

ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

###### МОСКОВСКИЙ ИНСТИТУТ ЭЛЕКТРОНИКИ И МАТЕМАТИКИ

**ЗАДАНИЕ**

**на междисциплинарную курсовую работу бакалавра**

студенту группы БИВ 154 Коптурову Дмитрию Александровичу

1. Тема работы

|  |
| --- |
| Разработка сетевой игры « Гомоку (Пять в ряд) » с использованием фреймворка Qt  Development of online game « Gomoku (Five in a Row) » using framework Qt |

1. Требования к работе

Разработать два приложения: игровой клиент и сервер.

|  |  |
| --- | --- |
| Клиент – приложение с графическим пользовательским интерфейсом, которое должно предоставлять следующий функционал: регистрация в базе данных игры, авторизация по логину и паролю, просмотр игровой статистики, совместная игра по сети интернет с другим пользователем с заданным логином или с любым другим случайным игроком, соответственно, организация самого игрового процесса, а также связи с оппонентом посредством текстовых сообщений.  Системные требования: клавиатура, мышь, процессор с тактовой частотой не менее 1 ГГц, 2 Гб ОЗУ, видеоадаптер с поддержкой DirectX 9 или выше, операционная система Windows 8 или выше, дисплей с разрешением 800x600, версия Qt 5.5. |  |
| Сервер – приложение с текстовым интерфейсом, которое должно иметь следующий функционал: хранение и обработка базы данных игры, система ответов на такие клиентские запросы, как авторизация, регистрация, начало игровой сессии, обработка ходов каждого игрока во всех текущих игровых сессиях, обработка завершения игры.  Системные требования: клавиатура, операционная система Debian GNU/Linux 8.0 или выше, процессор с тактовой частотой не менее 1 ГГц, 1 Гб ОЗУ, версия Qt 5.5. |  |

1. Содержание работы

Задание на междисциплинарную курсовую работу; график выполнения междисциплинарной курсовой работы; аннотация; оглавление; введение; Серверная часть: База данных, серверная часть организации игровой сессии, алгоритмы игровой сессии; Клиентская часть: интерфейс, организация запросов к серверу; Список использованных источников.

1. Сроки выполнения этапов работы

Первый вариант МКР предоставляется студентом в срок до «\_14\_»\_марта\_2016г.

Итоговый вариант МКР предоставляется студентом в срок до «\_19\_»\_мая\_\_\_2016г.

|  |  |  |
| --- | --- | --- |
| Задание выдано | «\_\_\_» \_\_\_\_\_\_\_\_\_ 201\_ г. | \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Е.А. Ерохина  подпись научного руководителя |
| Задание принято к исполнению | «\_\_\_» \_\_\_\_\_\_\_\_\_ 201\_ г. | \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Д.А. Коптуров подпись студента |

ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

###### МОСКОВСКИЙ ИНСТИТУТ ЭЛЕКТРОНИКИ И МАТЕМАТИКИ

**График выполнения**

**междисциплинарной курсовой работы бакалавра**

студента группы БИВ 154 Коптурова Дмитрия Александровича

Тема работы

Разработка сетевой игры « Гомоку (Пять в ряд) » с использованием фреймворка Qt

Development of online game « Gomoku (Five in a Row) » using framework Qt

|  |  |  |
| --- | --- | --- |
| Дата согласования первого варианта МКР | «\_14\_»\_марта\_2016г. | \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Е.А. Ерохина подпись научного руководителя |
| Дата согласования итогового варианта МКР | «\_19\_»\_мая\_\_\_2016г. | \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Е.А. Ерохина подпись научного руководителя |

**Аннотация**

В данной работе рассматривается реализация сетевой игры Гомоку. Гомоку – классическая японская игра. Ее суть заключается в составлении комбинации из 5 идущих подряд камней на поле, расчерченном линиями 15 x 15. Данная программа предоставляет возможность организации такой игры между двумя людьми посредством сети интернет.

Для ее написания использовались средства кроссплатформенного фреймворка Qt, поэтому клиентское приложение может быть использовано как на Windows, так и на GNU/Linux машине. В дальнейшем существует возможность его портирования на мобильные устройства под управлением операционной системы Android.

