МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

ОТЧЕТ

по учебной вычислительной (ознакомительной) практике

Льва Марата Вадимовича

студента 1 курса 7 группы

специальность «Прикладная

математика»

Руководитель практики:

старший преподаватель

О.О. Колб

Минск, 2024

# ОГЛАВЛЕНИЕ

[**ОГЛАВЛЕНИЕ 2**](#_gjdgxs)

[**Введение 3**](#_30j0zll)

[**Глава 1 Теоретические основы**](#_1fob9te) **4**

[1.1 Знакомство с компанией SoftClub](#_3znysh7) 4

[1.2 Знакомство с системой контроля версий Git. Курс на EPAM.](#_2et92p0) 4

[1.2.1 Теоретическая часть 4-5](#_3dy6vkm)

[1.2.2 Практическая часть](#_w1n2gq1ozo3k) 6

[1.3 Unit-тестирование 6-7](#_1t3h5sf)

[1.4 Разработка приложения для работы с табличными данными в Qt](#_4bma36ejxnjj) 7

[1.5 Разработка простейшего SDI-приложения](#_x2edghz9nukt) 7-8

[1.6 Вывод к главе 1](#_ts715bue610j) 8

[**Глава 2 Практическая часть**](#_253box8v2xui) **9**

[2.1 Unit-тестирование](#_lmep7qcmj7qu) 9

[2.1.1 Unit-тест на дружественных числах](#_15vmrg6660l5) 9-10

[2.1.2 Unit-тест на подсчете цифр в числе 1](#_sskt3eyxluj0)0-11

[2.2 Разработка GUI-приложения](#_cq53nooti5zj) 11-13

[2.3 Создание приложения для работы с табличными данными 1](#_u69092kcmyuf)3-14

[2.4 Создание текстового редактора 1](#_21m3ore8lnk1)4-15

[2.5 Создание векторного графического редактора 1](#_d1ltq5gz414g)6-17

[2.6 Создание игры 1](#_682pmhclch32)7-19

[2.7 Вывод к главе 2 1](#_b9psb012qva0)9

[**Заключение**](#_fl40o6mdaul) **20**

[**Список использованных источников**](#_4huzsftb5uuv) **21**

# Введение

В период с 24 июня до 6 июля была проведена учебная вычислительная практика. Практика включала в себя ознакомительную и практическую части.

Ознакомительная часть включала в себя изучение системы контроля версий Git и регистрацию на портале GitHub.

Практическая часть задания включала задачи разработки ряда:

- Разработка unit-тестов для консольных приложений в Qt;

- Разработку GUI-приложения;

- Создание приложения для работы с табличными данными;

- Разработку простейшего SDI-приложения (создание текстового редактора);

- Создание векторного графического редактора;

- Создание игры (Разработку проекта с графической анимацией объектов на экране).

Цель данной практики заключалась в расширении и углублении знаний в области разработки кросс-платформенных приложений. Была получена возможность не только закрепить навыки, полученные в течение семестра, но и приобрести новые компетенции, связанные с созданием современных, удобных для пользователя программных продуктов.

Основные задачи включали проектирование архитектуры и интерфейса приложения, разработку механизмов обработки событий и ввода данных, а также реализацию функций сохранения и загрузки информации, в том числе в бинарном формате. Особое внимание уделялось вопросам, позволяющим обеспечить автономность работы готового приложения и возможность его портирования на другие устройства. Параллельно с этим предстояло провести всестороннее тестирование и отладку, гарантирующие надежность и безопасность конечного продукта.

Таким образом, в ходе практики, не только применялись знания, полученные ранее, но и пополнялся арсенал новыми навыками и инструментами, необходимыми для создания современных, функциональных и кроссплатформенных программных решений.

# Глава 1 Теоретические основы

## 1.1 Знакомство с компанией SoftClub

В первый день учебной практики 24.06 была произведена лекционная встреча с представителями компании SoftClub. SoftClub является ведущей продуктовой IT-компанией в разработке программных решений для банков финансовых компаний, электронной торговли, биржевой отрасли и цифровой экономики в Восточной Европе и странах СНГ. Представители компании рассказали о вакансиях таких как DevOps Engineer, Angular-разработчик, SQL Developer и других. Также нам рассказывали о стажировках для студентов. Стажировка длится 3 месяца, за это время специалист сможет принять участие в крупнейших проектах и поучиться у опытных специалистов, готовых поделиться своими знаниями и опытом. В случае, успешного прохождения стажировки компания готова предложить трудоустройство в компанию.

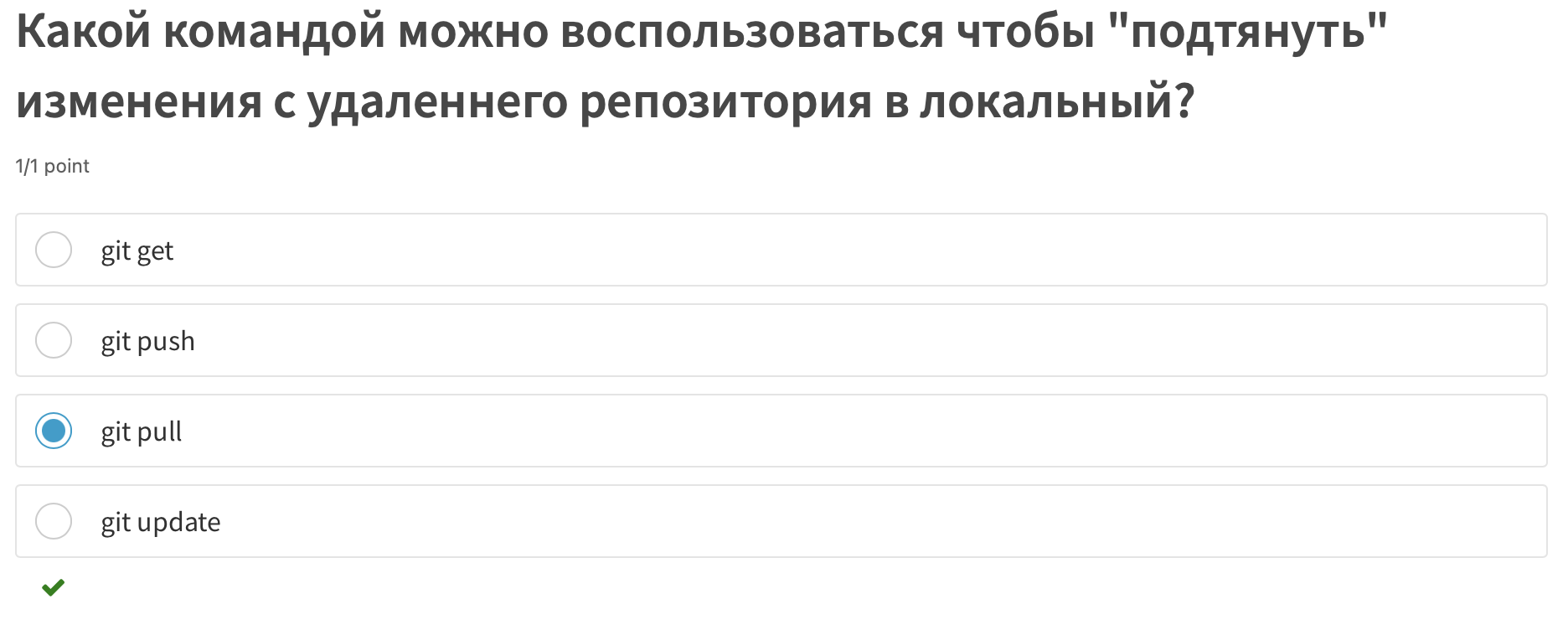
## 1.2 Знакомство с системой контроля версий Git. Курс на EPAM.

