

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО
ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

НАБЕРЕЖНОЧЕЛНИНСКИЙ ИНСТИТУТ (ФИЛИАЛ) ФЕДЕРАЛЬНОГО
ГОСУДАРСТВЕННОГО АВТОНОМНОГО
ОБРАЗОВАТЕЛЬНОГО УЧРЕЖДЕНИЯ ВЫСШЕГО ОБРАЗОВАНИЯ
«КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

ВЫСШАЯ ТЕХНИЧЕСКАЯ ШКОЛА
КАФЕДРА ИНФОРМАЦИОННЫХ СИСТЕМ

Направление подготовки: 01.03.02 Прикладная математика и информатика

КУРСОВАЯ РАБОТА

IV СЕМЕСТР

Тема: «Разработка алгоритма и программная реализация

задачи поиска выхода из прямоугольного лабиринта»

Дисциплина: «Языки и методы программирования»

Выполнил

студент Ахмадишин И.Р.,

группа 2221101 курс 2

Проверил

доцент, к.п.н.

Гумерова Л.З. _____

подпись

Оценка _____

Дата _____

Набережные Челны –2023

Содержание

Введение	3
1. Теоретическая часть разработки	6
1.1 Основные элементы программы для построения графиков	6
1.2 Методы и подходы к решению задачи	6
2. Практическая часть разработки программы.....	12
2.1 Анализ задачи.....	12
2.2 Разработка программы	12
2.2.1 Создание проекта	12
2.2.2 Создание интерфейса приложения	15
2.2.3 Разработка кода приложения.....	15
Заключение	24
Список используемых источников.....	25
Приложение 1	26

Введение

Лабиринт — это структура, состоящая из одного или нескольких извилистых путей, ведущих от входа к цели. Лабиринты несут в себе ощущение тайны и загадки. Одним из самых известных был лабиринт Минотавра. В современном мире такие лабиринты являются объектами развлечения. В парках и садах можно найти запутанные лабиринты из живых изгородей, а также зеркальные и снежные лабиринты.

Их использовали для различных целей, например:

- **В Древнем Египте:** для создания сложных структур.
- **В греческой мифологии:** для содержания мифических существ.
- **В средневековой Европе:** для духовного отражения.

Даже сегодня лабиринты по-прежнему используются для развлечения, отдыха и художественного самовыражения. Они символизируют путешествия, исследования и самоанализ.

Классификация лабиринтов

Лабиринты можно классифицировать по различным критериям, включая их структуру, предназначение и исторический контекст. Вот основные типы лабиринтов:

1. По структуре

1.1. Классический лабиринт:

1.1.1. Один путь: Лабиринт с одним путём без разветвлений, который ведёт от входа к центру и обратно.

1.2. Многоходовой (многофазный) лабиринт: Лабиринт с множеством пересекающихся путей, тупиков и развилок.

Примеры: Садовые и парковые лабиринты, такие как лабиринт в Хэмптон-Корт в Англии.

2. По предназначению

2.1. Медитативные и религиозные лабиринты:

2.1.1. Использование: Для духовных и медитативных практик, как символическое путешествие.

Примеры: Лабиринты в христианских соборах и монастырях.

2.2.Развлекательные лабиринты:

2.2.1. Использование: Для развлечения и отдыха.

Примеры: Парковые лабиринты, лабиринты в аттракционах.

2.3.Ритуальные и церемониальные лабиринты:

2.3.1. Использование: В обрядах и ритуалах, часто связанные с древними культурами.

2.4.Примеры: Кельтские и скандинавские лабиринты, использовавшиеся в ритуальных целях.

3. По материалам и конструкции

3.1.Садовые и живые лабиринты:

3.1.1. Материалы: Живые изгороди, кустарники.

Примеры: Лабиринт в Хэмптон-Корт.

3.2.Каменные и наземные лабиринты:

3.2.1. Материалы: Камни, выкладки на земле.

Примеры: Древние лабиринты в Северной Европе.

3.3.Лабиринты в помещениях:

3.3.1. Материалы: Стены и перегородки внутри зданий.

Примеры: Лабиринты в аттракционах, тематических парках.

4. По историческому контексту

4.1.Древние лабиринты:

Примеры: Кносский лабиринт на Крите, лабиринты в Скандинавии.

4.2.Средневековые лабиринты:

Примеры: Лабиринты в готических соборах (например, в Шартре).

4.3.Современные лабиринты:

Примеры: Лабиринты в современных парках и тематических аттракционах.

Каждый из этих типов лабиринтов имеет свои особенности и используется в различных контекстах и целях.

Разработка программы для построения лабиринта

В соответствии с поставленной целью необходимо решить следующие задачи:

- Изучить учебную и справочную литературу по теме исследования;
- Закрепить знания, полученные в процессе изучения теории по курсу «Языки и методы программирования»;
- применить полученные знания и навыки при решении проектных задач;
- Закрепить умения использовать справочную и нормативную литературу.

1. Теоретическая часть разработки

1.1 Основные элементы программы для построения лабиринта

Приложение должно строиться из трёх окон. Первое - для указания размера лабиринта, второе - для указания начальной позиции, третье - главное окно с лабиринтом и вкладками с командами для запуска поиска лабиринта, для изменения цвета из интерфейса. Для главного лабиринта создаётся вектор кнопок и вектор характеристик кнопок.

1.2 Методы и подходы к решению задачи

C++ предлагает множество инструментов и возможностей для создания высокоэффективных и масштабируемых приложений. Благодаря своим мощным средствам объектно-ориентированного программирования (ООП), управлению памятью, шаблонам и обширной стандартной библиотеке, C++ является отличным выбором для разнообразных задач, от системного программирования до разработки сложных прикладных программ.

C++: Основные аспекты и особенности

История:

Разработчик: Язык был создан Бьярне Страуструпом в 1979 году в Bell Labs.

Первоначальное название: C с классами (C with Classes).

Переименование: В 1983 году язык был переименован в C++.

Основные характеристики:

Объектно-ориентированное программирование (ООП): поддерживает классы, объекты, наследование, полиморфизм, инкапсуляцию и абстракцию.

Низкоуровневое управление: позволяет работать с памятью напрямую с помощью указателей, что обеспечивает высокую производительность.

Обобщённое программирование: Шаблоны (templates) позволяют создавать универсальные классы и функции, что способствует повторному использованию кода.

Поддержка нескольких парадигм: включает процедурное, объектно-ориентированное и обобщённое программирование.

Богатая стандартная библиотека: включает контейнеры, алгоритмы, итераторы и другие компоненты, облегчающие разработку.

Применение:

Системное программное обеспечение: Операционные системы, драйверы.

Игровая индустрия: Игровые движки (например, Unreal Engine).