Работа содержит 18 страниц, 9 рисунков, включая 3 фрагмента кода. Суммарный объем кода для программы-сервера составил 803 строки, а для программы-клиента 882 строки.

**Annotation**

This coursework considers the implementation of a network game Gomoku. Gomoku is the classic Japanese game. The main idea of game is the preparation of a combination of 5 consecutive stones at the field, lined with lines of 15 x 15. The program provides the opportunity to organize a game between two people via the Internet.  
Tools of cross-platform framework Qt were used for writing it. This means the client application can be used as on Windows machine as on GNU / Linux machine. In the future there is the possibility of porting it to mobile devices running the Android operating system.

The coursework contains 18 pages, 9 pictures, including 3 code fragments. The total amount of code for server program is 803 strings and for client program is 882 strings.

**Оглавление**

[**Введение** 6](#_Toc450335751)

[**1.** **Серверная часть** 7](#_Toc450335752)

[1.1. База данных 7](#_Toc450335753)

[1.2. Организация соединения 8](#_Toc450335754)

[1.3. Серверная часть организации игровой сессии 10](#_Toc450335755)

[1.4. Алгоритмы игровой сессии 11](#_Toc450335756)

[**2.** **Клиентская часть** 13](#_Toc450335757)

[2.1. Организация запросов к серверу 13](#_Toc450335758)

[2.2. Интерфейс 13](#_Toc450335759)

[**3.** **Клиент-серверное взаимодействие** 16](#_Toc450335760)

[**Заключение** 18](#_Toc450335761)

[**Список использованных источников** 18](#_Toc450335762)

# **Введение**

Человечество за свою историю придумало немало интересных игр, и некоторые из них являлись не просто средством развлечения, а способствовали развитию логического и тактического мышления. Одна из таких игр – Гомоку. У этой игры существует множество поклонников, а также различных вариаций. Здесь представлена наиболее распространенная реализация с полем 15 x 15 без каких-либо дополнительных правил.

Программа позволяет играть как со случайным, так и с приглашенным по имени пользователя противником посредством сети интернет, обмениваться текстовыми сообщениями во время игры и хранить игровую статистику. На данном этапе приложение работоспособно на платформах Windows и GNU/Linux.

# **Серверная часть**

## База данных

Хранение и использование информации о пользователе, необходимой для предоставления игрового функционала, обеспечивается классом db. Основной единицей класса является список структур user, в полях которого хранится вся необходимая информация, такая как имя пользователя, пароль, количество игр и так далее. Список организован с помощью стандартного в Qt шаблонного класса QList и хранится в private поле класса для предотвращения прямого доступа к нему из других частей программы, потому что это может повлечь нежелательные изменения данных или даже их потерю.

Для работы с этим списком организованы публичные методы: чтение и запись данных о пользователях в файл, добавление пользователя, поиск записи по имени пользователя или указателю на класс QTcpSocket, который в данном случае выполняет роль дескриптора клиентского приложения, на котором авторизован запрашиваемый пользователь, сортировка записей по соотношению побед/игр.

В качестве алгоритма сортировки выбрана сортировка пузырьком, несмотря на его максимальную сложность . На самом деле сортировка используется только для получения первых 10 самых успешных игроков. Так как пузырек за один проход находит ровно один максимум, то для получения 10 таких требуется максимум проходов. То есть работа алгоритма завершается за постоянное время, а в случае, когда в базе данных менее 10 пользователей, сложность составит , где – количество записей в базе.

Последний метод, реализованный в классе db - это получение статистики для каждого пользователя. Данная функция генерирует строку, которая содержит информацию о пользователе, который запросил статистику, а также о первых 10 игроках.

## Организация соединения

Весь функционал, реализованный в программе-сервере, предоставлен классом server. Класс server унаследован от стандартного в Qt класса QTcpServer, который предоставляет высокоуровневый и в то же время кроссплатформенный интерфейс для работы с сокетами по протоколу TCP в качестве сервера. Этот протокол был выбран, так как он предоставляет устойчивое соединение, а также гарантирует доставку и правильный порядок доставки пакетов, в отличии от UDP протокола. Это важно, потому что если пакеты приходили бы в случайном порядке, то порядок игровых ходов был бы нарушен, и игра имела бы непредсказуемый характер.

