СОДЕРЖАНИЕ

[СОДЕРЖАНИЕ 1](#_Toc199964399)

[ВВЕДЕНИЕ 3](#_Toc199964400)

[6.ТЕОРЕТИЧЕСКИЙ РАЗДЕЛ 5](#_Toc199964401)

[6.1 Краткое описание языка QML 5](#_Toc199964402)

[6.1 Анализ существующих аналогов 7](#_Toc199964403)

[6.2 Постановка задачи. 7](#_Toc199964404)

[7. ТЕХНИЧЕСКОЕ ЗАДАНИЕ 8](#_Toc199964405)

[7.1 Введение 8](#_Toc199964406)

[7.1.1 Наименование программы 8](#_Toc199964407)

[7.1.2 Краткая характеристика области применения программы 8](#_Toc199964408)

[7.2 Основание для разработки 8](#_Toc199964409)

[7.3 Назначение разработки 9](#_Toc199964410)

[7.4 Требования, предъявляемые к программе 9](#_Toc199964411)

[7.4.1 Требования к функциональным характеристикам программы 9](#_Toc199964412)

[7.4.2 Требования к техническим средствам, используемым при работе программы 9](#_Toc199964413)

[7.4.3 Требования к языкам программы и среде разработки программы 9](#_Toc199964414)

[7.5 Требования к программной документации 10](#_Toc199964415)

[7.6 Этапы разработки 10](#_Toc199964416)

[7.7 Обзор способов организации данных и обоснование структуры данных для эффективного выполнения операций. 11](#_Toc199964417)

[7.7.1 Двумерный массив 11](#_Toc199964418)

[7.7.2 Minesweeper 11](#_Toc199964419)

[7.7.3 Kurs\_m 12](#_Toc199964420)

[7.8 Выбор структур данных. 12](#_Toc199964421)

[8. РЕАЛИЗАЦИЯ 13](#_Toc199964422)

[8.1 Общие сведения 13](#_Toc199964423)

[8.1.1 Наименование программы 13](#_Toc199964424)

[8.1.2 Программное обеспечение, необходимое для функционирования программы 13](#_Toc199964425)

[8.1.2 Язык программирования, на котором написана программа 13](#_Toc199964426)

[8.2 Функциональное назначение программы (классы решаемых задач и функциональные ограничения на применения) 14](#_Toc199964427)

[8.3 Описание логической структуры программы 14](#_Toc199964428)

[8.3.1 Алгоритмы, используемые в программе 14](#_Toc199964429)

[8.3.2 Структура программы с описанием функций составных частей и связей между ними 15](#_Toc199964430)

[8.4 Вызов программы 15](#_Toc199964431)

[8.6 Входные данные 17](#_Toc199964432)

[8.7 Выходные данные 17](#_Toc199964433)

[ЗАКЛЮЧЕНИЕ 18](#_Toc199964434)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 19](#_Toc199964435)

[Приложение А 20](#_Toc199964436)

ВВЕДЕНИЕ

Курсовая работа по дисциплине «Разработка приложений на языке C++» посвящена созданию мобильной игры «Сапер» для операционной системы Aurora OS с использованием фреймворка Qt.

Фреймворк (англ. framework) ˗ это набор библиотек, инструментов, принципов и подходов, которые предоставляют удобный функционал для разработки более сложного программного обеспечения. Он задает структуру проекта.

Qt ‒ кроссплатформенный фреймворк, а для ОС Аврора и ряда десктопных дистрибутивов Linux на нем создаются нативные приложения.

Qt предоставляет большой набор возможностей для работы с GUI (графический интерфейс пользователя), включая графические элементы управления, взаимодействие с мышью и клавиатурой, работу с окнами и диалоговыми окнами, прорисовку графических элементов и многое другое.

Включает в себя все основные классы, которые могут потребоваться при разработке прикладного программного обеспечения, начиная от элементов графического интерфейса и заканчивая классами для работы с сетью, базами данных и XML.

Qt Quick ‒ модуль для поддержки QML.

