Министерство образования Республики Беларусь

Учреждение образования

БелорусскиЙ государственный университет

информатики и радиоэлектроники

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему

**Программное средство «ИГРОВОЙ ЦЕНТР»**

|  |  |  |
| --- | --- | --- |
| Студент |  | И. В. Ермолович |
| Руководитель |  | Д.С. Шулицкий |

Минск 2019

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 5](#_Toc28040420)

[1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 7](#_Toc28040421)

[1.1 Обзор аналогов 7](#_Toc28040422)

[1.2 Постановка задачи 10](#_Toc28040423)

[2 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА 11](#_Toc28040424)

[2.1 Структура программы 11](#_Toc28040425)

[2.2 Интерфейс программного средства 11](#_Toc28040426)

[2.3 Разработка алгоритмов программного средства 15](#_Toc28040427)

[3 ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА 19](#_Toc28040428)

[4 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 22](#_Toc28040429)

[ЗАКЛЮЧЕНИЕ 25](#_Toc28040430)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 26](#_Toc28040431)

[ПРИЛОЖЕНИЕ А 27](#_Toc28040432)

[Исходный код программы 27](#_Toc28040433)

ВВЕДЕНИЕ

История компьютерных игр начинается в 1940-х и 1950-х годах, когда в академической среде разрабатывались простые игры и симуляции. Компьютерные игры длительное время не были популярны, и только в 1970-х и 1980-х годах, когда появились для широкой публики аркадные автоматы, игровые консоли и домашние компьютеры, компьютерные игры становятся частью поп-культуры.

В далеком 1978 году коммерчески успешная игра Space Invaders сильно мотивировала крупных инвесторов обратить внимание на игровую индустрию. Эта видеоигра была выпущена в Японии на платформе, которую мы знаем под названием аркадные аппараты. Игра стала настолько популярной, что вызвала в стране дефицит монет, которые требовались для того, чтобы поиграть в нее. Из Японии игра быстро нашла свое распространение в Америке, где за 2 года было продано 60 000 аппаратов. Они стояли во всех барах, боулинг-клубах и кинотеатрах. Благодаря Space Invaders начался бум развития аркадных аппаратов, ставших первой ступенью эволюции индустрии игр. В 1979 году это было настоящим прорывом: американский дистрибьютор игры Belly Midway утроил рынок игровых автоматов в США, доведя объёмы продаж до 1,33 млрд. долларов за год. Многие известные компьютерные игры впервые появились именно на игровых аппаратах и уже позднее были портированы на другие платформы. Например, Pac-Man, Street Fighter, Killer Instinct.

Вместе с игровыми аппаратами игры развивались на платформе игровых приставок. И хотя история приставок началась еще в 1972 году вместе с релизом первой в мире домашней игровой приставки Magnavox Odyssey, прорыв в этом направлении начинается в 1977 году, когда в продажу поступает игровая приставка Atari 2600. Благодаря инвестициям Atari популяризация компьютерных и видеоигр выходит на новый уровень. Появляются новые крупные игроки, желающие занять теплое место на рынке видеоигр. В качестве крупнейших кроме Atari в историю вошли Sega, Nintendo, Sony и Microsoft.

Другой крупной игровой платформой «с древнейших времен» стали персональные компьютеры. Первые игры на них появились еще в 1960-х годах. Это были текстовые приключенческие игры, в которых общение между игроком и компьютером осуществлялось посредством ввода команд через клавиатуру. Дальнейшая эволюция игр тесно связана с ростом производительности и возможностей компьютерного оборудования. Это и популяризация компьютерных мышей в качестве интерфейса взаимодействия человек-компьютер, и появление звуковых карт, дающих возможность воспроизводить полноценные звуки, и ежегодное улучшение качества выводимого изображения, и конечно развитие компьютерный сетей, вылившихся в итоге в появление Интернета. С эволюцией сетей на смену привычным всем однопользовательским клиентским играм приходит новое поколение игр, дающих возможность играть совместно с другими игроками. Самая первая онлайн игра была сделана в текстовом виде и запущена в сети TelNet в 1978 году. Разработчиками этой игры были Ричард Бартл и Рой Трабшоу. Они назвали свое творение Multi-user Dungeon, сокращенно MUD. В дальнейшем эта аббревиатура стала использоваться в качестве классификации для многопользовательских онлайн игр, где абсолютно все отображается в виде текста. MUD является истоком всех виртуальных миров, которые мы сегодня знаем.

Игровая индустрия развивалось стремительно. Вскоре начали появляться прототипы современных игровых центров. Игровой центр – централизированное бесплатное приложение, позволяющее покупать, скачивать и запускать игры из одного места.

С успехом таких игр, как Counter Strike и Team Fortress, в 1999 началась история игрового центра Steam. Ещё до выхода финальных версий эти игры стали одними из самых популярных многопользовательских игр. В то время популярность игр достигала максимум 2—3 тысяч активных игроков. Число игроков TFC и CS спустя пару лет возросло в несколько раз. В результате Valve приступила к пересмотру традиционных методов организации онлайн-игр.

Когда разработчики работали над этой системой, им пришла идея использовать эту платформу для продажи игр. Реализация этой идеи представилась слишком дорогой, поэтому в Valve искали сотрудничества с различными компаниями, такими как Amazon, Yahoo и Cisco, предлагая данный подход, но все ограничивались обещаниями. Поскольку система была так необходима Valve, компания решила, что будет сама осуществлять эту идею.

Steam впервые был представлен на Game Developers Conference 22 марта 2002 года. Steam является первым успешным игровым центров для PC игр. В дальнейшем успех Steam повлёк за собой огромное множество других игровых центров от крупных мировых разработчиков игр.

Целью данного курсового проекта является разработка приложения «Игровой центр».

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1 Обзор аналогов

Существует огромное множество игровых центров. Самым главным представителем этого семейства является Steam, представленный на рисунке 1.1.

В Steam реализованы функции авторизации и регистрации пользователей, есть возможность максимально защитить свой аккаунт благодаря системе двухэтапной аутентификации. Очень удобно реализована структура самого приложения. Игры, скаченные или приобретенные пользователем, содержатся в окне “Библиотека”, остальные содержатся в окне “Магазин”. Реализован поиск и сортировка игр жанрам, году выпуска и т.п.. Удобная функция Steam Cloud позволяет хранить игровые данные пользователя на облаке Steam, с целью избегания повторной настройки игрового процесса. Steam поддерживает потоковую загрузку контента. Это позволяет распределить приоритеты загрузки содержимого. Таким образом, сначала загружается часть игры, необходимая для запуска. Остальные файлы (в правильном порядке) — в фоновом режиме. Загрузка уровня игры приостанавливается, если ещё не загружены необходимые файлы. Разработана функция Big Picture — это режим работы Steam, оптимизированный для больших экранов телевизоров и управления геймпадом. Нажатием одной кнопки Steam будет переключаться в полноэкранный режим, оптимизированный для удобства чтения и использования на телевизоре, без клавиатуры и мышки, хотя они тоже поддерживаются. В вкладке “Сообщество” реализован целый форум пользователей с большим количеством обсуждения, гайдов, модификаций для игр и многого другого. Доступ к этой функции можно получить прямо поверх игрового процесса благодаря функции Overlay, которая активируется клавишами Shift + Tab. Также в Steam добавлена поддержка достижений. А с недавних пор добавлена поддержка VR гарнитуры.

В целом, Steam – это эталонное приложение, пользующиеся огромным спросом как среди геймеров, так и среди обычных пользователей.