Приложения в реальном времени: Системы управления, финансовые приложения.

Встроенные системы: Программирование микроконтроллеров и встроенных устройств.

Научные вычисления: Моделирование и анализ данных.

Примеры кода

Простой пример программы на C++:

cpp

Копировать код

```
#include <iostream>
```

```
int main() {
```

```
    std::cout << "Hello, World!" << std::endl;
```

```
    return 0;
```

```
}
```

Пример использования класса:

cpp

Копировать код

```
#include <iostream>
```

```
#include <string>
```

```
class Person {
```

```
private:
```

```
    std::string name;
```

```
    int age;
```

```

public:
    Person(std::string n, int a) : name(n), age(a) {}

    void introduce() {
        std::cout << "Hi, I'm " << name << " and I'm " << age << " years old."
<< std::endl;
    }
};

int main() {
    Person person("Alice", 30);
    person.introduce();
    return 0;
}

```

Компиляторы и среды разработки

Компиляторы: GCC, Clang, MSVC.

Среды разработки: Visual Studio, CLion, Code::Blocks, Qt Creator.

Стандарты C++

C++ развивается через стандарты, поддерживаемые комитетом ISO.

Основные версии стандартов:

C++98: Первый ISO-стандарт (1998 год).

C++03: Небольшие исправления к C++98.

C++11: Значительные улучшения, включая лямбда-функции, автоопределение типов и умные указатели.

C++14, C++17, C++20: Последующие улучшения и новые возможности.

C++23: Ожидаемая версия с новыми фичами.

C++ остаётся одним из самых популярных языков программирования благодаря своей эффективности, гибкости и широкому применению в различных областях.

Qt: Мощный фреймворк для разработки кроссплатформенных приложений на C++

Qt предоставляет широкий спектр инструментов и библиотек для создания графических интерфейсов, работы с базами данных, сетевого программирования и многого другого. Разработка приложений с использованием Qt на C++ позволяет создавать производительные и удобные в использовании программы, которые могут работать на различных операционных системах, таких как Windows, macOS, Linux и другие.

Основные компоненты Qt

Qt Widgets: для создания классических десктопных графических интерфейсов.

Qt Quick: для создания современных, динамичных интерфейсов с использованием QML (язык описания интерфейсов) и JavaScript.

Qt Core: Основные классы и функции для работы с файлами, строками, потоками и т.д.

Qt Network: для работы с сетевыми протоколами и интернет-приложениями.

Qt SQL: для работы с базами данных через унифицированный интерфейс.

Пример приложения на Qt### Пример приложения на Qt и C++

Простой пример приложения, создающего окно с кнопкой, при нажатии на которую выводится сообщение:

Шаг 1: Установка Qt

Для начала необходимо установить Qt. Это можно сделать, скачав Qt Installer с официального сайта qt.io.

Шаг 2: Создание проекта

Используйте Qt Creator (среду разработки, поставляемую вместе с Qt) для создания нового проекта.

Шаг 3: Пример кода

main.cpp:

cpp

Копировать код

```
#include <QApplication>
```

```
#include <QPushButton>
```

```
int main(int argc, char *argv[]) {  
    QApplication app(argc, argv);  
  
    QPushButton button("Click me");  
  
    QObject::connect(&button, &QPushButton::clicked, []() {  
        qDebug("Button clicked!");  
    });  
  
    button.show();  
  
    return app.exec();  
}
```

.pro файл (Qt project file):

plaintext

Копировать код

```
QT += core gui  
  
greaterThan(QT_MAJOR_VERSION, 4): QT += widgets  
  
TARGET = example  
TEMPLATE = app  
  
SOURCES += main.cpp
```

Основные шаги в разработке приложений с использованием Qt

Создание проекта: С помощью Qt Creator создайте новый проект, выбрав тип проекта (например, Qt Widgets Application или Qt Quick Application).

Дизайн интерфейса: Используйте графический редактор в Qt Creator для создания интерфейсов или пишите QML-код для более сложных и анимированных интерфейсов.

Написание логики: Реализуйте логику приложения на C++ в соответствующих файлах. Используйте сигналы и слоты для обработки событий.

Сборка и отладка: Используйте встроенные инструменты для сборки и отладки приложения.

Тестирование и деплоймент: Тестируйте ваше приложение на различных платформах и подготавливайте его к выпуску.

Преимущества использования Qt

Кроссплатформенность: Возможность создания приложений, которые работают на различных операционных системах без изменений в коде.

Богатый набор библиотек: Наличие большого количества готовых к использованию классов и функций.

Модернизация интерфейса: Поддержка QML для создания современных и привлекательных интерфейсов.

Поддержка сообщества: Активное сообщество разработчиков и обширная документация.

Qt и C++ — это мощное сочетание для создания высокопроизводительных, кроссплатформенных приложений с современными интерфейсами и богатой функциональностью.

2. Практическая часть разработки программы

2.1 Анализ задачи

Данная курсовая работа посвящена разработке приложения для создания выхода из лабиринта на языке программирования C++. Главной целью проекта является написание программы с вектором из кнопок, и при нажатии на кнопку из вектора мы знали её индекс. Тем самым, если пользователь будет взаимодействовать с одной из кнопок, она изменялась бы, также настройка внешнего вида интерфейса.

2.2 Разработка программы

2.2.1 Создание проекта

Чтобы начать новый проект в Qt Creator, мы можем использовать шаблон под названием Qt Widgets, который представляет собой простое окно для создания программ на обычных компьютерах. Qt Creator это своего рода специальный инструмент, который помогает людям создавать компьютерные программы. Его можно использовать для создания различных типов программ, например, с изображениями или просто словами, с использованием разных языков, таких как C++ и QML. Qt Creator позволяет быстрее и проще создавать программы с помощью платформы Qt. Он содержит полезные инструменты для каждого этапа создания программы.