Модуль Qt Quick ˗ это стандартная библиотека для написания приложений QML. В то время как модуль Qt QML предоставляет механизм QML и языковую инфраструктуру, модуль Qt Quick предоставляет все основные типы, необходимые для создания пользовательских интерфейсов с помощью QML. Он предоставляет визуальный холст и включает типы для создания и анимации визуальных компонентов, получения пользовательского ввода, создания моделей и представлений данных, а также отложенного создания экземпляров объектов

Игра «Сапер» является одной из самых узнаваемых компьютерных игр, получившая свою известность благодаря простоте, увлекательности и глубине игрового процесса. Основная задача игрока – открыть все ячейки игрового поля, не попав на мину, используя подсказки в виде цифр, указывающих количество мин в соседних клетках.

Несмотря на кажущуюся простоту, разработка игры «Сапер» затрагивает ряд важных аспектов программирования, включая алгоритмы генерации случайных игровых полей, вычисления подсказок, обработку пользовательского ввода и реализацию интуитивно понятного графического интерфейса.

Цель курсовой работы: получение практических навыков программирования в области разработки игровых приложений под систему Аврора, путем создания реализованной и отлаженной игры «Сапер».

Задачи, необходимые для достижения поставленной цели:

1. Рассмотреть методы и алгоритмы программирования, подходящие для разработки программы.
2. Реализовать приложение.
3. Протестировать и отладить программу.

# 6.ТЕОРЕТИЧЕСКИЙ РАЗДЕЛ

## 6.1 Краткое описание языка QML

Существует ряд элементов QML, используемых для позиционирования объектов. Они называются ПОЗИЦИОНЕРАМИ, из которых модуль Qt Quick предоставляет следующее: Row, Column, Grid и Flow.

Column – это тип, который размещает свои дочерние элементы вдоль одного столбца. Его можно использовать как удобный способ вертикального позиционирования ряда элементов без использования якорей.

Свойство spacing можно использовать для изменения расстояния между дочерними элементами. Если элемент в столбце не виден или если его ширина или высота равны 0, элемент не будет размещен и не будет виден внутри столбца.

Кроме того, поскольку column автоматически позиционирует свои дочерние элементы по вертикали, дочерний элемент внутри столбца не должен устанавливать свое положение y или привязывать себя по вертикали, используя привязки top, bottom, anchors.verticalCenter, fill или centerIn.

Row - это тип, который позиционирует дочерние элементы вдоль одного ряда. Он может использоваться как удобный способ горизонтального позиционирования ряда элементов без использования якорей.

Flow - размещает свои дочерние элементы как слова на странице, обертывая их для создания строк или столбцов элементов.

Grid (сетка) - это тип, который позиционирует свои дочерние элементы при формировании сетки.

Grid создает сетку ячеек, достаточно большую, чтобы вместить все ее дочерние элементы, и размещает эти элементы в ячейках слева направо и сверху вниз. Каждый элемент расположен в верхнем левом углу своей ячейки с позицией (0, 0).

Grid по умолчанию состоит из четырех столбцов и создает столько строк, сколько необходимо для размещения всех ее дочерних элементов. Количество строк и столбцов можно ограничить, установив свойства строк и столбцов.

Rectangle расширяет Item и добавляет к нему цвет заливки. Кроме того, он поддерживает границы, определяемые свойствами border.color и border.width. Для создания прямоугольников со скругленными углами вы можете использовать свойство radius.

Каждый элемент Rectangle закрашивается либо сплошным цветом заливки, заданным с помощью свойства color, либо градиентом, заданным с помощью типа Gradient и установленным с помощью свойства gradient. Если указаны и цвет, и градиент, то используется градиент.

Для отображения текста можно использовать элемент Text. Наиболее заметным его свойством является свойство text типа string. Элемент вычисляет свою начальную ширину и высоту на основе заданного текста и используемого шрифта.

Текст можно выровнять по каждой стороне и по центру с помощью свойств horizontalAlignment и verticalAlignment.

Для дальнейшего улучшения рендеринга текста вы можете использовать свойства style и styleColor, которые позволяют вам отображать текст в контурном, приподнятом и вдавленном режиме.

Элемент Image может отображать изображения в различных форматах (например, PNG, JPG, GIF, BMP, WEBP).

Компоненты – это повторно используемые инкапсулированные типы QML с четко определенными интерфейсами.