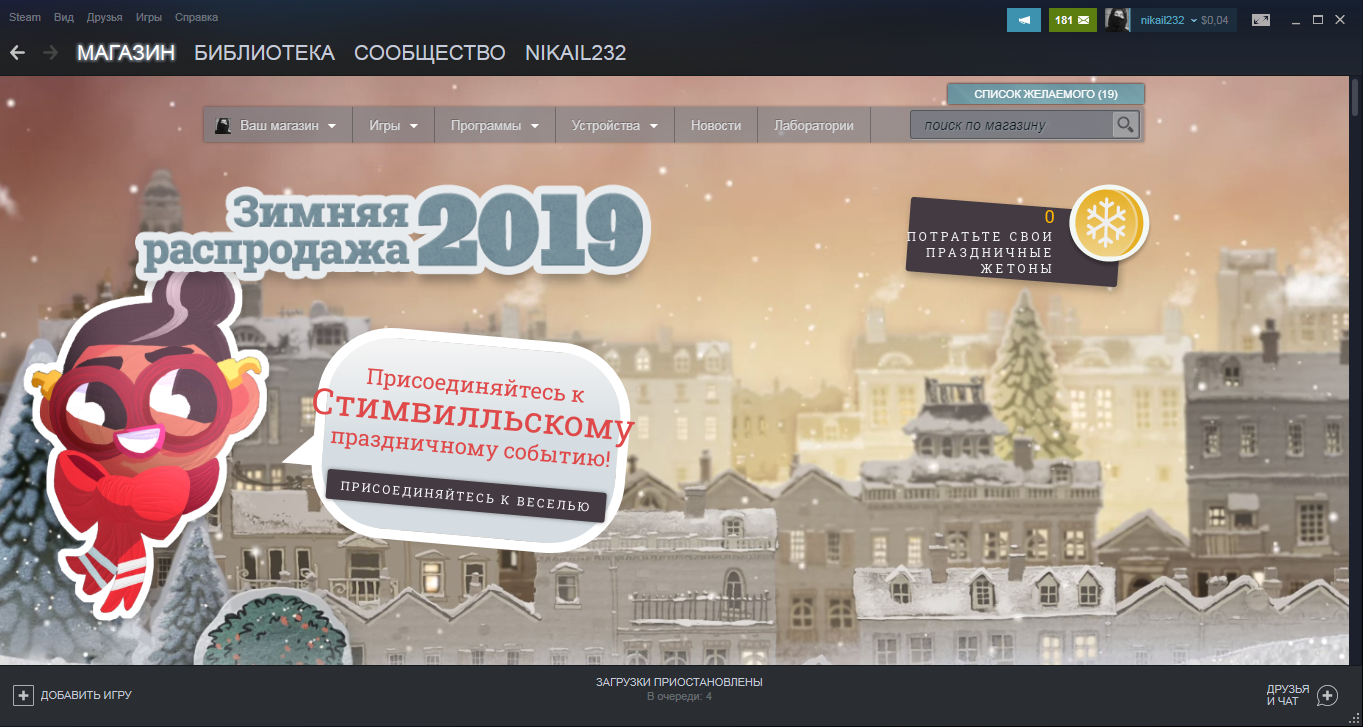


Рисунок 1.1 – Главное окно Steam

Следующим аналогом рассмотрим игровой центр Epic Games Store от компании Epic Games. Внешний вид приложения показана на рисунке 1.2.

По функционалу Epic Games Store уступает своим конкурентам, что может быть обусловлено очень недолгим существованием самой платформы, которая была выпущена 6 декабря 2018 года.  В данный момент у него нет таких функций, как достижения, сообщество или поддержка гарнитуры виртуальной реальности, но сам игровой центр остается популярным благодаря мудрой ценовой политике относительно разработчиков игр, выпускающих свои продукты под началом Epic Games и благодаря раздаче одной игры в неделю абсолютно бесплатно.

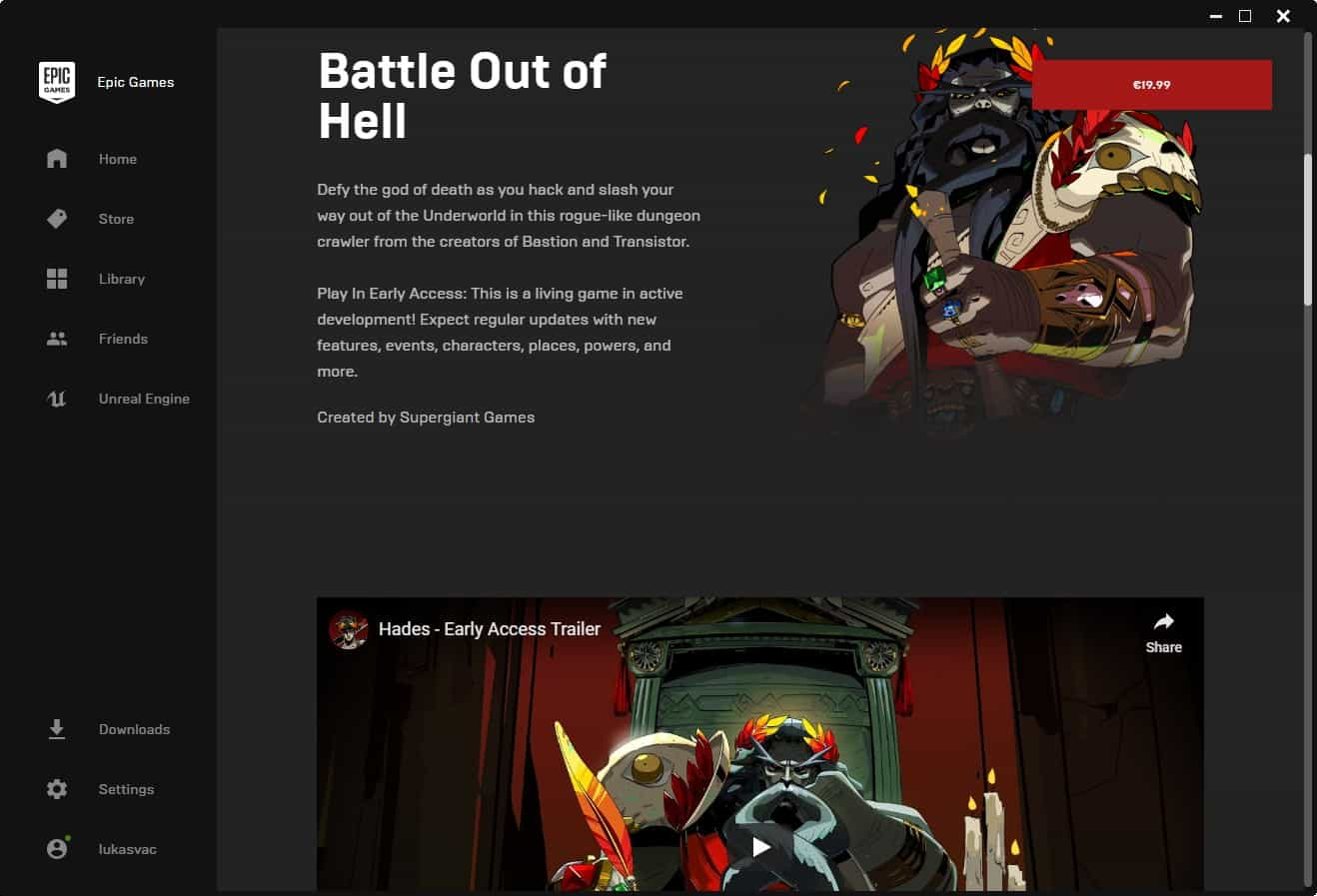


Рисунок 1.2 – Главное окно Epic Games Store

И последним нами рассмотренным аналогом игрового центра будет GOG.com. Интерфейс GOG.com представлен на рисунке 1.3.

Основное огромнейшее отличие Gog.com от предыдущих аналогов содержится не в функционале самого приложения, а в политике распространения цифровых копий. У Steam и Epic Games Store на игры накладывалась DRM защита авторских прав, здесь же разработчики платформы пошли полностью противоположным путем. Такое решение не могло обрадовать разработчиков игр, поэтому самих игр в GOG.com не очень много, а малое количество игр для игрового центра – это большая проблема. Сам игровой центр GOG Galaxy выполнен на отлично, но функционал находится на уровне Epic Games Store.