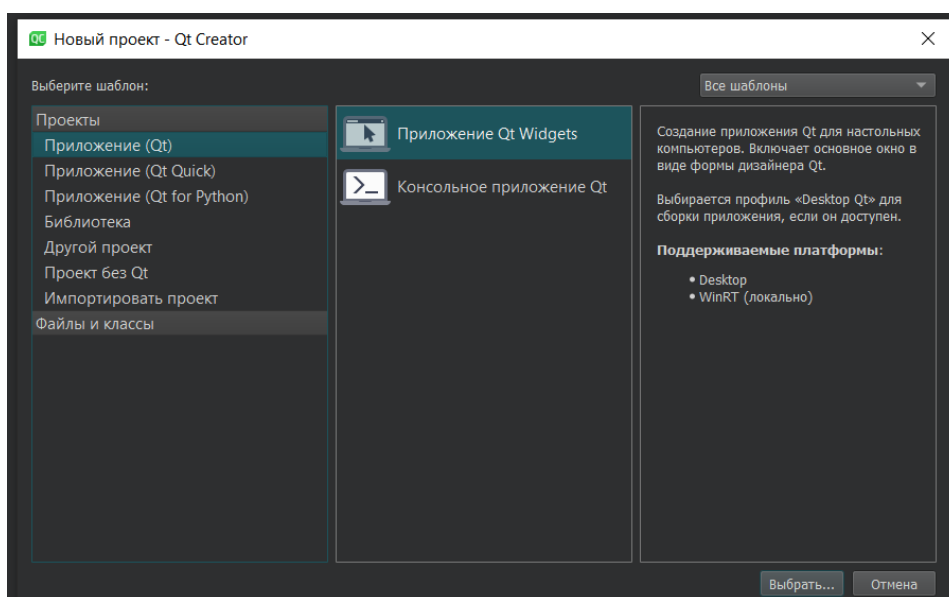


Рисунок 1 — Процесс выбора проекта.

После выбора проекта вам необходимо выбрать имя и указать местоположение проекта, затем выбрать систему сборки qmake (рисунок 2).

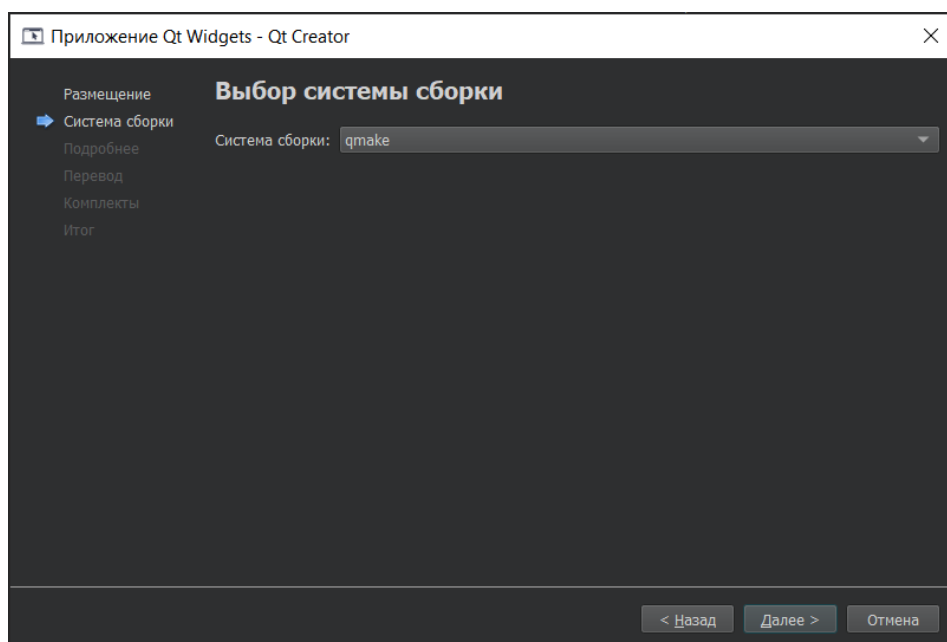


Рисунок 2 — Процесс выбора системы сборки.

Я оставляю класс, файл заголовка и исходный текстовый файл в системной форме, выбираю базовый класс QMainWindow и создаю дополнительную форму пользовательского интерфейса. Все описанные действия можно найти на рисунке 3.

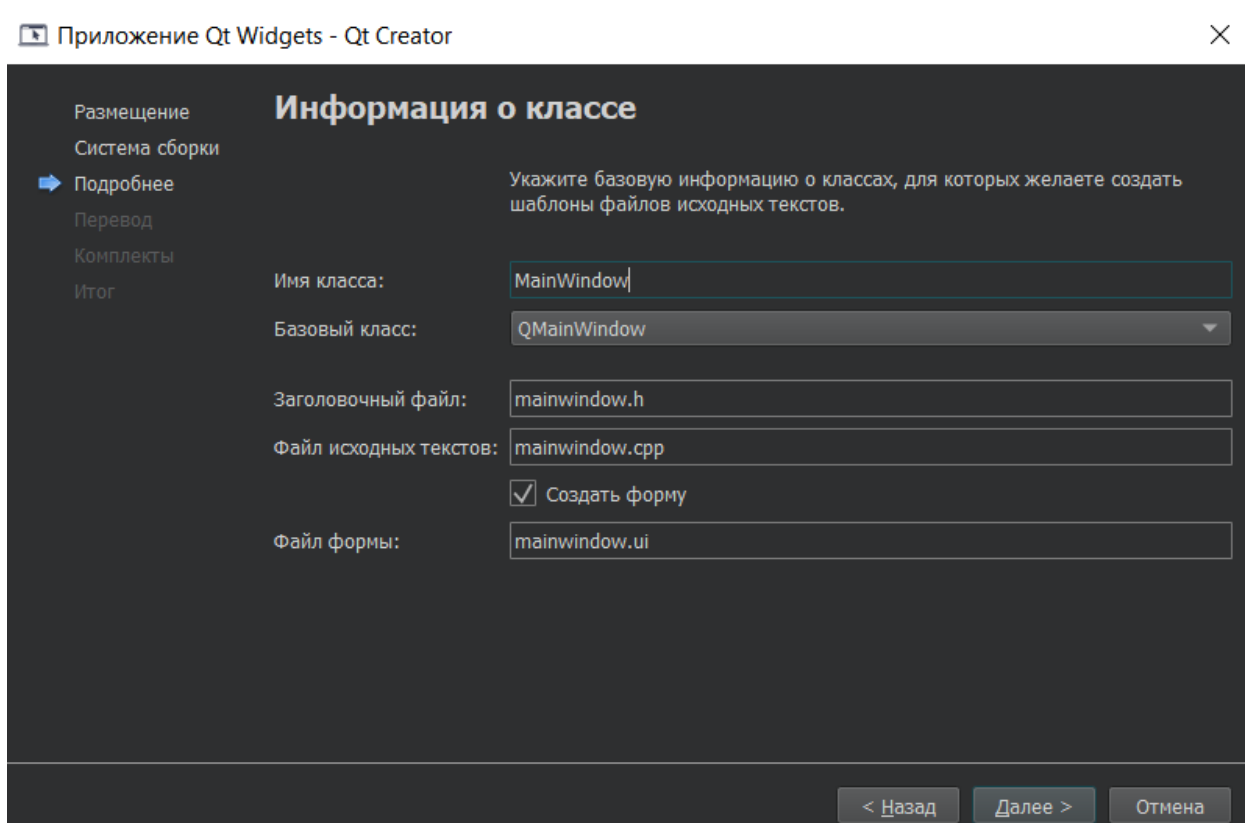


Рисунок 3 — Процесс указания информации о классах.