Компоненты часто определяются файлами компонентов, то есть файлами. qml. Тип компонента, по сути, позволяет определять компоненты QML встроенно, внутри документа QML, а не как отдельный файл QML. Это может быть полезно для повторного использования небольшого компонента в файле QML или для определения компонента, который логически связан с другими компонентами QML в файле.

Qt Quick позволяет прикрепить один элемент управления к другому элементу. Для этого применяется свойство якорь (anchors), которое позволяет установить расположение относительно 7 условных линий и для установки относительно каждой линии предоставляет соответствующее свойство:

-left: устанавливает расположение относительно левого края элемента; -right: устанавливает расположение относительно правого края элемента;

-top: устанавливает расположение относительно верхнего края элемента;

-bottom: устанавливает расположение относительно нижнего края элемента;

-baseline: устанавливает расположение относительно базовой линии элемента (по линии текста; если текста нет, то аналогично top);

-horizontalCenter: устанавливает расположение по горизонтали относительно центра элемента;

-verticalCenter: устанавливает расположение по вертикали относительно центра элемента;

## 6.1 Анализ существующих аналогов

Благодаря простоте игры, в мире существует очень много реализаций игры сапер, которые реализованы по разному на разных платформах, начиная от «классического» сапера для ОС windows, заканчивая веб-реализацией от google.

Несмотря на различия в графике, сама суть игры остается неизменной, меняются только платформы для игры. Поэтому главное отличие реализации, представленной в данной курсовой работе является факт того, что игра написана под систему Аврора нативно.

## 6.2 Постановка задачи.

В данной курсовой работе требуется разработать игру «Сапер» под ОС Аврора. Реализовать требуемый функционал, протестировать программу на наличие ошибок.

# 7. ТЕХНИЧЕСКОЕ ЗАДАНИЕ

## Введение

Составленное техническое задание по дисциплине «Разработка приложений на C++» является документом к курсовой работе, который отражает все этапы разработки программного продукта, а также процесс проектирования и выявление требований, предъявляемых конечному продукту.

### 7.1.1 Наименование программы

Название данного приложения «Сапер» будет напрямую связываться с темой курсовой работы «Игра “Сапер”». Сапер – это простая компьютерная игра, в которой игрок должен открыть все поля игровой области, избегая мин. Для этого у каждого игрового поля генерируется число – кол-во мин по соседству. На основе этих данных игрок должен определить какие клетки безопасны, а какие – нет. Каждую новую игру поле с минами генерируется случайным образом.

### 7.1.2 Краткая характеристика области применения программы

Программа является головоломкой для проведения досуга. Приложение будет полезно тем, кто хочет потренировать свой мозг с помощью мобильного устройства.

## 7.2 Основание для разработки

Основанием для разработки является курсовая работа по дисциплине Разработка приложений на C++», предусмотренная учебным планом направления подготовки 09.03.01 «Информатика и вычислительная техника» профиля «Цифровые комплексы, системы и сети».

## 7.3 Назначение разработки

Приложение предназначено как полезная игра для времяпрепровождения за мобильным устройством.

## 7.4 Требования, предъявляемые к программе

### 7.4.1 Требования к функциональным характеристикам программы

В приложении должны быть реализованы следующие функции:

* Начало новой игры со случайно сгенерированным полем
* Вывод чисел на открытых полях, соответствующих кол-ву соседних мин
* Открытие всех соседних пустых полей при нажатии на поле без мины.
* Вывод сообщений об выигрыше и проигрыше игрока
* Начало новой игры при проигрыше или выигрыше.

### 7.4.2 Требования к техническим средствам, используемым при работе программы

Персональный компьютер пользователя должен быть оснащён графическим адаптером, также должна быть установлена ОС Auora.

### 7.4.3 Требования к языкам программы и среде разработки программы

Для разработки используются языки программирования C++ и QML, в качестве среды разработки выступает Auora IDE, набор инструментов – Auora SDK.

## 7.5 Требования к программной документации

1. Пояснительная записка оформляется в соответствии с ЛНА РТУ МИРЭА.
2. Проектная документация, составленная в соответствии с ГОСТ.

В процессе создания приложения вся проделанная работа документируется, должны быть сохранены все детали разработки, а также трудности, с которыми пришлось столкнуться. Всё вышеперечисленное должно быть отражено в пояснительной записке, которая прилагается к работе.