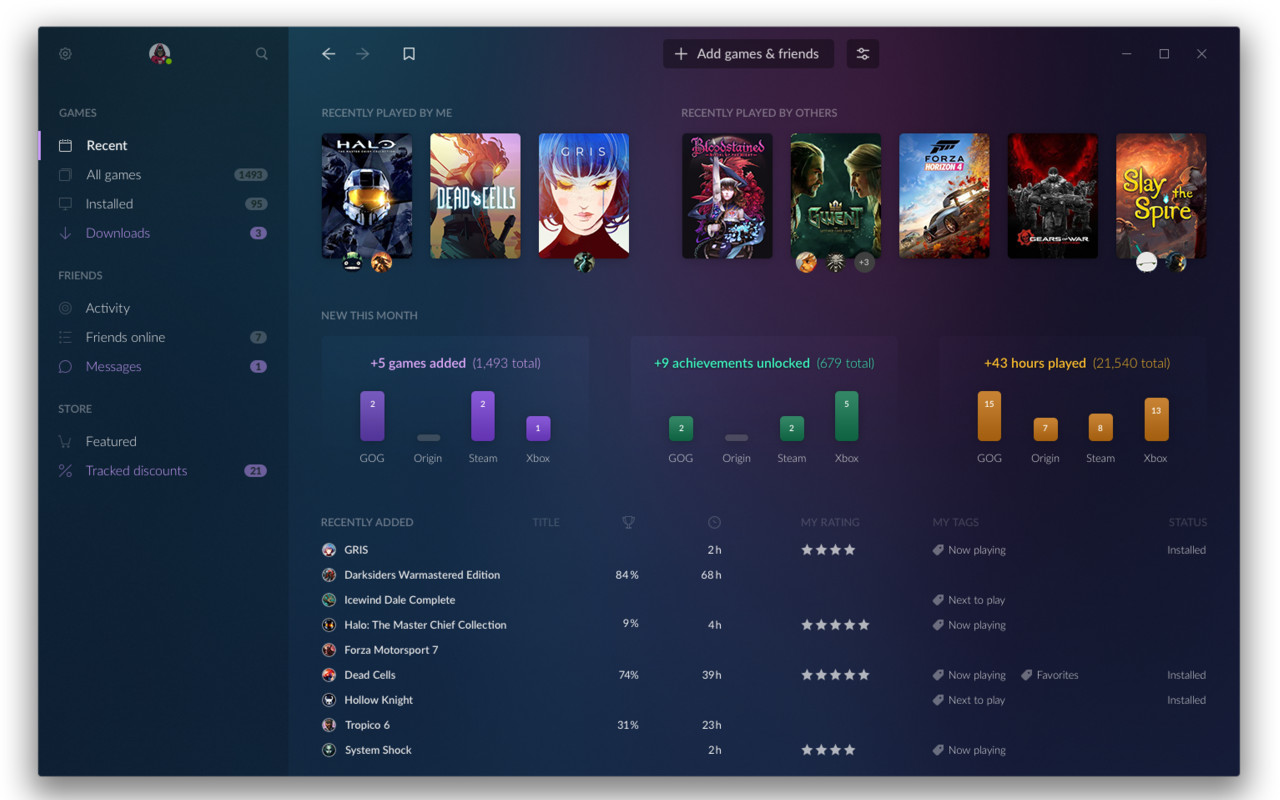


Рисунок 1.3 – Главное окно Gog Galaxy

## **1.2 Постановка задачи**

В рамках данного курсового проекта планируется разработка программного средства «Игровой центр».

Будут разработаны алгоритмы работы с базой данных SQLITE, а также алгоритмы работы с файлами.

В программном средстве планируется реализовать следующие функции:

* Авторизация и регистрация пользователей;
* Возможность сменить пользователя;
* Отображение фотографий профиля пользователей;
* Смена фотографий пользователя;
* Запуск игр “Doodle Jump”, “Snake” и “Miner”;
* Сохранение результатов игровых сеансов в текстовый файл;
* Загрузка результатов в таблицу лидеров.

Для разработки программного средства будет использоваться язык программирования C++, операционная система Windows 7 и среда разработки Qt 5.2.1.

2 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА



2.1 Структура программы

Данное приложение включает в себя пять составляющих: модули авторизации и регистрации, модуль для запуска игр, модуль для отображения таблицы лидеров, а также модуль для работы с SQLITE:

– MainWindow – модуль, отображающий окно авторизации пользователя;   
– Register – модуль, отображающий окно регистрации пользователя;

– Games – модуль, отображающий главную форму игрового центра – форму запуска игр;

– Leaders – модуль, отображающий таблицу лидеров среди пользователей;

– Sqlwork – модуль, содержащий в себе алгоритмы работы с базой данных SQLITE.

В ходе разработки программного продукта была использована одна среда разработки, на это есть ряд причин. Qt 5.2.1 предоставляет огромные возможности для редактирования исходного кода, лексического, синтаксического и даже семантического анализа написанного кода в реальном времени, и удобную подсветку синтаксиса, в [1, 4, 5] описана структура языка. Также Qt – это нечто большее, чем просто GUI Framework. Помимо возможности построения графического интерфейса, Qt поддерживает работу с XML, базами данных, OpenGL и т.д..

Именно благодаря тому, что Qt поддерживает функции работы с базами данных, в частности SQLITE, было решено выбрать его как инструмент разработки игрового центра.

Программное средство проектировалось, основываясь на методах, описанных в [2, 3].

2.2 Интерфейс программного средства

Внешний вид и удобность в использовании являются одними из главных критериев качества программного средства. Поэтому взаимодействие приложения с пользователем необходимо организовать максимально интуитивно и просто.

В качестве навигации были выбраны очень простые и понятные большинству пользователей компоненты QPushButton, QTableWidget. Также пользователю предлагается ввести логин и пароль в соответствующие компоненты QLineEdit на форме авторизации и, соответственно, имя пользователя, логин и пароль на форме регистрации.

На рисунке 2.1 представлено окно авторизации игрового центра, а на рисунке 2.2 окно регистрации нового пользователя.

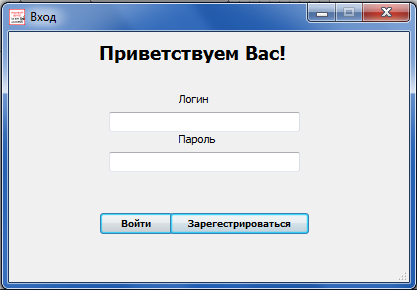


Рисунок 2.1 – Окно авторизации

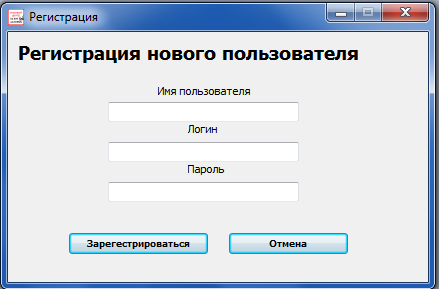


Рисунок 2.2 – Окно регистрации

Интерфейс авторизации и регистрации был сделан в классическом и проверенном временем стилем, на форме не присутствует ничего лишнего, что является признаком её компактности и простоты для пользователя. Была проведена работа над шрифтами текста в QLabel, также добавлена функция перехода фокуса между компонентами формы по нажатию клавиши Tab. При нахождении фокуса на кнопках и нажатии кнопки Enter будет эквивалентно нажатию левой кнопкой мыши по кнопке. Также добавлен пустой компонент QLabel на обе формы, в котором будет отображаться сообщение об ошибке при некорректном вводе данных.

Далее рассмотрим интерфейс формы для запуска игр. Внешний вид формы приведен на рисунке 2.3.