Следующим шагом будет выбор комплекта для сборки приложения, выбрав его, проверяю все свои действия и получаю проект, как показано на рисунке 4.

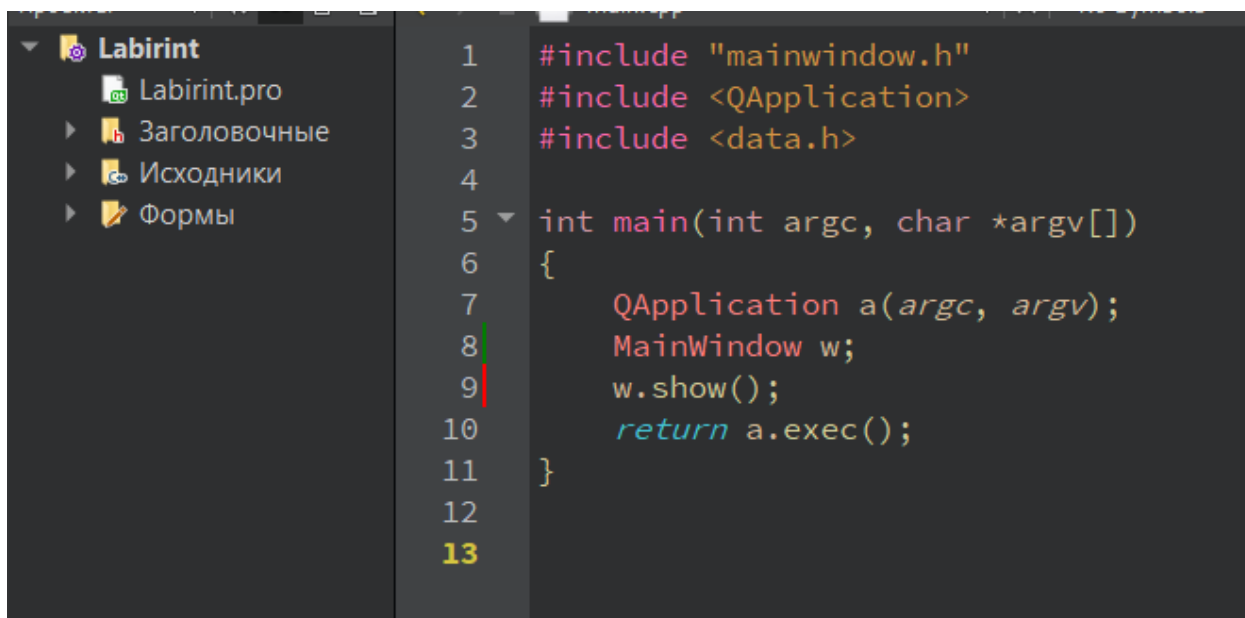


Рисунок 4 — начало проекта по созданию приложения.

На этом все шаги по созданию проекта завершены и переходим к созданию приложения.

2.2.2 Создание интерфейса приложения

Я решил, что первым делом должно выходить диалоговое окно с двумя редакторами строк, одна для указания количества столбцов, другая для строк. В файле `data.ui` располагаем все редакторы строк. Так же и в файле `start.ui`, только для указания начальной позиции. Затем выходить главное окно с двумя вкладками и их командами.

2.2.3 Разработка кода приложения

Создаём функцию, которая будет вызывать диалоговое окно, где пользователь будет вводить количество столбцов и строк. Если пользователь не ввёл количество столбцов и строк, программа полностью закрывается. Также в этой функции заполняем вектор характеристик кнопок `ct`, где размер вектора `ct` равен произведению количества строк и столбцов. Все элементы будут равны одному.

`Get_row` и `Get_column` — функции, которые возвращают значения для `rows` (строк) и `cols` (столбцов).

Листинг 1 — Вызов функции для указания размера лабиринта.

```
bool MainWindow::zap(){
    DATA d;
    d.setWindowIcon(QIcon("data.png"));
    QLabel l1{ &d };
    l1.setGeometry(40,40,200,24);
    l1.setText("Ведите кол-во столбцов:");
    QLabel l2{ &d };
    l2.setGeometry(40,100,200,24);
    l2.setText("Ведите кол-во строк:");
    if (d.exec() == QDialog::Accepted){
        rows = d.Get_row();
        cols = d.Get_column();
    }
    for (int i = 0; i < (rows * cols); i++) {
        ct.push_back(1);
    }
    if (rows == 0 && cols == 0){return false;}
    else{return true;}
}
```

Создаем функцию для указания пользователем начальной позиции. В этой функции все тоже самое, только, если начальная позиция выходит за рамки прямоугольника, функция перезапускается.

Листинг 2 — Вызов функции для указания начальной позиции.

```
bool MainWindow::st(){
    Start s;
    QLabel l1{ &s };
    l1.setGeometry(40,30,250,24);
    l1.setText("Ведите начальную позицию.");
    QLabel l2{ &s };
    l2.setGeometry(40,80,250,24);
    l2.setText("Ведите строку на которой находитесь.");
    QLabel l3{ &s };
    l3.setGeometry(40,130,250,24);
    l3.setText("Ведите столбец на которой находитесь.");
    if (s.exec() == QDialog::Accepted){
        st_r = s.st_row() - 1;
        st_c = s.st_column() - 1;
    }
    if((st_r > rows && st_c > cols) || (st_r < rows && st_c > cols) || (st_r > rows && st_c < cols)){return false;}
    else{
        int ogr = 0;
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                if (st_r == i && st_c == j){
                    ct[c] = 3;
                    ogr = 1;
                    break;
                }
            }
            c++;
        }
        if(ogr == 1){
            break;
        }
    }
    return true;
}
```


Затем создаем функцию, где создаются кнопки и помещаются в вектор кнопок `butn`, и добавляются в матрицу с использованием `QGridLayout`. `QGridLayout` используется для размещения кнопок в виде сетки (матрицы). Кнопки добавляются в `QGridLayout` с помощью метода `addWidget`. Выводим кнопки в главное окно. С каждой кнопкой связывается слот `Clk` с помощью `connect`.

Листинг 3 — Вызов функции для заполнения вектора кнопок и вывода в главное окно.

```
void MainWindow::btn(){

    QWidget *centralWidget = new QWidget(this);
    QGridLayout *grid = new QGridLayout(centralWidget);

    for(int i = 0; i < rows; ++i) {
        for(int j = 0; j < cols; ++j) {
            int ii = i * cols + j;
            QPushButton *button = new QPushButton("  ", this);
            button->setMaximumSize(50,50);
            butn.append(button);
            grid->addWidget(button, i, j);
            connect(button, &QPushButton::clicked, this, &MainWindow::Clk);
        }
    }

    centralWidget->setLayout(grid);
    setCentralWidget(centralWidget);
}
```

В слоте `Clk` определяется, какая кнопка была нажата, с использованием `sender`. Затем ищется индекс нажатой кнопки в векторе. Для определения, была ли кнопка нажата, у нас есть вектор `st`, в нём хранятся характеристики кнопок в виде чисел 0, 1, 2, где 0 означает стену, 2 – выход, 1 – коридор.

