СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 5](#_Toc533683672)

[**1** ОБЗОР ЛИТЕРАТУРЫ 6](#_Toc533683673)

[**2** СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ 10](#_Toc533683674)

[**3** ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ 13](#_Toc533683675)

[**3.1** Модуль интерфейса клиента. 13](#_Toc533683676)

[**3.1.1** Класс Dialog. 13](#_Toc533683677)

[**3.1.2** Класс Dialog интерфейса сервера. 17](#_Toc533683678)

[**3.1.3** Класс клиента MyClient. 19](#_Toc533683679)

[**3.1.4** Класс MyServer. 21](#_Toc533683680)

[**4** РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ 26](#_Toc533683681)

[**4.1** Клиент чата 26](#_Toc533683682)

[**4.2** Серверная часть 30](#_Toc533683683)

[**5** ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ 31](#_Toc533683684)

[**6** РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 34](#_Toc533683685)

[**6.1** Требования к аппаратному и программному обеспечению 34](#_Toc533683686)

[**6.2** Руководство по использованию программного средства. 34](#_Toc533683687)

[ЗАКЛЮЧЕНИЕ 40](#_Toc533683688)

[СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ 41](#_Toc533683689)

[Приложение А 42](#_Toc533683690)

[Приложение Б 44](#_Toc533683691)

ВВЕДЕНИЕ

Видеоигра – это игра с использованием изображений, сгенерированных электронной аппаратурой. Другими словами, видеоигра является электронной игрой, которая базируется на взаимодействии человека и устройства посредством визуального интерфейса, например телевизора или монитора компьютера.

В данном курсовом проекте была разработана двухмерная мультиплеерная игра. Игра написана на языке С++ с использованием спецификации OpenGL для вывода графики на экран. Проект содержит как клиентскую часть игры, так и сервер. Игра имеет низкие системные требования, что позволит запустить ее на старых компьютерах. Геймплей игры легок в освоении, что позволит даже неопытным игрокам сразу приступить к комфортной игре.

# **1** ОБЗОР ЛИТЕРАТУРЫ

Игра разработана с помощью среды разработки Microsoft Visual Studio 2019 на языке C++. За вывод текстур на экран отвечает спецификация OpenGL 3.3. Данная игра является представителем жанра платформера с элементами шутера и с мультиплеерной составляющей.

C++ — [компилируемый](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%B8%D0%BB%D0%B8%D1%80%D1%83%D0%B5%D0%BC%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), [статически типизированный](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D0%B0%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D1%82%D0%B8%D0%BF%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F) [язык программирования](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) общего назначения.

Поддерживает такие [парадигмы программирования](https://ru.wikipedia.org/wiki/%D0%9F%D0%B0%D1%80%D0%B0%D0%B4%D0%B8%D0%B3%D0%BC%D1%8B_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), как [процедурное программирование](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D0%B4%D1%83%D1%80%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), [объектно‑ориентированное программирование](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), [обобщённое программирование](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D0%BE%D0%B1%D1%89%D1%91%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5). Язык имеет богатую стандартную библиотеку, которая включает в себя распространённые контейнеры и алгоритмы, ввод-вывод, регулярные выражения, поддержку многопоточности и другие возможности. C++ сочетает свойства как [высокоуровневых](https://ru.wikipedia.org/wiki/%D0%92%D1%8B%D1%81%D0%BE%D0%BA%D0%BE%D1%83%D1%80%D0%BE%D0%B2%D0%BD%D0%B5%D0%B2%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), так и [низкоуровневых языков](https://ru.wikipedia.org/wiki/%D0%9D%D0%B8%D0%B7%D0%BA%D0%BE%D1%83%D1%80%D0%BE%D0%B2%D0%BD%D0%B5%D0%B2%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F). В сравнении с его предшественником — языком [C](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)), — наибольшее внимание уделено поддержке [объектно-ориентированного](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) и [обобщённого программирования](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D0%BE%D0%B1%D1%89%D1%91%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5).

C++ широко используется для разработки программного обеспечения, являясь одним из самых популярных языков программирования. Область его применения включает создание [операционных систем](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0), разнообразных прикладных программ, [драйверов](https://ru.wikipedia.org/wiki/%D0%94%D1%80%D0%B0%D0%B9%D0%B2%D0%B5%D1%80) устройств, приложений для встраиваемых систем, высокопроизводительных серверов, а также игр.