## 7.6 Этапы разработки

1. 28.04.2025 – Составление технического задания.

2. 30.04.2025 – Разработка программы

3. 10.05.2025 – Разработка программной документации

4. 05.06.2025 – Защита курсовой работы

## 7.7 Обзор способов организации данных и обоснование структуры данных для эффективного выполнения операций.

Данные могут быть организованы различными способами. Тип структуры данных в программе оказывает большое влияние на ее производительность. Для того чтобы выбрать наиболее простой и эффективный способ организации данных в программе рассмотрим несколько типов структур данных.

### 7.7.1 Двумерный массив

Двумерный массив - Двумерный массив представляет собой упорядоченную совокупность элементов одного типа, организованную в виде таблицы с фиксированным количеством строк и столбцов. Доступ к элементам осуществляется по двум индексам i и j, где i обозначает номер строки, а j — номер столбца. Основным преимуществом двумерного массива является быстрый доступ к элементам за время O(1) благодаря вычислению адреса через смещение. Эта структура проста в реализации, особенно в статически типизированных языках программирования, таких как C или Java. Двумерные массивы эффективны для численных операций, включая матричные вычисления и обработку изображений. Кроме того, при построчном хранении обеспечивается хорошая локализация данных, что повышает производительность за счёт кеш-эффективности.

### 7.7.2 Minesweeper

Minesweeper – главный модуль, написанный на C++ и хранящий все данные, выполняющий все функции игры, и обрабатывающий логику. Этот модуль является самым главным в программе, ведь без выполнения логики графический интерфейс не представляет из себя ничего особенного.

### 7.7.3 Kurs\_m

Модуль QML, отвечающий за внешний вид. В нем описано как должна выглядеть игра, а также реализована связь с главным модулем всей программы.

## 7.8 Выбор структур данных.

В ходе написания программы в основном использовалось 3 типа – Qstring для описания текстовых значений, таких как строка с состоянием игры, bool для логических данных, таких как наличие мин в полях и проиграл ли игрок, и int для числовых, таких как количество закрытых полей, количество соседних мин у поля.

Для хранения поля в памяти использовался двумерный вектор QVector типа bool.

# 8. РЕАЛИЗАЦИЯ

## 8.1 Общие сведения

В ходе выполнения курсовой работы была создана игра «Сапер» с пользовательским графическим интерфейсом под ОС Аврора. В ней выполняются все условия, обозначенные в техническом задании, и содержатся все необходимые компоненты, инструменты для корректной работы

### 8.1.1 Наименование программы

Название программы: «Сапер» или на английском языке «Minesweeper». Оно отражает предназначение и главную функцию созданного приложения.

### 8.1.2 Программное обеспечение, необходимое для функционирования программы

Для корректного функционирования данного программного продукта необходимо, чтобы на персональном компьютере или ноутбуке пользователя была установлена ОС АВРОРА. Также требуется наличие графического адаптера, чтобы устройство могло справляться с обработкой отображения приложения. Другие требования к устройству пользователя не предусмотрены.

### 8.1.2 Язык программирования, на котором написана программа

Для написания программы были использованы языки программирования С++ и QML, за их доступность, понятность и высокую производительность

## 8.2 Функциональное назначение программы (классы решаемых задач и функциональные ограничения на применения)

Данная игра написана для полезного проведения досуга за электронным устройством. Функциональные цели приложения включают операции с начало игры при выигрыше/проигрыше, генерацию случайного поля при начале новой игры, полный функционал самой игры “сапер”

## 8.3 Описание логической структуры программы

В программе используется Объектно-ориентированный подход реализации алгоритмов для упрощения данного приложения. Для хранения данных о поле и состояния игры используются встроенные структуры данных, такие как двумерные массивы и примитивные типы данных. Исходный код программы представлен в Приложении А.

### 8.3.1 Алгоритмы, используемые в программе

Для написания программы необходимо подключить библиотеку «rand» для работы с генератором случайных чисел, работа с векторами осуществляется средствами библиотеки «QVector», для нормальной генерации поля необходимо подключить библиотеку «ctime».

Основными алгоритмами для работы данной игры являются генерация минного поля, проверка ячейки поля и открытие всех свободных клеток поля.