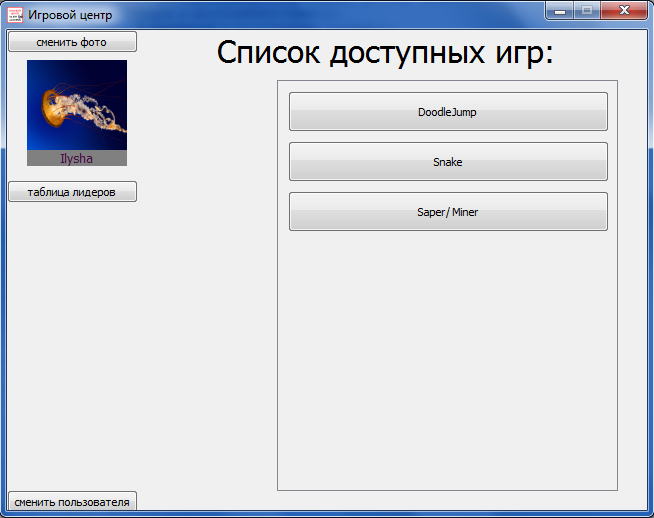


Рисунок 2.3 – Форма запуска игр

Эта форма является основной для игрового центра. На ней реализовано множество функций. Реализована отображение фотографии профиля вместе с именем пользователя в левом верхнем углу формы. Добавлена возможность смены фотографии по нажатию кнопки “Сменить фото”, при входе будет загружена фотография, которая было при предыдущем прекращении работы программного средства.

В левом нижнем углу находится кнопка “Сменить пользователя”. По нажатию этой кнопки пользователь будет возвращен на предыдущее окно, где сможет ввести другие данные и войти на другой аккаунт.

Справа на форме находится область с играми. По нажатию соответствующей кнопки будет открыта соответствующая игра. При выходе из игры, очки набранные пользователем могут записаться в таблицу лидеров, если будут удовлетворять определенным условиям.

Последним графическим интерфейсом, разработанным в данном приложении, является интерфейс формы таблицы лидеров. Внешний вид формы распологается на рисунке 2.4.

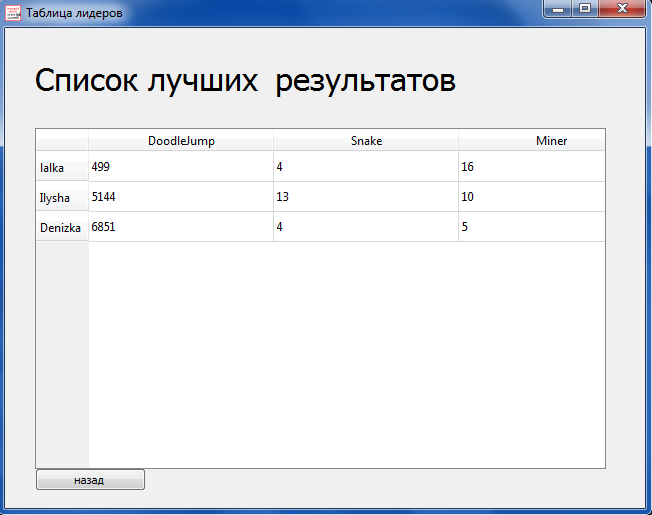


Рисунок 2.4 – Форма таблицы лидеров

Основным компонентом на форме является QTableWidget. Он является очень удобным инструментом для построения любых таблиц, поэтому было принято решение выводить лучшие результаты пользователей на этот компонент. В заголовках строк находятся имена пользователей, в заголовках столбцов находятся названия игр. Добавлена функция сортировки по алфавиту пользователей, сортировка по убыванию/возрастанию результата пользователей в той или иной игре.

2.3 Разработка алгоритмов программного средства

Было принято решение хранить логин и пароль пользователя в базе данных SQLITE. Для этого, как уже описывалось ранее, был создан отдельный модуль, содержащий в себе алгоритмы работы с этой базой данных.

Помимо алгоритмов, было создана сама база данных, содержащая таблицу Users, внешний вид которой показан на рисунке 2.3, которая состоит из 4 полей:

- Поле ID – номер записи в таблице. Этому полю были заданы параметры INTEGER AUTOINCREMENT. Это означает, что при добавление в таблицу новой записи, полю ID будет присвоено значение ID + 1 предыдущей записи таблицы.

- Поле Nickname – имя пользователя. Полю был задан только один параметр - VARCHAR (20) . Из этого следует, что поле хранит данные последовательностей символьного типа, длиной до 20 символов включительно.

- Поле Login – уникальный идентификатор пользователя. Также, как и Nickname может хранить в себе строки длиной до 20 символов, однако добавлено свойство UNIQUE, что означает невозможность повторений двух логинов у разных пользователей.

- Поле Password – пользовательский ключ доступа к своему аккаунту. Является защитой аккаунта пользователя от попыток получения доступа людьми, не владеющими паролем. Хранит данные до символьного типа длиной до 20 символов.

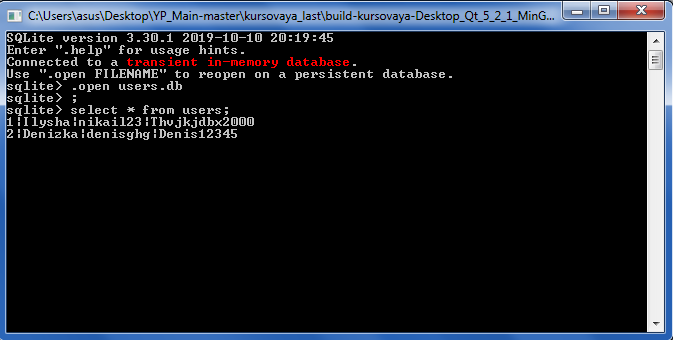


Рисунок 2.5 – Таблица Users в базе данных

Рассмотрим алгоритмы работы с базой данных. Чтобы получить доступ к базе данных через Qt, необходимо объявить объект класса типа QSqlDatabase, который отвечает за установление соединения с базой данных, а также объект типа QSqlQuery, который позволяет делать запросы к базе данных. Для объявления объектов такого типа, необходимо подключить к проекту файлы, в которых они описаны:

#include "QtSql/QSqlDatabase"

#include "QSqlQuery"

Далее был создан класс SqlWork, в разделе private которого были объявлены объекты соответствующих типов:

QSqlDatabase database;

QSqlQuery sqlquery;

Работа с базой данной SQLITE была реализована объектно-ориентированным путем. Был создан класс, полями которого являются database и sqlquery, а методами соответствующие функции добавления пользователя в базу данных и проверки существования пользователя с конкретными данными. Ниже приведен пример реализации функции добавления пользователя в базу данных:

void SqlWork::AddNewUser(QString username, QString login, QString password)

{

if (database.open())

{

sqlquery = QSqlQuery(database);

sqlquery.prepare("INSERT INTO users (nickname, login, password) VALUES (:username,:login,:password);");

sqlquery.bindValue(":username", username);

sqlquery.bindValue(":login", login);

sqlquery.bindValue(":password", password);

sqlquery.exec();

database.close();

}

}

Важно добавить, что после выполнения работы с базой данных, необходимо разорвать соединение с ней. Поэтому установление соединения с базой данных было добавлено в конструктор класса, а разрыв соединения происходил в деструкторе класса:

SqlWork::SqlWork()

{

database = QSqlDatabase::addDatabase("QSQLITE", "Connection");

database.setDatabaseName(Path\_users\_db);

}

SqlWork::~SqlWork()

{

database = QSqlDatabase();

database.removeDatabase("Connection");

}

Далее рассмотрим реализацию других алгоритмов программного средства.

Как описывалось выше, после окончания игрового сеанса, в игровой центр будет поступать результат этого сеанса. Просто добавить его в таблицу лидеров было бы нелогичным, поэтому был разработан алгоритм сравнения текущего результата с результатом этого же пользователя в этой же игре, находящимся в файле. Основная идея алгоритма – перед записью в файл, проверить:

1) Есть ли в файле результат игры этого пользователя в этой игре.