Рассмотрим детальней, как конкретно происходит сетевое взаимодействие на стороне сервера. Как только к порту, который слушает сервер, будет произведено новое подключение, произойдет выброс сигнала newConnection. Этот сигнал соединен с реализованным в классе server слотом slotNewConnection. Внутри этого слота происходит получение указателя на объект класса QTcpSocket, который предоставляет интерфейс взаимодействия с только что подключившимся клиентом. Далее этот указатель соединяется с двумя другими слотами из server.

Первый - слот slotDisconected, вызывающийся при получении сигнала disconnected, который выбрасывается при отключении клиента от сервера. Данный слот предназначен для очистки памяти и перевода аккаунта в режим «Не в сети», если пользователь был авторизован. Другой слот slotReadClient (рис. 2) соединен с сигналом readyRead, который вызывается, если от клиента пришли какие-то данные. В связи с этим отпадает необходимость дополнительно где-то хранить указатели на все объекты QTcpSocket, подключенных клиентов, так как всегда можно обращаться к объекту, пославшему сигнал. Прежде чем начать рассмотрение чтения полученных данных, следует рассказать про то, как они были отправлены, потому что отправка и чтение данных реализовано практически одинаково, как на стороне клиента, так и на стороне сервера.

За отправку отвечает слот sendToClient (рис. 1). Он принимает два аргумента – указатель на QTcpSocket, который предоставляет клиент получателя, и строку с данными, которые мы хотим передать. Чтобы быть уверенными, что были получены все данные целиком, при отправке резервируются первые 16 бит сообщения, которые содержат размер отправляемых данных, после чего происходит непосредственно отправка полученного массива байт через механизмы класса QTcpSocket.

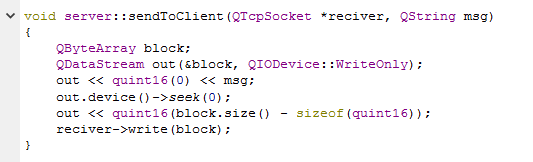


Рис.1. Слот sendToClient.

Таким образом, при чтении в первую очередь происходит проверка количества полученных байт. Как только это число будет больше или равно зарезервированных 2-х байт, считывается размер ожидаемого сообщения. Следующая проверка сравнивает количество готовых для чтения байт с числом, которое мы получили в результате предыдущей проверки, и, если оно совпадает, происходит чтение всего сообщения, которое в дальнейшем передается в функцию handler, которая обрабатывает все возможные запросы.

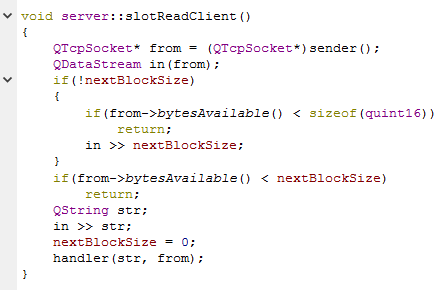


Рис. 2. Слот slotReadClient.

Типичный ответ сервера или запрос к нему строится по простой схеме: сначала указывается число, соответствующие запрашиваемой операции, а затем данные необходимые для ее выполнения. Для удобства числа определены с использованием препроцессора в отдельном заголовочном файле. Из полученного сообщения handler выделяет первую подстроку до пробела и преобразует ее в число, а затем отправляет в большую условную конструкцию, откуда дальше и происходит обработка запроса и генерация ответа.

## Серверная часть организации игровой сессии

Все, что касается обработки непосредственно запросов во время игры, вынесено в отдельный класс. Чтобы обеспечить должное быстродействие сервера при большом количестве игровых сессий, каждая из них обрабатывается в отдельном потоке. Средства Qt позволяют перенести все операции, связанные с каким-либо конкретным объектом класса QObject или унаследованного от него, в отдельный поток через методы класса QThread.

