.
* Если имя файла не имеет расширения .xlsx, оно добавляется.
* Данные из модели считываются и записываются в файл в формате CSV, но с расширением .xlsx, что позволяет Excel корректно открывать этот файл

6. Загрузка из Excel:

* Действие Load Excel вызывает слот loadFromExcel().
* Открывается диалог для выбора файла.
* Если имя файла не имеет расширения .xlsx, оно добавляется.
* Данные из выбранного файла читаются и загружаются в модель.

**Методы и функции:**

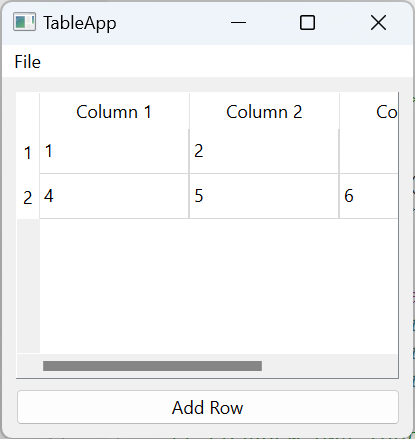
* QApplication: объект, управляющий жизненным циклом приложения.
* QMainWindow: базовый класс для главного окна приложения.
* QTableView: виджет для отображения табличных данных.
* QStandardItemModel: модель данных, позволяющая хранить и управлять табличными данными.
* QAction: класс, представляющий действие, которое можно добавить в меню или на панель инструментов.
* QPushButton: кнопка, которая используется для добавления новой строки.
* QFileDialog: диалог для открытия и сохранения файлов.
* QInputDialog: диалог для ввода текста пользователем.
* QVBoxLayout: компоновщик, который размещает виджеты вертикально.

**Результат**

В результате программа предоставляет пользователю интерфейс для работы с табличными данными. Пользователь может:

* Добавлять новые строки через всплывающие окна.
* Сохранять данные в файлы с расширением .csv или .xlsx.
* Загружать данные из файлов с расширением .csv или .xlsx.

Данные сохраняются в формате CSV, который может быть прочитан и изменен Excel, если файл имеет расширение .xlsx. Это позволяет легко обмениваться данными с пользователями, использующими Excel.



*Рисунок 3 - Таблица*

## **2.1.4 SDI-приложение**

Разработка простейшего SDI-приложения, иллюстрирующего использование меню, корректировку, сохранение и восстановление текстовой, числовой информации в текстовом (QTextStream) и бинарном(QDataStream) формате.

**Основные компоненты программы:**

1. MainWindow (Главное окно):

* Наследует от QMainWindow.
* Содержит центральный виджет QTextEdit для редактирования текста.
* Имеет меню с пунктами для сохранения и загрузки данных в текстовом формате.

2. Слоты (slots):

* saveText(): Сохраняет данные из QTextEdit в текстовый файл.
* loadText(): Загружает данные из текстового файла в QTextEdit.

**Инициализация интерфейса:**

* QTextEdit: Центральный виджет для редактирования текста.
* QMenu: Создает меню "File".
* QAction: Создает действия для пунктов меню, каждое из которых подключается к соответствующему слоту с помощью connect.

**Сохранение в текстовый файл:**

* QFileDialog::getSaveFileName: Открывает диалог для выбора файла.
* QFile: Открывает файл для записи.
* QTextStream: Записывает текстовые данные из QTextEdit в файл.

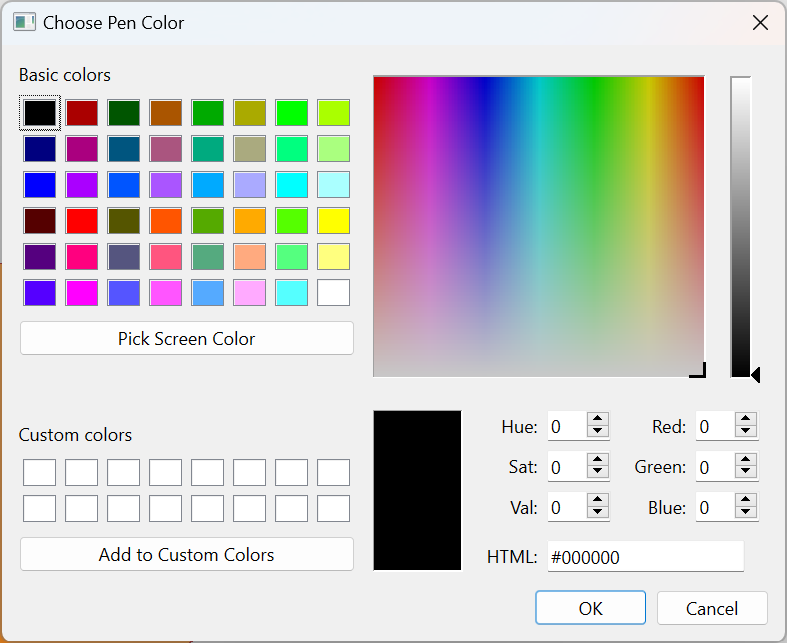
**Загрузка из текстового файла:**

* QFileDialog::getOpenFileName: Открывает диалог для выбора файла.
* QFile: Открывает файл для чтения.
* QTextStream: Читает текстовые данные из файла и загружает их в QTextEdit.