2) Больше ли значение нового результата предыдущего.

При выполнении первого условия мы переходим к второму, иначе просто записываем результат в файл. При выполнении второго условия мы перезаписываем файл с новым значением результата, иначе ничего не делаем. Схема алгоритма приведена на рисунке 2.6.



Рисунок 2.6 – Схема алгоритма проверки результата сеанса игры

# **3 ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА**

В ходе тестирования приложения были выявлены некоторые недостатки программного средства. Была составлена таблица 3.1, показывающая ожидаемые и реальные результаты, полученные при заданных условиях, она представлена ниже.

Таблица 3.1 – Ожидаемые и реальные результаты тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| № | Тестовые случаи | Ожидаемый результат | Полученный результат |
| 1. | Ввод правильного логина и пароля | Успешный вход пользователя | Вывод сообщения об ошибке |
| 2. | Ввод неправильного логина и пароля | Вывод сообщения об ошибке | Вывод сообщения об ошибке |
| 3. | Регистрация пользователя с повторным логином | Сообщение с просьбой поменять логин | Сообщение с просьбой поменять логин |
| 4. | Регистрация пользователя с правильными данными | Сообщение об успешной регистрации | Сообщение об успешной регистрации |
| 5. | Нажатие на кнопку с игрой | Открытие игры | Ничего не произошло |
| 6. | Нажатие на кнопку “Таблица лидеров” | Открытие правильной таблицы лидеров | Открытие правильной таблицы лидеров |
| 7. | Нажатие на кнопку “Сменить пользователя” | Переход на форму авторизации | Переход на форму авторизации |

Разработка приложения велась с использованием системы контроля версий GitHub, позволившая сохранять состояние программы на каждом отдельном этапе по ходу добавления нового функционала или изменения уже существующего. Появление новых точек возврата происходит посредством группировки изменённых файлов, затем они объединяются под общим именем «коммита», в котором кратко изложена суть изменений. Также можно добавлять к каждому этапу новые файлы, или удалять устаревшие варианты. После накопления определённого количества групп изменений, их следует отправить на удалённый репозиторий, где видна вся история приложения и разница между каждым новым «коммитом».

В ходе отладки программного обеспечения была выявлена ошибка. При правильном вводе логина и пароля выводилось сообщение, что логин и пароль неправильные. Текст подпрограммы, в которой происходила ошибка:

void MainWindow::on\_pushButton\_clicked()

{

QString login;

QString password;

bool isOpen = false;

{

SqlWork sql;

getLoginData(&login, &password);

if (sql.IsExistingUser(login, password))

isOpen = true;

else

ui->label\_4->setText("Неверный логин или пароль!");

}

if (isOpen)

{

GamesForm->show();

emit OpenGamesForm(login);

this->close();

}

}

bool SqlWork::IsExistingUser(QString login, QString password)

{

QString DBlogin;

QString DBpassword;

if (database.open())

{

sqlquery = QSqlQuery(database);

sqlquery.exec("SELECT \* FROM users;");

while (sqlquery.next())

{

DBlogin = sqlquery.value(2).toString();

if (password != "")

{

DBpassword = sqlquery.value(3).toString();

if ((login == DBlogin) && (password == DBpassword))

{

return true; break;

}

}

else

if (login == DBlogin)

{

return true; break;

}

}

database.close();

}

return false;

}

Ошибка явно происходила в методе IsExistingUser объекта sql. В ходе отладки программного средства удалось выяснить то, что сам метод IsExistingUser работает полностью правильно, и то, что ошибка в том, что не устанавливалось подключение к базе данных в конструкторе класса объекта sql и, следовательно, функция database.open() возвращала всё время false. Причиной этого послужило то, что символы ‘\’ воспринимались не по назначению. Ошибка исправилась простым экранированием этого символа:

#define Path\_users\_db "build-kursovaya-Desktop\_Qt\_5\_2\_1\_MinGW\_32bit-Debug\\sqlite\\users.db"

Вторая ошибка была связана с тем, что по нажатию кнопки с именем игры, сама игра не открывалась. На её устранение ушло гораздо больше времени. Однако, через Microsoft Visual Studio 2019 они открывались и работали очень стабильно. Проблема решилась тогда, когда была произведена попытка запуска .exe файла игры. Windows выдала ошибку, изображенную на рисунке 3.1.

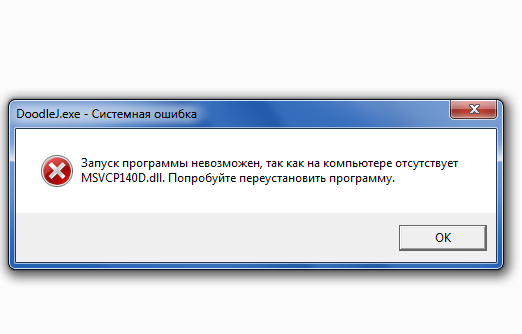


Рисунок 3.1 – Окно с ошибкой

DLL или dynamic-link library - это файл с дополнительными ресурсами, которые могут использоваться программами и самой операционной системой.

Проблема была решена перемещением в папку с проектом соответствующих файлов .dll.

4 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Перед запуском игрового центра пользователю настоятельно рекомендуется убедиться в целостности корневого каталога игрового центра.

Необходимо наличие следующих файлов и директорий:

* “build-kursovaya-Desktop\_Qt\_5\_2\_1\_MinGW\_32bit-Debug\sqlite\users.db”
* “build-kursovaya-Desktop\_Qt\_5\_2\_1\_MinGW\_32bit-Debug\score.txt”
* Игровой .exe файл в одно папке с необходимыми .dll файлами, расположенный по пути:

“Games0\...”

При соблюдении всех условий программное средство “Игровой центр” успешно запуститься. На рисунке 4.1 изображено приветственное окно авторизации.

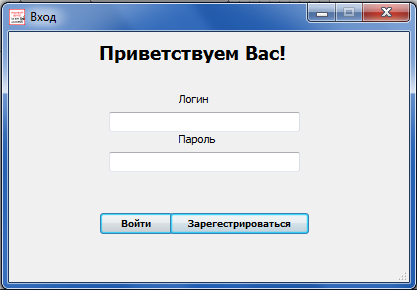


Рисунок 4.1 – Окно авторизации

Если вход в программу осуществляется впервые, то необходимо выполнить следующую последовательность действий:

* Нажать кнопку “Зарегестрироваться”.
* В открывшемся окне ( рисунок 4.2 ) ввести данные для регистрации, в случае возникновении сообщения о существовании пользователя с таким логином необходимо подобрать другой логин.
* В случае возникновении любого другого сообщения четко следовать написанному в нем тексту.
* Нажать на кнопку “Зарегестрироваться”.
* Закрыть окно регистрации.

- Перейти к последовательности действий при повторном входе.

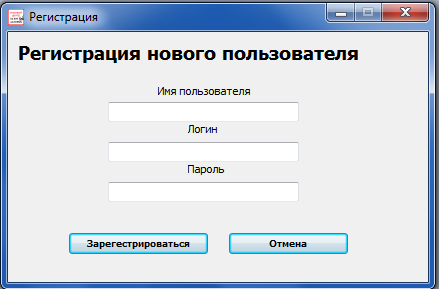


Рисунок 4.2 – Окно регистрации

Если пользователь осуществляет повторный вход в программу, то необходимо выполнить следующую последовательность этапов:

* Ввести логин и пароль от своего аккаунта правильно.
* В случае неправильного ввода, возникнет сообщение об ошибке.
* Нажать на кнопку “Войти”. В случае правильного ввода всех данных откроется окно для запуска игр.
* Перейти к возможным действиям пользователя на форме запуск игр.

( рисунок 4.3 ).