Как говорилось выше, все запросы обрабатываются handler, в том числе и все запросы к сессии. Однако, они сразу же передаются в другой handler, который уже является функцией-членом session.

Каждая сессия содержит указатели на структуры user в базе данных для быстрого обновления статистики, а также задает значения текущего камня в поле currentStone. Но наиболее важным элементом класса session является двухмерный массив, который представляет собой игровое поле, и, более того, это единственное игровое поле, используемое в сессии. Клиентское приложение не хранит и не проверяет ходы игроков. Все это осуществляется только на сервере и сделано с целью защиты от написания стороннего клиента для жульничества.

## Алгоритмы игровой сессии

Внутри класса session также описаны две функции, которые непосредственно обрабатывают игровой процесс.

Первая, checkTurn, получает в качестве аргументов строку - камень игрока, сделавшего ход, а также координаты на поле, где этот ход был сделан и обрабатывает три возможных ситуации.

- Ничья, то есть количество ходов достигло 255 и на поле не осталось свободных точек, куда можно поставить камень, но победная комбинация так и не была собрана. В этом случае сессия помечается как завершенная, а клиентам отправляется ответ о том, что достигнута ничья.

- Сделан ход, но победа не была достигнута. Тогда функция проверяет корректность хода, то есть обрабатывает случай, когда на заявленных координатах уже есть камень. В этом случает никакого результата клиентам возвращено не будет, но, если ход корректный, будут высланы координаты хода и символ игрока, который его сделал, а также переменная, хранящая символ игрока, который должен сделать текущий ход перейдет в противоположное значение.

- Сделан ход и достигнута победа. В такой ситуации сессия будет помечена как завершенная, игроку, сделавшему победных ход, будет выслано сообщение о победе, а его оппоненту сообщение о поражении.

Вторая функция - chekWin (рис. 3) - проверяет, является ли ход победным. Здесь определен счетчик и вызваны 8 циклов, которые «как лучи» расходятся от последнего сделанного хода. Суть алгоритма заключается в том, что как только на очередной линии прохода будут расставлены пять подряд идущих камней текущего игрока, функция вернет значение истина, в противном случае - ложь.

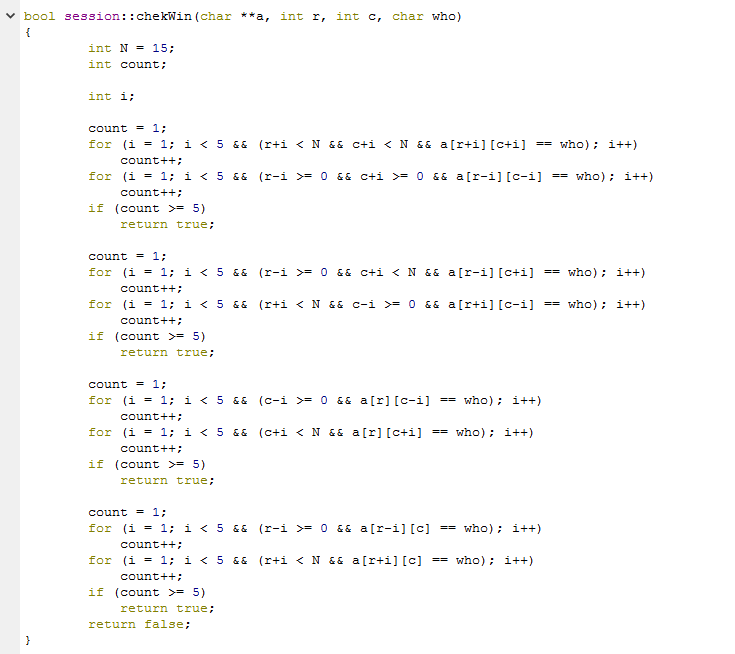


Рис. 3. Метод chekWin.

Пример работы алгоритма (рис. 4): 1) сделан ход, 2) радиально от него расходятся проверки и тут же завершаются, если камень не соответствует красному (на самом деле, обход каждого радиуса идет последовательно, чтобы не потерять непрерывность расставленных камней), 3) счетчик по диагонали достиг 5, алгоритм завершился, функция вернула значение Истина.