### 8.3.2 Структура программы с описанием функций составных частей и связей между ними

Для грамотного структурирования программы внутри объекта Minesweeper были созданы несколько функций, которые вызываются при разных действиях пользователя:

Reset() – сброс игры в начальное состояние

Reveal(int row, int col) – открытие значения клетки

generateBoard() – случайная генерация поля

getCellValue(int row, int col) – получение значения на вывод минного поля (сколько мин рядом)

reveaEmptyCells(int row, int col) – Открытие всех минных полей с 0 мин соседствующих с нажатой клеткой.

## 8.4 Вызов программы

В данном пункте будет продемонстрирована работа программы с помощью снимков экрана во время ее работы:

Состояние игры при запуске:

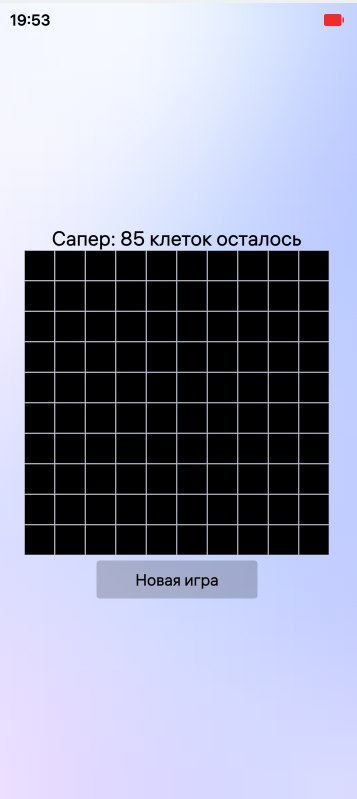


Рисунок 8.1 – Начало игры

Открытие свободных полей:



Рисунок 8.2 – Открытие свободных полей

Проигрыш:



Рисунок 8.3 – Состояние игры при проигрише

Новая игра:

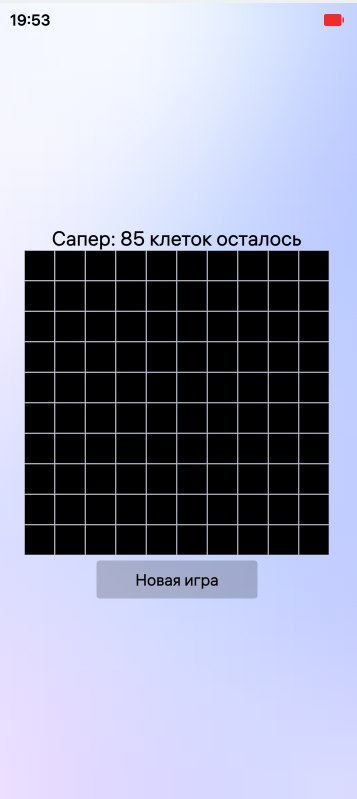


Рисунок 8.4 – Начало новой игры

Победа игрока:



Рисунок 8.5 - Состояние игры при победе

## 8.6 Входные данные

В качестве входных данных программа принимает нажатия пользователя

## 8.7 Выходные данные

Выходные данные – графическое отображение минного поля.

ЗАКЛЮЧЕНИЕ

На протяжении всего процесса проектирования и создания программного продукта были получены практические навыки в разработке приложений с графическим интерфейсом на C++ и QML

Успешно выполнены поставленные задачи: создание игры “сапер”, реализация игрового цикла, генерация случайного поля, начало игры с начала после проигрыша или выигрыша

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Лозовский В.В. Алгоритмические основы обработки данных: учебное пособие / Лозовский В.В., Платонова О.В., Штрекер Е.Н. — М.: МИРЭА – Российский технологический университет, 2022. – 337 с.

2. Платонова О.В. Алгоритмические основы обработки данных: методические указания / Платонова О.В., Асадова Ю.С., Расулов М.М. — М.: МИРЭА – Российский технологический университет, 2022. — 73 с.