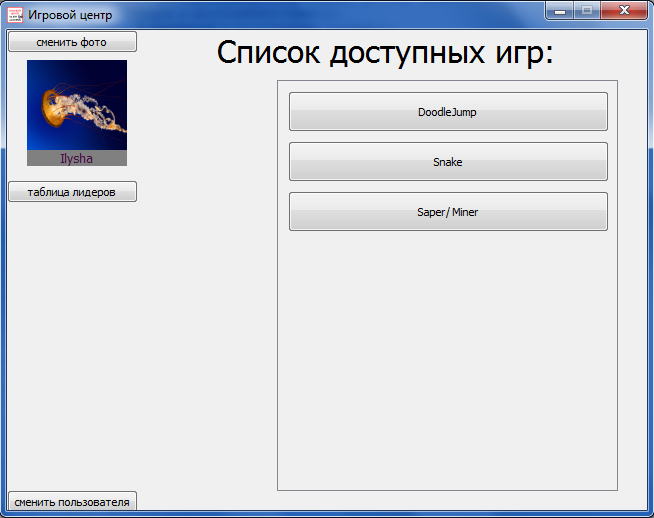


Рисунок 4.3 – Форма запуска игр

Возможные действия пользователей на форме для запуска игр:

- Открыть таблицу лидеров, нажав на кнопку “Таблица лидеров”. На ней будут располагаться Ваши лучшие результаты и результаты других пользователей.

- Сменить фотографию, нажав на кнопку “Сменить фотографию”. Фотография останется текущей, пока вы ее не смените. Фотография масштабируется, поэтому Вы можете загрузить фотографию любого размера.

- Сменить пользователя, нажав на кнопку “Сменить пользователя”. Вы переместитесь на предыдущую форму и сможете зайти в другой аккаунт.

- Запустить игру, нажав на ее имя в правом списке, результат игрового сеанса автоматически попадет в таблицу лидеров, если он будет лучше, чем предыдущий лучший результат пользователя.

- В любой момент можете закрыть приложение.

ЗАКЛЮЧЕНИЕ

Игровая индустрия – одна из самых перспективных отраслей IT. С каждым днем человечество всё больше и больше увеличивает требования к игровым продуктам, требуя чего-то нового. Игровые центры несомненно помогают индустрии: позволяют разработчикам выпускать свою проекты под покровительством крупнейших корпораций, тем самым заставляя их чувствовать себя увереннее, ведь их авторское право защищено, их проект не останется незамеченным, если он действительно сделан с умом, и они получат фиксированную прибыль от своего проекта. При этом, обычные пользователи игрового центра знают, что покупают сертифицированный продукт, могут оставить отзыв и, даже, в могут вернуть свои деньги, если продукт их не устроит.

В рамках данного курсового проекта было разработано программное средство «Игровой центр». Получившийся игровой центр является аналогом своих прототипов, таких как Steam, Epic Games Store, GOG.com с урезанным в рамках текущий знаний разработчика.

Были разработаны алгоритмы работы с базой данных SQLITE, файлами, различными компонентами форм Qt.

В программном средстве реализованы следующие функции:

* Авторизация и регистрация пользователей;
* Возможность сменить пользователя;
* Отображение фотографий профиля пользователей;
* Смена фотографий пользователя;
* Запуск игр “Doodle Jump”, “Snake” и “Miner”;
* Сохранение результатов игровых сеансов в текстовый файл;
* Загрузка результатов в таблицу лидеров.

Для разработки программного средства использовался язык программирования C++, операционная система Windows 7 и среда разработки Qt 5.2.1.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Программирование на языке C++ / Гирберд Шилдт. Учебный курс: / Гирберд Шилдт. – СПб: изд. С. В. Малгачёва, 2001. – 231 с;
2. Орлов, С. А. Технологии разработки программного обеспечения: учеб. Пособие. – СПб, 2003;
3. Уилсон, С. Принципы проектирования и разработки программного обеспечения, yчебн. курс. – СПб, 2003;
4. Культин, Н. А. Основы программирования в C++ 2-е издание – учеб. пособие. – Москва: изд. «НТ Пресс», 2008. – 182с;
5. Культин, Н. А. Основы программирования в C++ – учеб. пособие. – Москва: изд. «НТ Пресс», 2008. – 167с;

ПРИЛОЖЕНИЕ А

Исходный код программы

**Main.cpp:**

#include "mainwindow.h"

#include <QApplication>

int main(int argc, char \*argv[])

{

QApplication a(argc, argv);

MainWindow w;

w.show();

return a.exec();

}

**MainWindow.h:**

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include "QtSql/QSqlDatabase"

#include "QSqlQuery"

#include <QDebug>

#include "register.h"

#include "games.h"

#include "sqlwork.h"

namespace Ui {

class MainWindow;

}

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

explicit MainWindow(QWidget \*parent = 0);

~*MainWindow*();

void getLoginData(QString \*login, QString \*password);

bool IsExistingUser(QString login, QString password);

signals:

void OpenGamesForm(QString login);

private slots:

void on\_pushButton\_2\_clicked();

void on\_pushButton\_clicked();

void CreateMainWindow();

private:

Ui::MainWindow \*ui;

Register \*RegisterForm;

games \*GamesForm;

};

#endif // MAINWINDOW\_H

**MainWindow.cpp:**

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include <fstream>

#include <QSqlError>

#include <set>

MainWindow::MainWindow(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::MainWindow)

{

ui->setupUi(this);

RegisterForm = new Register();

GamesForm = new games();

connect(RegisterForm, &Register::mainWindowOpen, this, &MainWindow::show);

connect(this, &MainWindow::OpenGamesForm, GamesForm, &games::PrepareGameFormToShow);

connect(GamesForm, &games::ShowMainForm, this, &MainWindow::CreateMainWindow);

}

MainWindow::~*MainWindow*()

{

delete ui;

}

void MainWindow::getLoginData(QString \*login, QString \*password){

\*login = \*login + ui->lineEdit->text();

\*password = \*password + ui->lineEdit\_2->text();

}

void MainWindow::on\_pushButton\_2\_clicked()

{

RegisterForm->show();

this->close();

}

void MainWindow::on\_pushButton\_clicked()

{

QString login;

QString password;

bool isOpen = false;

{

SqlWork sql;

getLoginData(&login, &password);

if (sql.IsExistingUser(login, password))

isOpen = true;

else

ui->label\_4->setText("Неверный логин или пароль!");

}

if (isOpen)

{

GamesForm->show();

emit OpenGamesForm(login);

this->close();

}

}

void MainWindow::CreateMainWindow()

{

this->show();

this->ui->lineEdit\_2->clear();

this->ui->lineEdit->clear();

this->ui->label\_4->clear();

}

**Register.h:**

#ifndef REGISTER\_H

#define REGISTER\_H

#include <QMainWindow>

#include "sqlwork.h"

namespace Ui {

class Register;

}

class Register : public QMainWindow

{

Q\_OBJECT

public:

explicit Register(QWidget \*parent = 0);

~*Register*();

void FormClose(QWidget \*form);

void GetRegisterData(QString \*username, QString \*login, QString \*password);

bool CheckRegisterDataFormat(QString username, QString login, QString password);

signals:

void mainWindowOpen();

private slots:

void on\_pushButton\_3\_clicked();

void on\_pushButton\_2\_clicked();

private:

Ui::Register \*ui;

bool Check;

};

#endif // REGISTER\_H

**Register.cpp:**

#include "register.h"

#include "ui\_register.h"

Register::Register(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::Register)

{

ui->setupUi(this);

}

Register::~*Register*()

{

delete ui;

}

void Register::GetRegisterData(QString \*username, QString \*login, QString \*password)

{

\*username = ui->lineEdit->text();

\*login = ui->lineEdit\_2->text();

\*password = ui->lineEdit\_3->text();

}

void Register::FormClose(QWidget \*form)

{

form->close();

emit mainWindowOpen();

}

void Register::on\_pushButton\_3\_clicked()

{

FormClose(this);

}

bool Register::CheckRegisterDataFormat(QString username, QString login, QString password)

{

if (!(username == "") && !(login == "") && !(password == ""))

{

if (username.length() < 5)