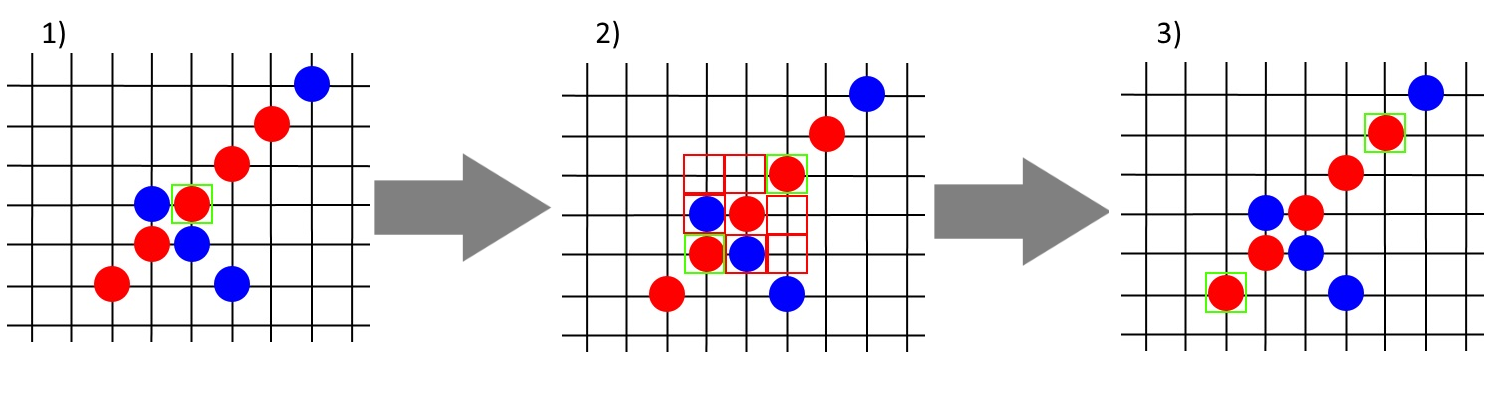


Рис. 4. Пример работы алгоритма.

# **Клиентская часть**

## Организация запросов к серверу

Если в серверном приложении главный класс server был унаследован от QTcpServer, то главный клиентский класс net - от QTcpSocket. Он содержит методы для отправки и получения данных, а также обработчик ответов. Вообще говоря, это два очень похожих по структуре класса, однако у них абсолютно разное назначение.

Типичная строка запроса к серверу имеет следующую структуру: *[число-идентификатор] + [данные, необходимые для выполнения запрашиваемой операции]*. Число-идентификатор - это определенная числовая константа, которая соответствует запрашиваемой операции. Все эти константы определены в отдельном, точно таком же, как и на сервере, заголовочном файле. Фреймворк Qt предоставляет удобные средства для потокового чтения данных из различных источников, в том числе и строк, поэтому на деле структурные единицы в строке отделяются друг от друга пробелами.

Структура любого запроса во время игры имеет практически такой же вид, как и обычный запрос: *[число-идентификатор, что запрос нужно передать в обработчик сессии] + [номер сессии] + [команда для сессионного обработчика] + [данные для выполнения запрашиваемой операции]*. В связи с тем, что каждая сессия на сервере обрабатывается независимо, число-идентификатор постоянно и соответствует команде для передачи обработки сообщения в локальный обработчик сессии, номер которой содержится в следующей подстроке, ограниченной пробелами. Оставшаяся часть сообщения отправляется в локальный обработчик для каждой конкретной сессии.

## Интерфейс

Весь интерфейс построен на уже готовых элементах фреймворка Qt. Главные окна: окно авторизации, основное меню игры, окно с игрой - предоставлены классами, унаследованными от QWidget, так как он является основным для реализации вообще какого-либо графического интерфейса в Qt; дополнительные окна, такие как окна с правилами и информацией о программе, предоставлены классами-потомками QDialog, который уже является унаследованным от QWidget, но имеет более узкую специализацию – создание диалоговых окон.