Индекс в векторе кнопок равен индексу вектора `st`, то есть если пользователь нажал на кнопку, в `st` изменяется число на 0, 1 или 2. Изначально элементы вектора `st` равны 1 (коридор), затем при нажатии на кнопку изменяется элемент в `st`, равный индексу кнопки, на 0 (стена). Если еще раз

была нажата та же кнопка, элемент становится равным 2 (выход), затем при нажатии еще раз – на 1 (коридор). Также каждый раз при нажатии кнопки текст кнопки меняется в зависимости от значения элемента *ct*.

Листинг 4 — События нажатия на кнопки.

```
void MainWindow::Clk()
{
    QPushButton *clickedButton = qobject_cast<QPushButton*>(sender());
    if(clickedButton) {
        int i = btnn.indexOf(clickedButton);
        if(i != -1) {
            if(ct[i] == 1){
                clickedButton->setStyleSheet("color:green");
                ct[i] = 0;
                clickedButton->setText("Стена");
                return;
            }
            if(ct[i] == 0){
                clickedButton->setStyleSheet("color:red");
                clickedButton->setText("Выход");
                ct[i] = 2;
                return;
            }
            if(ct[i] == 2){
                clickedButton->setText(" ");
                ct[i] = 1;
                return;
            }
        }
    }
}
```

После того как пользователь создал лабиринт, он сможет через вкладку «Лабиринт» и команду «Запустить» запустить функцию для подготовки к поиску выхода из лабиринта *on_action_triggered*. В ней создаем две матрицы *ma* и *pt*.

Матрицу *ma* заполняем элементами *ct*; *ma* имеет вид, как расположены кнопки на главном окне, а с элементами *ct* будет представление, как выглядит

наш лабиринт. У матрицы `pt` изначально элементы равны 0. В дальнейшем она будет показывать путь к выходу из лабиринта.

Вызываем функцию `findExit` для поиска выхода из лабиринта. Если функция находит выход из лабиринта, вызывается окно сообщения с текстом «Выход найден!», иначе вызывается окно сообщения с текстом «Выход не найден!». В `findExit` передаются матрицы `ma` и `pt` и начальная позиция. После того как функция `findExit` завершила работу, заполняем вектор `st` элементами матрицы `pt`, затем изменяем кнопки: если до этого 1 – это коридор, то теперь это путь.

Листинг 5 — Вызов функции для подготовки к поиску выхода из лабиринта.

```
void MainWindow::on_action_triggered()
{
    QVector <QVector<int>> ma(rows);
    QVector <QVector<int>> pt(rows);
    c = 0;
    for (int i = 0; i < rows; i++) {
        ma[i].resize(cols);
        pt[i].resize(cols);
        for (int j = 0; j < cols; j++) {
            ma[i][j] = ct[c];
            pt[i][j] = 0;
            c++;
        }
    }

    if (findExit(ma, pt, st_r, st_c)) {
        pt[st_r][st_c] = 3;
        QMessageBox msg;
        msg.setGeometry(800,300,800,800);
        msg.setText("Выход найден!");
        msg.exec();
    } else {
        pt[st_r][st_c] = 3;
        QMessageBox msg;
        msg.setGeometry(800,300,800,800);
        msg.setText("Выход не найден!");
```

```

        msg.exec();
    }

    c = 0;
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            ct[c] = pt[i][j];
            if (pt[i][j] == 1 ){
                btn[c]->setStyleSheet("color:purple");
                btn[c]->setText("Путь");
                c++;
            }
            if(pt[i][j] == 0){
                btn[c]->setStyleSheet("color:green");
                btn[c]->setText("Стена");
                c++;
            }
            if(pt[i][j] == 2){
                btn[c]->setStyleSheet("color:red");
                btn[c]->setText("Выход");
                c++;
            }
            if(pt[i][j] == 3){
                btn[c]->setStyleSheet("color:orange");
                btn[c]->setText("Тут");
                c++;
            }
        }
    }
    c = 1;
}

```

Наконец, создаем функцию для поиска выхода из лабиринта findExit, представленного матрицей maze. Функция принимает также матрицу sol, которая будет содержать решение - путь к выходу из лабиринта.

На каждом шаге функция проверяет текущую клетку (с координатами row и col) на соответствие условиям:

- Если мы вышли за границы лабиринта или попали в стену (клетка равна 0), функция возвращает false.

- Если мы достигли выхода из лабиринта (клетка равна 2), функция добавляет текущую клетку в `sol` и возвращает `true`.
- Если текущая клетка не является выходом и также не является стеной, то она помечается как посещенная (значение клетки меняется на 0), и для каждой из соседних клеток (сверху, снизу, слева, справа) рекурсивно вызывается функция `findExit`.
- Если хотя бы одна из соседних клеток содержит выход (рекурсивные вызовы вернули `true`), то текущая клетка добавляется в `sol` и функция возвращает `true`.
- Если выход не найден в соседних клетках, функция возвращает `false`.

Таким образом, функция рекурсивно проходит все возможные пути в лабиринте, пока не найдет выход.

Листинг 6 — Поиск выхода из лабиринта.

```

Bool MainWindow::findExit(QVector<QVector<int>>& maze, QVector<QVector<int>>& sol , int row, int
col) {
    if (row < 0 || col < 0 || row >= maze.size() || col >= maze[0].size() || maze[row][col] == 0) {
        return false;
    }

    if (maze[row][col] == 2) {
        sol[row][col] = 2;
        return true;
    }

    maze[row][col] = 0;

    if (findExit(maze, sol, row + 1, col) || findExit(maze, sol, row - 1, col) || findExit(maze, sol, row, col + 1) ||
findExit(maze, sol, row, col - 1)) {
        sol[row][col] = 1;
        return true;
    }

    return false;
}

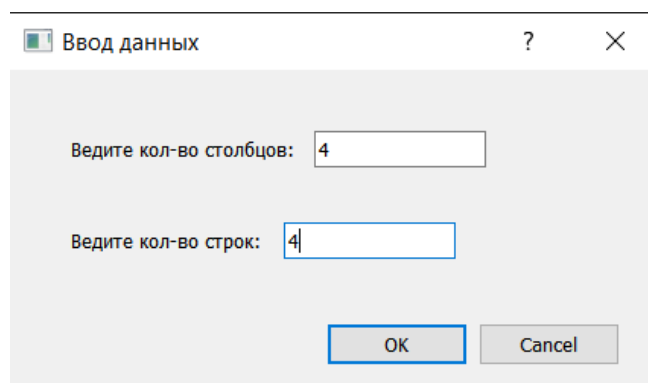
```

Также через вкладку «Лабиринт» можно запустить команду «Очистить». Она очищает лабиринт, если пользователь захотел создать новый лабиринт. В ней элементы вектора характеристик кнопок вновь равны 1, кроме начальной позиции. Все элементы вектора кнопок меняются, кроме той, которая имеет индекс, равный индексу элемента вектора характеристик кнопок, который равен 3, то есть начальному местоположению.

