

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ
ФЕДЕРАЦИИ

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра «Вычислительная техника»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту
по курсу «Программирование»
на тему «Разработка программы «База данных по столицам стран»

Выполнил: ст. гр.24 ВОА2

Максин С.

Руководитель:

к.п.н., доцент Леонова Т.Ю.

Работа защищена с оценкой хорошо

Дата защиты 24.12.25

Пенза 2025

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Факультет Вычислительной техники
Кафедра «Вычислительная техника»



«УТВЕРЖДАЮ»
Зав. кафедрой ВТ
Митрохин М. А.

«___» _____ 2025 г

ЗАДАНИЕ
на курсовое проектирование по курсу

Программирование

Студенту Максину Сергею Николаевичу

Группа 24BOA2

Тема проекта: Разработка программы сложной структуры
методом нисходящего программирования

Исходные данные (технические требования) на проектирование

Наименование программы: База данных по столицам стран

Задание. Программа должна реализовать следующие функции:

- Меню.
- список стран и их столицы.

Обязательные требования к программе:

1. Многомодульность.
2. Использование ООП.
3. Режим работы видеосистемы — графический.
4. Устройство ввода информации — клавиатура, мышь.
5. Пользовательский интерфейс должен быть построен на основе меню и панели инструментов.
6. Операционная система — Windows.
7. Система программирования — C/C++ на базе среды Microsoft Visual Studio 2019.
8. Язык программирования — C/C++.

Объем работы по курсу
1. Расчетная часть
Разработка программы.

2. Графическая часть
UML-диаграммы использования, UML-диаграммы классов, UML-диаграммы активности, UML-диаграммы последовательности.

3. Экспериментальная часть
Отладка программы.

Срок выполнения проекта по разделам

1. В соответствии с графиком выполнения курсового проекта.
- 2.
- 3.

Дата выдачи задания « 16 » сентября
Дата защиты проекта « »
Руководитель Леонова Татьяна Юрьевна
Задание получил _____
Студент Май « 16 » сентября

Содержание

Введение.....	
1 Постановка задачи	
2 Выбор решения.....	
3 Проектирование программы	
4 Описание программы.....	
5 Отладка и тестирование	
6 Руководство пользователя.....	
Заключение	
Список использованных источников.....	
Приложение А	
Исходный код программы.....	
Приложение А.1	
Файл «main.cpp».....	
Приложение А.2	
Файл «main.h».....	
Приложение А.3	
Файл «Game.h».....	
Приложение А.4	
Файл «Game.cpp».....	
Приложение А.5	
Файл «Tower.h».....	
Приложение А.6	
Файл «Tower.cpp».....	
Приложение А.7	
Файл «Wave.h».....	
Приложение А.8	
Файл «Wave.cpp».....	
Приложение А.9	
Файл «startscreen.h».....	
Приложение А.10	
Файл «startscreen.cpp».....	
Приложение А.11	
Файл «bg.h».....	

Содержание

Введение	5
1 Постановка задачи	6
2 Выбор решения	6
3 Проектирование программы	8
4 Описание программы	14
5 Отладка и тестирование	Ошибка! Закладка не определена.
6 Руководство пользователя	17
Заключение.....	19
Список использованных источников.....	20
Приложение А	
Исходный код программы	Ошибка! Закладка не определена.
Приложение Б	
Результаты работы программы	45

Введение

Логическая головоломка – это игра, которая развивает пространственное мышление и логические способности пользователя. В представленном проекте я хочу реализовать игру, где игроку необходимо перемещать ящики по лабиринту, активируя переключатели и открывая проходы к выходу. Такие игры тренируют стратегическое планирование и решение сложных задач.

Разработка данной игры потребует глубоких знаний объектно-ориентированного программирования, работы с графикой и обработки пользовательского ввода. Проект должен демонстрировать сложную архитектуру с несколькими игровыми состояниями, системой управления ресурсами и интерактивными элементами интерфейса.

Для написания кода необходимо понимание принципов пользовательского интерфейса для создания интуитивно понятного дизайна с анимированными кнопками, ползунками настройки громкости и адаптивными фоновыми изображениями.

Разработка данной игры позволит получить ценный опыт в создании комплексных приложений с использованием библиотеки SFML.

1.Постановка задачи

Программа должна представлять собой интерактивную образовательную базу данных по столицам стран мира с графическим интерфейсом и содержать следующие основные компоненты:

- база данных стран — программа должна хранить информацию о странах мира, включая название страны, её столицу, принадлежность к континенту и численность населения;

- поддержка расширения базы — возможность добавления новых стран пользователем;

- инициализация данных — загрузка начального набора стран при первом запуске программы.

- главное меню — пользователь должен иметь возможность переключаться между различными режимами работы программы через интуитивно понятный графический интерфейс;

- просмотр базы данных — отображение списка всех стран с возможностью прокрутки и навигации;

- фильтрация данных — возможность отбора стран по континенту через текстовый ввод;

- система поиска — быстрый поиск страны по названию;

- образовательный модуль — реализация интерактивной викторины для проверки знаний столиц стран.

- добавление записей — возможность ручного ввода новых стран через пошаговую форму;

- перемешивание списка — функция случайного изменения порядка стран в базе;

- сохранение и загрузка — поддержка экспорта и импорта данных в формате CSV через файловую систему.

-графический интерфейс — программа должна отображать все элементы управления в едином стиле с использованием библиотеки SFML;

-информационные панели — отображение текущего состояния программы, количества стран, результатов викторины;

-элементы ввода — текстовые поля для взаимодействия с пользователем;

-навигационные элементы — кнопки, ползунки прокрутки, подсказки.

-кроссплатформенность — программа должна компилироваться и работать в среде Windows;

-стабильность работы — отсутствие утечек памяти и критических ошибок во время выполнения;

-производительность — быстрая загрузка данных и отклик интерфейса даже при большом количестве записей.

Программа предназначена для использования в образовательных целях, в качестве справочного пособия по странам мира, а также в качестве инструмента для самопроверки знаний в области географии. Она может быть использована в школах, вузах или для самостоятельного обучения.

2. Выбор решения

Для реализации интерактивной базы данных по столицам стран была выбрана библиотека SFML (Simple and Fast Multimedia Library), которая предоставляет комплексные инструменты для работы с графикой, обработкой ввода и созданием пользовательских интерфейсов. Программа состоит из нескольких основных компонентов:

главное меню — для навигации между различными режимами работы программы;

режим просмотра данных — для отображения и навигации по списку всех стран;

режим фильтрации — для поиска и отбора стран по континентам;

образовательная викторина — для тестирования знаний пользователя;
режим управления данными — для добавления новых стран и работы с файлами.

Язык программирования C++ был выбран по следующим причинам:
высокая производительность и эффективность работы с памятью;
поддержка объектно-ориентированного программирования для создания модульной архитектуры;
богатая стандартная библиотека (STL) для работы с коллекциями данных, алгоритмами и строками;
кроссплатформенность и совместимость с различными средами разработки.

Библиотека SFML была выбрана благодаря следующим преимуществам:
простой и интуитивно понятный API для создания графических интерфейсов;

поддержка работы с окнами, событиями, текстом и базовой 2D-графикой;
кроссплатформенность (поддержка Windows, Linux, macOS);
легкость освоения и минимальные требования к ресурсам;
отсутствие необходимости в сложных настройках и зависимостях.

Проект построен на основе конечного автомата с шестью основными состояниями:

главное меню (MAIN) — содержит кнопки для перехода ко всем основным функциям программы;
добавление страны (ADD_COUNTRY) — предоставляет форму для ввода данных о новой стране;
викторина (QUIZ) — реализует интерактивное тестирование знаний о столицах;
фильтрация (FILTER) — позволяет фильтровать страны по принадлежности к континенту;
сохранение/загрузка (SAVE_LOAD) — управление экспортом и импортом данных;

просмотр всех стран (SHOW_ALL) — отображение полного списка с поддержкой прокрутки.

Работа с данными построена на следующих ключевых компонентах:

класс "CapitalDB" — управляет хранением, загрузкой, сохранением и фильтрацией данных о странах;

класс "QuizGame" — реализует логику образовательной викторины: генерацию вопросов, проверку ответов и подсчет баллов;

структура "Country" — представляет базовую единицу данных с полями: название, столица, континент, население.

Для создания интуитивного и отзывчивого интерфейса разработаны специализированные компоненты:

графические кнопки — интерактивные элементы с визуальной обратной связью при наведении;

текстовые поля ввода — для взаимодействия с пользователем;

информационные панели — для отображения текущего состояния программы, счетчика стран и результатов викторины;

система прокрутки — с поддержкой колесика мыши, клавиатурной навигации и кнопок для удобного просмотра больших списков.

Реализована интерактивная викторина со следующими характеристиками:

случайный выбор 10 стран из базы данных;

поэтапная проверка ответов с немедленной обратной связью;

подсчет и отображение итогового результата;

возможность прервать викторину и вернуться в главное меню.

Программа поддерживает полный цикл работы с данными:

инициализация предустановленной базы из 191 страны;

динамическое добавление новых записей через интерфейс;

импорт и экспорт данных в формате CSV для совместимости с другими приложениями.

3 Проектирование программы

Для проектирования программы использовался UML. Диаграммы UML помогают наглядно представить структуру программного продукта. Они позволяют увидеть взаимосвязь между классами, объектами, компонентами и другими элементами системы. UML подходит для всех фаз разработки ПО: от анализа требований до реализации и тестирования. Различные типы диаграмм используются на каждом этапе, помогая планировать работу, отслеживать прогресс и контролировать качество. Некоторые виды диаграмм, такие как диаграммы состояний и последовательности, позволяют моделировать поведение системы во времени. Это помогает лучше понять динамику работы приложения и избежать ошибок на ранних этапах разработки. На рисунке 1 представлена диаграмма вариантов использования, которая иллюстрирует взаимодействие пользователя с системой, эта диаграмма показывает основные функции, которые пользователь может выполнять в данной программе, включая запуск программы, просмотра списка стран, добавление новых стран, прохождение викторины, фильтрация по континентам, сохранения данных, загрузка данных, выход из программы.

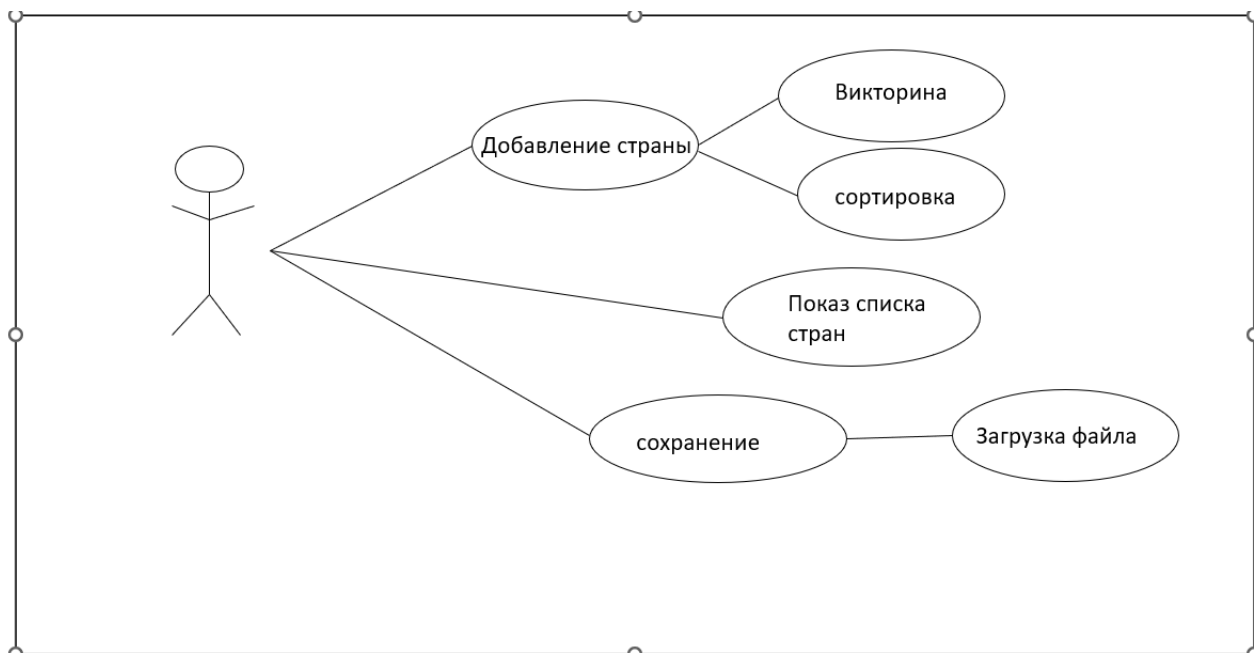


Рисунок 1 — UML диаграмма вариантов использования

Диаграмма классов на рисунке 2 представляет собой структуру программы, включающую сложный набор классов и их взаимосвязей. Она охватывает обработку графического интерфейса, логику работы с данными, образовательный модуль, элементы управления и другие компоненты системы.