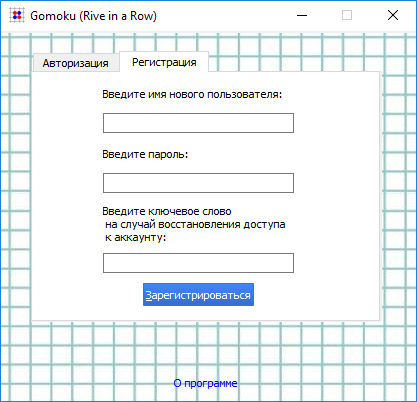
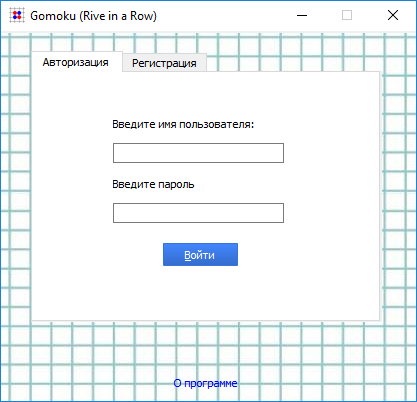


Рис. 5. Окно авторизации и регистрации пользователя.

Из главного меню программы (рис. 6) можно открыть окно с правилами игры, получить статистику, пригласить пользователя по его имени или ждать случайного игрока.

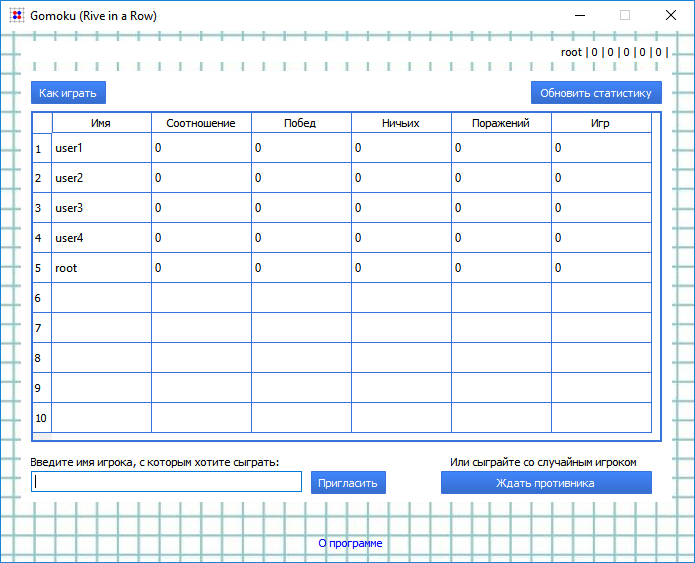


Рис. 6. Главное меню программы.

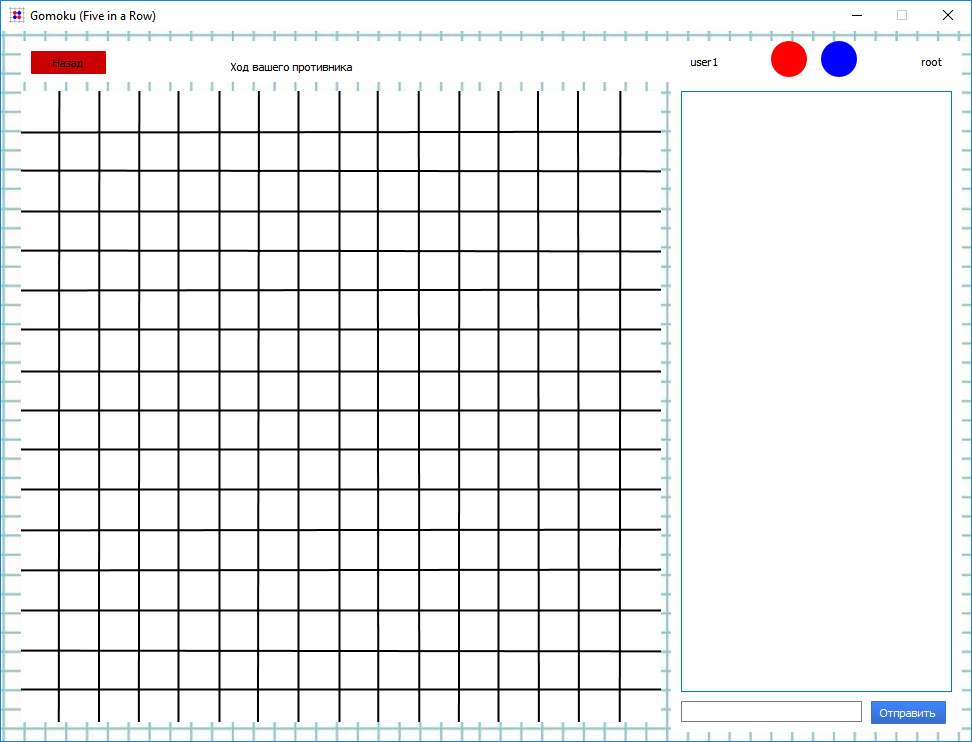
Во время самой игры окно выглядит следующим образом (рис. 7): справа от игрового поля – текстовый чат с противником, над ним - имена игроков; прямо по центру выше игрового поля – статусная строка, говорящая, чьим является текущий ход, а также сообщающая о завершении игры.