Практически каждый современный проект использует системы версионного контроля. Git - самый популярный инструмент для контроля версий. Поэтому на второй день учебной практики был установлен Git(система контроля версий) и началось прохождение курса EPAM Version Control with Git (https://learn.epam.com/detailsPage?id=601f195a-d408-4439-a16d-0630ed2a412e). Курс был рассчитан на 8 часов и пройден в тот же день. Этот курс помогает научиться использовать git для решения ежедневных задач. Курс также помогает для понимания различных стратегий организации ветвления в проекте и выборе оптимальной. Основной целью курса было подготовить фундамент для изучения программирования, эффективного использования современных инструментов разработчика. Курс подразделялся на практическую и теоретическую части.

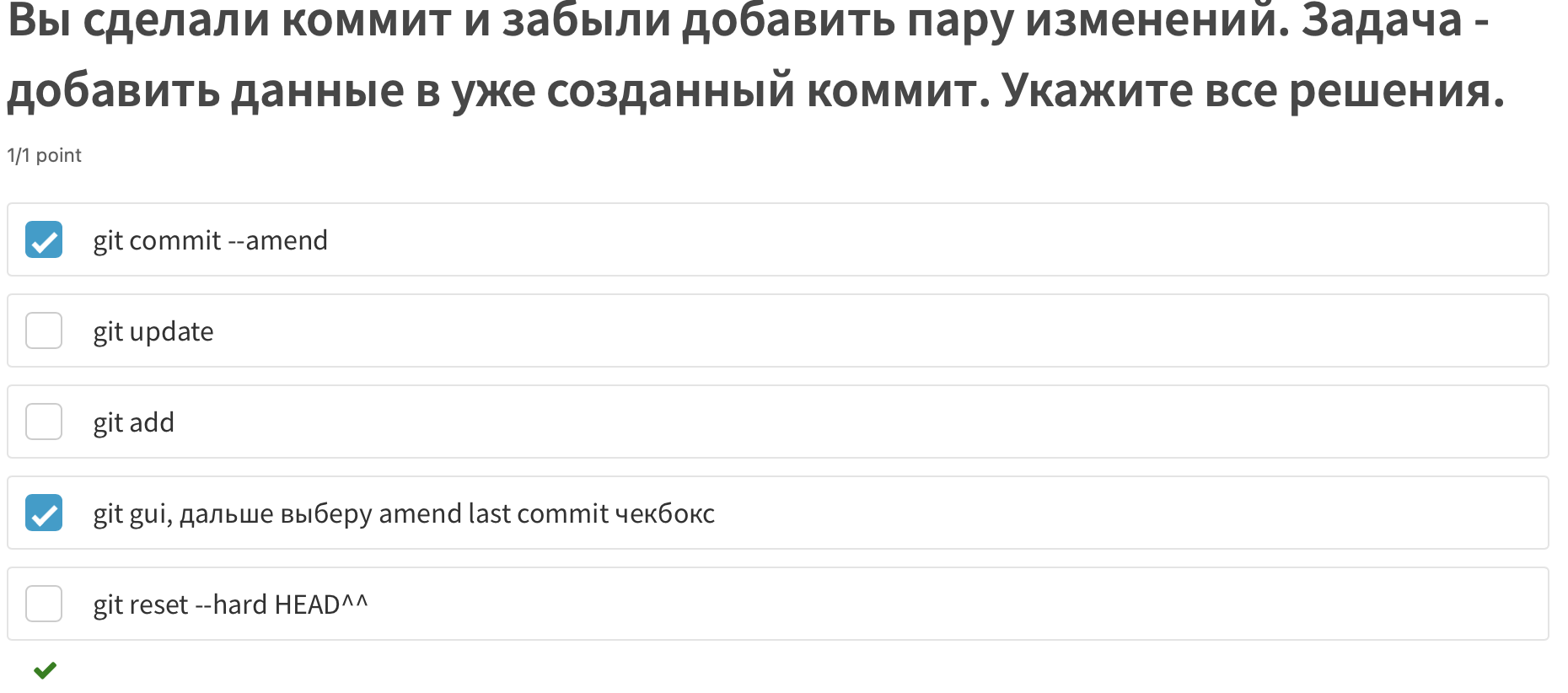
## 1.2.1 Теоретическая часть

Теоретическая часть включала в себя просмотр видео и выполнение тестов. Она состояла из 21 уроков (6 тестов и 15 уроков). Первые уроки были посвящены объяснению концепта Version Control System, установке и настройке Git. Следующий урок был посвящен созданию и клонированию собственного репозитория на GitHub. Следующие уроки были посвящены освоению базовых команд таких как git add - добавление изменений в рабочую директорию, git status - проверка статуса репозитория, git commit -m - создание нового коммита, git log - просмотр истории коммитов, git commit –amend -m - отмена последнего коммита, git revert HEAD - откат последнего коммита, git branch new\_branch\_name - создание новой ветки, git checkout -b branch\_name - переключение между ветками, git merge - слияние веток, git push - отправка изменений в удаленный репозиторий, git pull - получение изменений из удаленного репозитория и другие.

Примеры заданий из тестов.



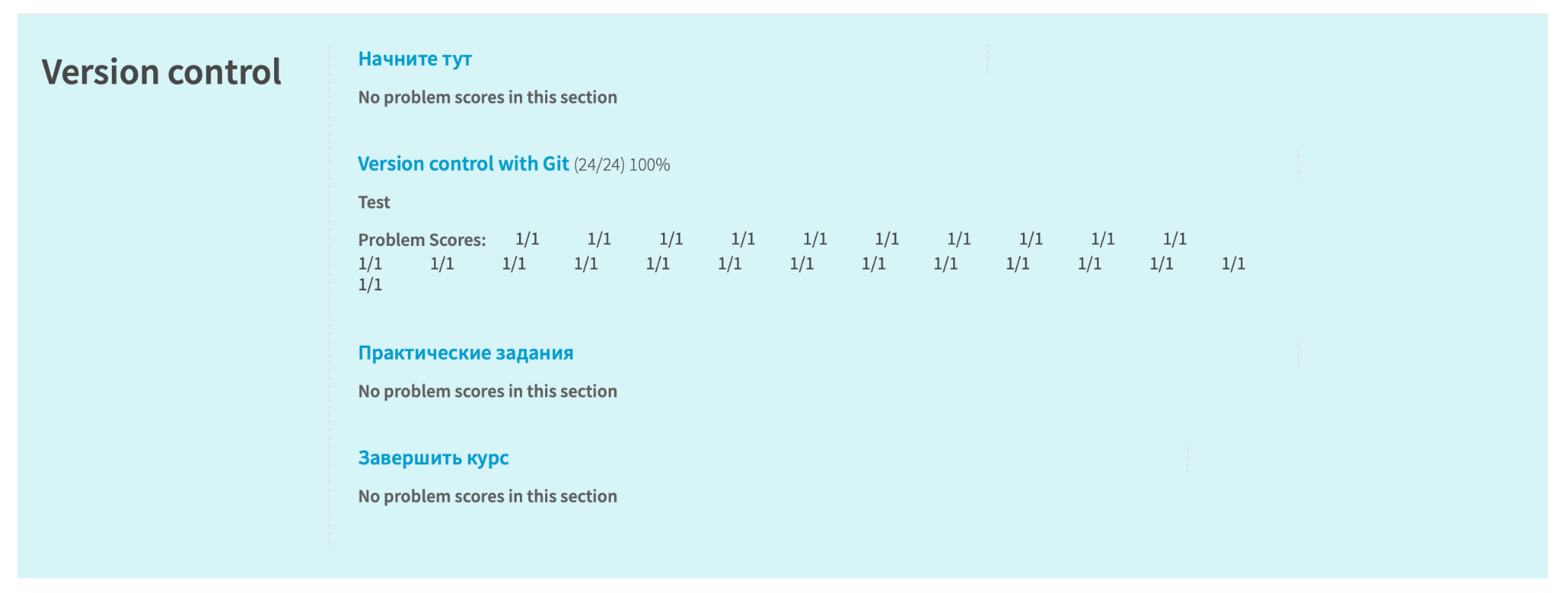
*Рисунок 1 - тестовое задание на знание команды git pull*