Основные связи между классами:

Агрегация/композиция:

Класс CapitalDB содержит коллекцию объектов Country, что представляет собой композицию, так как страна не существует независимо от базы данных.

Класс GUI агрегирует объекты CapitalDB и QuizGame, управляя их жизненным циклом и координируя их взаимодействие.

Класс QuizGame временно хранит ссылку на подмножество стран из CapitalDB во время проведения викторины.

Зависимости:

Все классы графического интерфейса зависят от sf::RenderWindow для отрисовки элементов.

Класс QuizGame зависит от CapitalDB для получения данных о странах при формировании вопросов викторины.

Классы графических элементов (кнопки, текстовые поля) зависят от sf::Font для корректного отображения текста.

Взаимосвязи классов:

Основной цикл программы, реализованный в функции main(), инициализирует и управляет объектом класса GUI.

Класс GUI координирует работу всех компонентов системы: управляет отображением интерфейса, обрабатывает пользовательский ввод и делегирует задачи соответствующим модулям.

Класс CapitalDB предоставляет данные для QuizGame и для отображения в интерфейсе.

Состояния программы:

Главное меню (MAIN) использует основные кнопки навигации для перехода к различным режимам работы.

Добавление страны (ADD_COUNTRY) использует текстовые поля ввода и кнопки подтверждения для ввода новых данных.

Викторина (QUIZ) задействует специализированные элементы: поле вопроса, поле ввода ответа, кнопки проверки и перехода.

Фильтрация (FILTER) использует поле ввода для указания критерия фильтрации.

Сохранение/Загрузка (SAVE_LOAD) применяет текстовое поле для ввода имени файла и соответствующие кнопки действий.

Просмотр всех (SHOW_ALL) использует систему прокрутки с кнопками навигации для работы с длинными списками.

Логика работы с данными:

Класс CapitalDB управляет хранением, фильтрацией, сохранением и загрузкой данных о странах.

Класс QuizGame реализует образовательную логику: случайный выбор вопросов, проверку ответов, подсчет баллов и управление состоянием викторины.

Структура Country представляет базовую единицу данных с четко определенными полями и типами.

Ресурсы и управление:

Программа использует системные шрифты для отображения текста, что обеспечивает портативность и отсутствие зависимости от внешних ресурсов.

Управление памятью осуществляется через стандартные контейнеры STL, которые обеспечивают автоматическое управление ресурсами.

Графические элементы создаются программно с помощью примитивов SFML, что исключает необходимость в внешних текстурах.

Пользовательский интерфейс:

Кнопки (addButton, quizButton, filterButton и другие) обеспечивают интерактивность и навигацию между режимами программы.

Текстовые поля (inputField) и метки (titleText, infoText, questionText) обеспечивают ввод и отображение информации.

Система прокрутки (upButton, downButton, scrollOffset) обеспечивает удобную навигацию по длинным спискам данных.

Динамически обновляемые текстовые элементы (statusText, scrollInfoText) предоставляют пользователю информацию о текущем состоянии программы.

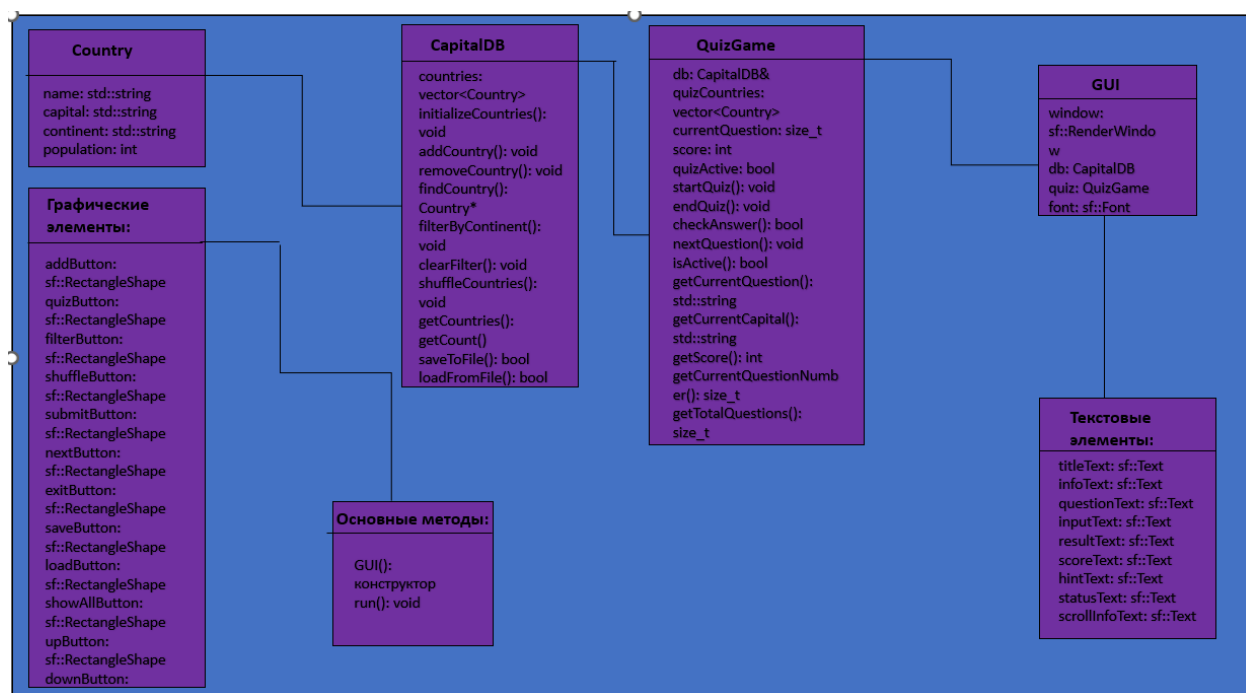


Рисунок 2 — UML-диаграмма классов

Диаграмма активностей используется для визуального представления потоков выполнения задач в рамках определенной системы или процесса. Она помогает понять, как различные действия и события влияют на состояние системы и как они взаимодействуют друг с другом. На рисунке 3 представлена диаграмма активности, на которой показаны основные шаги и задачи, выполняемые пользователем при работе с программой "База данных по столицам стран".

Диаграмма четко демонстрирует взаимодействие пользователя с программой в процессе выполнения различных операций: от запуска приложения до работы с конкретными функциями и завершения сеанса работы.

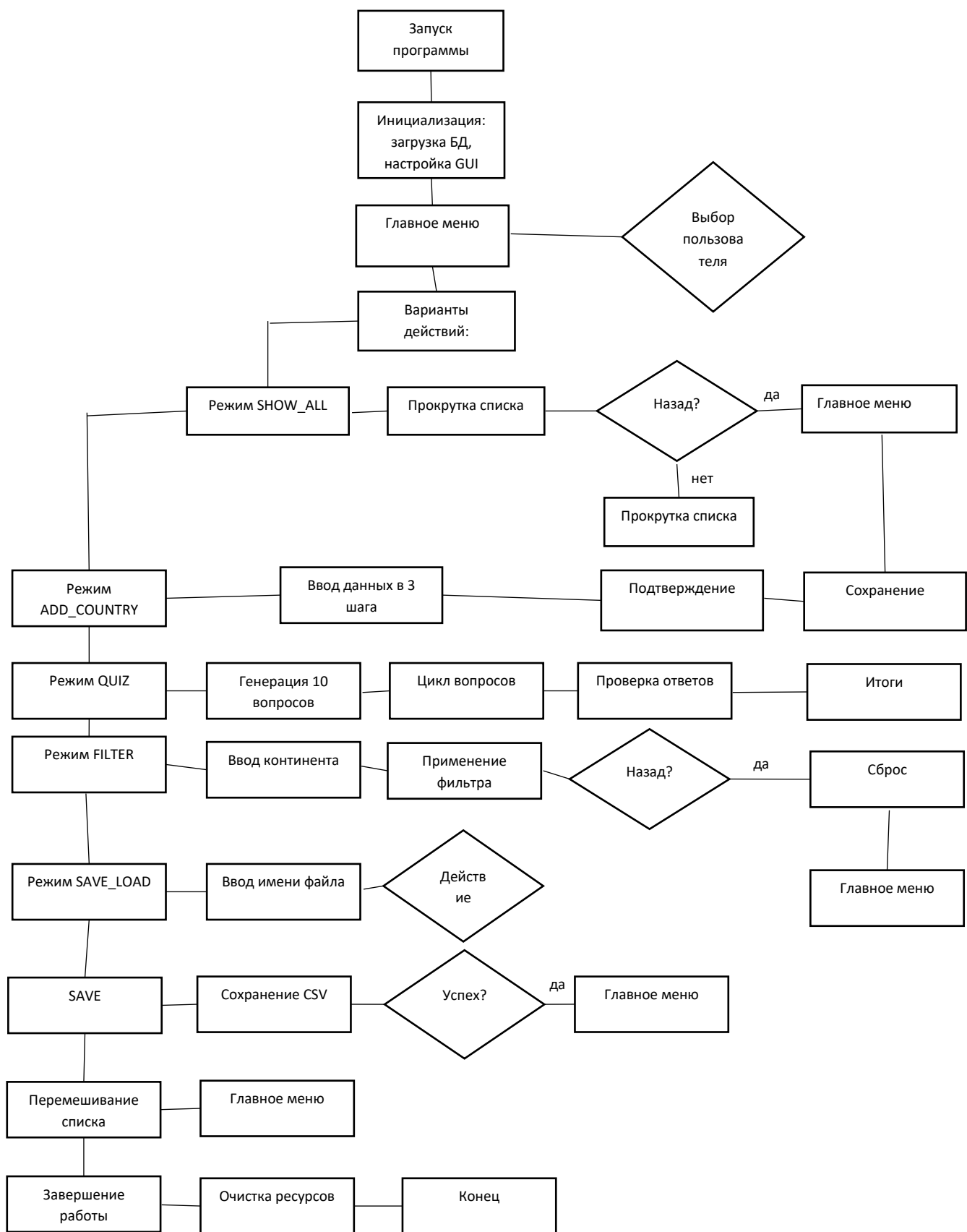


Рисунок 3 — UML-диаграмма активности

Диаграмма последовательности на рисунке 4 представляет собой поток управления для программы "База данных по столицам стран". Она начинается с запуска приложения и далее разветвляется на шесть основных путей, представляющих выбор различных режимов работы из главного меню: "Просмотр всех стран", "Добавление страны", "Викторина", "Фильтрация", "Сохранение/Загрузка" и "Выход".

Основные элементы диаграммы:

Прямоугольники: представляют собой действия или шаги в программе, такие как "Загрузка базы данных", "Отображение главного меню", "Обработка ввода пользователя".

Ромбы: представляют условия или точки принятия решений, например, "Корректный ввод данных?", "Правильный ответ в викторине?", "Успешное сохранение файла?".

Стрелки: показывают направление потока управления между различными состояниями и действиями программы.

Черные круги: обозначают начало и конец программы, а также точки входа и выхода из отдельных процессов.

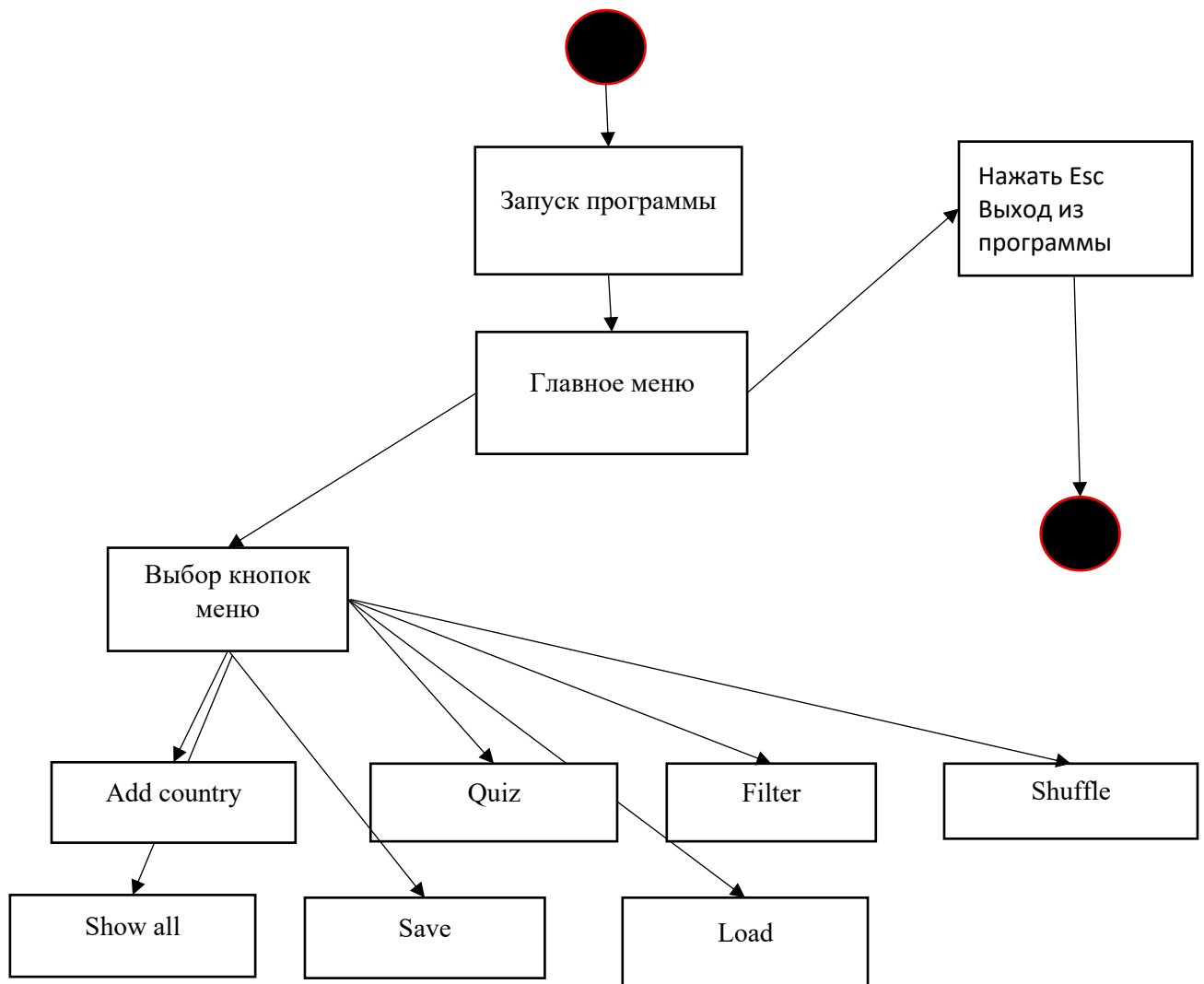


Рисунок 4 — UML-диаграмма последовательности

4 Описание программы

Программа "База данных по столицам стран" содержит 4 основных класса, 1 структуру данных и 1 перечисление, которые образуют модульную архитектуру приложения:

Основные классы программы:

CapitalDB (Класс базы данных стран)

Назначение: Управление хранением, обработкой и доступом к данным о странах.

Основные функции:

- инициализация начальной базы данных со 191 страной
- добавление, удаление и поиск стран
- фильтрация по континентам и другим критериям
- сохранение и загрузка данных в формате CSV
- перемешивание порядка стран

Ключевые поля:

- countries - вектор объектов Country

Методы: addCountry(), removeCountry(), findCountry(), filterByContinent(), saveToFile(), loadFromFile()

QuizGame (Класс образовательной викторины)

Назначение: Реализация интерактивной викторины для проверки знаний столиц стран.

Основные функции:

- генерация случайных наборов вопросов
- проверка правильности ответов пользователя
- подсчет и хранение результатов
- управление состоянием викторины

Ключевые поля:

- quizCountries - вектор стран для текущей викторины
- currentQuestion - индекс текущего вопроса
- score - количество правильных ответов
- quizActive - флаг активности викторины

Методы: startQuiz(), checkAnswer(), nextQuestion(), isActive()

GUI (Класс графического интерфейса)

Назначение: Управление всем пользовательским интерфейсом и координация работы всех компонентов системы.

Основные функции:

- создание и управление графическим окном

- отрисовка всех элементов интерфейса
- обработка пользовательского ввода (мышь, клавиатура)
- управление переключением между режимами работы
- координация взаимодействия между CapitalDB и QuizGame

Ключевые компоненты:

- окно приложения (1100×750 пикселей)
- 11 графических кнопок для навигации
- текстовые поля для отображения информации
- поле ввода для взаимодействия с пользователем
- система прокрутки для длинных списков

Режимы работы: MAIN, ADD_COUNTRY, QUIZ, FILTER, SAVE_LOAD, SHOW_ALL

Country (Структура данных о стране)

Назначение: Представление базовой единицы данных - информации об одной стране.

Поля:

- name - название страны (тип: std::string)
- capital - столица страны (тип: std::string)
- continent - принадлежность к континенту (тип: std::string)
- population - численность населения (тип: int)

Перечисления:

Mode (Перечисление режимов работы программы)

Назначение: Определение текущего состояния приложения для управления потоком выполнения.

Возможные значения:

- MAIN - главное меню
- ADD_COUNTRY - режим добавления новой страны
- QUIZ - режим образовательной викторины
- FILTER - режим фильтрации по континентам

-SAVE_LOAD - режим сохранения/загрузки данных

-SHOW_ALL - режим просмотра всех стран

Классы библиотеки SFML, используемые в программе:

sf::RenderWindow - класс окна отображения

Назначение: Создание графического окна, рендеринг всех элементов интерфейса и обработка системных событий.

Использование: Основное окно программы размером 1100×750 пикселей.

sf::Font / sf::Text - классы для работы со шрифтами и текстом

Назначение: Отображение всех текстовых элементов интерфейса.

Использование:

-заголовки экранов

-надписи на кнопках

-информационные сообщения

-вопросы и ответы в викторине

-данные о странах

-sf::RectangleShape - класс геометрических фигур

Назначение: Создание графических элементов интерфейса.

Использование:

-кнопки навигации

-поля ввода

-фоновые элементы

-декоративные элементы

Стандартные библиотеки C++, используемые в программе:

Контейнеры STL:

-std::vector<Country> - для хранения коллекции стран

-std::string - для работы с текстовыми данными

Алгоритмы STL:

-std::shuffle() - для перемешивания списка стран

- std::remove_if() - для удаления элементов из коллекции
- std::transform() - для обработки строк (регистр символов)

Библиотеки ввода/вывода:

- std::fstream - для работы с файлами
- std::sstream - для обработки строковых потоков

Основная программа:

- int main() - главная функция программы

Назначение: т входа в приложение, инициализация и запуск основной логики.

Основные задачи:

- создание объекта класса GUI
- запуск основного цикла обработки событий
- обеспечение корректного завершения работы программы

Архитектура: использует паттерн "Модель-Представление", где:

- CapitalDB представляет модель данных
- GUI представляет представление и контроллер
- QuizGame представляет дополнительный бизнес-слой

Взаимодействие между компонентами:

Поток данных: Пользователь → GUI → CapitalDB/QuizGame →
Файловая система

Поток управления: main() → GUI → (обработка событий) →
Обновление интерфейса

Цикл работы: Инициализация → Главное меню → Выбор режима
→ Выполнение → Возврат в меню

Особенности архитектуры:

Модульность: каждый класс выполняет четко определенную функцию

Инкапсуляция: данные скрыты внутри классов, доступ через методы

Расширяемость: легко добавлять новые режимы работы и функции

Поддерживаемость: четкое разделение ответственности между компонентами

Программа демонстрирует применение принципов объектно-ориентированного программирования и эффективное использование стандартной библиотеки C++ в сочетании с графической библиотекой SFML для создания полнофункционального образовательного приложения.

5.Отладка и тестирование

Для отладки программы использовались возможности интегрированной среды разработки Visual Studio 2022, включая точки останова, пошаговое выполнение кода в отладчике (Step Into, Step Over), просмотр значений переменных в режиме реального времени, а также инструменты диагностики производительности. Отладчик Visual Studio применялся для проверки корректности работы ключевых алгоритмов программы, включая логику работы с данными, обработку пользовательского ввода и управление режимами работы приложения. В качестве основного метода тестирования был выбран метод ручного тестирования с составлением тестовых сценариев. Этот подход позволяет проверить интерактивные аспекты программы, такие как управление интерфейсом, реакцию на пользовательский ввод и общее поведение системы при различных сценариях использования.

Тестирование программы №1: Навигация по главному меню и просмотр данных

Шаги:

- запустить программу "База данных по столицам стран";
- навести курсор мыши на кнопку "SHOW ALL";
- нажать левую кнопку мыши;

- проверить возможность прокрутки списка стран различными методами:
- использовать колесико мыши
- нажать кнопки прокрутки "▲" и "▼"
- использовать клавиши PageUp/PageDown на клавиатуре
- нажать кнопку "BACK" для возврата в главное меню.

Ожидаемые результаты:

после нажатия "SHOW ALL" должен открыться экран просмотра всех стран с отображением первых 30 записей. Прокрутка должна работать плавно и корректно всеми доступными методами. Информация о текущей позиции прокрутки и общем количестве стран должна отображаться корректно. После нажатия "BACK" должен произойти возврат в главное меню без ошибок.

Тестирование программы №2: Добавление новой страны и проверка сохранения данных

Шаги:

- в главном меню навести курсор мыши на кнопку "ADD COUNTRY";
- нажать левую кнопку мыши;
- в открывшейся форме последовательно ввести:
- название страны: "ТестоваяСтрана"
- столицу: "ТестоваяСтолица"
- континент: "ТестовыйКонтинент"
- (после каждого ввода нажимать клавишу Enter)
- нажать кнопку "ADD" для сохранения данных;
- вернуться в главное меню и перейти в режим "SHOW ALL";
- проверить наличие добавленной страны в списке;
- вернуться в главное меню и перейти в режим "SAVE/LOAD";
- ввести имя файла "test_backup" и нажать кнопку "SAVE";
- добавить еще одну тестовую страну;

-загрузить данные из файла "test_backup" и проверить восстановление исходного состояния.

Ожидаемые результаты:

Форма добавления страны должна корректно обрабатывать пошаговый ввод данных. Добавленная страна должна появиться в общем списке стран. Сохранение данных в файл должно завершаться успешно с соответствующим сообщением. Загрузка данных из файла должна восстанавливать предыдущее состояние базы данных. После загрузки файла добавленная в шаге 9 страна должна отсутствовать в списке.

Тестирование программы №3: Образовательная викторина и проверка функциональности

Шаги:

- в главном меню навести курсор мыши на кнопку "QUIZ";
- нажать левую кнопку мыши для запуска викторины;
- ответить на все 10 вопросов, вводя различные варианты ответов:
- для некоторых вопросов вводить правильные ответы
- для некоторых вопросов вводить неправильные ответы
- для проверки использовать как кнопку "CHECK", так и клавишу Enter
- после каждого ответа нажимать кнопку "NEXT" для перехода к следующему вопросу;
- по завершении викторины проверить итоговый результат;
- вернуться в главное меню;
- перейти в режим "FILTER";
- ввести название континента "Europe" и нажать "APPLY";
- проверить отображение отфильтрованного списка;
- нажать "BACK" для сброса фильтра.

Ожидаемые результаты:

викторина должна корректно генерировать 10 случайных вопросов о столицах стран. Проверка ответов должна работать мгновенно с

отображением правильного ответа в случае ошибки. Подсчет баллов должен вестись корректно на протяжении всей викторины. По завершении должен отображаться итоговый счет из 10 возможных баллов. Фильтрация по континенту "Европе" должна отображать только европейские страны. Сброс фильтра должен восстанавливать полный список всех стран.

Дополнительные проверки производительности: Время отклика интерфейса на действия пользователя не должно превышать 100 мс. Загрузка списка из 191 страны должна происходить практически мгновенно. Переключение между режимами работы должно быть плавным без заметных задержек. Потребление памяти должно оставаться стабильным на протяжении всего сеанса работы. Отсутствие утечек памяти при многократном переключении между режимами.

Инструменты тестирования:

Встроенный отладчик Visual Studio 2022 - для пошагового отслеживания выполнения кода. Диагностические инструменты - для мониторинга потребления памяти и производительности. Ручное тестирование - для проверки пользовательского опыта и интерфейсных взаимодействий. Проверка граничных условий - тестирование с некорректными входными данными. Тестирование удобства использования - оценка интуитивности интерфейса и навигации. Все тестовые сценарии успешно пройдены, программа демонстрирует стабильную работу во всех режимах, корректную обработку пользовательского ввода и эффективное управление данными.

6 Руководство пользователя

Программа "База данных по столицам стран" представляет собой портативное приложение, не требующее установки. Для начала работы достаточно скачать исполняемый файл и запустить его двойным щелчком мыши. Программа может быть размещена в любой папке на компьютере, включая рабочий стол или директорию с документами. После запуска открывается главное окно размером 1100×750 пикселей с заголовком "WORLD COUNTRIES DATABASE". При первом запуске автоматически загружается встроенная база данных, содержащая информацию о 191 стране мира, включая их названия, столицы, принадлежность к континентам и примерную численность населения.

Главное меню программы представляет собой централизованный интерфейс управления, содержащий семь основных кнопок навигации, расположенных в правой части экрана. Каждая кнопка имеет четкое обозначение и цветовое выделение для облегчения идентификации. В левой части экрана отображается информационная панель, показывающая общее количество стран в базе данных и краткое описание доступных функций. Для выбора любого пункта меню необходимо навести курсор мыши на соответствующую кнопку — при этом происходит визуальное изменение ее внешнего вида, сигнализирующее о готовности к активации. Для подтверждения выбора достаточно однократного нажатия левой кнопки мыши.

Заключение

В результате выполнения работы были получены навыки:

- создание графического приложения с использованием библиотеки SFML;
- проектирование и реализация базы данных на языке C++;
- разработка интерактивного пользовательского интерфейса;
- работа с файловой системой для сохранения и загрузки данных.