Microsoft Visual Studio — линейка продуктов компании [Microsoft](https://ru.wikipedia.org/wiki/Microsoft), включающих [интегрированную среду разработки](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D0%B3%D1%80%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D1%80%D0%B5%D0%B4%D0%B0_%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B8) программного обеспечения и ряд других инструментальных средств. Данные продукты позволяют разрабатывать как [консольные](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BA%D1%81%D1%82%D0%BE%D0%B2%D1%8B%D0%B9_%D0%B8%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8F) [приложения](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B8%D0%BA%D0%BB%D0%B0%D0%B4%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5), так и приложения с [графическим интерфейсом](https://ru.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D1%84%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B8%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8F), в том числе с поддержкой технологии [Windows Forms](https://ru.wikipedia.org/wiki/Windows_Forms), а также [веб‑сайты](https://ru.wikipedia.org/wiki/%D0%A1%D0%B0%D0%B9%D1%82), [веб‑приложения](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5), [веб‑службы](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D0%BB%D1%83%D0%B6%D0%B1%D0%B0) как в [родном](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%88%D0%B8%D0%BD%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4), так и в [управляемом](https://ru.wikipedia.org/wiki/%D0%A3%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D1%8F%D0%B5%D0%BC%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4) кодах для всех платформ, поддерживаемых [Windows](https://ru.wikipedia.org/wiki/Windows), [Windows Mobile](https://ru.wikipedia.org/wiki/Windows_Mobile), [Windows CE](https://ru.wikipedia.org/wiki/Windows_CE), [.NET Framework](https://ru.wikipedia.org/wiki/.NET_Framework), [Xbox](https://ru.wikipedia.org/wiki/Xbox), [Windows Phone](https://ru.wikipedia.org/wiki/Windows_Phone) [.NET Compact Framework](https://ru.wikipedia.org/wiki/.NET_Compact_Framework) и [Silverlight](https://ru.wikipedia.org/wiki/Silverlight).

Visual Studio включает в себя [редактор исходного кода](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%B4%D0%B0%D0%BA%D1%82%D0%BE%D1%80_%D0%B8%D1%81%D1%85%D0%BE%D0%B4%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BA%D0%BE%D0%B4%D0%B0) с поддержкой технологии [IntelliSense](https://ru.wikipedia.org/wiki/IntelliSense) и возможностью простейшего [рефакторинга кода](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D1%84%D0%B0%D0%BA%D1%82%D0%BE%D1%80%D0%B8%D0%BD%D0%B3). Встроенный [отладчик](https://ru.wikipedia.org/wiki/Microsoft_Visual_Studio_Debugger) может работать как отладчик уровня исходного кода, так и отладчик машинного уровня. Остальные встраиваемые инструменты включают в себя редактор форм для упрощения создания графического интерфейса приложения, веб-редактор, дизайнер [классов](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D1%81%D1%81_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) и дизайнер [схемы базы данных](https://ru.wikipedia.org/wiki/%D0%A1%D1%85%D0%B5%D0%BC%D0%B0_%D0%B1%D0%B0%D0%B7%D1%8B_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85). Visual Studio позволяет создавать и подключать сторонние дополнения ([плагины](https://ru.wikipedia.org/wiki/%D0%9F%D0%BB%D0%B0%D0%B3%D0%B8%D0%BD)) для расширения функциональности практически на каждом уровне, включая добавление поддержки систем [контроля версий исходного кода](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%83%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F_%D0%B2%D0%B5%D1%80%D1%81%D0%B8%D1%8F%D0%BC%D0%B8) (как, например, [Subversion](https://ru.wikipedia.org/wiki/Subversion) и [Visual SourceSafe](https://ru.wikipedia.org/wiki/Microsoft_Visual_SourceSafe)), добавление новых наборов инструментов (например, для редактирования и визуального проектирования кода на [предметно-ориентированных языках программирования](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B5%D0%B4%D0%BC%D0%B5%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA)) или инструментов для прочих аспектов [процесса разработки программного обеспечения](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81_%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B8_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F) (например, клиент Team Explorer для работы с [Team Foundation Server](https://ru.wikipedia.org/wiki/Team_Foundation_Server)).

OpenGL (Open Graphics Library) - спецификация, определяющая платформонезависимый (независимый от языка программирования) программный интерфейс для написания приложений, использующих двумерную и трёхмерную компьютерную графику.