Рис. 7. Окно с запущенной игрой.

Следует отметить, что оформление элементов полностью сделано с использованием Qt Style Sheet – аналогом Cascade Style Sheet (средств создания дизайна веб-документов), применимым к элементам Qt. Поэтому пользователь сам, при желании, может изменить оформление. Для этого необходимо запустить программу-клиент из терминала с параметром *-customCss ./theme.css*, где *./theme.css* - это путь до пользовательского css-файла.

# **Клиент-серверное взаимодействие**

На рисунке 8 предоставлена общая схема взаимодействия объектов реализованных классов между собой.

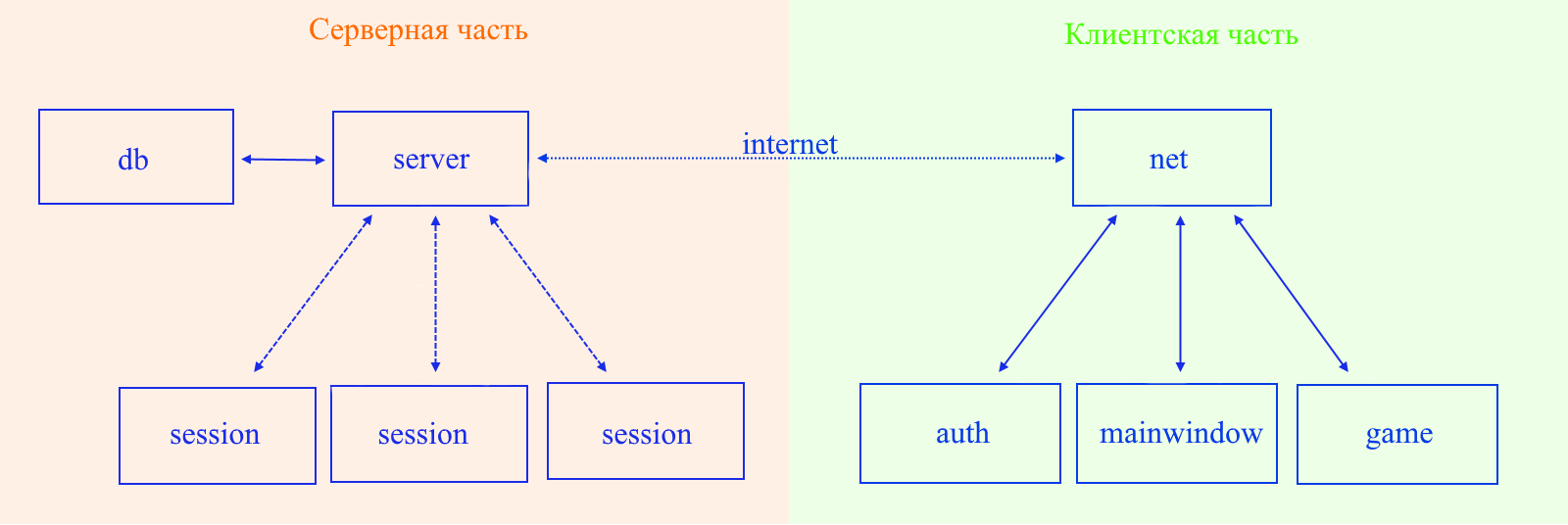


Рис. 8. Связь объектов классов.