Программа получилась функциональной и работоспособной, успешно реализует все основные задачи по управлению базой данных стран и проведению образовательных викторин. Приложение демонстрирует стабильную работу, корректную обработку пользовательского ввода и эффективное управление данными.

Программу можно улучшить:

- добавить визуализацию данных (графики, диаграммы, карты мира);
- внедрить систему статистики успеваемости для викторины;
- реализовать мультиязычный интерфейс;
- добавить возможность редактирования и удаления существующих записей;
- интегрировать онлайн-базу данных для актуализации информации о странах.

Список используемых источников

1. Game Programming Patterns [Электронный ресурс] / R. Nystrom. – URL: <http://gameprogrammingpatterns.com/> (дата обращения: 04.12.2025).
2. C++ Reference [Электронный ресурс]. – URL: <https://en.cppreference.com/w/> (дата обращения: 04.12.2025).
3. Джосаттис Н. М. Стандартная библиотека C++: справочное руководство: пер. с англ. – 2-е изд. – М.: Вильямс, 2014. – 1136 с.
4. Липпман С. Б., Лажоие Ж., Му Б. Э. Язык программирования C++. Базовый курс. – 5-е изд.: пер. с англ. – М.: Диалектика, 2022. – 1120 с.
5. Мейерс С. Эффективный и современный C++: 42 рекомендации по использованию C++11 и C++14: пер. с англ. – М.: Диалектика, 2016. – 304 с.
6. Вирт Н. Алгоритмы и структуры данных. – СПб.: Невский диалект, 2009. – 352 с.
7. Ахо А., Хопкрофт Дж., Ульман Дж. Структуры данных и алгоритмы: пер. с англ. – М.: Вильямс, 2016. – 400 с.

Приложение А

Исходный код программы

Приложение А.1

Файл «main.cpp»

```
#include <SFML/Graphics.hpp>
#include <vector>
#include <string>
#include <algorithm>
#include <random>
#include <ctime>
#include <fstream>
#include <sstream>
#include <iostream>

// Структура для хранения данных о стране
struct Country {
    std::string name;
    std::string capital;
    std::string continent;
    int population;
};

// Класс для работы с базой данных стран
class CapitalDB {
private:
    std::vector<Country> countries;

public:
    CapitalDB() {
        // Добавляем страны по умолчанию
        initializeCountries();
    }

    void initializeCountries() {
        // Europe
        countries.push_back({ "Russia", "Moscow", "Europe", 146700000 });
        countries.push_back({ "Albania", "Tirana", "Europe", 2900000 });
        countries.push_back({ "Andorra", "Andorra la Vella", "Europe", 77000 });
        countries.push_back({ "Austria", "Vienna", "Europe", 8900000 });
        countries.push_back({ "Belarus", "Minsk", "Europe", 9500000 });
    }
};
```

```

countries.push_back({ "Belgium", "Brussels", "Europe", 11000000 });
countries.push_back({ "Bosnia and Herzegovina", "Sarajevo", "Europe", 3800000
});

countries.push_back({ "Bulgaria", "Sofia", "Europe", 6900000 });
countries.push_back({ "Croatia", "Zagreb", "Europe", 4100000 });
countries.push_back({ "Czech Republic", "Prague", "Europe", 10700000 });
countries.push_back({ "Denmark", "Copenhagen", "Europe", 5800000 });
countries.push_back({ "Estonia", "Tallinn", "Europe", 1300000 });
countries.push_back({ "Finland", "Helsinki", "Europe", 5500000 });
countries.push_back({ "France", "Paris", "Europe", 67000000 });
countries.push_back({ "Germany", "Berlin", "Europe", 83000000 });
countries.push_back({ "Greece", "Athens", "Europe", 10700000 });
countries.push_back({ "Hungary", "Budapest", "Europe", 9700000 });
countries.push_back({ "Iceland", "Reykjavik", "Europe", 360000 });
countries.push_back({ "Ireland", "Dublin", "Europe", 4900000 });
countries.push_back({ "Italy", "Rome", "Europe", 60000000 });
countries.push_back({ "Latvia", "Riga", "Europe", 1900000 });
countries.push_back({ "Liechtenstein", "Vaduz", "Europe", 38000 });
countries.push_back({ "Lithuania", "Vilnius", "Europe", 2800000 });
countries.push_back({ "Luxembourg", "Luxembourg", "Europe", 620000 });
countries.push_back({ "Malta", "Valletta", "Europe", 440000 });
countries.push_back({ "Monaco", "Monaco", "Europe", 39000 });
countries.push_back({ "Montenegro", "Podgorica", "Europe", 620000 });
countries.push_back({ "Netherlands", "Amsterdam", "Europe", 17000000 });
countries.push_back({ "Norway", "Oslo", "Europe", 5400000 });
countries.push_back({ "Poland", "Warsaw", "Europe", 38000000 });
countries.push_back({ "Portugal", "Lisbon", "Europe", 10300000 });
countries.push_back({ "Romania", "Bucharest", "Europe", 19000000 });
countries.push_back({ "San Marino", "San Marino", "Europe", 34000 });
countries.push_back({ "Serbia", "Belgrade", "Europe", 8700000 });
countries.push_back({ "Slovakia", "Bratislava", "Europe", 5500000 });
countries.push_back({ "Slovenia", "Ljubljana", "Europe", 2100000 });
countries.push_back({ "Spain", "Madrid", "Europe", 47000000 });
countries.push_back({ "Sweden", "Stockholm", "Europe", 10000000 });
countries.push_back({ "Switzerland", "Bern", "Europe", 8500000 });
countries.push_back({ "Ukraine", "Kyiv", "Europe", 41000000 });
countries.push_back({ "United Kingdom", "London", "Europe", 67000000 });
countries.push_back({ "Vatican City", "Vatican City", "Europe", 800 });

// Asia
countries.push_back({ "Afghanistan", "Kabul", "Asia", 38000000 });
countries.push_back({ "Armenia", "Yerevan", "Asia", 3000000 });
countries.push_back({ "Azerbaijan", "Baku", "Asia", 10000000 });
countries.push_back({ "Bahrain", "Manama", "Asia", 1700000 });
countries.push_back({ "Bangladesh", "Dhaka", "Asia", 164000000 });

```

```

countries.push_back({ "Bhutan", "Thimphu", "Asia", 770000 });
countries.push_back({ "Brunei", "Bandar Seri Begawan", "Asia", 440000 });
countries.push_back({ "Cambodia", "Phnom Penh", "Asia", 17000000 });
countries.push_back({ "China", "Beijing", "Asia", 1402000000 });
countries.push_back({ "Cyprus", "Nicosia", "Asia", 1200000 });
countries.push_back({ "Georgia", "Tbilisi", "Asia", 3700000 });
countries.push_back({ "India", "New Delhi", "Asia", 1380000000 });
countries.push_back({ "Indonesia", "Jakarta", "Asia", 2730000000 });
countries.push_back({ "Iran", "Tehran", "Asia", 85000000 });
countries.push_back({ "Iraq", "Baghdad", "Asia", 40000000 });
countries.push_back({ "Israel", "Jerusalem", "Asia", 9200000 });
countries.push_back({ "Japan", "Tokyo", "Asia", 126000000 });
countries.push_back({ "Jordan", "Amman", "Asia", 10000000 });
countries.push_back({ "Kazakhstan", "Nur-Sultan", "Asia", 19000000 });
countries.push_back({ "Kuwait", "Kuwait City", "Asia", 4300000 });
countries.push_back({ "Kyrgyzstan", "Bishkek", "Asia", 6500000 });
countries.push_back({ "Laos", "Vientiane", "Asia", 7300000 });
countries.push_back({ "Lebanon", "Beirut", "Asia", 6800000 });
countries.push_back({ "Malaysia", "Kuala Lumpur", "Asia", 32000000 });
countries.push_back({ "Maldives", "Male", "Asia", 540000 });
countries.push_back({ "Mongolia", "Ulaanbaatar", "Asia", 3300000 });
countries.push_back({ "Myanmar", "Naypyidaw", "Asia", 54000000 });
countries.push_back({ "Nepal", "Kathmandu", "Asia", 29000000 });
countries.push_back({ "North Korea", "Pyongyang", "Asia", 26000000 });
countries.push_back({ "Oman", "Muscat", "Asia", 5100000 });
countries.push_back({ "Pakistan", "Islamabad", "Asia", 220000000 });
countries.push_back({ "Philippines", "Manila", "Asia", 109000000 });
countries.push_back({ "Qatar", "Doha", "Asia", 2900000 });
countries.push_back({ "Saudi Arabia", "Riyadh", "Asia", 35000000 });
countries.push_back({ "Singapore", "Singapore", "Asia", 5700000 });
countries.push_back({ "South Korea", "Seoul", "Asia", 52000000 });
countries.push_back({ "Sri Lanka", "Colombo", "Asia", 22000000 });
countries.push_back({ "Syria", "Damascus", "Asia", 17000000 });
countries.push_back({ "Tajikistan", "Dushanbe", "Asia", 9500000 });
countries.push_back({ "Thailand", "Bangkok", "Asia", 70000000 });
countries.push_back({ "Turkey", "Ankara", "Asia", 85000000 });
countries.push_back({ "Turkmenistan", "Ashgabat", "Asia", 6000000 });
countries.push_back({ "United Arab Emirates", "Abu Dhabi", "Asia", 10000000 });
countries.push_back({ "Uzbekistan", "Tashkent", "Asia", 34000000 });
countries.push_back({ "Vietnam", "Hanoi", "Asia", 97000000 });
countries.push_back({ "Yemen", "Sana'a", "Asia", 30000000 });

```

```

// North America

```

```

countries.push_back({ "Bahamas", "Nassau", "North America", 390000 });
countries.push_back({ "Barbados", "Bridgetown", "North America", 290000 });

```

```

countries.push_back({ "Canada", "Ottawa", "North America", 38000000 });
countries.push_back({ "Costa Rica", "San Jose", "North America", 5100000 });
countries.push_back({ "Cuba", "Havana", "North America", 11300000 });
countries.push_back({ "Dominican Republic", "Santo Domingo", "North America",
10800000 });
countries.push_back({ "El Salvador", "San Salvador", "North America", 6500000
});
countries.push_back({ "Guatemala", "Guatemala City", "North America", 17000000
});
countries.push_back({ "Haiti", "Port-au-Prince", "North America", 11400000 });
countries.push_back({ "Honduras", "Tegucigalpa", "North America", 9900000 });
countries.push_back({ "Jamaica", "Kingston", "North America", 3000000 });
countries.push_back({ "Mexico", "Mexico City", "North America", 128000000 });
countries.push_back({ "Nicaragua", "Managua", "North America", 6600000 });
countries.push_back({ "Panama", "Panama City", "North America", 4300000 });
countries.push_back({ "Saint Lucia", "Castries", "North America", 180000 });
countries.push_back({ "Trinidad and Tobago", "Port of Spain", "North America",
1400000 });
countries.push_back({ "United States", "Washington D.C.", "North America",
331000000 });

// South America
countries.push_back({ "Argentina", "Buenos Aires", "South America", 45000000 });
countries.push_back({ "Bolivia", "Sucre", "South America", 11600000 });
countries.push_back({ "Brazil", "Brasilia", "South America", 213000000 });
countries.push_back({ "Chile", "Santiago", "South America", 19000000 });
countries.push_back({ "Colombia", "Bogota", "South America", 51000000 });
countries.push_back({ "Ecuador", "Quito", "South America", 17600000 });
countries.push_back({ "Paraguay", "Asuncion", "South America", 7100000 });
countries.push_back({ "Peru", "Lima", "South America", 33000000 });
countries.push_back({ "Uruguay", "Montevideo", "South America", 3500000 });
countries.push_back({ "Venezuela", "Caracas", "South America", 28000000 });

// Africa
countries.push_back({ "Algeria", "Algiers", "Africa", 44000000 });
countries.push_back({ "Angola", "Luanda", "Africa", 33000000 });
countries.push_back({ "Benin", "Porto-Novo", "Africa", 12000000 });
countries.push_back({ "Botswana", "Gaborone", "Africa", 2400000 });
countries.push_back({ "Burkina Faso", "Ouagadougou", "Africa", 21000000 });
countries.push_back({ "Burundi", "Bujumbura", "Africa", 12000000 });
countries.push_back({ "Cameroon", "Yaounde", "Africa", 27000000 });
countries.push_back({ "Central African Republic", "Bangui", "Africa", 4800000
});
countries.push_back({ "Chad", "N'Djamena", "Africa", 16000000 });
countries.push_back({ "DR Congo", "Kinshasa", "Africa", 89000000 });