{

ui->label\_5->setText("Длина имени пользователя должна быть больше 5 символов!");

return false;

}

if (login.length() < 5)

{

ui->label\_5->setText("Длина логина должна быть больше 5 символов!");

return false;

}

if (password.length() < 5)

{

ui->label\_5->setText("Длина пароля должна быть больше 5 символов!");

return false;

}

return true;

}

else

{

ui->label\_5->setText("Заполните все поля!");

return false;

}

}

void Register::on\_pushButton\_2\_clicked()

{

QString username;

QString login;

QString password;

{

SqlWork sql;

GetRegisterData(&username, &login, &password);

if ((CheckRegisterDataFormat(username, login, password)) && !(sql.IsExistingUser(login, "")))

{

ui->label\_5->setText("Вы успешно зарегестрировались!");

sql.AddNewUser(username, login, password);

}

else

ui->label\_5->setText("Пользователь с таким логином уже зарегестрирован!");

}

}

**Sqlwork.h:**

#ifndef SQLWORK\_H

#define SQLWORK\_H

#include <QString>

#include "QtSql/QSqlDatabase"

#include "QSqlQuery"

#include <QDebug>

class SqlWork

{

public:

SqlWork();

~SqlWork();

void AddNewUser(QString username, QString login, QString password);

bool IsExistingUser(QString login, QString password);

QString GetUsername(QString login);

private:

QSqlDatabase database;

QSqlQuery sqlquery;

};

#endif // SQLWORK\_H

**Sqlwork.cpp:**

#include "sqlwork.h"

#define Path\_users\_db "build-kursovaya-Desktop\_Qt\_5\_2\_1\_MinGW\_32bit-Debug\\sqlite\\users.db"

SqlWork::SqlWork()

{

database = QSqlDatabase::addDatabase("QSQLITE", "Connection");

database.setDatabaseName(Path\_users\_db);

}

SqlWork::~SqlWork()

{

database = QSqlDatabase();

database.removeDatabase("Connection");

}

void SqlWork::AddNewUser(QString username, QString login, QString password)

{

if (database.open())

{

sqlquery = QSqlQuery(database);

sqlquery.prepare("INSERT INTO users (nickname, login, password) VALUES (:username,:login,:password);");

sqlquery.bindValue(":username", username);

sqlquery.bindValue(":login", login);

sqlquery.bindValue(":password", password);

sqlquery.exec();

database.close();

}

}

bool SqlWork::IsExistingUser(QString login, QString password)

{

QString DBlogin;

QString DBpassword;

if (database.open())

{

sqlquery = QSqlQuery(database);

sqlquery.exec("SELECT \* FROM users;");

while (sqlquery.next())

{

DBlogin = sqlquery.value(2).toString();

if (password != "")

{

DBpassword = sqlquery.value(3).toString();

if ((login == DBlogin) && (password == DBpassword))

{

return true;

break;

}

}

else

if (login == DBlogin)

{

return true;

break;

}

}

database.close();

}

return false;

}

QString SqlWork::GetUsername(QString login)

{

QString username;

if (database.open())

{

sqlquery = QSqlQuery(database);

sqlquery.prepare("SELECT nickname FROM users WHERE login = :login;");

sqlquery.bindValue(":login", login);

sqlquery.exec();

sqlquery.first();

username = sqlquery.value(0).toString();

database.close();

}

return username;

}

**Games.h:**

#ifndef GAMES\_H

#define GAMES\_H

#include <QMainWindow>

#include "QtSql/QSqlDatabase"

#include "QSqlQuery"

#include <QFileDialog>

#include "sqlwork.h"

#include "leaders.h"

namespace Ui {

class games;

}

class games : public QMainWindow

{

Q\_OBJECT

public:

explicit games(QWidget \*parent = 0);

~*games*();

QString GetNickname();

void ChangePhoto(QPixmap photo);

void LoadPhoto();

public slots:

void PrepareGameFormToShow(QString loginPlayer);

void SetStyleAndShowUsername();

private slots:

void on\_pushButton\_clicked();

void on\_pushButton\_2\_clicked();

void on\_pushButton\_3\_clicked();

void on\_pushButton\_4\_clicked();

void on\_pushButton\_5\_clicked();

void on\_pushButton\_6\_clicked();

signals:

void LeadersFormOpen();

void CheckGameResult(QString login, Games game, int score);

void ShowMainForm();

private:

Ui::games \*ui;

QString login;

leaders\* LeadersForm;

};

#endif // GAMES\_H

**Games.cpp:**

#include "games.h"

#include "ui\_games.h"

#include <QProcess>

#include <iostream>

#include <QPainter>

#define Path\_ProfilePhotos "build-kursovaya-Desktop\_Qt\_5\_2\_1\_MinGW\_32bit-Debug//ProfilePhotos//"

games::games(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::games)

{

LeadersForm = new leaders();

connect(this, &games::LeadersFormOpen, LeadersForm, &leaders::LoadLeadersTable);

connect(this, &games::CheckGameResult, LeadersForm, &leaders::CheckGameResult);

ui->setupUi(this);

}

void games::ChangePhoto(QPixmap photo)

{

QString FileName = Path\_ProfilePhotos + login;

QFile PhotoFile(FileName);

PhotoFile.*open*(QIODevice::WriteOnly);

QDataStream Stream(&PhotoFile);

Stream<<photo;

PhotoFile.*close*();

}

void games::LoadPhoto()

{

QPixmap\* photo = new QPixmap;

QString FileName = Path\_ProfilePhotos + login;

QFile PhotoFile(FileName);

PhotoFile.*open*(QIODevice::ReadOnly);

QDataStream Stream(&PhotoFile);

Stream >> \*photo;

PhotoFile.*close*();

ui->photo->setPixmap(\*photo);

}

void games::PrepareGameFormToShow(QString loginPlayer)

{

login = loginPlayer;

SetStyleAndShowUsername();

LoadPhoto();

}

void games::SetStyleAndShowUsername()

{

SqlWork sql;

ui->photo->setStyleSheet("QLabel{margin-left: 0px; border-radius: 5px; background: grey; color: #4A0C46;}");

ui->nicknameLabel->setStyleSheet("QLabel{margin-left: 10px; border-radius: 25px; background: grey; color: #4A0C46;}");

ui->nicknameLabel->setText(sql.GetUsername(login));

}

games::~*games*()

{

delete ui;

}

void games::on\_pushButton\_clicked()

{

QString filename = QFileDialog::getOpenFileName(this, tr("Выберите фотографию профиля!"), QDir::currentPath(), tr("Картинки (\*.png \*.jpg \*.bmp)"));

if (!filename.isNull())

{

QPixmap \*Photo = new QPixmap;

Photo->load(filename);

Photo->scaled(QSize(100, 100));

ChangePhoto(\*Photo);

LoadPhoto();

}

}

void games::on\_pushButton\_2\_clicked()

{

QByteArray Score1("0");

QProcess DoodleJ;

DoodleJ.start("Games0\\DoodleJ\\x64\\Debug\\DoodleJ.exe");

DoodleJ.waitForFinished(-1);

Score1 = DoodleJ.readAllStandardOutput();

emit CheckGameResult(login, Doodle, Score1.toInt());

}

void games::on\_pushButton\_3\_clicked()

{

QByteArray Score2("0");

QProcess Snake1;

Snake1.start("Games0\\Snake\\Snake1\\x64\\Debug\\Snake1.exe");

Snake1.waitForFinished(-1);

Score2 = Snake1.readAllStandardOutput();

emit CheckGameResult(login, Snake, Score2.toInt());

}

void games::on\_pushButton\_4\_clicked()

{

QByteArray Score3("0");

QProcess Miner1;

Miner1.start("Games0\\Miner\\x64\\Debug\\Miner.exe");

Miner1.waitForFinished(-1);

Score3 = Miner1.readAllStandardOutput();

emit CheckGameResult(login, Miner, Score3.toInt());