Внутри основного класса сервера server создается объект класса db, который предоставляет удобный интерфейс для работы с базой данных. Помимо этого, на каждую игровую сессию создается объект класса session, который помещается в отдельный поток, на рисунке это обозначено штрих-линией. Так как классы server в серверном приложении и net в клиентском унаследованы от сетевых классов Qt, то именно между их объектами организуется Интернет-соединение. В клиентском приложении объекты классов, предоставляющих графический интерфейс, не имеют связи ни между собой, ни с интернетом. Взаимодействие между ними происходит полностью через класс net.

Рассмотрим это взаимодействие на примере авторизации (рис. 9). Запрос помечен красными линиями, ответ зелеными. Как только пользователь попытался авторизоваться в окне авторизации, объект класса auth передает в net строку с запросом. В классе net эта строка передается на сервер с помощью описанного в пункте 1.2 механизма. На сервере происходит разбор строки, после которого необходимая информация из нее передается в метод класса db для проверки данных. Если авторизация произошла успешно, в server генерируется ответная строка и все также отправляется клиенту. Как только положительный ответ на авторизацию придет клиенту, класс net удалит из памяти объект класса auth, а взамен создаст объект mainwindow, который содержит в себе главное окно программы.

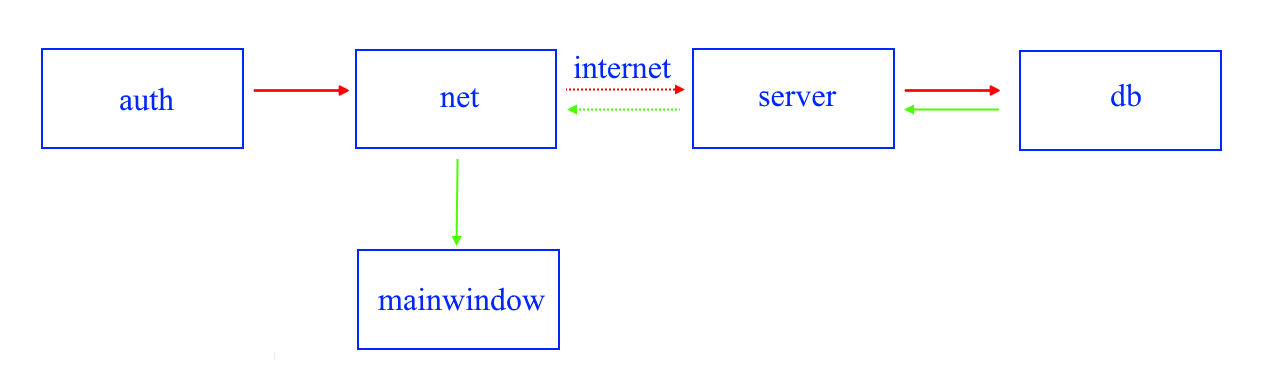


Рис. 9. Схема процесса авторизация

# **Заключение**

В конечном счете, результатом курсовой работы является работоспособное, готовое к массовому использованию клиент-серверное приложение, предоставляющее все необходимые средства для проведения комфортной игры между двумя игроками через сеть интернет.

# **Список использованных источников**

1. Гомоку Википедия. URL: [https://ru.wikipedia.org/wiki/Гомоку](https://ru.wikipedia.org/wiki/%D0%93%D0%BE%D0%BC%D0%BE%D0%BA%D1%83) (Дата обращения 22.04.2016)
2. Qt Documentation. URL: <http://doc.qt.io/>
3. Шлее М. Qt 5.3. Профессиональное программирование на C++. – СПб.: БХВ-Петербург, 2015. – 928с.: ил. – (В подлиннике)