## **2.1.5 Аналог векторного графического редактора**

Разработка приложения, аналога Векторного графического редактора, с обработкой событий от мыши, сериализацией набора графических объектов в файлах.

Программа представляет собой графическое приложение на базе фреймворка Qt, которое позволяет пользователю рисовать и сохранять простые геометрические фигуры (линии, прямоугольники и круги). А также позволяет выбирать любой цвет для пера и заливки. Выполняется сериализация набора графических объектов в файлах. Приложение включает в себя несколько классов, каждый из которых отвечает за определенные аспекты функциональности.



*Рисунок 4 - Палитра*

**Детализация реализации и работы компонентов:**

**Класс Shape**

* Абстрактный класс, который служит базовым для всех типов фигур. Его основные обязанности:
* Хранение координат начальной и конечной точек (QPoint startPoint и QPoint endPoint).
* Хранение цветов пера и заливки (QColor penColor и QColor fillColor).
* Флаг, определяющий, заполнена ли фигура (bool filled).
* Методы доступа (геттеры и сеттеры) для этих свойств.
* Методы serialize и deserialize для сохранения и загрузки состояния фигуры.

**Классы Rectangle, Circle и Line**

* Классы-наследники от Shape, реализующие конкретные типы фигур.
* Каждый из этих классов:

1. Переопределяет метод draw(QPainter \*painter), который конкретно рисует фигуру, используя QPainter.
2. Переопределяет метод getType(), возвращающий соответствующий тип фигуры.
3. Переопределяет методы serialize и deserialize для добавления типа фигуры при сериализации и десериализации.

**Класс DrawArea**

* Класс области рисования, который наследует QWidget.
* Основные методы:

1. paintEvent(QPaintEvent \*event): Отвечает за перерисовку всех фигур при каждом обновлении виджета.
2. mousePressEvent(QMouseEvent\*event), mouseMoveEvent(QMouseEvent\*event), mouseReleaseEvent(QMouseEvent \*event):

Обрабатывают события мыши для создания и изменения фигур.

* Сериализация:

1. saveToFile(const QString &fileName) и loadFromFile(const QString &fileName): Сохранение и загрузка списка фигур в/из файл(а) JSON.
2. Вспомогательные методы shapeToJson(Shape \*shape) и shapeFromJson(const QJsonObject &json) для преобразования фигур в JSON-формат и обратно.

## **2.1.6 Простейшая игра**

Разработка проекта с графической анимацией объектов на экране.

**Основные компоненты и их функции**

1.Конструктор GameWindow(QWidget \*parent = nullptr)\*\*:

* Инициализирует основное окно игры с фиксированным размером 800x600 пикселей.
* Скрывает стандартный курсор мыши с помощью setCursor(Qt::BlankCursor).
* Создает и настраивает метки для изображения преподавателя (teacherLabel), бейсбольной биты (batLabel) и катафалка (hearseLabel).
* Устанавливает изображения для этих меток.
* Инициализирует и запускает таймер (moveTimer)

2.Метод mousePressEvent(QMouseEvent \*event)\*\*:

* Переопределяет поведение при нажатии кнопки мыши.
* Проверяет нажатие левой кнопки мыши и вызывает метод checkCollision() для проверки столкновения биты с изображением преМетод moveTeacher() moveTeacher()\*\*:
* Перемещает изображение преподавателя в случайное направление через каждые 300 миллисекунд.
* Вычисляет новое случайное смещение в пределах максимального значения maxOffset (100 пикселей) и обновляет позицию преподавателя с учетом границ окна.

## **2.1.7 Хеш-таблица**

Описание программы и её работы

Программа представляет собой GUI-приложение для управления библиотекой книг с использованием хеш-таблицы для хранения информации о книгах. В приложении реализованы функции добавления, удаления, поиска книг по ISBN и просмотра всех книг в библиотеке.

**Основные компоненты и их функции**

1. Класс LibraryManager:

* Этот класс является основным виджетом приложения, реализующим интерфейс и логику работы с библиотекой книг.

2.Поля ввода и кнопки:

* isbnInput, titleInput, authorInput: Поля ввода для ISBN, названия книги и имени автора соответственно.
* addButton, removeButton, findButton, viewAllButton: Кнопки для выполнения соответствующих действий (добавление, удаление, поиск и просмотр всех книг).

3. Таблица bookTable:

* Таблица для отображения списка всех книг в библиотеке. Она имеет три столбца: ISBN, название книги и автор.

**Методы класса LibraryManager**

1. Конструктор LibraryManager(QWidget \*parent = nullptr):

- Инициализирует интерфейс, создает поля ввода, кнопки и таблицу. Настраивает макеты и подключает сигналы кнопок к соответствующим слотам.