*Рисунок 2 - тестовое задание на изменение коммита*

## 1.2.2 Практическая часть

Практическая часть состояла из 5 заданий. Благодаря хорошему изложению теоретической части задания показались интересными и не такими уж трудными, как казалось сначала. В ходе выполнения практической части курса были применены на практике те знания, которые накапливались по ходу курса, освоены навыки работы с системой контроля версий Git, получены практические навыки работы с основными командами Git(add, commit, push, pull, branch, merge), а также освоена практика работы с удаленными репозиториями.

Ссылка на репозиторий GitHub с прохождением практической части:  
<https://github.com/MaratLeu/git-demo.git>



*Рисунок 3 - сертификат об окончании курса*

## 1.3 Unit-тестирование

Unit-тестирование, или модульное - это разновидность тестирования в программной разработке, которое заключается в проверке работоспособности отдельных функциональных модулей, процессов или частей кода приложения. Unit-тестирование позволяет избежать ошибок или быстро исправить их при обновлении или дополнении ПО новыми компонентами, не тратя времени на проверку программного обеспечения целиком.

Главная причина написания юнит-тестов — тестирование отдельных модулей. Поскольку каждый модуль пишется отдельно, тестировать его тоже можно изолированно, без связки с другими. Получается простая схема: программист написал модуль → протестировал его → продолжил разработку для связи с другими модулями и других тестов.

Если пропустить этап юнит-тестирования, в следующий раз не получится понять, что именно вызвало ошибку: какой-то из модулей или неправильно настроенная интеграция между ними. Придётся разбираться, тратить время и всё равно тестировать отдельные юниты.

На занятиях 26.06 было предложено написание unit-тестов в Qt. Среда разработки QtCreator включает в себя инструменты для работы с юнит-тестами.

## 1.4 Разработка приложения для работы с табличными данными в Qt

На занятиях 27.06 было предложено разработать приложение, отражающее табличные данные с возможностью корректировки (на основе шаблона Модель-Вид), вывода и ввода данных.

Основной задачей было создать приложение, способное считывать данные из файла с электронными таблицами и отображать их в виджете QTableView, а также сохранять внесенные изменения обратно в файл с электронными таблицами.

Алгоритм создания приложения:

1. Загрузка данных из файла с электронными таблицами: для этого можно воспользоваться библиотекой Qt, такой как QXlsx, которая представляет удобные методы для чтения и записи файлов с электронными таблицами.

Данные из файла будут загружены в модель QStandartItemModel.

1. Отображение данных в QTableView: виджет QTableView будет использоваться для отображения табличных данных. Модель данных будет передана в QTableView для отображения табличных данных, обеспечивая связь между данными и их визуальным представлением.
2. Редактирование данных: пользователь сможет редактировать данные в виджете QTableView. Изменения будут автоматически отображаться в модели данных.
3. Сохранение данных в файл с электронными таблицами: после внесения изменений в виджете QTableView пользователь может сохранить

изменения в файл с электронными таблицами.

## 1.5 Разработка простейшего SDI-приложения

На занятиях 28-29 июня было предложено разработать простейшее SDI-приложение, иллюстрирующее использование меню, корректировку, сохранение и восстановление текстовой, числовой информации в текстовом (QTextStream) и бинарном(QDataStream) формате.

Алгоритм создания приложения:

1. Cоздание класса DocWindow: унаследованного от класса TextEdit, представляющее собой окно для редактирования. Создание сигнала changeWindowTitle() предназначенная для того, чтобы информировать о том, что текстовая область была изменена. Создание слотов slotLoad(), slotSave(), slotSaveAs() необходимы для проведения операция чтения и записи файлов.
2. Создание класса SDIProgram: унаследованного от класса QMainWindow. В его конструкторе создаются 2 виджета: всплывающее меню File, Help
3. Чтение и запись файлов.

## 1.6 Вывод к главе 1

В ходе работы над данным разделом отчета были изучены и освоены следующие ключевые аспекты:

1. Знакомство с компанией SoftClub, ее деятельностью и направлением работы. Это дало общее понимание контекста в котором проводилась практическая часть стажировки.
2. Изучены системы контроля версий Git. В теоретической части рассмотрены основные концепции и принципы работы с Git, такие как репозиторий, ветвление, слияние. Практическая часть включала выполнение упражений и заданий.
3. Освоение методики unit-тестирования.
4. Разработка приложений для работы с табличными данными с использованием Qt.
5. Создание простейшего SDI-приложения

# Глава 2 Практическая часть

## 2.1 Unit-тестирование

[Ссылка](#_1t3h5sf) на теоретическую часть по unit-тестам

## 2.1.1 Unit-тест на дружественных числах

В рамках занятия 26.06 надо было продемонстрировать умение работы с unit-тестированием. Для этого было предложено написать unit-тесты для своих лабораторных. Одной из лабораторных работ является лабораторная на проверку являются ли числа дружественными. Два числа называются дружественными если сумма делителей первого числа(кроме себя самого) равна второму числу, и наоборот сумма делителей второго числа равна первому числу.

Был создан класс Friends, в котором была реализована функция bool isFriends() для проверки чисел на дружественность

**bool** isFriends() {

**int** sum1 = 0, sum2 = 0;

**for** (**int** i = 1; i < a; i++) {

**if** (a % i == 0)

sum1 += i;

**else**

sum1 += 0;

}

**for** (**int** k = 1; k < b; k++) {

**if** (b % k == 0)

sum2 += k;

**else**

sum2 += 0;

}

**if** ((sum2 == a) && (sum1 == b) && (a != b)) {

**return** **true**;

}

**return** **false**;

}

Для самих unit-тестов был создан файл tst\_test\_friends.cpp, к этому файлу мы подключаем наш файл #include "Friends.h" , в котором реализован класс Friends с функцией реализованной выше.