Для OpenGL существует большое количество различных библиотек. В этом проекте использовалась библиотека GLEW, а также математическая библиотека GLM.

OpenGL Extension Wrangler Library (GLEW) - кроссплатформенная [библиотека](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D0%B1%D0%BB%D0%B8%D0%BE%D1%82%D0%B5%D0%BA%D0%B0_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) на [C](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F))/[C++](https://ru.wikipedia.org/wiki/C%2B%2B), которая упрощает запрос и загрузку [расширений OpenGL](https://ru.wikipedia.org/wiki/OpenGL#%D0%A0%D0%B0%D1%81%D1%88%D0%B8%D1%80%D0%B5%D0%BD%D0%B8%D1%8F). GLEW обеспечивает эффективные [run-time](https://ru.wikipedia.org/wiki/%D0%92%D1%80%D0%B5%D0%BC%D1%8F_%D0%B2%D1%8B%D0%BF%D0%BE%D0%BB%D0%BD%D0%B5%D0%BD%D0%B8%D1%8F_(%D1%84%D0%B0%D0%B7%D0%B0_%D0%B6%D0%B8%D0%B7%D0%BD%D0%B5%D0%BD%D0%BD%D0%BE%D0%B3%D0%BE_%D1%86%D0%B8%D0%BA%D0%BB%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D1%8B)) механизмы для определения того, какие OpenGL расширения поддерживаются на целевой платформе. Все расширения OpenGL размещаются в одном [заголовочном файле](https://ru.wikipedia.org/wiki/%D0%97%D0%B0%D0%B3%D0%BE%D0%BB%D0%BE%D0%B2%D0%BE%D1%87%D0%BD%D1%8B%D0%B9_%D1%84%D0%B0%D0%B9%D0%BB), который автоматически генерируется из официального списка расширений. GLEW доступна на множестве [операционных систем](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0), включая [Windows](https://ru.wikipedia.org/wiki/Microsoft_Windows), [Linux](https://ru.wikipedia.org/wiki/Linux), [Mac OS X](https://ru.wikipedia.org/wiki/Mac_OS_X), [FreeBSD](https://ru.wikipedia.org/wiki/FreeBSD), [IRIX](https://ru.wikipedia.org/wiki/IRIX) и [Solaris](https://ru.wikipedia.org/wiki/Solaris).

OpenGL Matmatics (GLM) - это математическая библиотека C ++ предназначенная для графического программного обеспечения, основанного на спецификациях OpenGL Shading Language (GLSL).

GLM предоставляет классы и функции, разработанные и реализованные с теми же соглашениями и функциональными возможностями именования, что и GLSL, так что любой, кто знает GLSL, может использовать GLM также и в C ++.

Эта библиотека прекрасно работает с OpenGL, но также обеспечивает совместимость с другими сторонними библиотеками и SDK. Это хороший кандидат на рендеринг программного обеспечения (трассировка лучей / растеризация), обработку изображений, физическое моделирование и любой контекст разработки, для которого требуется простая и удобная математическая библиотека.

Windows Sockets API (WSA), название которого было укорочено до Winsock. Это техническая спецификация, которая определяет, как сетевое программное обеспечение Windows будет получать доступ к сетевым сервисам, в том числе TCP/IP. Он определяет стандартный интерфейс между клиентским приложением (таким как FTP-клиент или веб-браузер) и внешним стеком протоколов TCP/IP.

# **2** СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ

Игра состоит преимущественно из двух основных частей: клиента и сервера. Сам же клиент состоит из графического движка, написанного с помощью OpenGL, и логики игры.

Изучив теоретические аспекты разрабатываемой системы и выработав список требований, необходимых для разработки системы, разбиваем систему на функциональные блоки(модули). Это необходимо для обеспечения гибкой архитектуры. Такой подход позволяет изменять или заменять модули без изменения всей системы в целом.

В разрабатываемом приложении можно выделить несколько блоков:

- блок ввода текстур (относящийся к клиенту);

- блок приема и отправки команд (относящийся к клиенту);

- блок приема команд с клавиатуры (относящийся к клиенту);

- блок проверки состояния игрока (относящийся к клиенту);

- блок отрисовки коллизии и текстур карты (относящийся к клиенту);

- блок отрисовки и логики интерфейса (относящийся к клиенту);

- блок логики стрельбы (относящийся к клиенту);

- блок передачи команд и сообщений (относящийся к серверу);

- блок приема команд и подключений (относящийся к серверу).