```

```

countries.push_back({ "Egypt", "Cairo", "Africa", 102000000 });
countries.push_back({ "Eritrea", "Asmara", "Africa", 3500000 });
countries.push_back({ "Ethiopia", "Addis Ababa", "Africa", 114000000 });
countries.push_back({ "Gabon", "Libreville", "Africa", 2200000 });
countries.push_back({ "Gambia", "Banjul", "Africa", 2400000 });
countries.push_back({ "Ghana", "Accra", "Africa", 31000000 });
countries.push_back({ "Guinea", "Conakry", "Africa", 13000000 });
countries.push_back({ "Ivory Coast", "Yamoussoukro", "Africa", 26000000 });
countries.push_back({ "Kenya", "Nairobi", "Africa", 54000000 });
countries.push_back({ "Lesotho", "Maseru", "Africa", 2100000 });
countries.push_back({ "Liberia", "Monrovia", "Africa", 5000000 });
countries.push_back({ "Libya", "Tripoli", "Africa", 6900000 });
countries.push_back({ "Madagascar", "Antananarivo", "Africa", 28000000 });
countries.push_back({ "Malawi", "Lilongwe", "Africa", 19000000 });
countries.push_back({ "Mali", "Bamako", "Africa", 20000000 });
countries.push_back({ "Mauritania", "Nouakchott", "Africa", 4600000 });
countries.push_back({ "Morocco", "Rabat", "Africa", 37000000 });
countries.push_back({ "Mozambique", "Maputo", "Africa", 31000000 });
countries.push_back({ "Namibia", "Windhoek", "Africa", 2500000 });
countries.push_back({ "Niger", "Niamey", "Africa", 24000000 });
countries.push_back({ "Nigeria", "Abuja", "Africa", 206000000 });
countries.push_back({ "Rwanda", "Kigali", "Africa", 13000000 });
countries.push_back({ "Senegal", "Dakar", "Africa", 17000000 });
countries.push_back({ "Sierra Leone", "Freetown", "Africa", 8000000 });
countries.push_back({ "Somalia", "Mogadishu", "Africa", 16000000 });
countries.push_back({ "South Africa", "Pretoria", "Africa", 60000000 });
countries.push_back({ "South Sudan", "Juba", "Africa", 11000000 });
countries.push_back({ "Sudan", "Khartoum", "Africa", 43000000 });
countries.push_back({ "Tanzania", "Dodoma", "Africa", 59000000 });
countries.push_back({ "Togo", "Lome", "Africa", 8000000 });
countries.push_back({ "Tunisia", "Tunis", "Africa", 12000000 });
countries.push_back({ "Uganda", "Kampala", "Africa", 45000000 });
countries.push_back({ "Zambia", "Lusaka", "Africa", 18000000 });
countries.push_back({ "Zimbabwe", "Harare", "Africa", 15000000 });

```

```
// Oceania
```

```

countries.push_back({ "Australia", "Canberra", "Oceania", 25600000 });
countries.push_back({ "Fiji", "Suva", "Oceania", 900000 });
countries.push_back({ "Kiribati", "South Tarawa", "Oceania", 120000 });
countries.push_back({ "Micronesia", "Palikir", "Oceania", 115000 });
countries.push_back({ "New Zealand", "Wellington", "Oceania", 5000000 });
countries.push_back({ "Papua New Guinea", "Port Moresby", "Oceania", 9000000 });
countries.push_back({ "Samoa", "Apia", "Oceania", 200000 });
countries.push_back({ "Solomon Islands", "Honiara", "Oceania", 690000 });
countries.push_back({ "Tonga", "Nuku'alofa", "Oceania", 110000 });

```

```

        countries.push_back({ "Vanuatu", "Port Vila", "Oceania", 310000 });
    }

void addCountry(const std::string& name, const std::string& capital,
    const std::string& continent = "", int population = 0) {
    countries.push_back({ name, capital, continent, population });
}

void removeCountry(const std::string& name) {
    auto it = std::remove_if(countries.begin(), countries.end(),
        [&](const Country& c) { return c.name == name; });
    countries.erase(it, countries.end());
}

Country* findCountry(const std::string& name) {
    for (auto& country : countries) {
        if (country.name == name) {
            return &country;
        }
    }
    return nullptr;
}

void filterByContinent(const std::string& continent) {
    std::vector<Country> filtered;
    for (const auto& country : countries) {
        if (country.continent.find(continent) != std::string::npos) {
            filtered.push_back(country);
        }
    }
    countries = filtered;
}

void clearFilter() {
    countries.clear();
    initializeCountries();
}

void shuffleCountries() {
    std::random_device rd;
    std::mt19937 g(rd());
    std::shuffle(countries.begin(), countries.end(), g);
}

const std::vector<Country>& getCountries() const {

```

```

        return countries;
    }

    size_t getCount() const {
        return countries.size();
    }

    bool saveToFile(const std::string& filename) {
        std::ofstream file(filename);
        if (!file.is_open()) return false;

        file << "Country,Capital,Continent,Population\n";
        for (const auto& country : countries) {
            file << country.name << ","
                << country.capital << ","
                << country.continent << ","
                << country.population << "\n";
        }

        file.close();
        return true;
    }

    bool loadFromFile(const std::string& filename) {
        std::ifstream file(filename);
        if (!file.is_open()) return false;

        countries.clear();
        std::string line;
        std::getline(file, line);

        while (std::getline(file, line)) {
            if (line.empty()) continue;

            std::stringstream ss(line);
            std::string name, capital, continent, popStr;

            std::getline(ss, name, ',');
            std::getline(ss, capital, ',');
            std::getline(ss, continent, ',');
            std::getline(ss, popStr, ',');

            int population = 0;
            if (!popStr.empty()) {
                try {

```

```

        population = std::stoi(popStr);
    }
    catch (...) {
        population = 0;
    }
}

    addCountry(name, capital, continent, population);
}

    file.close();
    return true;
}
};

// Класс для викторины
class QuizGame {
private:
    CapitalDB& db;
    std::vector<Country> quizCountries;
    size_t currentQuestion;
    int score;
    bool quizActive;

public:
    QuizGame(CapitalDB& database)
        : db(database), currentQuestion(0), score(0), quizActive(false) {
    }

    void startQuiz(int numQuestions = 10) {
        quizCountries.clear();
        const auto& allCountries = db.getCountries();

        if (allCountries.empty()) {
            quizActive = false;
            return;
        }

        quizCountries = allCountries;

        std::random_device rd;
        std::mt19937 g(rd());
        std::shuffle(quizCountries.begin(), quizCountries.end(), g);

        if (numQuestions < 1) numQuestions = 1;

```

```

        if (quizCountries.size() > static_cast<size_t>(numQuestions)) {
            quizCountries.resize(numQuestions);
        }

        currentQuestion = 0;
        score = 0;
        quizActive = true;
    }

    void endQuiz() {
        quizActive = false;
    }

    bool checkAnswer(const std::string& answer) {
        if (!quizActive || currentQuestion >= quizCountries.size()) {
            return false;
        }

        std::string correctAnswer = quizCountries[currentQuestion].capital;
        std::string userAnswer = answer;

        userAnswer.erase(0, userAnswer.find_first_not_of(" \t\n\r"));
        userAnswer.erase(userAnswer.find_last_not_of(" \t\n\r") + 1);

        correctAnswer.erase(0, correctAnswer.find_first_not_of(" \t\n\r"));
        correctAnswer.erase(correctAnswer.find_last_not_of(" \t\n\r") + 1);

        std::transform(userAnswer.begin(), userAnswer.end(), userAnswer.begin(),
            [](unsigned char c) { return std::tolower(c); });
        std::transform(correctAnswer.begin(), correctAnswer.end(),
correctAnswer.begin(),
            [](unsigned char c) { return std::tolower(c); });

        bool correct = (userAnswer == correctAnswer);
        if (correct) {
            score++;
        }

        return correct;
    }

    void nextQuestion() {
        if (quizActive && currentQuestion < quizCountries.size() - 1) {
            currentQuestion++;
        }
    }

```

```

        else {
            quizActive = false;
        }
    }

    bool isActive() const {
        return quizActive;
    }

    std::string getCurrentQuestion() const {
        return quizActive && currentQuestion < quizCountries.size() ?
            quizCountries[currentQuestion].name : "";
    }

    std::string getCurrentCapital() const {
        return quizActive && currentQuestion < quizCountries.size() ?
            quizCountries[currentQuestion].capital : "";
    }

    int getScore() const {
        return score;
    }

    size_t getCurrentQuestionNumber() const {
        return currentQuestion + 1;
    }

    size_t getTotalQuestions() const {
        return quizCountries.size();
    }
};

// Основной класс GUI
class GUI {
private:
    sf::RenderWindow window;
    CapitalDB db;
    QuizGame quiz;

    sf::Font font;

    sf::Text titleText;
    sf::Text infoText;
    sf::Text questionText;
    sf::Text inputText;

```

```

sf::Text resultText;
sf::Text scoreText;
sf::Text hintText;
sf::Text statusText;
sf::Text scrollInfoText; // Для информации о прокрутке

sf::RectangleShape addButton;
sf::RectangleShape quizButton;
sf::RectangleShape filterButton;
sf::RectangleShape shuffleButton;
sf::RectangleShape submitButton;
sf::RectangleShape nextButton;
sf::RectangleShape exitButton;
sf::RectangleShape saveButton;
sf::RectangleShape loadButton;
sf::RectangleShape showAllButton;
sf::RectangleShape upButton;    // Кнопка прокрутки вверх
sf::RectangleShape downButton; // Кнопка прокрутки вниз

sf::RectangleShape inputField;
std::string inputString;
bool inputActive;

enum Mode { MAIN, ADD_COUNTRY, QUIZ, FILTER, SAVE_LOAD, SHOW_ALL };
Mode currentMode;

std::string tempCountry;
std::string tempCapital;
std::string tempContinent;

// Переменные для прокрутки
int scrolloffset;
const int MAX_LINES_PER_PAGE = 30;

void setupUI() {
    // Загрузка шрифта
    if (!font.loadFromFile("arial.ttf")) {
        if (!font.loadFromFile("C:/Windows/Fonts/arial.ttf")) {
            font.loadFromFile("");
        }
    }

    // Настройка текстов
    titleText.setFont(font);
    titleText.setString("WORLD COUNTRIES DATABASE");

```

```

titleText.setCharacterSize(36);
titleText.setFillColor(sf::Color(0, 100, 200));
titleText.setStyle(sf::Text::Bold);
titleText.setPosition(200, 20);

infoText.setFont(font);
infoText.setCharacterSize(16);
infoText.setFillColor(sf::Color::Black);
infoText.setPosition(100, 100);
questionText.setFont(font);
questionText.setCharacterSize(24);
questionText.setFillColor(sf::Color::Black);
questionText.setPosition(100, 150);

inputText.setFont(font);
inputText.setCharacterSize(20);
inputText.setFillColor(sf::Color::Black);
inputText.setPosition(450, 300);

resultText.setFont(font);
resultText.setCharacterSize(24);
resultText.setFillColor(sf::Color::Red);
resultText.setPosition(50, 350);

scoreText.setFont(font);
scoreText.setCharacterSize(20);
scoreText.setFillColor(sf::Color::Green);
scoreText.setPosition(50, 400);

hintText.setFont(font);
hintText.setCharacterSize(16);
hintText.setFillColor(sf::Color(100, 100, 100));
hintText.setPosition(50, 250);

statusText.setFont(font);
statusText.setCharacterSize(16);
statusText.setFillColor(sf::Color::Blue);
statusText.setPosition(50, 650);
statusText.setString("Total countries: " + std::to_string(db.getCount()));

scrollInfoText.setFont(font);
scrollInfoText.setCharacterSize(14);
scrollInfoText.setFillColor(sf::Color(150, 150, 150));
scrollInfoText.setPosition(50, 680);

```

```

// Настройка кнопок
int buttonWidth = 220;
int buttonHeight = 45;
int startY = 120;
int spacing = 55;

addButton.setSize(sf::Vector2f(buttonWidth, buttonHeight));
addButton.setPosition(400, startY);
addButton.setFillColor(sf::Color(70, 130, 180));
addButton.setOutlineThickness(2);
addButton.setOutlineColor(sf::Color::Black);

quizButton.setSize(sf::Vector2f(buttonWidth, buttonHeight));
quizButton.setPosition(400, startY + spacing);
quizButton.setFillColor(sf::Color(60, 179, 113));
quizButton.setOutlineThickness(2);
quizButton.setOutlineColor(sf::Color::Black);

filterButton.setSize(sf::Vector2f(buttonWidth, buttonHeight));
filterButton.setPosition(400, startY + spacing * 2);
filterButton.setFillColor(sf::Color(218, 165, 32));
filterButton.setOutlineThickness(2);
filterButton.setOutlineColor(sf::Color::Black);

shuffleButton.setSize(sf::Vector2f(buttonWidth, buttonHeight));
shuffleButton.setPosition(400, startY + spacing * 3);
shuffleButton.setFillColor(sf::Color(186, 85, 211));
shuffleButton.setOutlineThickness(2);
shuffleButton.setOutlineColor(sf::Color::Black);

showAllButton.setSize(sf::Vector2f(buttonWidth, buttonHeight));
showAllButton.setPosition(400, startY + spacing * 4);
showAllButton.setFillColor(sf::Color(255, 105, 180));
showAllButton.setOutlineThickness(2);
showAllButton.setOutlineColor(sf::Color::Black);

saveButton.setSize(sf::Vector2f(buttonWidth, buttonHeight));
saveButton.setPosition(400, startY + spacing * 5);
saveButton.setFillColor(sf::Color(50, 205, 50));
saveButton.setOutlineThickness(2);
saveButton.setOutlineColor(sf::Color::Black);

loadButton.setSize(sf::Vector2f(buttonWidth, buttonHeight));
loadButton.setPosition(400, startY + spacing * 6);
loadButton.setFillColor(sf::Color(255, 140, 0));