**void** Test\_Friends::test\_Friends\_data() {

QTest::addColumn<**int**>("m\_a");

QTest::addColumn<**int**>("m\_b");

QTest::addColumn<**bool**>("res");

QTest::newRow("220 и 284") << 220 << 284 << **true**;

QTest::newRow("10 и 9") << 10 << 9 << **false**;

QTest::newRow("22 и 24") << 22 << 24 << **false**;

QTest::newRow("1184 и 1210") << 1184 << 1210 << **true**;

}

**void** Test\_Friends::test\_Friends()

{

QFETCH(**int**, m\_a);

QFETCH(**int**, m\_b);

QFETCH(**bool**, res);

mCalc.SetA(m\_a);

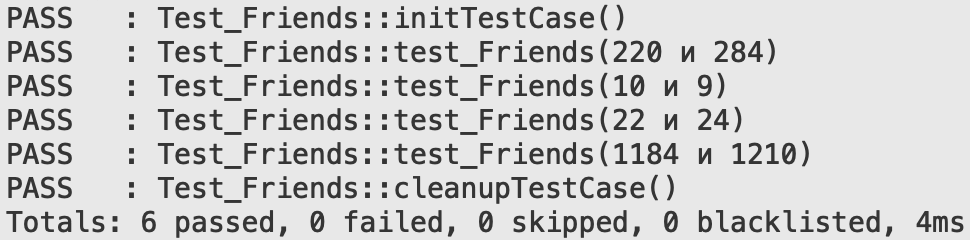
mCalc.SetB(m\_b);

QCOMPARE(mCalc.isFriends(), res);

}

QTEST\_APPLESS\_MAIN(Test\_Friends)

#include "tst\_test\_friends.moc"



*Рисунок 4 - прохождение unit-тестов*

## 2.1.2 Unit-тест на подсчете цифр в числе

В качестве второго примера для проверки умения работы с unit-тестами была выбрана, лабораторная, в которой требуется найти повторение каждой цифры в числе.

Для этого был создан класс Сount.h в котором была реализована функция vector<**int**> count(), возвращающий вектор, содержащий повторение каждой цифры в числе

vector<**int**> count() {

vector<**int**> mas(10, 0);

**while** (num != 0) {

**int** digit = num % 10;

mas[digit]++;

num /= 10;

}

**return** mas;

}

Также для unit-тестов был создан файл tst\_test\_count.cpp, к которому подключается файл #include "Count.h" в котором реализован класс Count с функцией которая была реализована выше.

**void** TestCount::test\_Count\_data() {

QTest::addColumn<**int**>("num");

QTest::addColumn<QVector<**int**>>("mas");

QVector<**int**> mas1 = {0, 0, 0, 1, 0, 1, 0, 1, 0, 0};

QVector<**int**> mas2 = {0, 0, 1, 2, 1, 0, 0, 1, 1, 0};

QVector<**int**> mas3 = {1, 0, 1, 1, 0, 0, 2, 1, 1, 0};

QVector<**int**> mas4 = {0, 0, 3, 0, 0, 0, 0, 0, 0, 0};

QTest::newRow("573") << 573 << mas1;

QTest::newRow("374823") << 374823 << mas2;

QTest::newRow("3762806") << 3762806 << mas3;

QTest::newRow("222") << 222 << mas4;

}

**void** TestCount::test\_Count() {

QFETCH(**int**, num);

QFETCH(QVector<**int**>, mas);

mCalc.SetNum(num);

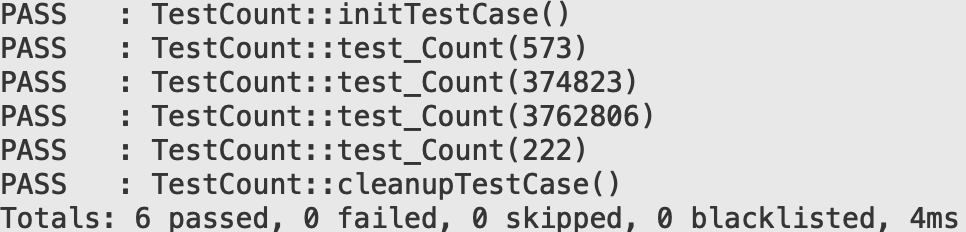
vector<**int**> masStd(mas.begin(), mas.end());

QCOMPARE(mCalc.count(), masStd);

}

QTEST\_APPLESS\_MAIN(TestCount)

#include "tst\_test\_count.moc"



*Рисунок 5 - прохождение unit-тестов*

## 2.2 Разработка GUI-приложения

В рамках разработки GUI-приложения было предложено разработать класс предназначенный для осуществления операций с датами. Причем в качестве приватных значений в этом классе должны использоваться компоненты день, месяц, год. Используя абстрактный тип данных Date и используя программное получить программное средство, которое позволяет по двум датам(дате рождения и по текущеей дате или по определённой пользователем дате) построить и вывести на экран график биоритмов. Биоритмы показывают Ваше состояние в "идеальном" случае, без учета конкретных случайных факторов.

Вы можете рассматривать графики как усредненное состояние и сопоставлять с вашим прошлым и текущим состоянием.

MainWindow.cpp (Отрывок кода для отрисовки графика)

**void** MainWindow::paintEvent(QPaintEvent\* ) {

QPainter painter;

painter.begin(**this**);

painter.setRenderHint(QPainter::Antialiasing, **true**);

painter.translate(width()/2,height()/2 + 50);

painter.drawLine(-width(),0,width(),0);

QPoint point;

painter.save();

**for**(**double** j = -width()/60, i = -width() / 60; i < width() / 60; i += 0.01) {

j += 0.01;

point.setX(j\*30);

QPen linepen(Qt::red);

linepen.setWidth(3);

painter.setPen(linepen);

point.setY(sin(-(i+days)\*2\*M\_PI/23)\*100);

painter.drawPoint(point);

linepen.setColor(Qt::blue);

painter.setPen(linepen);

point.setY(sin(-(i+days)\*2\*M\_PI/28)\*100);

painter.drawPoint(point);

linepen.setColor(Qt::green);

painter.setPen(linepen);

point.setY(sin(-(i+days)\*2\*M\_PI/33)\*100);

painter.drawPoint(point); }

painter.restore();

**for** (**int** j=-width()/60; j <= width()/60; j++) {

**if**(j == 0) {

painter.save();

QPen linepen(Qt::black);

linepen.setWidth(3);

painter.setPen(linepen);

QFont font = painter.font();

font.setPointSize(10);

painter.setFont(font);

painter.drawLine(j\*30,-20, j\*30,20);

painter.rotate(90);

QString date= ui->dateEdit->date().addDays(-j).toString();

painter.drawText(10, j\*30, date);

painter.restore(); }

**else** {

painter.drawLine(j\*30,-10, j\*30,10);

painter.save();

painter.rotate(90);

QString date= ui->dateEdit->date().addDays(-j).toString();