Структурная схема, иллюстрирующая перечисленные блоки и связи между ними приведена на рисунке 2.1.

Каждый модуль выполняет свою задачу. Чтобы система работала каждый модуль взаимодействует с другими модулями путем обмена данными.

Рассмотрим функциональные блоки приложения.

Блок ввода текстур в данном курсовом проекте является также и движком игры. Отвечает за ввод и обработку текстур, помещение текстуры в соответствующий объект, для дальнейшей работы с текстурой. Реализация этих операций осуществляется в классах IndexBuffer, VertexBuffer, VertexArray, VertexBufferLayout, Texture, Shader и Renderer.

Блок приема и отправки команд отвечает за осуществление соединение клиента и сервера, отправку пакетов с данными на сервер, а также прием этих пакетов от самого сервера для передачи этих данных клиенту. Работа с сетью организована с помощью библиотеки Winsock2.h.

Блок приема команд с клавиатуры регистрацию нажатий клавиш на клавиатуре и, следовательно, реакцию на эти нажатия в игре. Отвечает преимущественно за управление персонажем: перемещение, стрельба, взаимодействие с объектами. Осуществляются эти действия в классе Player1, при этом используются параметры классов HUD, Fire и объектов класса Map.

Блок проверки состояния игрока предназначен регистрации попаданий по игроку и отправки соответствующих данных в класс HUD. Осуществляет изменение состояния игрока “жив” – “мертв”. Эти действия осуществляются в классе Player1.

Блок отрисовки коллизии и текстур карты реализуется в классе Map. В этом классе осуществляется отрисовка коллизии карты, которая нужна для отслеживания объектов карты, отслеживания положения игроков на карте, нахождения пули в пространстве, положение противника, данные о котором поступают с сервера. Данные блок также осуществляет прием текстур из блока вывода текстур и осуществляет расстановку текстур на экране (объекты карты, противник, пули). Также содержит параметры передачи данных для других классов.

Блок отрисовки и логики интерфейса реализована в классе HUD. Этот блок предназначен для отображения интерфейса на экране игрока, а также отслеживание состояния здоровья игрока, в следствии чего отменялась текстура интерфейса.

Блок логики стрельбы осуществлен в классе Fire. В этом классе осуществлены все функции стрельбы в этой игре: урон, координаты пули, которые передаются в класс Map, отрисовка пули на экране, функции передачи параметров другим классам.

Блок передачи команд и сообщений на сервере предназначен для передачи команд и сообщений, поступивших от одного из клиентов, для последующей отправки другим клиентам.

Блок приема команд и подключений на сервера предназначен для приема подключений и команд, которые поступают от клиентов. В этом блоке осуществляется учет поступивших клиентов и их запись, а также осуществляется подготовка для передачи пришедших от них пакетов данных.

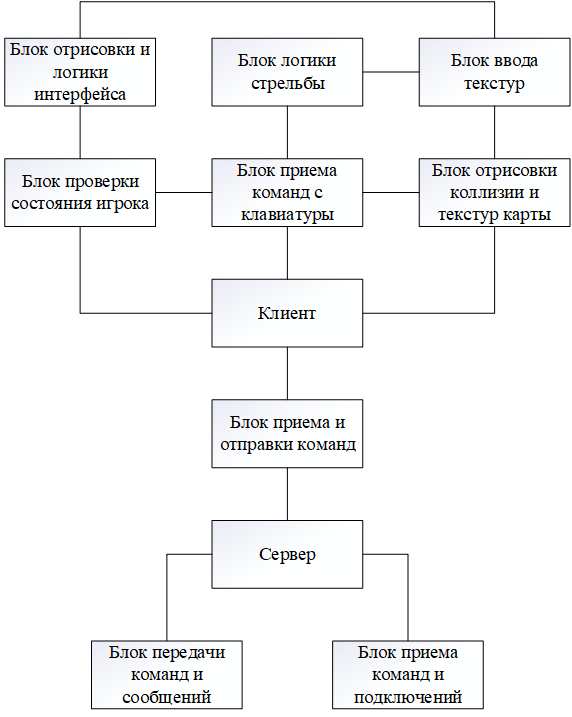


Рисунок 2.1 – Структурная схема чат-программы

# **3** ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

Рассмотрим подробно функционирование программы. Для этого проведем анализ основных модулей программы и рассмотрим их зависимости. А также проанализируем все модули, которые входят в состав кода программы, и рассмотрим назначение всех методов и переменных этих модулей.