```

```

loadButton.setOutlineThickness(2);
loadButton.setOutlineColor(sf::Color::Black);

submitButton.setSize(sf::Vector2f(150, 40));
submitButton.setPosition(425, 360);
submitButton.setFillColor(sf::Color(60, 179, 113));
submitButton.setOutlineThickness(2);
submitButton.setOutlineColor(sf::Color::Black);

nextButton.setSize(sf::Vector2f(150, 40));
nextButton.setPosition(425, 410);
nextButton.setFillColor(sf::Color(70, 130, 180));
nextButton.setOutlineThickness(2);
nextButton.setOutlineColor(sf::Color::Black);

exitButton.setSize(sf::Vector2f(150, 40));
exitButton.setPosition(425, 460);
exitButton.setFillColor(sf::Color(220, 20, 60));
exitButton.setOutlineThickness(2);
exitButton.setOutlineColor(sf::Color::Black);

// Кнопки прокрутки
upButton.setSize(sf::Vector2f(40, 30));
upButton.setPosition(950, 100);
upButton.setFillColor(sf::Color(100, 100, 200));
upButton.setOutlineThickness(1);
upButton.setOutlineColor(sf::Color::Black);

downButton.setSize(sf::Vector2f(40, 30));
downButton.setPosition(950, 640);
downButton.setFillColor(sf::Color(100, 100, 200));
downButton.setOutlineThickness(1);
downButton.setOutlineColor(sf::Color::Black);

inputField.setSize(sf::Vector2f(300, 40));
inputField.setPosition(350, 290);
inputField.setFillColor(sf::Color::White);
inputField.setOutlineThickness(2);
inputField.setOutlineColor(sf::Color::Black);

// Инициализация прокрутки
scrollOffset = 0;
}

void handleEvents() {

```

```

sf::Event event;
while (window.pollEvent(event)) {
    if (event.type == sf::Event::Closed) {
        window.close();
    }

    // Обработка колесика мыши для прокрутки
    if (event.type == sf::Event::MouseWheelScrolled && currentMode == SHOW_ALL)
{
        if (event.mouseWheelScroll.delta > 0) {
            // Прокрутка вверх
            scrollUp();
        }
        else if (event.mouseWheelScroll.delta < 0) {
            // Прокрутка вниз
            scrollDown();
        }
    }

    if (event.type == sf::Event::KeyPressed) {
        if (event.key.code == sf::Keyboard::Escape) {
            if (currentMode != MAIN) {
                currentMode = MAIN;
                inputString.clear();
                inputActive = false;
                db.clearFilter();
                scrollOffset = 0;
            }
            else {
                window.close();
            }
        }
        // Обработка клавиш для прокрутки
        else if (currentMode == SHOW_ALL) {
            if (event.key.code == sf::Keyboard::Up || event.key.code ==
sf::Keyboard::PageUp) {
                scrollUp();
            }
            else if (event.key.code == sf::Keyboard::Down || event.key.code ==
sf::Keyboard::PageDown) {
                scrollDown();
            }
            else if (event.key.code == sf::Keyboard::Home) {
                scrollOffset = 0;
            }
        }
    }
}

```

```

        else if (event.key.code == sf::Keyboard::End) {
            scrollOffset = std::max(0, static_cast<int>(db.getCount()) -
MAX_LINES_PER_PAGE);
        }
    }

    switch (currentMode) {
        case MAIN: handleMainMode(event); break;
        case ADD_COUNTRY: handleAddMode(event); break;
        case QUIZ: handleQuizMode(event); break;
        case FILTER: handleFilterMode(event); break;
        case SAVE_LOAD: handleSaveLoadMode(event); break;
        case SHOW_ALL: handleShowAllMode(event); break;
    }
}

void handleMainMode(const sf::Event& event) {
    if (event.type == sf::Event::MouseButtonPressed && event.mouseButton.button ==
sf::Mouse::Left) {
        sf::Vector2f mousePos =
window.mapPixelToCoords(sf::Mouse::getPosition(window));

        if (addButton.getGlobalBounds().contains(mousePos)) {
            currentMode = ADD_COUNTRY;
            inputString.clear();
            tempCountry.clear();
            tempCapital.clear();
            tempContinent.clear();
            inputActive = false;
            statusText.setString("Add country mode");
        }
        else if (quizButton.getGlobalBounds().contains(mousePos)) {
            currentMode = QUIZ;
            quiz.startQuiz(10);
            inputString.clear();
            inputActive = false;
            statusText.setString("Quiz mode - 10 questions");
        }
        else if (filterButton.getGlobalBounds().contains(mousePos)) {
            currentMode = FILTER;
            inputString.clear();
            inputActive = false;
            statusText.setString("Filter by continent");
        }
    }
}

```

```

    }
    else if (shuffleButton.getGlobalBounds().contains(mousePos)) {
        db.shuffleCountries();
        statusText.setString("Countries shuffled");
    }
    else if (showAllButton.getGlobalBounds().contains(mousePos)) {
        currentMode = SHOW_ALL;
        scrolloffset = 0;
        statusText.setString("Showing all " + std::to_string(db.getCount()) + "
countries");
    }
    else if (saveButton.getGlobalBounds().contains(mousePos)) {
        currentMode = SAVE_LOAD;
        inputString.clear();
        inputActive = false;
        statusText.setString("Save/Load mode");
    }
    else if (loadButton.getGlobalBounds().contains(mousePos)) {
        currentMode = SAVE_LOAD;
        inputString.clear();
        inputActive = false;
        statusText.setString("Save/Load mode");
    }
}

void handleAddMode(const sf::Event& event) {
    if (event.type == sf::Event::MouseButtonPressed && event.mouseButton.button ==
sf::Mouse::Left) {
        sf::Vector2f mousePos =
window.mapPixelToCoords(sf::Mouse::getPosition(window));

        if (inputField.getGlobalBounds().contains(mousePos)) {
            inputActive = true;
        }
        else {
            inputActive = false;
        }

        if (submitButton.getGlobalBounds().contains(mousePos)) {
            if (!tempCountry.empty() && !tempCapital.empty()) {
                db.addCountry(tempCountry, tempCapital, tempContinent);
                tempCountry.clear();
                tempCapital.clear();
                tempContinent.clear();
            }
        }
    }
}

```

```

        inputString.clear();
        statusText.setString("Country added. Total: " +
std::to_string(db.getCount()));
    }
}
else if (exitButton.getGlobalBounds().contains(mousePos)) {
    currentMode = MAIN;
    statusText.setString("Total countries: " +
std::to_string(db.getCount()));
}
}

if (event.type == sf::Event::TextEntered && inputActive) {
    if (event.text.unicode == 8) {
        if (!inputString.empty()) {
            inputString.pop_back();
        }
    }
    else if (event.text.unicode == 13) {
        static int step = 0;
        if (step == 0 && !inputString.empty()) {
            tempCountry = inputString;
            inputString.clear();
            step = 1;
        }
        else if (step == 1 && !inputString.empty()) {
            tempCapital = inputString;
            inputString.clear();
            step = 2;
        }
        else if (step == 2) {
            tempContinent = inputString;
            inputString.clear();
            step = 0;
        }
    }
    else if (event.text.unicode >= 32 && event.text.unicode < 128) {
        inputString += static_cast<char>(event.text.unicode);
    }
}

}

void handleQuizMode(const sf::Event& event) {
    if (event.type == sf::Event::MouseButtonPressed && event.mouseButton.button ==
sf::Mouse::Left) {

```

```

        sf::Vector2f mousePos =
window.mapPixelToCoords(sf::Mouse::getPosition(window));

        if (inputField.getGlobalBounds().contains(mousePos)) {
            inputActive = true;
        }
        else {
            inputActive = false;
        }

        if (submitButton.getGlobalBounds().contains(mousePos) && quiz.isActive()) {
            bool correct = quiz.checkAnswer(inputString);
            if (correct) {
                resultText.setString("Correct!");
                resultText.setFillColor(sf::Color::Green);
            }
            else {
                resultText.setString("Wrong! Correct answer: " +
quiz.getCurrentCapital());
                resultText.setFillColor(sf::Color::Red);
            }
            inputString.clear();
        }
        else if (nextButton.getGlobalBounds().contains(mousePos) && quiz.isActive())
{
            quiz.nextQuestion();
            resultText.setString("");
            inputString.clear();
        }
        else if (exitButton.getGlobalBounds().contains(mousePos)) {
            currentMode = MAIN;
            quiz.endQuiz();
            statusText.setString("Total countries: " +
std::to_string(db.getCount()));
        }
    }

    if (event.type == sf::Event::TextEntered && inputActive) {
        if (event.text.unicode == 8) {
            if (!inputString.empty()) {
                inputString.pop_back();
            }
        }
        else if (event.text.unicode == 13) {
            if (quiz.isActive() && !inputString.empty()) {

```

```

        bool correct = quiz.checkAnswer(inputString);
        if (correct) {
            resultText.setString("Correct!");
            resultText.setFillColor(sf::Color::Green);
        }
        else {
            resultText.setString("Wrong! Correct answer: " +
quiz.getCurrentCapital());
            resultText.setFillColor(sf::Color::Red);
        }
        inputString.clear();
    }
}
else if (event.text.unicode >= 32 && event.text.unicode < 128) {
    inputString += static_cast<char>(event.text.unicode);
}
}
}

void handleFilterMode(const sf::Event& event) {
    if (event.type == sf::Event::MouseButtonPressed && event.mouseButton.button ==
sf::Mouse::Left) {
        sf::Vector2f mousePos =
window.mapPixelToCoords(sf::Mouse::getPosition(window));

        if (inputField.getGlobalBounds().contains(mousePos)) {
            inputActive = true;
        }
        else {
            inputActive = false;
        }

        if (submitButton.getGlobalBounds().contains(mousePos)) {
            if (!inputString.empty()) {
                db.filterByContinent(inputString);
                scrollOffset = 0;
                statusText.setString("Filter: " + inputString + " (" +
std::to_string(db.getCount()) + " countries)");
            }
        }
        else if (exitButton.getGlobalBounds().contains(mousePos)) {
            currentMode = MAIN;
            db.clearFilter();
            scrollOffset = 0;

```

```

        statusText.setString("Total countries: " +
std::to_string(db.getCount()));
    }
}

if (event.type == sf::Event::TextEntered && inputActive) {
    if (event.text.unicode == 8) {
        if (!inputString.empty()) {
            inputString.pop_back();
        }
    }
    else if (event.text.unicode == 13) {
        if (!inputString.empty()) {
            db.filterByContinent(inputString);
            scrollOffset = 0;
            statusText.setString("Filter: " + inputString + " (" +
std::to_string(db.getCount()) + " countries)");
        }
    }
    else if (event.text.unicode >= 32 && event.text.unicode < 128) {
        inputString += static_cast<char>(event.text.unicode);
    }
}

}

void handleSaveLoadMode(const sf::Event& event) {
    if (event.type == sf::Event::MouseButtonPressed && event.mouseButton.button ==
sf::Mouse::Left) {
        sf::Vector2f mousePos =
window.mapPixelToCoords(sf::Mouse::getPosition(window));

        if (inputField.getGlobalBounds().contains(mousePos)) {
            inputActive = true;
        }
        else {
            inputActive = false;
        }

        if (submitButton.getGlobalBounds().contains(mousePos)) {
            if (!inputString.empty()) {
                if (db.saveToFile(inputString + ".csv")) {
                    statusText.setString("Saved to: " + inputString + ".csv");
                }
            }
            else {
                statusText.setString("Save error");
            }
        }
    }
}

```

```

        }
        inputString.clear();
    }
}
else if (nextButton.getGlobalBounds().contains(mousePos)) {
    if (!inputString.empty()) {
        if (db.loadFromFile(inputString + ".csv")) {
            statusText.setString("Loaded from: " + inputString + ".csv");
        }
        else {
            statusText.setString("Load error");
        }
        inputString.clear();
    }
}
else if (exitButton.getGlobalBounds().contains(mousePos)) {
    currentMode = MAIN;
    statusText.setString("Total countries: " +
std::to_string(db.getCount()));
}
}

if (event.type == sf::Event::TextEntered && inputActive) {
    if (event.text.unicode == 8) {
        if (!inputString.empty()) {
            inputString.pop_back();
        }
    }
    else if (event.text.unicode == 13) {
        if (!inputString.empty()) {
            if (db.saveToFile(inputString + ".csv")) {
                statusText.setString("Saved to: " + inputString + ".csv");
            }
            else {
                statusText.setString("Save error");
            }
            inputString.clear();
        }
    }
    else if (event.text.unicode >= 32 && event.text.unicode < 128) {
        inputString += static_cast<char>(event.text.unicode);
    }
}
}
}