Листинг 7 — функция очистки лабиринта.

```
void MainWindow::on_action_5_triggered()
{
    for(int i = 0; i < butn.size(); i++){
        if(ct[i] == 3){
            butn[i]->setStyleSheet("background-color: white");
            butn[i]->setStyleSheet("color:orange");
            butn[i]->setText("Тут");
        }
        else{
            butn[i]->setStyleSheet("background-color: white");
            butn[i]->setText(" ");
            ct[i] = 1;
        }
    }
}
```

Результат выполнения программы можно увидеть на рисунках 5-8.



Ввод данных

Ведите кол-во столбцов: 4

Ведите кол-во строк: 4

OK Cancel

Рисунок 5 – Ввод столбцов и строк.

Начальная позиция

Ведите начальную позицию.

Ведите строку на которой находитесь: 2

Ведите столбец на которой находитесь: 2

OK Cancel

Рисунок 6 – Ввод начальной позиции.

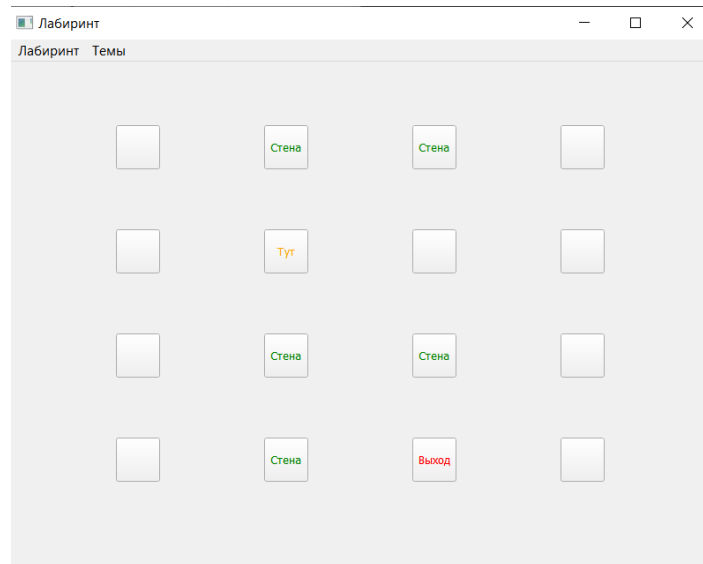


Рисунок 7 – Лабиринт.

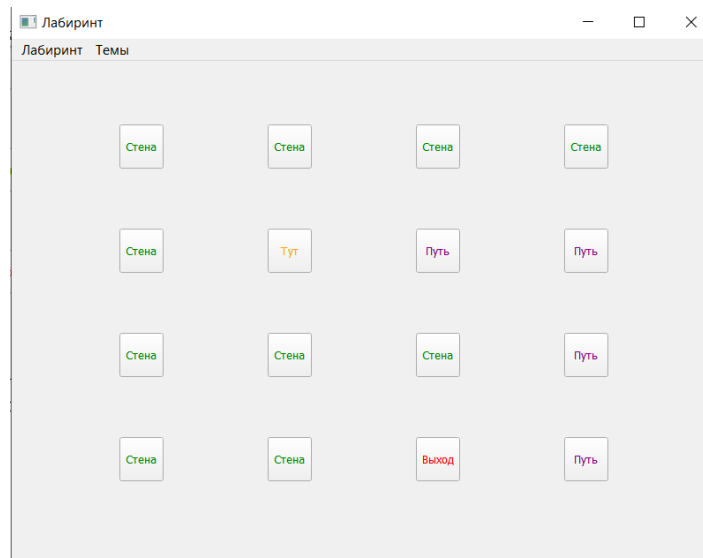


Рисунок 8 – Лабиринт с путем к выходу.

Заключение

В ходе выполнения курсового проекта были достигнуты следующие результаты:

- Изучены и проанализированы исторические аспекты и культурное значение лабиринтов;
- Закреплены теоретические знания по языкам и методам программирования, изученные в курсе;
- Применены полученные знания для решения практической задачи разработки приложения на языке C++;
- Освоены навыки использования среды разработки Qt Creator и фреймворка Qt для создания графического интерфейса.

Список используемых источников

1. Дэвис Стефан Р. С++ для «чайников» / Р. Стефан Дэвис. —4-е изд. Москва: Издательский дом «Вильямс»,2003.—336 с. Текст непосредственный.
2. ШлееМ.Qt5.10. Профессиональное программирование на С++/М. Шлее. —Санкт-Петербург: БХВ-Петербург,2018.—1072с. Текст непосредственный.
3. Алексеев Е. Р. Программирование на языке С++ в среде QtCreator / Е. Р. Алексеев—Москва: ALTLinux,2015—448 с.—Текст непосредственный.
4. QtCreator [Электронный ресурс]: электронный журнал//. Официальная страница QtCreator – Режим доступа: <https://www.Qt.io/product/development-tools> (датаобращения:26.03.2021).
5. Саммерфилд М. Qt. Профессиональное программирование. Разработка кроссплатформенных приложений на С++, - Санкт-Петербург: «Символ-Плюс», 2011 -560с. – Текст: Непосредственный.
6. Страуструп Бьярне. Программирование. Принципы и практика с использованием С++, - Москва: «Вильямс», 2018 – 1328. – Текст непосредственный.
7. Страуструп Бьярне. Программирование. зык программирования С++. Страуструп, - Москва: «Бином», 2022 – 1216. – Текст непосредственный.

data.h

```
#ifndef DATA_H
#define DATA_H
#include <QDialog>
namespace Ui {
class DATA;
}
class DATA : public QDialog
{
    Q_OBJECT
public:
    explicit DATA(QWidget *parent = nullptr);
    ~DATA();

    int Get_row();

    int Get_column();

private:
    Ui::DATA *ui;
};
#endif // DATA_H
```

mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H
#include <QMainWindow>
#include <QDebug>
#include <QPushButton>
#include <QVector>
QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT
public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();
    void btn();
```