В разрабатываемой чат программе выделим 4 модуля:

* модуль интерфейса клиента;
* модуль клиента;
* модуль сервера;
* модуль интерфейса сервера.

Изначально пользователь попадает в окно программы клиента, которое реализуется классом Dialog*.* Основное взаимодействие пользователя с интерфейсом программы осуществляется через этот класс. Аналог этого класса имеется и в серверной части клиента программы. Работа клиента осуществляется через класс MyClient, который находится в составе серверной части программы, т.к. основные функции чата работают через сервер, без авторизации через сервер полноценная работа программы не возможна. Работа самого сервера осуществляется через класс MyServer.

## **3.1** Модуль интерфейса клиента.

### **3.1.1** Класс Dialog.

Работа интерфейса клиента осуществлена при помощи класса Dialog. Данный класс наследует параметры класса QDialog являющегося частью библиотеки QDialog включенную в фреймворк Qt*.* Для обеспечения сетевых коммуникаций используются сокеты. Сокет - это конечная точка сетевых коммуникаций. Сокеты будут использоваться в данной программе, чтобы регистрировать сигналы и запускать необходимые функции. Для создания объекта сокета \_sok используется класс QTcpSocket. В пространство имен Uiдобавляем класс Dialogдля создания объекта типа Ui и взаимодействия с элементами интерфейса. Создается объект \*ui. Далее в данном классе присутствуют следующие методы для работы с клиентом:

* void onSokConnected();
* void onSokDisconnected();
* void onSokDisplayError(QAbstractSocket::SocketError socketError);
* void on\_pbConnect\_clicked();
* void on\_pbDisconnect\_clicked();
* void on\_cbToAll\_clicked();
* void on\_pbSend\_clicked();
* void AddToLog(QString text, QColor color = Qt::black);

А также присутствует конструктор и деструктор. Данный конструктор принимает значения типа QWidget и наследуется от класса QDialog. Внутри самого класса происходит запуск интерфейса и передача ему параметров класса Dialog при помощи ui->setupUi(this). Также в классе Dialog был создан объект типа QString, содержимое этого объекта мы обнуляем, чтобы не возникло никакого мусора. Затем выделяется память под сокет \_sok. При помощи команды connect осуществляется подключение сокета к различным видам сигналов и включение функций объекта Dialog при срабатывании этих сигналов. Список сигналов и функций, привязанных к ним:

* connect (\_sok, SIGNAL(readyRead()), this, SLOT(onSokReadyRead()));
* connect (\_sok, SIGNAL(connected()), this, SLOT(onSokConnected()));
* connect (\_sok, SIGNAL(disconnected()), this, SLOT(onSokDisconnected());
* connect (\_sok, SIGNAL, (error(QAbstractSocket::SocketError)), this, SLOT(onSokDisplayError(QAbstractSocket::SocketError)));

Деструктор содержит команду удаления объекта ui.

Рассмотрим сигналы и функции, которые они вызывают. Сигнал readyRead() реагирует на прием сигналов от сервера. Она вызывает функцию onSokReadyRead(), которая создает объект inтипа QDataStream и передает в него данные сокета \_sok. Класс QDataStream позволяет отправлять данные из клиента на сервер. Передача данных осуществляется путем передачи блоков данных, первые 2 байта которых – это размер, далее 3 байта – это команда серверу. Полученная команда обрабатывается и при помощи switсh() выполняется набор инструкций исходя из кода команды. Команды берутся из библиотеки myclient.h, которая находится в программе сервера, она будет рассмотрена позже.

Рассмотрим инструкцию каждой команды:

* comUsersOnline – отправляет при помощи функции AddToLog() сообщение на экран о добавленном пользователе. Функция AddToLog() будет рассмотрена ниже. После этого пользователю становится доступна кнопка отправки сообщения. Создается объект users типа QString и в него помещается имя подключенного пользователя, затем создается объект l типа QStringList и в него помещается объект типа QString, этот список нужен для содержания всех пользователей, находящихся в сети в данный момент. Затем этот список отправляется на экран интерфейса в Поле пользователей при помощи ui->lwusers->addItems(l);
* comPublicServerMessege *–* как понятно из названия, эта команда предназначена для отправки сообщения в чат от имени сервера. В нем создается объект message типа QString, в него помещается само сообщение и выводится в Поле сообщений при помощи функции AddToLog();
* comMessageToAll – предназначена для отправки публичного сообщения в Поле сообщений. Создается объект QString, в него помещается имя отправившего сообщение, затем создается объект QString message и в него помещается само сообщение. При помощи функции AddToLog() сообщение попадает на экран;
* comMessageToUsers – предназначена для отправки приватных сообщений от пользователей. Создается объект QString, в него помещается имя отправившего сообщение, затем создается объект QString message и в него помещается само сообщение. При помощи функции AddToLog() сообщение попадает на экран;
* comPrivateServerMessage - предназначена для отправки приватных сообщений от сервера. Создается объект QString, в него помещается имя отправившего сообщение, затем создается объект QString message и в него помещается само сообщение. При помощи функции AddToLog() сообщение попадает на экран;
* comUserJoin – отправляет сообщение о присоединении пользователя к чату в Поле сообщений. Создается объект типа QString name и в него помещается имя пользователя при помощи ui->lwUsers->addItem(name), затем имя помещается в список пользователей и выводит сообщение о нем при помощи AddToLog()*.*
* comUserLeft – уведомляет об отключении пользователя от сервера. Создается объект типа QString, в него помещается имя покинувшего чат, затем производится поиск этого имени в Поле пользователей и удаляется из него, при этом уведомляет всех пользователей о его отключении через функцию AddToLog()*;*
* comErrNameIvalid *–* предназначена для выведения сообщения об ошибке, в случае если введенный пользователем “никнейм” уже есть в системе. Сообщение выводится при помощи AddToLog() затем происходит принудительное отключение клиента через \_sok->disconnectFromHost();
* comErrNameUsed *-* предназначена для выведения сообщения об ошибке, в случае если введенный пользователем “никнейм” уже используется в системе. Сообщение выводится при помощи AddToLog() затем происходит принудительное отключение клиента через \_sok->disconnectFromHost();
* comServerClose - предназначена для выведения сообщения об ошибке, в случае если сервер прекратил свою работу. Сообщение выводится при помощи AddToLog() затем происходит принудительное отключение клиента через \_sok->disconnectFromHost();

Сигнал connected() вызывается при подключении пользователя к серверу. Этот сигнал вызывает функцию onSokConnected(). Эта функция меняет состояния кнопок Connect и Disconnect, выводит сообщение о подключении к соответствующему адресу при помощи AddToLog(), создается объект QDataStream, который будет использоваться для передачи данных на сервер. Для начала на первые 2 байта помещаем цифру 0 с типом данных quint16, этим способом задается размер, поначалу он равен 0, затем передается команда comAutchReq для класса MyClient, уже после этого отправляется на сервер имя пользователя. Указатель объекта QDataStream перемещается в начало и помещается на это место размер сообщения.

Сигнал disconnected() вызывается при отключении сервера. Этот сигнал вызывает функцию onSokDisconnected(*)*. В этой функции осуществляется переключение кнопок Connect и Disconnect в обратные состояния, переключается состояние кнопки Send в режим false, затем очищается список пользователей. Заканчивается функция выводом сообщения об отключении от сервера при помощи функции AddToLog()*.*

Далее следует описания функций, которые включаются при взаимодействии с интерфейсом.

При помощи функции on\_pbConnect\_clicked() осуществляется подключение к серверу. В эту функцию передается адрес и порт сервера.

При помощи функции on\_pbDisconnect\_clicked() осуществляется отключение от сервера.

При помощи функции on\_cbToAll\_clicked() осуществляется реакция на снятие, либо установки галочки возле ячейки To All, если галочка установлена, то меняется состояние на true, а текст кнопки отправки сообщений меняется на To All, если же галочку убрать, то текст кнопки отправки сообщений меняется на Send To Slected.

При нажатии на кнопку отправки сообщений создается объект QDataStream, который будет использоваться для передачи данных на сервер. Для начала на первые 2 байта помещаем цифру 0 с типом данных quint16, этим способом задается размер, поначалу он равен 0, затем идет проверка ячейки ToAll, если галочка стоит, то отправляется команда comMessageToAll, если же галочки нет, то отправляется команда comMessageToUsers. В случае второго варианта выполняется следующее, идет проверка выбранного пользователя, его имя копируется в объект типа QString и отправляется на сервер, а уже после отправляется сообщение. Указатель объекта QDataStream перемещается в начало и помещается на это место размер сообщения.