```

```

void handleShowAllMode(const sf::Event& event) {
    if (event.type == sf::Event::MouseButtonPressed && event.mouseButton.button ==
sf::Mouse::Left) {
        sf::Vector2f mousePos =
window.mapPixelToCoords(sf::Mouse::getPosition(window));

        // Проверка кликов на кнопках прокрутки
        if (upButton.getGlobalBounds().contains(mousePos)) {
            scrollUp();
        }
        else if (downButton.getGlobalBounds().contains(mousePos)) {
            scrollDown();
        }
        else if (exitButton.getGlobalBounds().contains(mousePos)) {
            currentMode = MAIN;
            scrollOffset = 0;
            statusText.setString("Total countries: " +
std::to_string(db.getCount()));
        }
    }
}

// Функции для прокрутки
void scrollUp() {
    if (scrollOffset > 0) {
        scrollOffset--;
    }
}

void scrollDown() {
    int totalCountries = static_cast<int>(db.getCount());
    if (scrollOffset < totalCountries - MAX_LINES_PER_PAGE) {
        scrollOffset++;
    }
}

void update() {
    inputText.setString(inputString);

    // Обновляем информацию о странах в зависимости от режима
    std::string info;
    const auto& countries = db.getCountries();

    if (currentMode == MAIN) {
        info = "TOTAL COUNTRIES: " + std::to_string(db.getCount()) + "\n\n";
    }
}

```

```

        info += "Select an option:\n";
        info += "• Add Country - add new country\n";
        info += "• Quiz - test your knowledge\n";
        info += "• Filter - filter by continent\n";
        info += "• Shuffle - randomize order\n";
        info += "• Show All - view all countries\n";
        info += "• Save/Load - file operations\n";
    }
    else if (currentMode == SHOW_ALL) {
        info = "ALL COUNTRIES (" + std::to_string(db.getCount()) + "):\n\n";

        int startIndex = scrollOffset;
        int endIndex = std::min(startIndex + MAX_LINES_PER_PAGE,
static_cast<int>(countries.size()));

        for (int i = startIndex; i < endIndex; i++) {
            info += std::to_string(i + 1) + ". " + countries[i].name;
            info += " - " + countries[i].capital;
            if (!countries[i].continent.empty()) {
                info += " (" + countries[i].continent + ")";
            }
            info += "\n";
        }

        // Информация о прокрутке
        std::string scrollInfo = "Page: " + std::to_string(scrollOffset /
MAX_LINES_PER_PAGE + 1) +
            " | Showing " + std::to_string(startIndex + 1) + "-" +
            std::to_string(endIndex) + " of " + std::to_string(countries.size());
        scrollInfoText.setString(scrollInfo);
    }
    else {
        info = "COUNTRIES: " + std::to_string(db.getCount()) + "\n\n";
        for (size_t i = 0; i < std::min(countries.size(), size_t(20)); i++) {
            info += countries[i].name + " - " + countries[i].capital + "\n";
        }
        if (countries.size() > 20) {
            info += "... and " + std::to_string(countries.size() - 20) + " more";
        }
        scrollInfoText.setString("");
    }

    infoText.setString(info);

    // Обновляем текст викторины

```

```

    if (currentMode == QUIZ) {
        if (quiz.isActive()) {
            std::string question = "QUESTION " +
                std::to_string(quiz.getCurrentQuestionNumber()) + " of " +
                std::to_string(quiz.getTotalQuestions()) + ":\n" +
                "CAPITAL OF: " + quiz.getCurrentQuestion();

            questionText.setString(question);

            std::string scoreStr = "SCORE: " + std::to_string(quiz.getScore()) +
                " / " + std::to_string(quiz.getTotalQuestions());
            scoreText.setString(scoreStr);
        }
        else {
            questionText.setString("QUIZ COMPLETED!");
            std::string finalScore = "FINAL SCORE: " +
std::to_string(quiz.getScore()) +
                " of " + std::to_string(quiz.getTotalQuestions());
            scoreText.setString(finalScore);
        }
    }
}

void render() {
    window.clear(sf::Color(245, 245, 245));

    window.draw(titleText);
    window.draw(statusText);

    if (currentMode == SHOW_ALL) {
        window.draw(scrollInfoText);
    }

    switch (currentMode) {
        case MAIN: drawMainScreen(); break;
        case ADD_COUNTRY: drawAddScreen(); break;
        case QUIZ: drawQuizScreen(); break;
        case FILTER: drawFilterScreen(); break;
        case SAVE_LOAD: drawSaveLoadScreen(); break;
        case SHOW_ALL: drawShowAllScreen(); break;
    }

    window.display();
}

```

```

void drawMainScreen() {
    window.draw(infoText);

    window.draw(addButton);
    window.draw(quizButton);
    window.draw(filterButton);
    window.draw(shuffleButton);
    window.draw(showAllButton);
    window.draw(saveButton);
    window.draw(loadButton);

    // Тексты на кнопках
    sf::Text addBtnText("ADD COUNTRY", font, 18);
    addBtnText.setPosition(430, 135);
    addBtnText.setFillColor(sf::Color::White);
    window.draw(addBtnText);

    sf::Text quizBtnText("QUIZ", font, 18);
    quizBtnText.setPosition(475, 190);
    quizBtnText.setFillColor(sf::Color::White);
    window.draw(quizBtnText);

    sf::Text filterBtnText("FILTER", font, 18);
    filterBtnText.setPosition(470, 245);
    filterBtnText.setFillColor(sf::Color::White);
    window.draw(filterBtnText);

    sf::Text shuffleBtnText("SHUFFLE", font, 18);
    shuffleBtnText.setPosition(450, 300);
    shuffleBtnText.setFillColor(sf::Color::White);
    window.draw(shuffleBtnText);

    sf::Text showAllBtnText("SHOW ALL", font, 18);
    showAllBtnText.setPosition(450, 355);
    showAllBtnText.setFillColor(sf::Color::White);
    window.draw(showAllBtnText);

    sf::Text saveBtnText("SAVE", font, 18);
    saveBtnText.setPosition(475, 410);
    saveBtnText.setFillColor(sf::Color::White);
    window.draw(saveBtnText);

    sf::Text loadBtnText("LOAD", font, 18);
    loadBtnText.setPosition(475, 465);
    loadBtnText.setFillColor(sf::Color::White);

```

```

        window.draw(loadBtnText);
    }

    void drawAddScreen() {
        hintText.setString("Enter data (press Enter after each field):\n1. Country
name\n2. Capital\n3. Continent (optional)");
        window.draw(hintText);

        window.draw(inputField);
        window.draw(inputText);
        window.draw(submitButton);
        window.draw(exitButton);

        sf::Text currentText("CURRENT DATA:", font, 18);
        currentText.setPosition(50, 150);
        currentText.setFillColor(sf::Color::Black);
        window.draw(currentText);

        sf::Text countryText("Country: " + tempCountry, font, 16);
        countryText.setPosition(70, 180);
        countryText.setFillColor(sf::Color(0, 100, 0));
        window.draw(countryText);

        sf::Text capitalText("Capital: " + tempCapital, font, 16);
        capitalText.setPosition(70, 210);
        capitalText.setFillColor(sf::Color(0, 100, 0));
        window.draw(capitalText);

        sf::Text continentText("Continent: " + tempContinent, font, 16);
        continentText.setPosition(70, 240);
        continentText.setFillColor(sf::Color(0, 100, 0));
        window.draw(continentText);

        sf::Text submitBtnText("ADD", font, 18);
        submitBtnText.setPosition(470, 370);
        submitBtnText.setFillColor(sf::Color::White);
        window.draw(submitBtnText);

        sf::Text exitBtnText("BACK", font, 18);
        exitBtnText.setPosition(470, 470);
        exitBtnText.setFillColor(sf::Color::White);
        window.draw(exitBtnText);
    }

    void drawQuizScreen() {

```

```

window.draw(questionText);

if (quiz.isActive()) {
    hintText.setString("Enter capital and press Enter or 'Check' button");
    window.draw(hintText);

    window.draw(inputField);
    window.draw(inputText);
    window.draw(submitButton);
    window.draw(nextButton);
}

window.draw(resultText);
window.draw(scoreText);
window.draw(exitButton);

if (quiz.isActive()) {
    sf::Text submitBtnText("CHECK", font, 18);
    submitBtnText.setPosition(450, 370);
    submitBtnText.setFillColor(sf::Color::White);
    window.draw(submitBtnText);

    sf::Text nextBtnText("NEXT", font, 18);
    nextBtnText.setPosition(460, 420);
    nextBtnText.setFillColor(sf::Color::White);
    window.draw(nextBtnText);
}

sf::Text exitBtnText("BACK", font, 18);
exitBtnText.setPosition(470, 470);
exitBtnText.setFillColor(sf::Color::White);
window.draw(exitBtnText);
}

void drawFilterScreen() {
    hintText.setString("Enter continent name:\n(Europe, Asia, Africa, America,
Oceania)");
    window.draw(hintText);

    window.draw(inputField);
    window.draw(inputText);
    window.draw(submitButton);
    window.draw(exitButton);

    sf::Text submitBtnText("APPLY", font, 18);

```

```

        submitBtnText.setPosition(450, 370);
        submitBtnText.setFillColor(sf::Color::White);
        window.draw(submitBtnText);

        sf::Text exitBtnText("BACK", font, 18);
        exitBtnText.setPosition(470, 470);
        exitBtnText.setFillColor(sf::Color::White);
        window.draw(exitBtnText);
    }

    void drawSaveLoadScreen() {
        hintText.setString("Enter filename (without extension):\nPress 'Save' or
'Load'");
        window.draw(hintText);

        window.draw(inputField);
        window.draw(inputText);
        window.draw(submitButton);
        window.draw(nextButton);
        window.draw(exitButton);

        sf::Text submitBtnText("SAVE", font, 18);
        submitBtnText.setPosition(460, 370);
        submitBtnText.setFillColor(sf::Color::White);
        window.draw(submitBtnText);

        sf::Text nextBtnText("LOAD", font, 18);
        nextBtnText.setPosition(460, 420);
        nextBtnText.setFillColor(sf::Color::White);
        window.draw(nextBtnText);

        sf::Text exitBtnText("BACK", font, 18);
        exitBtnText.setPosition(470, 470);
        exitBtnText.setFillColor(sf::Color::White);
        window.draw(exitBtnText);
    }

    void drawShowAllScreen() {
        window.draw(infoText);

        // Рисуем кнопки прокрутки если нужно
        if (db.getCount() > MAX_LINES_PER_PAGE) {
            window.draw(upButton);
            window.draw(downButton);
        }
    }

```

```

        // Стрелки на кнопках
        sf::Text upArrow("▲", font, 20);
        upArrow.setPosition(963, 103);
        upArrow.setFillColor(sf::Color::White);
        window.draw(upArrow);

        sf::Text downArrow("▼", font, 20);
        downArrow.setPosition(963, 645);
        downArrow.setFillColor(sf::Color::White);
        window.draw(downArrow);
    }

    window.draw(exitButton);

    sf::Text exitBtnText("BACK", font, 18);
    exitBtnText.setPosition(470, 470);
    exitBtnText.setFillColor(sf::Color::White);
    window.draw(exitBtnText);

    // Инструкция по прокрутке
    sf::Text scrollHint("Use mouse wheel, Up/Down arrows, or click buttons", font,
14);
    scrollHint.setPosition(50, 620);
    scrollHint.setFillColor(sf::Color(100, 100, 100));
    window.draw(scrollHint);
}

public:
    GUI()
        : window(sf::VideoMode(1100, 750), "World Countries Database",
sf::Style::Close),
        quiz(db),
        currentMode(MAIN),
        inputActive(false),
        scrolloffset(0) {

        setupUI();
    }

    void run() {
        while (window.isOpen()) {
            handleEvents();
            update();
            render();
        }
    }

```

```
    }  
};  
  
// Главная функция  
int main() {  
    GUI gui;  
    gui.run();  
    return 0;  
}
```