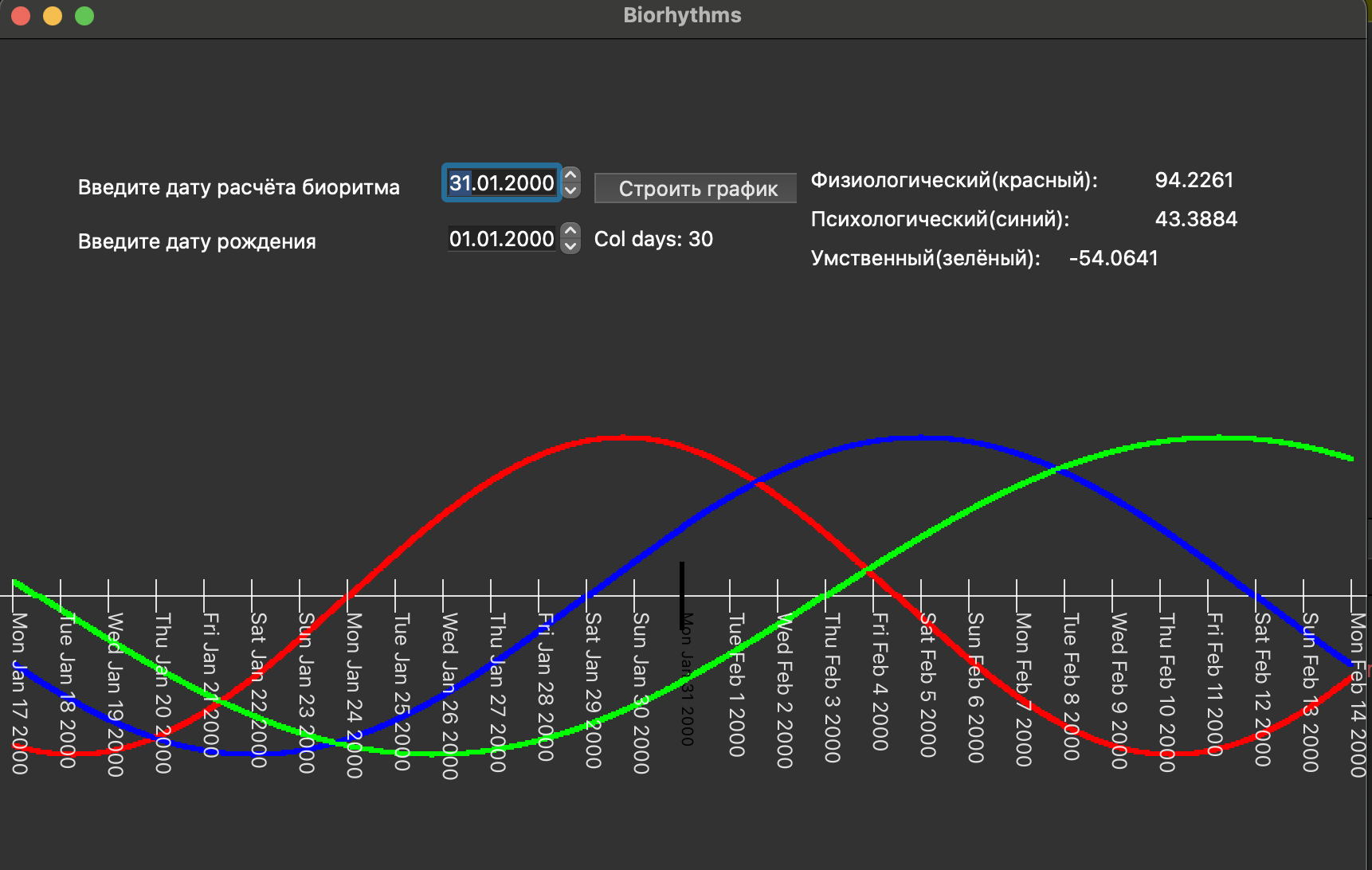
painter.drawText(10, j\*30, date);

painter.restore();

}

}

}



*Рисунок 6 - график биоритмов*

## 2.3 Создание приложения для работы с табличными данными

На занятиях 27.06 было предложено разработать приложение, отражающее табличные данные с возможностью корректировки (на основе шаблона Модель-Вид), вывода и ввода данных.

[Алгоритм](#_4bma36ejxnjj) создания приложения приведен в теоретической части

MainWindow.cpp (Отрывок кода для чтения табличных данных)

MainWindow::MainWindow(QWidget \*parent) :

QMainWindow(parent),

ui(**new** Ui::MainWindow) {

ui->setupUi(**this**);

csvModel = **new** QStandardItemModel(**this**);

ui->tableView->setModel(csvModel);

ui -> tableView -> setEditTriggers(QAbstractItemView::DoubleClicked);

QString filepath = QDir::currentPath();

QFile file(filepath + "/work.csv");

QFile file\_2("work\_2.csv");

**if** ( !file.open(QIODevice::ReadOnly | QFile::Text) || !file\_2.open(QIODevice::WriteOnly | QFile::Text)) {

qDebug() << "File not exists";

} **else** {

QTextStream in(&file);

QTextStream out(&file\_2);

**while** (!in.atEnd()) {

QString line = in.readLine();

QList<QStandardItem \*> standardItemsList;

**for** (QString item : line.split(";")) {

standardItemsList.append(**new** QStandardItem(item));

}

csvModel->insertRow(csvModel->rowCount(), standardItemsList);

out << standardItemsList; }}}



*Рисунок 7 - чтение данных из excel-файла в виджет QTableView*

## 2.4 Создание текстового редактора

На занятиях 28-29 июня было предложено разработать простейшее SDI-приложение, иллюстрирующее использование меню, корректировку, сохранение и восстановление текстовой, числовой информации в текстовом (QTextStream) и бинарном(QDataStream) формате.

[Алгоритм](#_x2edghz9nukt) создания приложения приведен в теоретической части

DocWindow.cpp (Отрывок кода для сохранения и загрузки в текстовом и бинарном виде)

**void** DocWindow::slotSave() {

**if** (m\_strFilename.isEmpty()) {

slotSaveAs();

**return**;

}

QFile file(m\_strFilename);

**if** (file.open(QIODevice::WriteOnly)) {

**if** (m\_strFilename.contains(".txt")) {

QTextStream(&file) << toPlainText();

}

**else** **if** (m\_strFilename.contains(".bin")) {

QByteArray data = toPlainText().toUtf8();

QDataStream fin(&file);

fin << data;

}

file.close();

emit changeWindowTitle(m\_strFilename);

}

}

**void** DocWindow::slotLoad()

{

QString str = QFileDialog::getOpenFileName(**this**, tr("Open file"), QDir::currentPath(), tr("\*txt; \*bin"));

**if** (str.isEmpty()) {

**return**;

}

QFile file(str);

**if** (file.open(QIODevice::ReadOnly)) {

**if** (str.contains(".txt")) {

QTextStream stream(&file);

setPlainText(stream.readAll());

}

**else** **if** (str.contains(".bin")) {

QByteArray bytes = file.readAll();

QString hexString = QString::fromUtf8(bytes);

qDebug() << hexString << "\n";