3. Белик А.Г. Алгоритмы и структуры данных: учебное пособие / А.Г. Белик, В.Н. Цыганенко. — Омск: ОмГТУ, 2022. — 104 с. — ISBN 978-5-8149-3498-7. — Текст: электронный // Лань: электронно-библиотечная система. — URL: https://e.lanbook.com/book/343688 (дата обращения: 03.09.2024)

4. Павлов Л.А. Структуры и алгоритмы обработки данных / Л.А. Павлов, Н.В. Первова. — 2-е изд., стер. — Санкт-Петербург: Лань, 2022. — 256 с. — ISBN 978-5-507-44105-1. — Текст: электронный // Лань: электронно-библиотечная система. — URL: https://e.lanbook.com/book/207563 (дата обращения: 03.09.2024)

5. Пантелеев Е.Р. Алгоритмы и структуры данных: учебное пособие / Е.Р. Пантелеев, А.Л. Алыкова. — Иваново: ИГЭУ, 2018. — 142 с. — Текст: электронный // Лань: электронно-библиотечная система. — URL: https://e.lanbook.com/book/154576 (дата обращения: 03.09.2024)

Приложение А

Код программы приведен в листингах 11.1, 11.2, 11.3, 11.4

Листинг 11.1 – код main.cpp

|  |
| --- |
| #include <auroraapp.h>  #include <QtQuick>  #include <QGuiApplication>  #include <QQmlApplicationEngine>  #include <QQmlContext>  #include "custom.h"  #include "minesweeper.h"  int main(int argc, char \*argv[])  {  QScopedPointer<QGuiApplication> application(Aurora::Application::application(argc, argv));  application->setOrganizationName(QStringLiteral("ru.template"));  application->setApplicationName(QStringLiteral("kurs"));  qmlRegisterType<Custom>("ru.auroraos.Custom", 1, 0, "Custom");  qmlRegisterType<Minesweeper>("Minesweeper", 1, 0, "Minesweeper");  QScopedPointer<QQuickView> view(Aurora::Application::createView());  view->setSource(Aurora::Application::pathTo(QStringLiteral("qml/kurs.qml")));  view->show();  return application->exec();  } |

Листинг 11.2 – код minesweeper.h

|  |
| --- |
| #ifndef MINESWEEPER\_H  #define MINESWEEPER\_H  #include <QObject>  #include <QVector>  class Minesweeper : public QObject  {  Q\_OBJECT  Q\_PROPERTY(int rows READ rows CONSTANT)  Q\_PROPERTY(int cols READ cols CONSTANT)  Q\_PROPERTY(int mineCount READ mineCount CONSTANT)  Q\_PROPERTY(int revealedCount READ revealedCount NOTIFY revealedCountChanged)  Q\_PROPERTY(bool gameOver READ gameOver NOTIFY gameOverChanged)  Q\_PROPERTY(bool gameWon READ gameWon NOTIFY gameWonChanged)  public:  explicit Minesweeper(int rows = 10, int cols = 10, int mineCount = 15, QObject \*parent = nullptr);  int rows() const { return m\_rows; }  int cols() const { return m\_cols; }  int mineCount() const { return m\_mineCount; }  int revealedCount() const { return m\_revealedCount; }  bool gameOver() const { return m\_gameOver; }  bool gameWon() const { return m\_gameWon; }  Q\_INVOKABLE void reset(); |

Продолжение листинга 11.2

|  |
| --- |
| Q\_INVOKABLE void reveal(int row, int col);  Q\_INVOKABLE void toggleFlag(int row, int col);  Q\_INVOKABLE int getCellValue(int row, int col) const;  Q\_INVOKABLE bool isCellRevealed(int row, int col) const;  Q\_INVOKABLE bool isCellFlagged(int row, int col) const;  Q\_INVOKABLE bool isCellMine(int row, int col) const;  // Q\_INVOKABLE void generateBoard(int roww, int coll);  signals:  void revealedCountChanged();  void gameOverChanged();  void gameWonChanged();  void boardChanged();  private:  void generateBoard(int roww, int coll);  void revealEmptyCells(int row, int col);  int countAdjacentMines(int row, int col) const;  bool isValidCell(int row, int col) const;  int m\_rows;  int m\_cols;  int m\_mineCount;  int m\_revealedCount;  bool m\_gameOver;  bool m\_gameWon;  bool first\_time;  QVector<QVector<bool>> m\_mines;  QVector<QVector<bool>> m\_revealed;  QVector<QVector<bool>> m\_flags;  };  #endif // MINESWEEPER\_H |