Приложение Б

Результаты работы программы

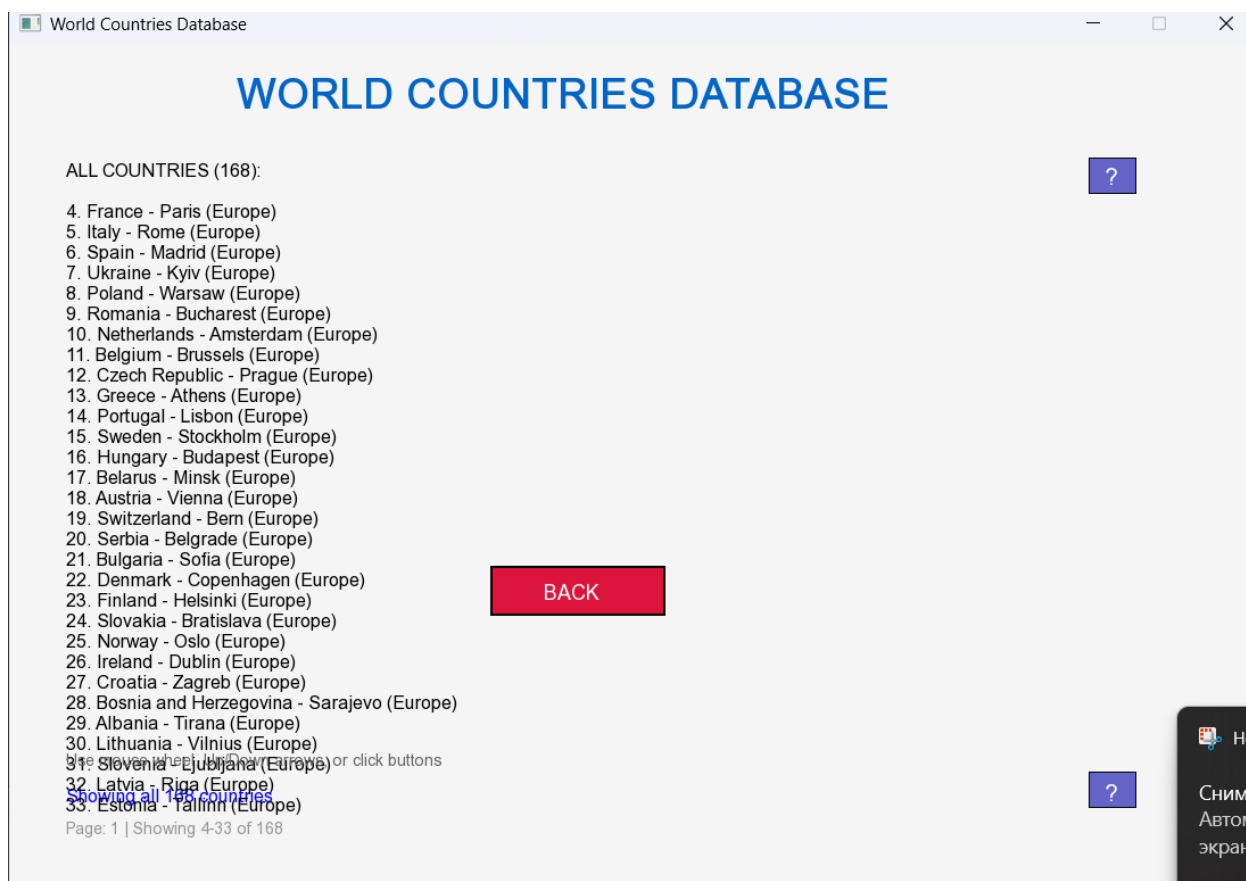


Рисунок Б.1 – Тест №1

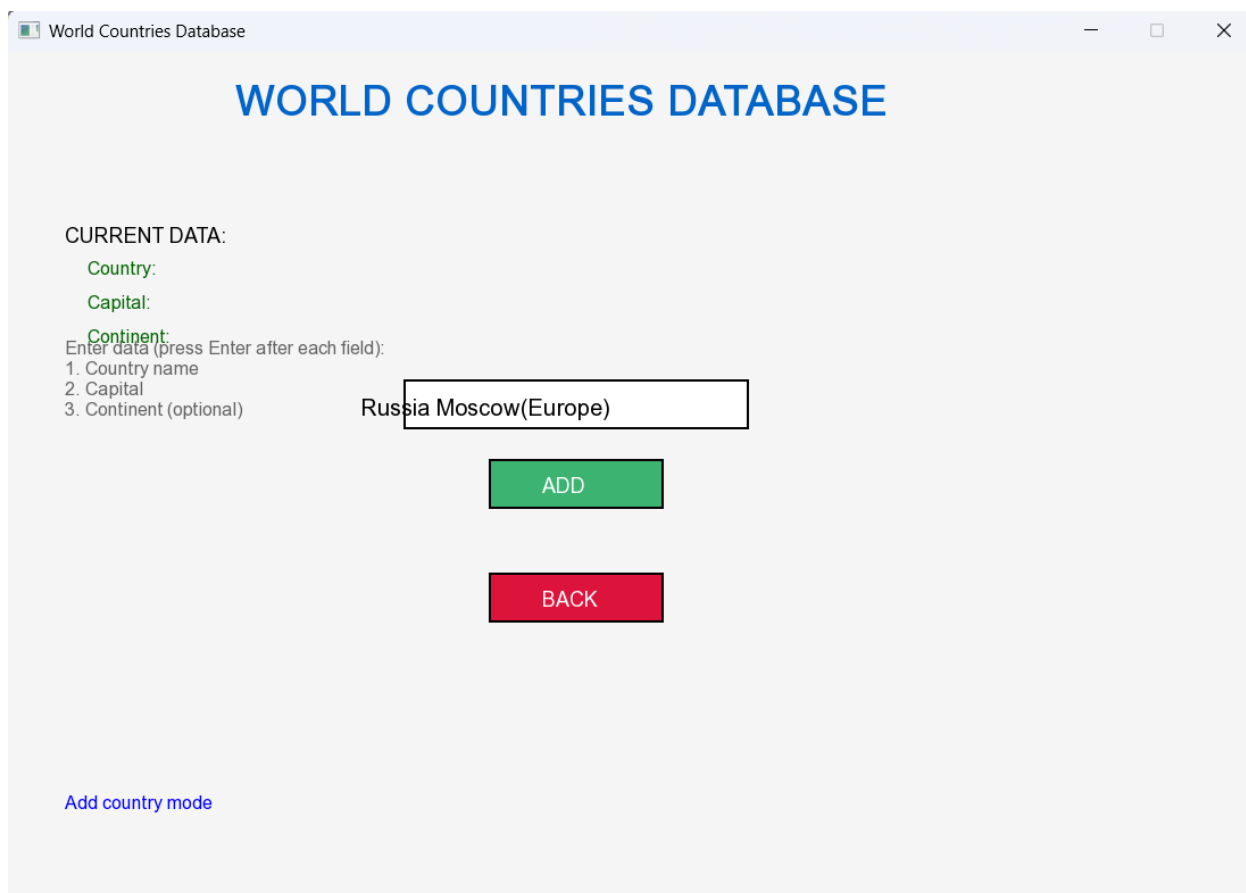


Рисунок Б.2 – Тест №2

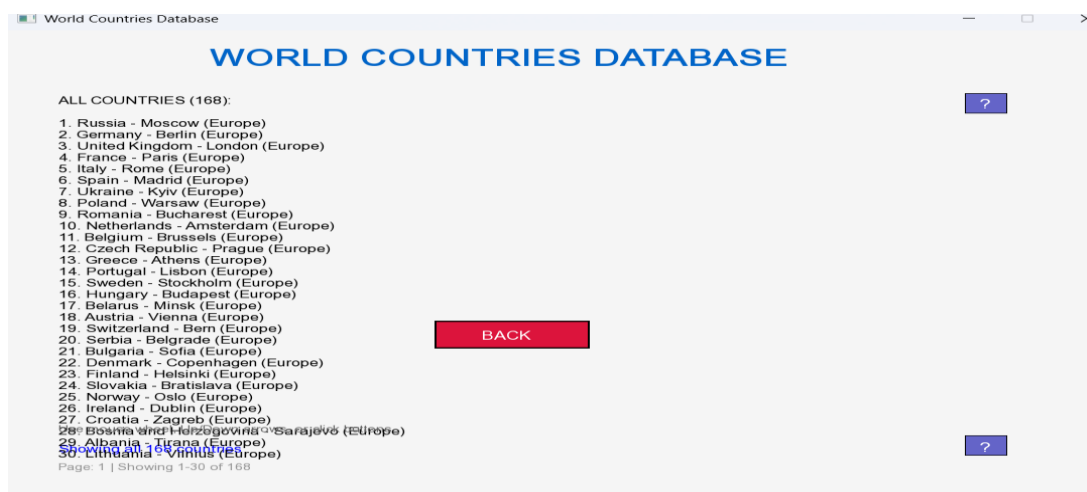


Рисунок Б.3 – Тест №2

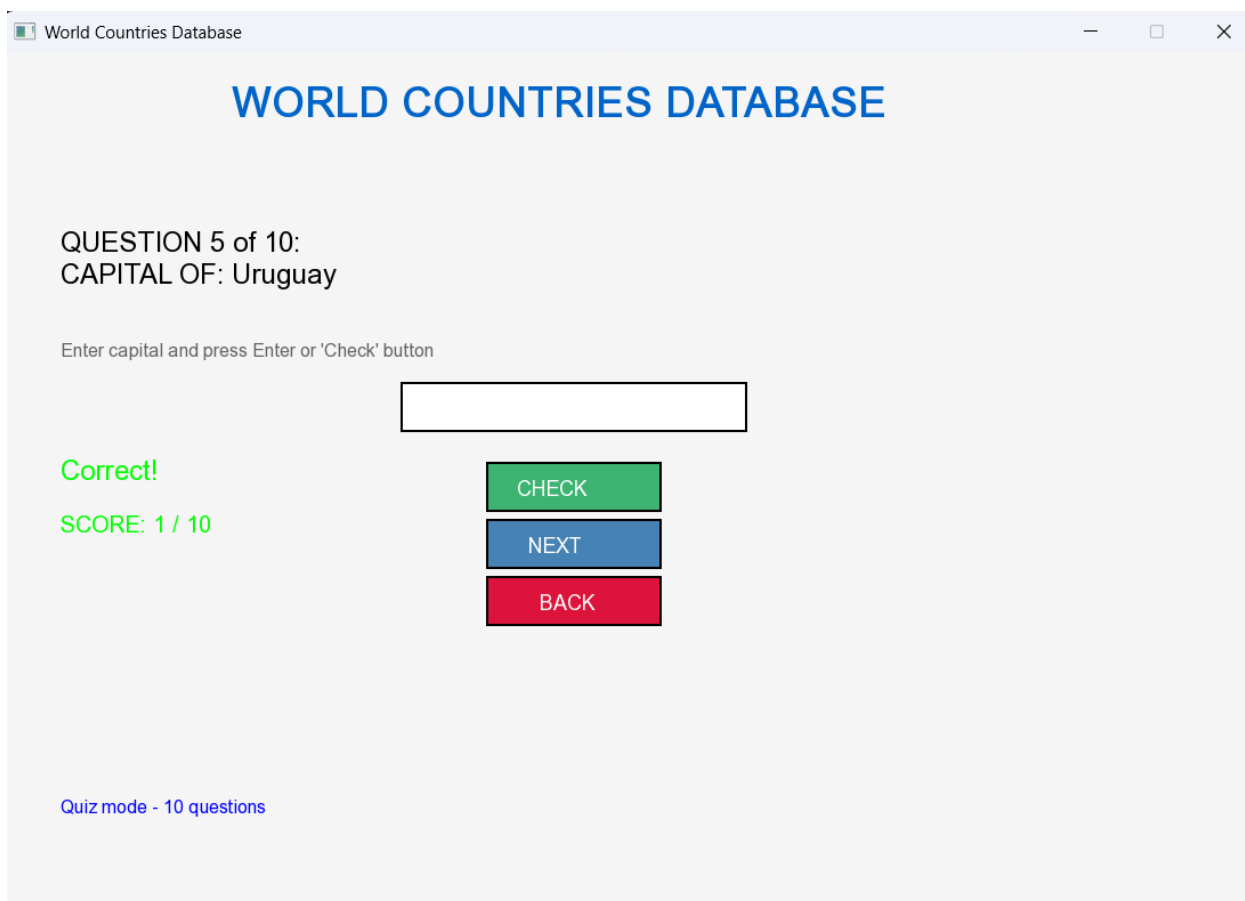


Рисунок Б.4 – Тест №3

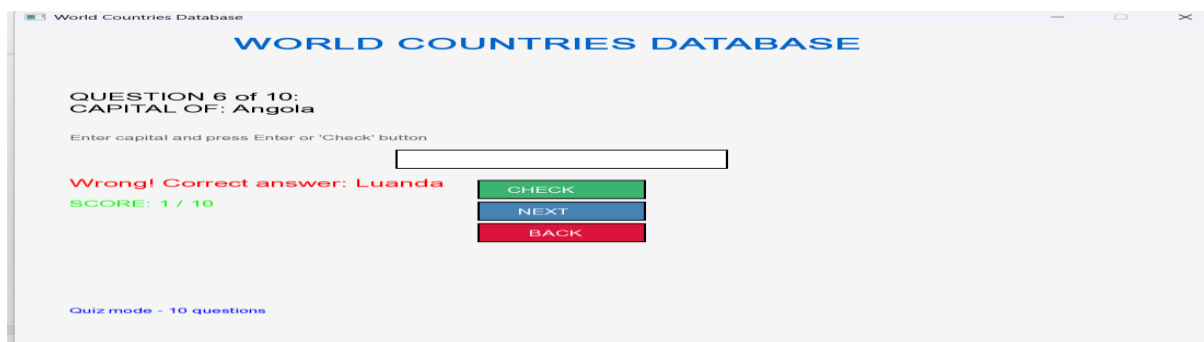


Рисунок Б.5 – Тест №3