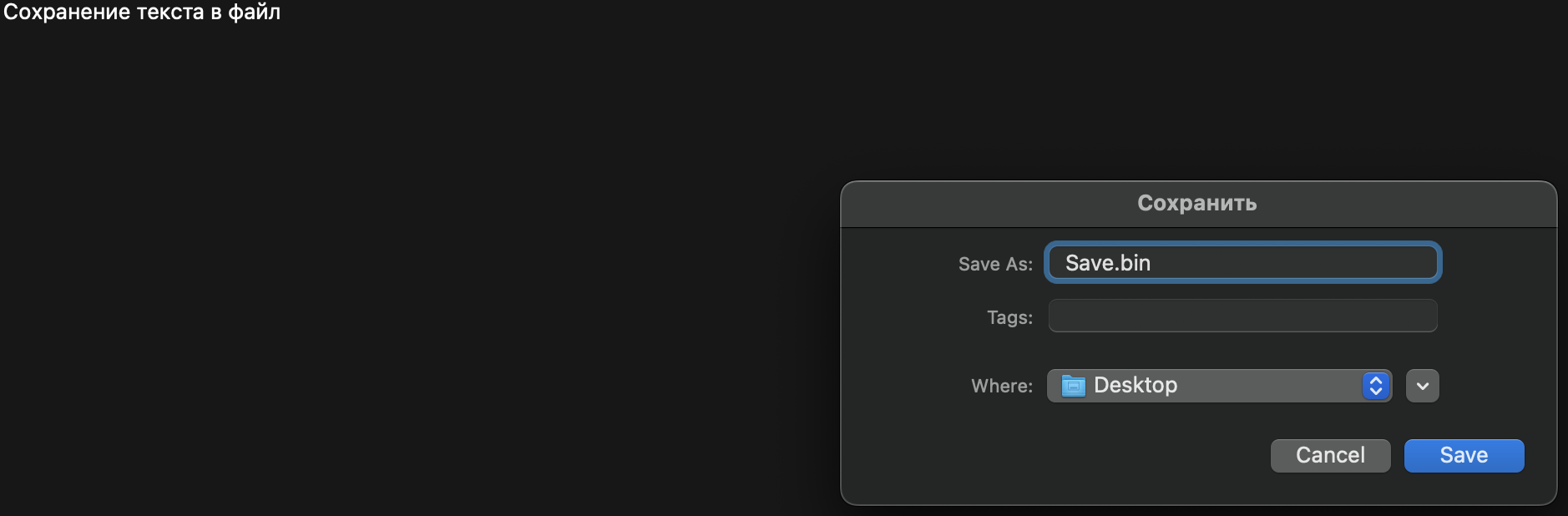
setPlainText(hexString);

}

file.close();

m\_strFilename = str;

emit changeWindowTitle(m\_strFilename); }}

*Рисунок 8 - сохранение текста в файл*

## 2.5 Создание векторного графического редактора

На занятиях 1-2 июля была предложена разработка приложения, аналога Векторного графического редактора, с обработкой событий от мыши, сериализацией набора графических объектов в файлах.

Алгоритм создания приложения:

1. Создание абстрактного класса Shape унаследованного от класса QGraphicsItem, который будет содержать такие параметры как позиция, ширина, высота фигуры.
2. Создание классов Rectangle (прямоугольник), Oval (овал), Rhombus (ромб), Line(линия) унаследованных от класса Shape для отрисовки различных фигур.
3. Создание класса paintScene унаследованного от класса QGraphicsScene, который будет содержать перечисление enum в зависимости от режима выбора фигуры (Drawing\_Rectangle, Drawing\_Oval, Drawing\_Rhombus, Drawing\_Line, Drawing\_None), в этом классе будут реализованы обработка событий от мыши mousePressEvent (при нажатии), mouseMoveEvent (при перемещении), mouseReleaseEvent(отпускание мыши). Также здесь будут реализованы методы сериализации и десереализации для работы с графическими объектами.

*paintscene.cpp (сериализация сцены в формат JSON, десереализация из файла JSON обратно в сцену)*

**void** paintScene::serializeToJson(QJsonObject& json) **const** {

QJsonArray shapesArray;

**for** (**const** **auto**& shape : items()) {

QJsonObject shapeJson;

**if** (**auto** rectangle = qgraphicsitem\_cast<Rectangle\*>(shape)) {

rectangle->serializeToJson(shapeJson);

} **else** **if** (**auto** oval = qgraphicsitem\_cast<Oval\*>(shape)) {

oval->serializeToJson(shapeJson);

} **else** **if** (**auto** rhombus = qgraphicsitem\_cast<Rhombus\*>(shape)) {

rhombus->serializeToJson(shapeJson);

} **else** **if** (**auto** line = qgraphicsitem\_cast<Line\*>(shape)) {

line->serializeToJson(shapeJson);

}

shapesArray.append(shapeJson);

}

json["shapes"] = shapesArray;

json["penColor"] = penColor().name();

json["drawingMode"] = **static\_cast**<**int**>(drawingMode);

}

**void** paintScene::deserializeFromJson(**const** QJsonObject& json) {

clear();

QJsonArray shapesArray = json["shapes"].toArray();

**for** (**const** **auto**& shapeValue : shapesArray) {

QJsonObject shapeJson = shapeValue.toObject();

QString type = shapeJson["type"].toString();

Shape\* shape = **nullptr**;

qDebug() << "Shape type:" << type;

**if** (type == "rectangle") {

shape = **new** Rectangle;

} **else** **if** (type == "oval") {

shape = **new** Oval;

} **else** **if** (type == "rhombus") {

shape = **new** Rhombus;

} **else** **if** (type == "line") {

shape = **new** Line;

}

**if** (shape) {

shape->deserializeFromJson(shapeJson);

addItem(shape);

} **else** {

qWarning() << "Unknown shape type:" << type;

}

}

**if** (json.contains("penColor")) {

setPenColor(QColor(json["penColor"].toString()));