Листинг 11.3 – код minesweeper.cpp

|  |
| --- |
| #include "minesweeper.h"  // #include <QRandomGenerator>  #include <random>  #include <QDebug>  #include <ctime>  Minesweeper::Minesweeper(int rows, int cols, int mineCount, QObject \*parent)  : QObject(parent), m\_rows(rows), m\_cols(cols), m\_mineCount(mineCount),  m\_revealedCount(0), m\_gameOver(false), m\_gameWon(false)  {  std::srand(std::time({}));  reset();  }  void Minesweeper::reset()  {  m\_mines = QVector<QVector<bool>>(m\_rows, QVector<bool>(m\_cols, false));  m\_revealed = QVector<QVector<bool>>(m\_rows, QVector<bool>(m\_cols, false));  m\_flags = QVector<QVector<bool>>(m\_rows, QVector<bool>(m\_cols, false));  m\_revealedCount = 0;  m\_gameOver = false;  m\_gameWon = false;  first\_time = true;  generateBoard(-1, -1);  emit revealedCountChanged(); |

Продолжение листинга 11.3

|  |
| --- |
| emit gameOverChanged();  emit gameWonChanged();  emit boardChanged();  }  void Minesweeper::generateBoard(int roww, int coll)  {  // Place mines randomly  int minesPlaced = 0;  while (minesPlaced < m\_mineCount) {  int row = rand() % m\_rows;  int col = rand() % m\_cols;  if (!m\_mines[row][col] && !(row == roww && coll == col)) { // && !(row == roww && coll == col) int roww, int coll  m\_mines[row][col] = true;  minesPlaced++;  }  }  }  void Minesweeper::reveal(int row, int col)  {  if (m\_gameOver m\_gameWon !isValidCell(row, col) m\_revealed[row][col] m\_flags[row][col])  return;  if (m\_mines[row][col] && !first\_time) {  // Game over - hit a mine  m\_revealed[row][col] = true;  m\_gameOver = true;  emit gameOverChanged();  emit boardChanged();  return;  }  if(first\_time){  m\_mines = QVector<QVector<bool>>(m\_rows, QVector<bool>(m\_cols, false));  m\_revealed = QVector<QVector<bool>>(m\_rows, QVector<bool>(m\_cols, false));  m\_flags = QVector<QVector<bool>>(m\_rows, QVector<bool>(m\_cols, false));  m\_revealedCount = 0;  m\_gameOver = false;  m\_gameWon = false;  generateBoard(row, col);  }  first\_time = false;  revealEmptyCells(row, col);  // Check for win condition  if (m\_revealedCount == (m\_rows \* m\_cols - m\_mineCount)) {  m\_gameWon = true;  emit gameWonChanged();  }  emit boardChanged();  }  void Minesweeper::revealEmptyCells(int row, int col)  { |

Продолжение листинга 11.4

|  |
| --- |
| if (!isValidCell(row, col) m\_revealed[row][col] m\_mines[row][col])  return;  m\_revealed[row][col] = true;  m\_revealedCount++;  emit revealedCountChanged();  // If this cell has no adjacent mines, reveal all adjacent cells  if (countAdjacentMines(row, col) == 0) {  for (int r = -1; r <= 1; r++) {  for (int c = -1; c <= 1; c++) {  if (r == 0 && c == 0) continue;  revealEmptyCells(row + r, col + c);  }  }  }  }  void Minesweeper::toggleFlag(int row, int col)  {  if (m\_gameOver m\_gameWon !isValidCell(row, col) || m\_revealed[row][col])  return;  m\_flags[row][col] = !m\_flags[row][col];  emit boardChanged();  }  int Minesweeper::getCellValue(int row, int col) const  {  if (!isValidCell(row, col) !m\_revealed[row][col] m\_mines[row][col])  return -1;  return countAdjacentMines(row, col);  }  bool Minesweeper::isCellRevealed(int row, int col) const  {  return isValidCell(row, col) && m\_revealed[row][col];  }  bool Minesweeper::isCellFlagged(int row, int col) const  {  return isValidCell(row, col) && m\_flags[row][col];  }  bool Minesweeper::isCellMine(int row, int col) const  {  return isValidCell(row, col) && m\_mines[row][col];  }  int Minesweeper::countAdjacentMines(int row, int col) const  {  int count = 0;  for (int r = -1; r <= 1; r++) {  for (int c = -1; c <= 1; c++) {  if (r == 0 && c == 0) continue;  int newRow = row + r;  int newCol = col + c;  if (isValidCell(newRow, newCol) && m\_mines[newRow][newCol]) {  count++;  }  }  } |