```

    bool zap();
    bool findExit(QVector<QVector<int>>&, QVector<QVector<int>>&, int, int);
    int org();
    bool st();
public slots:
    void Clk();
private slots:
    void on_action_triggered();
    void on_action_2_triggered();
    void on_action_3_triggered();
    void on_action_4_triggered();
    void on_action_5_triggered();
private:
    Ui::MainWindow *ui;
    QVector<QPushButton*> butn;
    QVector<int> ct;
    int rows;
    int cols;
    int st_r;
    int st_c;
    int c = 0;
};
#endif // MAINWINDOW_H

start.h
#ifndef START_H
#define START_H

#include <QDialog>

namespace Ui {
class Start;
}

class Start : public QDialog
{
    Q_OBJECT
public:
    explicit Start(QWidget *parent = nullptr);
    ~Start();
    int st_row();
    int st_column();
private:
    Ui::Start *ui;

```

```

};
#endif // START_H

data.cpp
#include "data.h"
#include "ui_data.h"
DATA::DATA(QWidget *parent) :
    QDialog(parent),
    ui(new Ui::DATA)
{
    ui->setupUi(this);
}
int DATA::Get_row(){
    return ui->lineEdit->text().toInt();
}
int DATA::Get_column(){
    return ui->lineEdit_2->text().toInt();
}
DATA::~DATA()
{
    delete ui;
}

```

start.cpp

```

#include "start.h"
#include "ui_start.h"

Start::Start(QWidget *parent) :
    QDialog(parent),
    ui(new Ui::Start)
{
    ui->setupUi(this);
}
int Start::st_row(){
    return ui->lineEdit->text().toInt();
}
int Start::st_column(){
    return ui->lineEdit_2->text().toInt();
}
Start::~Start()
{
    delete ui;
}

```

mainwindow.cpp

```

#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "data.h"
#include "start.h"
#include <QPushButton>
#include <QGridLayout>
#include <QLabel>
#include <QDebug>
#include <iostream>
#include <QStyle>
#include <QSignalMapper>
#include <QMessageBox>
#include <iostream>
using namespace std;
MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);
}
MainWindow::~MainWindow()
{
    delete ui;
}
void MainWindow::btn(){
    QWidget *centralWidget = new QWidget(this);
    QGridLayout *grid = new QGridLayout(centralWidget);
    for(int i = 0; i < rows; ++i) {
        for(int j = 0; j < cols; ++j) {
            QPushButton *button = new QPushButton("  ", this);
            button->setMaximumSize(50,50);
            btn.append(button);
            grid->addWidget(button, i, j);
            connect(button, &QPushButton::clicked, this, &MainWindow::Clk);
        }
    }
    btn[c]->setStyleSheet("color:orange");
    btn[c]->setText("TyT");
    centralWidget->setLayout(grid);
    setCentralWidget(centralWidget);
}
void MainWindow::Clk()

```

```

{
    QPushButton *clickedButton = qobject_cast<QPushButton*>(sender());
    if(clickedButton) {
        int i = btn.indexOf(clickedButton);
        if(i != -1) {
            if(ct[i] == 1){
                clickedButton->setStyleSheet("color:green");
                ct[i] = 0;
                clickedButton->setText("Стена");
                return;
            }
            if(ct[i] == 0){
                clickedButton->setStyleSheet("color:red");
                clickedButton->setText("Выход");
                ct[i] = 2;
                return;
            }
            if(ct[i] == 2){
                clickedButton->setText(" ");
                ct[i] = 1;
                return;
            }
        }
    }
}

bool MainWindow::zap(){
    DATA d;
    d.setWindowIcon(QIcon("data.png"));
    QLabel l1{&d};
    l1.setGeometry(40,40,200,24);
    l1.setText("Ведите кол-во столбцов:");
    QLabel l2{&d};
    l2.setGeometry(40,100,200,24);
    l2.setText("Ведите кол-во строк:");
    if(d.exec() == QDialog::Accepted){
        rows = d.Get_row();
        cols = d.Get_column();
    }
    for (int i = 0; i < (rows * cols); i++) {
        ct.push_back(1);
    }
    if(rows == 0 && cols == 0){return false;}
}

```

```

        else{return true;}
    }

bool MainWindow::st(){
    Start s;
    QIcon icon("start.ico");
    s.setWindowIcon(icon);
    QLabel l1{&s};
    l1.setGeometry(40,30,250,24);
    l1.setText("Ведите начальную позицию.");
    QLabel l2{&s};
    l2.setGeometry(40,80,250,24);
    l2.setText("Ведите строку на которой находитесь.");
    QLabel l3{&s};
    l3.setGeometry(40,130,250,24);
    l3.setText("Ведите столбец на которой находитесь.");
    if(s.exec() == QDialog::Accepted){
        st_r = s.st_row() - 1;
        st_c = s.st_column() - 1;
    }
    if((st_r > rows && st_c > cols) || (st_r < rows && st_c > cols) || (st_r > rows && st_c < cols)){return false;}
    else{
        int ogr = 0;
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                if(st_r == i && st_c == j){
                    ct[c] = 3;
                    ogr = 1;
                    break;
                }
            }
            c++;
        }
        if(ogr == 1){
            break;
        }
    }
    return true;
}

void MainWindow::on_action_triggered()
{
    QVector <QVector<int>> ma(rows);

```

```

QVector <QVector<int>> pt(rows);
c = 0;
for (int i = 0; i < rows; i++) {
    ma[i].resize(cols);
    pt[i].resize(cols);
    for (int j = 0; j < cols; j++) {
        ma[i][j] = ct[c];
        pt[i][j] = 0;
        c++;
    }
}

if (findExit(ma, pt, st_r, st_c)) {
    pt[st_r][st_c] = 3;
    QMessageBox msg;
    msg.setGeometry(800,300,800,800);
    msg.setText("Выход найден!");
    msg.exec();
} else {
    pt[st_r][st_c] = 3;
    QMessageBox msg;
    msg.setGeometry(800,300,800,800);
    msg.setText("Выход не найден!");
    msg.exec();
}

c = 0;
for (int i = 0; i < rows; i++) {
    for (int j = 0; j < cols; j++) {
        ct[c] = pt[i][j];
        if (pt[i][j] == 1 ){
            btn[c]->setStyleSheet("color:purple");
            btn[c]->setText("Путь");
            c++;
        }
        if(pt[i][j] == 0){
            btn[c]->setStyleSheet("color:green");
            btn[c]->setText("Стена");
            c++;
        }
        if(pt[i][j] == 2){
            btn[c]->setStyleSheet("color:red");
            btn[c]->setText("Выход");
            c++;
        }
    }
}

```