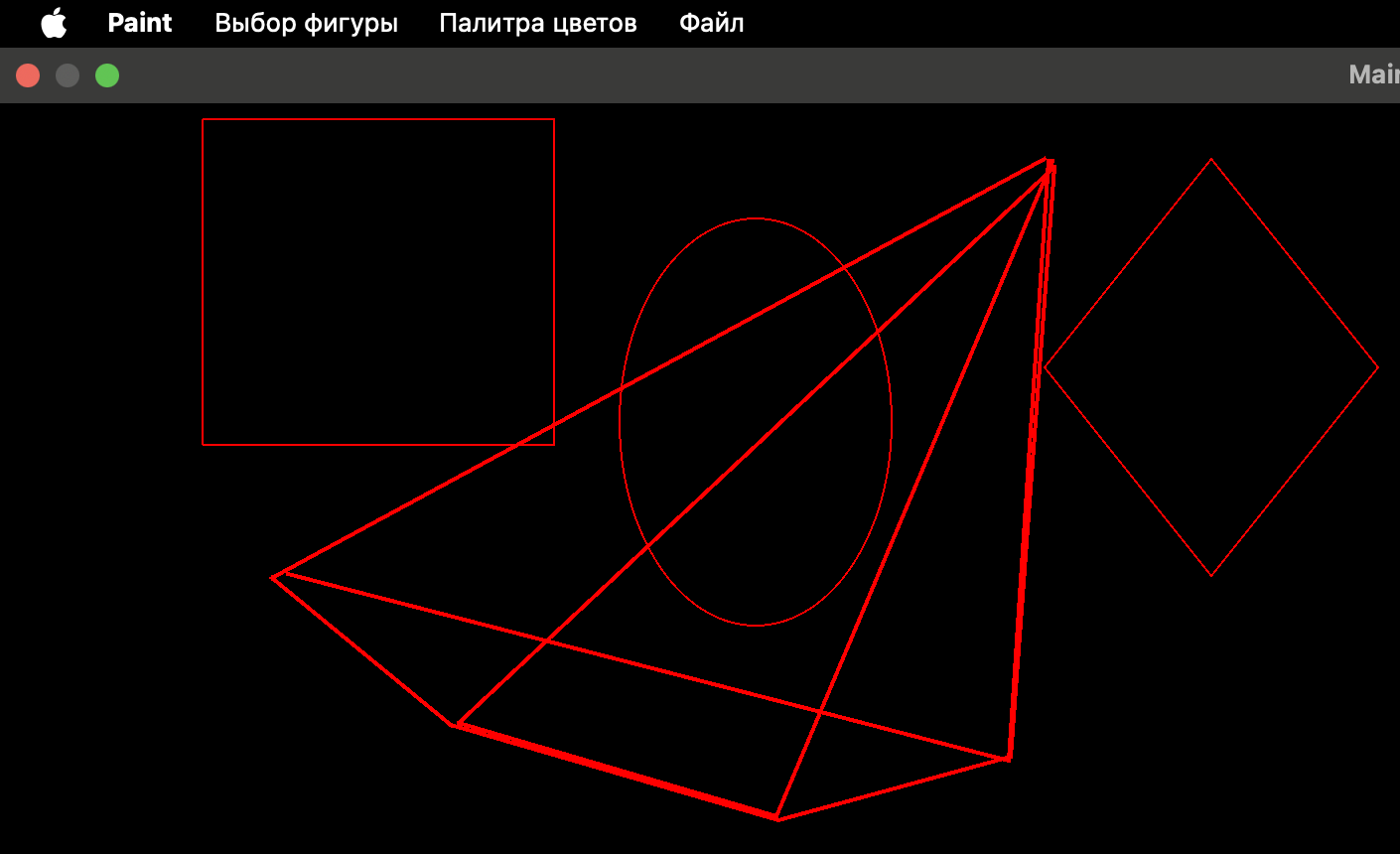
}

**if** (json.contains("drawingMode")) {

drawingMode = **static\_cast**<DrawingMode>(json["drawingMode"].toInt());