2. Метод hashFunction:

- Простейшая хеш-функция, которая используется для вычисления хеша ISBN. В данном случае она не используется для работы с QHash, так как QHash сам управляет

Метод addBook:

- Добавляет книгу в хеш-таблицу. Проверяет, заполнены ли все поля, и существует ли книга с таким ISBN. Если все проверки пройдены, добавляет книгу и показывает сообщение об успешном добавлении

Метод removeBook:

- Удаляет книгу из хеш-таблицы по ISBN. Проверяет, заполнено ли поле ISBN, и существует ли книга с таким ISBN. Если все проверки пройдены, удаляет книгу и показывает сообщение об успешном удалении

Метод findBook:

- Ищет книгу по ISBN в хеш-таблице. Проверяет, заполнено ли поле ISBN, и существует ли книга с таким ISBN. Если книга найдена, показывает информацию о ней

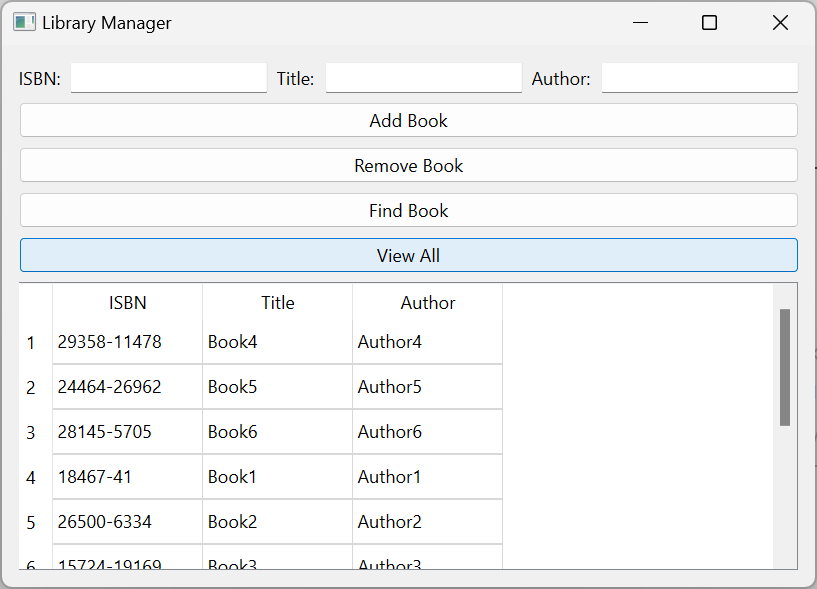
Метод viewAllBooks:

- Отображает все книги в таблице bookTable. Очищает таблицу и заново заполняет её данными из хеш-таблицы.

**Результат**

* Пользователь может удобно управлять библиотекой книг через графический интерфейс.
* Все операции (добавление, удаление, поиск, просмотр) выполняются быстро благодаря использованию хеш-таблицы.
* Программа предоставляет интуитивно понятный интерфейс для работы с книгами, отображая информативные сообщения при ошибках и успешных действиях.
* Данные предварительно заполняются случайными значениями, что демонстрирует работу программы при старте.
* Невозможно добавить несколько книг с одинаковым ISBN.

Эта реализация демонстрирует основы создания GUI-приложения с использованием Qt и работы с хеш-таблицей для хранения и управления данными.



*Рисунок 5- Хеш-таблица*

## **2.2 Выводы к главе 2**

В ходе учебной практики была выполнена работа по нескольким направлениям, включающим как теоретические, так и практические аспекты программирования. Основные результаты практической части:

1. Unit-тестирование:

Были разработаны и внедрены модульные тесты для различных компонентов приложения, что позволило обеспечить высокое качество кода и своевременное обнаружение ошибок.

2.Разработка GUI-приложения:

Создано GUI-приложениe, которое обеспечивает интуитивно понятное взаимодействие пользователя с программой. Были использованы современные фреймворки и библиотеки для создания удобного и функционального интерфейса.

3.Разработка приложения, отображающего табличные данные:

Реализовано приложение, отображающее табличные данные с возможностью их корректировки, ввода и вывода. В основе приложения лежит архитектурный шаблон Модель-Вид (MVC), обеспечивающий разделение логики, представления и данных.

4. SDI-приложение:

Создано SDI-приложение (Single Document Interface), поддерживающее работу с одним документом за раз, что упростило управление и взаимодействие с пользователем.

5. Аналог векторного графического редактора:

Разработан аналог векторного графического редактора, позволяющий пользователям создавать и редактировать векторные изображения, что продемонстрировало навыки работы с графическими интерфейсами и алгоритмами обработки графики.

6. Простейшая игра:

Разработана простейшая игра, включающая базовую игровую механику и взаимодействие с пользователем. Это позволило изучить основные принципы разработки игр и организации игрового процесса.

7. Хеш-таблица:

Реализована структура данных хеш-таблица, обеспечивающая эффективное хранение и доступ к данным по ключу. Это продемонстрировало понимание алгоритмов и структур данных, а также их практическое применение.

Выполнение этих задач позволило закрепить теоретические знания, получить практические навыки и подготовиться к решению более сложных задач в будущем.