```

    }
    if(pt[i][j] == 3){
        butn[c]->setStyleSheet("color:orange");
        butn[c]->setText("Tyт");
        c++;
    }
}
}
c = 1;
}

bool MainWindow::findExit(QVector<QVector<int>>& maze, QVector<QVector<int>>& sol , int row, int
col) {
    if (row < 0 || col < 0 || row >= maze.size() || col >= maze[0].size() || maze[row][col] == 0) {
        return false;
    }
    if (maze[row][col] == 2) {
        sol[row][col] = 2;
        return true;
    }
    maze[row][col] = 0;
    if (findExit(maze, sol, row + 1, col) || findExit(maze, sol, row - 1, col) || findExit(maze, sol, row, col + 1) ||
findExit(maze, sol, row, col - 1)) {
        sol[row][col] = 1;
        return true;
    }
    return false;
}

void MainWindow::on_action_2_triggered()
{
    QPalette darkPalette;
    darkPalette.setColor(QPalette::Window, QColor(53, 53, 53));
    darkPalette.setColor(QPalette::WindowText, Qt::white);
    darkPalette.setColor(QPalette::Base, QColor(25, 25, 25));
    darkPalette.setColor(QPalette::AlternateBase, QColor(53, 53, 53));
    darkPalette.setColor(QPalette::ToolTipBase, Qt::white);
    darkPalette.setColor(QPalette::ToolTipText, Qt::white);
    darkPalette.setColor(QPalette::Text, Qt::white);
    darkPalette.setColor(QPalette::Button, Qt::white);
    for(int i = 0; i < butn.size(); i++){
        if(ct[i] == 1 ){
            if(c == 1){
                butn[i]->setStyleSheet("color:purple");

```

```

        butn[i]->setText("Путь");
    }
    else{
        butn[i]->setStyleSheet("background-color: white");
        butn[i]->setText(" ");
    }
}

if(ct[i] == 0){
    butn[i]->setStyleSheet("background-color: white");
    butn[i]->setStyleSheet("color:green");
    butn[i]->setText("Стена");
}

if(ct[i] == 2){
    butn[i]->setStyleSheet("background-color: white");
    butn[i]->setStyleSheet("color:red");
    butn[i]->setText("Выход");
}

if(ct[i] == 3){
    butn[i]->setStyleSheet("background-color: white");
    butn[i]->setStyleSheet("color:orange");
    butn[i]->setText("Тыт");
}
}

darkPalette.setColor(QPalette::ButtonText, Qt::black);
darkPalette.setColor(QPalette::BrightText, Qt::red);
darkPalette.setColor(QPalette::Link, QColor(42, 130, 218));
darkPalette.setColor(QPalette::Highlight, QColor(42, 130, 218));
darkPalette.setColor(QPalette::HighlightedText, Qt::white);
qApp->setPalette(darkPalette);
}

void MainWindow::on_action_3_triggered()
{
    qApp->setPalette(style()->standardPalette());
    for(int i = 0; i < butn.size(); i++){
        if(ct[i] == 1 ){
            if(c == 1){
                butn[i]->setStyleSheet("color:purple");
                butn[i]->setText("Путь");
            }
            else{
                butn[i]->setStyleSheet("background-color: white");
                butn[i]->setText(" ");
            }
        }
    }
}

```

```

    }
}
if(ct[i] == 0){
    butn[i]->setStyleSheet("background-color: white");
    butn[i]->setStyleSheet("color:green");
    butn[i]->setText("Стена");
}
if(ct[i] == 2){
    butn[i]->setStyleSheet("background-color: white");
    butn[i]->setStyleSheet("color:red");
    butn[i]->setText("Выход");
}
if(ct[i] == 3){
    butn[i]->setStyleSheet("background-color: white");
    butn[i]->setStyleSheet("color:orange");
    butn[i]->setText("Тут");
}
}
}
void MainWindow::on_action_4_triggered()
{
    QPalette orgPalette;
    orgPalette.setColor(QPalette::Window, QColor(255,255,153));
    orgPalette.setColor(QPalette::WindowText, Qt::gray);
    orgPalette.setColor(QPalette::Base, Qt::white);
    orgPalette.setColor(QPalette::AlternateBase, Qt::white);
    orgPalette.setColor(QPalette::ToolTipBase, Qt::white);
    orgPalette.setColor(QPalette::ToolTipText, Qt::black);
    orgPalette.setColor(QPalette::Text, Qt::black);
    orgPalette.setColor(QPalette::Button, Qt::white);
    for(int i = 0; i < butn.size(); i++){
        if (ct[i] == 1 ){
            if (c == 1){
                butn[i]->setStyleSheet("color:purple");
                butn[i]->setText("Путь");
            }
            else{
                butn[i]->setStyleSheet("background-color: white");
                butn[i]->setText(" ");
            }
        }
        if(ct[i] == 0){

```

```

        butn[i]->setStyleSheet("background-color: white");
        butn[i]->setStyleSheet("color:green");
        butn[i]->setText("Стена");
    }
    if(ct[i] == 2){
        butn[i]->setStyleSheet("background-color: white");
        butn[i]->setStyleSheet("color:red");
        butn[i]->setText("Выход");
    }
    if(ct[i] == 3){
        butn[i]->setStyleSheet("background-color: white");
        butn[i]->setStyleSheet("color:orange");
        butn[i]->setText("Тыт");
    }
}

orgPalette.setColor(QPalette::ButtonText, Qt::black);
orgPalette.setColor(QPalette::BrightText, Qt::red);
orgPalette.setColor(QPalette::Link, Qt::white);
orgPalette.setColor(QPalette::Highlight, Qt::white);
orgPalette.setColor(QPalette::HighlightedText, Qt::black);
qApp->setPalette(orgPalette);
}

void MainWindow::on_action_5_triggered()
{
    for(int i = 0; i < butn.size(); i++){
        if(ct[i] == 3){
            butn[i]->setStyleSheet("background-color: white");
            butn[i]->setStyleSheet("color:orange");
            butn[i]->setText("Тыт");
        }
        else{
            butn[i]->setStyleSheet("background-color: white");
            butn[i]->setText(" ");
            ct[i] = 1;
        }
    }
}

```

main.cpp

```

#include "mainwindow.h"
#include <QDebug>
#include <QApplication>
#include <QStyleFactory>

```

```

#include <QMessageBox>
#include <data.h>
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    bool c;
    if(w.zap()){
        while (c == false) {
            c = w.st();
            if(c == false){
                QMessageBox::warning(&w, "Alarm!", "Вы вели не правильное место положение!!!");
            }
        }
        QIcon icon("Labirint.png");
        w.setWindowIcon(icon);
        w.btn();
        w.show();
        qApp->setStyle(QStyleFactory::create("Fusion"));
    }
    return a.exec();
}

```