Продолжение листинга 11.3

|  |
| --- |
| return count;  }  Bool Minesweeper::isValidCell(int row, int col) const  {  return row >= 0 && row < m\_rows && col >= 0 && col < m\_cols;  } |

Листинг 11.4 – код kurs.qml

|  |
| --- |
| import QtQuick 2.0  import Sailfish.Silica 1.0  import ru.auroraos.Custom 1.0  import Minesweeper 1.0  /\* ApplicationWindow {  // objectName: "applicationWindow"  // initialPage: Qt.resolvedUrl("pages/MainPage.qml")  // cover: Qt.resolvedUrl("cover/DefaultCoverPage.qml")  // allowedOrientations: defaultAllowedOrientations  // Custom{  // id: customObj  // }  // Text {  // id: name  // anchors.centerIn: parent  // text: customObj.getTextContent()  // font.pixelSize: customObj.getTextSize()  // }  } \*/  ApplicationWindow {  // id: window  // visible: true  // width: 400  // height: 500  //title: qsTr("Сапер")  Minesweeper {  id: game  // rows: 10  // cols: 10  // mineCount: 15  }  Column {  anchors.fill: parent  spacing: 10  Text {  id: top\_text  font.pixelSize: 40  text: {  if (game.gameOver) return "Игра окончена! Вы проиграли.";  if (game.gameWon) return "Поздравляем! Вы выиграли!";  return "Сапер: " + (game.rows \* game.cols - game.mineCount - game.revealedCount) + " клеток осталось";  }  anchors.horizontalCenter: parent.horizontalCenter  y: grid.y - 50  // padding: 10 |

Продолжение листинга 11.4

|  |
| --- |
| }  Grid {  id: grid  columns: game.cols  spacing: 1  anchors.centerIn: parent  property bool first\_time: true  Repeater {  model: game.rows \* game.cols  Rectangle {  id: cell  function onFieldupdate(){  cell.color = game.isCellRevealed(row, col) ? (game.isCellMine(row, col) ? "red" : "#ddd") : Theme.primaryColor;  if (!game.isCellRevealed(row, col)) {  rect\_text.text = game.isCellFlagged(row, col) ? "X" : "";  }  if (game.isCellMine(row, col)) rect\_text.text = "💣";  var value = game.getCellValue(row, col);  rect\_text.text = value > 0 ? value : "";  }  readonly property var refreshTrigger: game.boardChanged  width: 60  height: 60  // color: game.isCellRevealed(row, col) ? (game.isCellMine(row, col) ? "red" : "#ddd") : "#aaa";  color: {  refreshTrigger;  return game.isCellRevealed(row, col) ? (game.isCellMine(row, col) ? "red" : "#ddd") : Theme.primaryColor;  // if (game.isCellRevealed(Math.floor(index / game.cols), index % game.cols)) {  // return game.isCellMine(Math.floor(index / game.cols), index % game.cols) ? "red" : "#ddd";  // }  // return "#aaa";  }  border.color: "#777"  border.width: 1  property int row: Math.floor(index / game.cols)  property int col: index % game.cols  Text {  id:rect\_text  anchors.centerIn: parent  font.pixelSize: 32  }  MouseArea {  anchors.fill: parent  acceptedButtons: Qt.LeftButton | Qt.RightButton |

Продолжение листинга 11.4

|  |
| --- |
| onClicked: {  if (mouse.button === Qt.LeftButton) {  game.reveal(row, col);  } else if (mouse.button === Qt.RightButton) {  game.toggleFlag(row, col);  }  }  }  Component.onCompleted: {  game.boardChanged.connect(onFieldupdate)  }  }  }  }  Button {  text: "Новая игра"  anchors.horizontalCenter: parent.horizontalCenter  y: grid.y + 620  onClicked: game.reset()  }  }  } |