}

void games::on\_pushButton\_5\_clicked()

{

LeadersForm->show();

emit LeadersFormOpen();

}

void games::on\_pushButton\_6\_clicked()

{

emit ShowMainForm();

this->close();

}

**Leaders.h:**

#ifndef LEADERS\_H

#define LEADERS\_H

#include <QMainWindow>

#include <QFile>

#include <QDir>

#include <QTextStream>

#include <QTableWidgetItem>

#include "sqlwork.h"

enum Games {Doodle, Snake, Miner};

namespace Ui {

class leaders;

}

class leaders : public QMainWindow

{

Q\_OBJECT

public:

explicit leaders(QWidget \*parent = 0);

~*leaders*();

public slots:

void LoadLeadersTable();

void CheckGameResult(QString login, Games game, int score);

void WriteScoreToTableWidget(int score, int i, Games GameType);

int ReturnIndexOfMatchUsernameAndTableHeader(QString username, bool \*isFind);

void AddRowAndSetNewUsernameAsHeader(QString username);

private slots:

void on\_pushButton\_clicked();

private:

Ui::leaders \*ui;

};

#endif // LEADERS\_H

**Leaders.cpp:**

#include "leaders.h"

#include "ui\_leaders.h"

#define Path\_Score "build-kursovaya-Desktop\_Qt\_5\_2\_1\_MinGW\_32bit-Debug//score.txt"

leaders::leaders(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::leaders)

{

ui->setupUi(this);

}

leaders::~*leaders*()

{

delete ui;

}

QString getLine(QString login, Games game, int score)

{

QString Result;

switch(game)

{

case Doodle:

Result = "DoodleJump " + login + " " + QString::number(score);

break;

case Snake:

Result = "Snake " + login + " " + QString::number(score);

break;

case Miner:

Result = "Miner " + login + " " + QString::number(score);

break;

};

return Result;

}

void getData(QString line, Games\* gameRet, int\* scoreRet, QString\* loginRet)

{

QString game;

QString login;

QString score;

int i = 0;

while (line[i] != ' ')

{

game += line[i];

i++;

}

i++;

while (line[i] != ' ')

{

login += line[i];

i++;

}

i++;

while ((line[i] != ' ') && (i < line.size()))

{

score += line[i];

i++;

}

if (game == "DoodleJump")

\*gameRet = Doodle;

if (game == "Snake")

\*gameRet = Snake;

if (game == "Miner")

\*gameRet = Miner;

\*scoreRet = score.toInt();

\*loginRet = login;

}

bool CompareFileAndGameResult(QString line, QString result, bool \*Find)

{

QString ResultLogin;

QString LineLogin;

Games ResultGame;

Games LineGame;

int ResultScore;

int ResultStringListcore;

getData(line, &LineGame, &ResultStringListcore, &LineLogin);

getData(result, &ResultGame, &ResultScore, &ResultLogin);

if ((ResultGame == LineGame) && (ResultLogin == LineLogin))

{

if (ResultScore > ResultStringListcore)

{

return true;

}

\*Find = true;

}

return false;

}

bool isOpenToReadCompleteFile(QFile \*LeadersFile)

{

if ((\*LeadersFile).exists())

if ((\*LeadersFile).*open*(QIODevice::ReadOnly | QIODevice::Text))

{

if (!(\*LeadersFile).*atEnd*())

{

return true;

}

else

(\*LeadersFile).*close*();

}

return false;

}

bool isOpenToRewriteFile(QFile \*LeadersFile)

{

return ((\*LeadersFile).*open*(QIODevice::WriteOnly | QIODevice::Truncate| QIODevice::Text));

}

bool isOpenToAppendFile(QFile \*LeadersFile)

{

return ((\*LeadersFile).*open*(QIODevice::Append | QIODevice::Text));

}

void ReadDataToResultStringList(QFile \*LeadersFile, QStringList \*FileResults)

{

while (!(\*LeadersFile).*atEnd*())

{

\*FileResults << (\*LeadersFile).readLine();

}

(\*LeadersFile).*close*();

}

void WriteResultStringListInFile(QFile \*LeadersFile, QStringList \*NewFileResults)

{

QTextStream stream(LeadersFile);

for (int i = 0; i < (\*NewFileResults).size(); i++)

{

if (i != (\*NewFileResults).size()-1)

stream << (\*NewFileResults).at(i).trimmed() << '\r' << '\n';

else

stream << (\*NewFileResults).at(i).trimmed();

}

}

void WriteSingleResultInFile(QFile \*LeadersFile, QString GameResult)

{

QTextStream stream(LeadersFile);

stream << GameResult;

}

void leaders::CheckGameResult(QString login, Games game, int score)

{

QFile LeadersFile(Path\_Score);

QString GameResult = getLine(login, game, score);

QStringList ResultStringList;

int i = 0;

int SizeOfResultStringList;

bool isFind = false;

if (isOpenToReadCompleteFile(&LeadersFile))

{

ReadDataToResultStringList(&LeadersFile, &ResultStringList);

SizeOfResultStringList = ResultStringList.size();

foreach (QString FileResult, ResultStringList)

{

if (!CompareFileAndGameResult(FileResult, GameResult, &isFind))

{

i++;

if ((!isFind) && (i == SizeOfResultStringList) && isOpenToRewriteFile(&LeadersFile))

{

ResultStringList.insert(SizeOfResultStringList, GameResult);

WriteResultStringListInFile(&LeadersFile, &ResultStringList);

LeadersFile.*close*();

break;

}

}

else

if (isOpenToRewriteFile(&LeadersFile))

{

ResultStringList.removeAt(i);

ResultStringList.insert(i, GameResult);

WriteResultStringListInFile(&LeadersFile, &ResultStringList);

LeadersFile.*close*();

break;

}

}

}

else

if (isOpenToAppendFile(&LeadersFile))

{

WriteSingleResultInFile(&LeadersFile, GameResult);

LeadersFile.*close*();

}

}

void leaders::WriteScoreToTableWidget(int score, int i, Games GameType)

{

QTableWidgetItem \*newItem = new QTableWidgetItem(tr("%1").arg(score));

switch (GameType)

{

case Doodle:

ui->tableWidget->setItem(i, 0, newItem);

break;

case Snake:

ui->tableWidget->setItem(i, 1, newItem);

break;

case Miner:

ui->tableWidget->setItem(i, 2, newItem);

break;

};

}

int leaders::ReturnIndexOfMatchUsernameAndTableHeader(QString username, bool \*isFind)

{

int i = 0;

while (i < ui->tableWidget->rowCount())

{

QString TableHeader = ui->tableWidget->verticalHeaderItem(i)->text();

if (TableHeader == username)

{

\*isFind = true;

break;

}

i++;

}

return i;

}

void leaders::AddRowAndSetNewUsernameAsHeader(QString username)

{

ui->tableWidget->setRowCount(ui->tableWidget->rowCount() + 1);

QTableWidgetItem \*TableHeader = new QTableWidgetItem(username);

ui->tableWidget->setVerticalHeaderItem(ui->tableWidget->rowCount()-1, TableHeader);

}

void leaders::LoadLeadersTable()

{

QFile LeadersFile(Path\_Score);

QStringList ResultStringList;

QString login;

QString username;

int score;

Games GameType;

SqlWork sql;

bool isFind;

if (isOpenToReadCompleteFile(&LeadersFile))

{

ReadDataToResultStringList(&LeadersFile, &ResultStringList);

foreach (QString s, ResultStringList)

{

getData(s, &GameType, &score, &login);

username = sql.GetUsername(login);

isFind = false;

int i = ReturnIndexOfMatchUsernameAndTableHeader(username, &isFind);

if (isFind)

WriteScoreToTableWidget(score, i, GameType);

else

{

AddRowAndSetNewUsernameAsHeader(username);

WriteScoreToTableWidget(score, i, GameType);

}

}

}

}

void leaders::on\_pushButton\_clicked()

{

this->close();

}