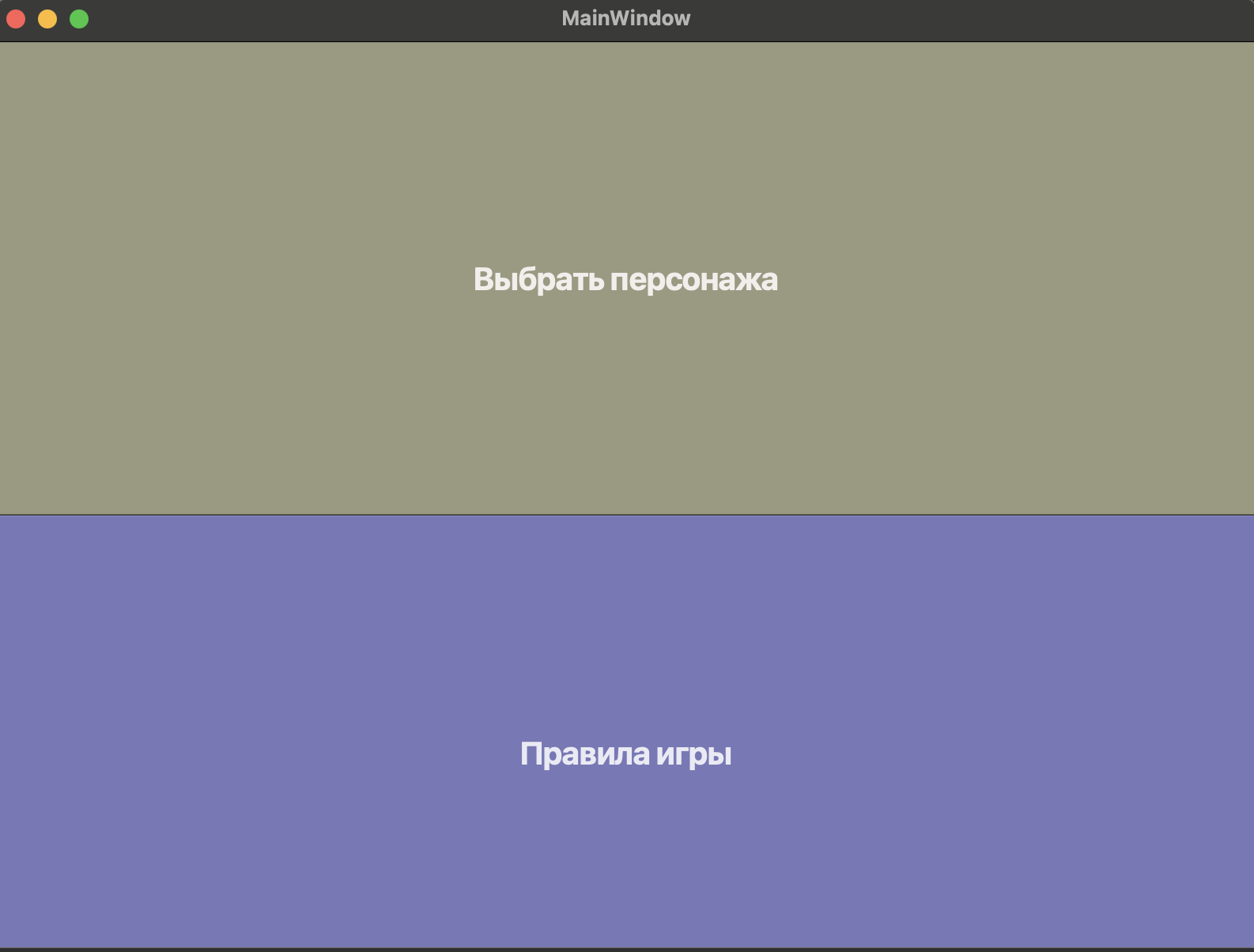
}

}

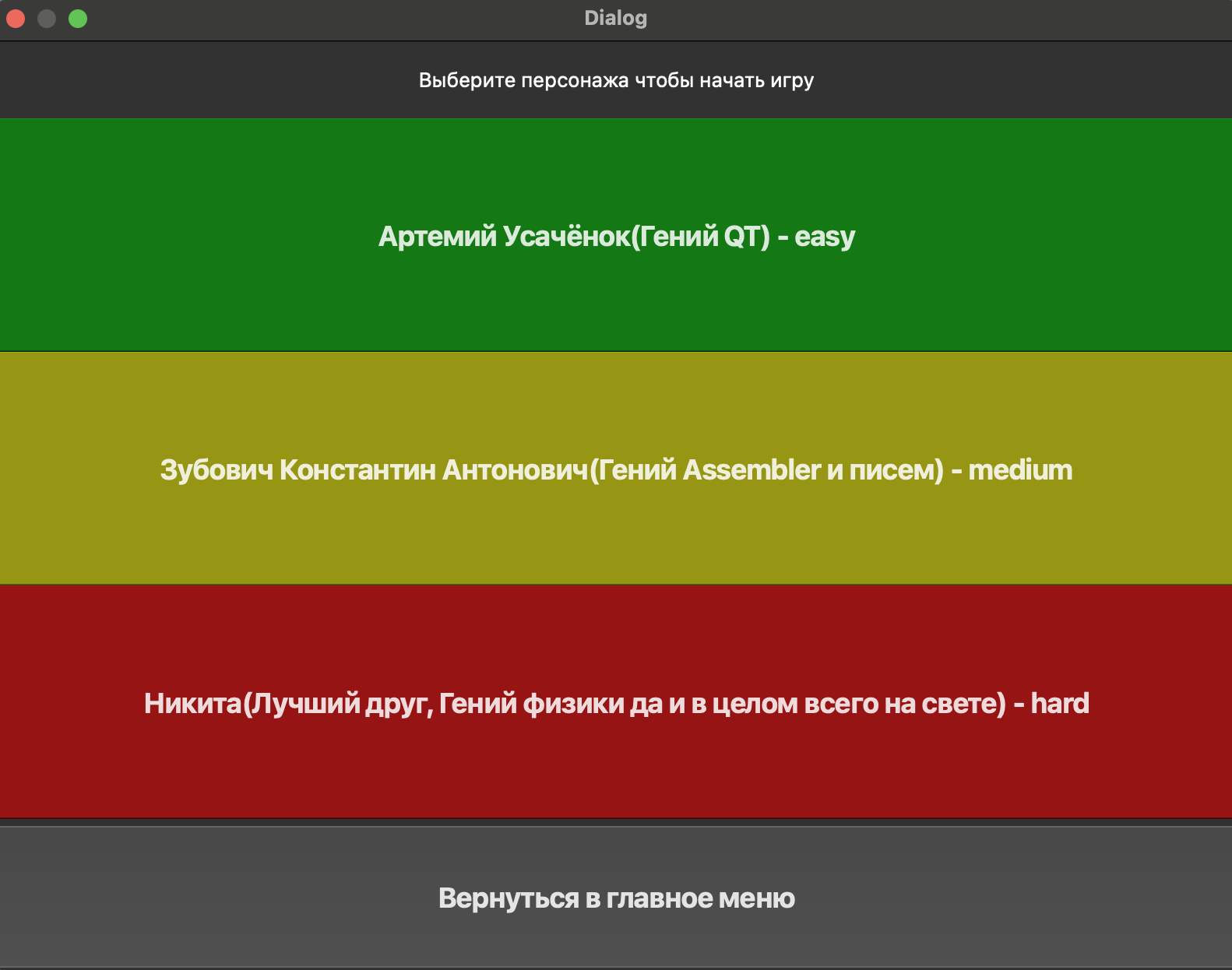


*Рисунок 9 - прорисовка фигур (прямоугольник, овал, ромб, линия) на экране*

## 2.6 Создание игры На предпоследнем занятии 05.07 требовалось получить программу чучело. Ее суть заключается в том что изначально на экране появляется фотография какого либо персонажа, а также на экране вместо курсора бейсбольная бита. Задача в том чтобы управляя мышкой пытаться догнать фотографию и стукнуть по фотографии битой нажимая левую клавишу мышки, при этом фото быстро убегает, гонка сопровождается музыкой(любимой музыкой), в том случае когда удается битой попасть по фото, на фото появляются синяки и т.д.,удар сопровождается соответствующим звуком (по типу ура вау). У фотографии есть три жизни. когда третий раз осуществляется попадание , на экране появляется катафалк и под похоронную музыку уезжает за границу экране. Также необходимо чтобы можно было использовать различные фото персонажей, желательно чтобы эти параметры задавались в меню.



*Рисунок 10 - главное меню*



*Рисунок 11 - выбор персонажа*

**

*Рисунок 12 - картинка бегающая по экрану*

## 2.7 Вывод к главе 2

В ходе работы над данным разделом отчета были изучены и освоены следующие ключевые аспекты:

1. Разработка unit-тестов для консольных приложений. Unit-тесты позволили выявить и оперативно устранить ошибки на ранних этапах разработки, когда их исправление требует меньших трудозатрат.
2. Разработка GUI - приложения. Создание такого приложения дает визуальное представление данных и результатов в графической форме, которое способствует быстрому восприятию и анализу информации.
3. Создание приложения для работы с табличными данными. Одним из преимуществ такого приложения является возможность выполнять операции с данными в табличном формате быстрее и проще, чем в общих программах таких как электронные таблицы.
4. Разработка простейшего SDI - приложения (создание текстового редактора). Важным преимуществом его является простой и интуитивно понятный пользовательский интерфейс, а именно SDI - редактор имеет единственное главное окно для работы с текстовыми документами, что создает более понятную и удобную структуру по сравнению с многооконными (MDI) текстовыми редакторами.
5. Создание векторного графического редактора. В контексте разработки этого приложения помимо создания самого редактора немаловажную роль играют сериализация и десериализация графических проектов. Эффективная реализация сериализации и десериализации является ключевой для обеспечения возможности сохранения и загрузки графических проектов в векторном редакторе.
6. Разработку проекта с графической анимацией объектов на экране. Создание такого проекта предоставляет динамичное и интерактивное взаимодействие, повышает визуальную привлекательность проекта.

# Заключение

Проделанная работа в рамках учебной практики продемонстрировала успешное освоение и применение на практике ключевых методик и инструментов разработки современных кроссплатформенных приложений с использованием фреймворка Qt.

Были реализованы:

1. Консольные приложения с применением unit-тестирования, позволившие выявлять и оперативно устранять ошибки на ранних этапах разработки
2. GUI-приложения, обеспечивающие наглядное графическое представление данных для быстрого восприятия информации пользователями
3. Приложение для работы с табличными данными, предоставляющее более эффективные возможности по сравнению с общими программами, такими как электронные таблицы
4. Простейший SDI-текстовый редактор с интуитивно понятным и простым пользовательским интерфейсом
5. Векторный графический редактор, где ключевую роль играет эффективная сериализация и десериализация графических проектов
6. Проект с графической анимацией объектов на экране, предоставляющий динамичное и интерактивное взаимодействие, повышающее визуальную привлекательность
7. Выполненные практические задания наглядно продемонстрировали освоение широкого спектра современных подходов и инструментов для разработки высококачественных кроссплатформенных приложений с использованием Qt. Полученные навыки будут ценны при создании дальнейших проектов.
8. Дополнительно было создано приложение “Хэш-таблица”.

Выполненные практические задания наглядно продемонстрировали освоение широкого спектра современных подходов и инструментов для разработки высококачественных кроссплатформенных приложений с использованием Qt. Полученные навыки будут ценны при создании дальнейших проектов.

## 

# Список использованных источников

1. "Qt Documentation." Qt. Режим доступа: <https://doc.qt.io/>. Дата доступа – 06.07.2024.
2. Qt Forum. Режим доступа:<https://forum.qt.io/>. Дата доступа – 25.06.2024. (Сайт целиком)
3. EPAM Learn. Режим доступа: <https://elearn.epam.com/courses/>. Дата доступа – 24.06.2024. (Статья с сайта)
4. Образовательный портал ФПМИ БГУ. Режим доступа: <https://edufpmi.bsu.by/>. Дата доступа – 26.06.2024. (Сайт целиком)
5. "Qt Project." GitHub. Режим доступа: <https://github.com/qt>. Дата доступа – 01.07.2024. (Сайт целиком)
6. Макс Шлее. Qt 4.8. Профессиональное программирование на C++. – БХВ-Петербург, 2012. – 462 c. (Книга с одним автором)
7. Version Control with Git [Электронный ресурс]. - Режим дотупа: <https://learn.epam.com/detailsPage?id=601f195a-d408-4439-a16d-0630ed2a412e> . – Дата доступа: 26.06.2024.
8. METANIT.COM - Сайт о программировании. Режим доступа: <https://metanit.com/>. - Дата доступа: 04.07.2024. (Сайт целиком)
9. Марк Саммерфилд - Qt Профессиональное программирование (High tech), 2011 – 1192 c. (Книга с одним автором)
10. Скотт Чакон и Бен Страуб - Pro Git (Apress), 2009 - 458 с. (Книга с двумя авторами)