Функция AddToLog() выполняет отправку сообщений в Поле сообщений. Эта функция принимает два параметра, текст типа QString и код цвета текста типа QColor*.* При помощи ui->lwLog->insertItem(0, QTime::currentTime().toString+” ” + text) осуществляется вывод на экран сообщения со временем, а затем при помощи ui->lwLog->item(0)->setTextColor(color) устанавливается цвет текста.

### **3.1.2** Класс Dialog интерфейса сервера.

Работа клиента сервера реализована при помощи класса Dialog, однако сам класс содержит другие функции. Также, как и предыдущий класс, класс Dialog наследует класс QDialog. Сам же класс Dialog добавлен в пространство имен Ui.

В данном классе создается объект \*ui. Создается объект \*\_serv типа MyServer. Данный класс необходим, чтобы использовать основные функции сервера и отображать их на экране.

Данный класс содержит следующие функции:

* конструктор explicit Dialog(QWidget \*parent = 0);
* деструктор ~Dialog();
* void addToLog(QString text, QColor color);
* void messageFromGui(QString message, const QStringList &users);
* void onAddUserToGui(QString name);
* void onRemoveUserFromGui(QString name);
* void onMessageToGui(QString message, QString from, const QStringList &users);
* void onAddLogToGui(QString string, QColor color);
* void on\_pbSend\_clicked();
* void on\_cbToAll\_clicked();
* void on\_pbStartStop\_toggled(bool checked);

Рассмотрим подробнее эти функции. Также, как и в конструкторе клиента, данный конструктор имеет входные параметры типа QWidget, в который передаются данные типа QDialog. Под объект \_serv выделяется память типа MyServer и передаются параметры класса Dialog, подробнее о конструкторе класса MyServer будет написано ниже. При помощи функции connect() осуществляется сокетов с сигналами и функциями. Следом осуществляется попытка автоматического запуска сервера по заданным параметрам, как правило введен IP-адрес и порт по умолчанию. В случае успеха запуска выводится соответствующее сообщение.

Деструктор данного класса содержит функцию удаления ui.

Функция onAdduserToGui(), как понятно из названия, предназначена для вывода списка пользователей в Поле пользователей, в него передается параметр типа QString, который содержит “никнейм” пользователя. Осуществляется это благодаря ui->lwUsers->addItem(name)*,* затем на экран выводится сообщение о подключенном пользователе и выделяет сообщение соответствующим цветом.

Функция onRemoveUserFromGui() предназначена для удаления пользователя из Поля пользователей в случае его отключения от сервера.

Функция onMessageToGui() предназначена для вывода сообщения от пользователя, в эту функцию поступают данные типа QString message*,* QString from, *QStringList &users*. Эти данные необходимы для отправки самого сообщения, от кого пришло это сообщения, и, если сообщение было отправлено приватно, кому адресовано это сообщение. В данном случае сервер способен видеть все сообщения, отправленные пользователями, в том числе и приватные.

Функция on\_pbSend\_clicked() предназначена для выполнения определенных инструкций, при нажатии на кнопку Отправки сообщения. В начале осуществляется проверка на наличие пользователей онлайн, если же пользователей на сервере нет, то при помощи класса QMessageBox::information выводится сообщения No clients connected. Создается объект типа QStringList. Осуществляется проверка на состояния ячейки параметра отправки сообщений, если же стоит галочка, то сообщение предназначено для всех пользователей, если же для кого-то конкретно пользователя, то галочки нет. Затем происходит отправка сообщения пользователям, в Поле сообщений сервера пишется Sended pubic server message, если же был выбран какой-то из пользователей и сообщение предназначается только ему, то выводится сообщение Sended private serve message to и дальше указывается имя пользователя. Уведомления об отправке сообщения осуществляется при помощи функции addToLog(), которая является аналогом такой же функции для клиента.

Следом идет функция on\_cbToAll\_clicked(). Эта функция предназначена для проверки состояния кнопки параметра вывода сообщений, если установлена галочка, то кнопка отправки сообщений получает имя Send *To All*, если же галочки нет, то пишется Send To Selected.

Функция on\_pbStartStop\_toggled() предназначенная для осуществления определенных действий после нажатии кнопки Start/Stopserver. В него передается параметр состояния нажатия кнопки. Если кнопка была нажата, то выводится сообщение об остановке сервера и указывается его адрес и порт, затем при помощи функции \_serv->doCloseServer() осуществляется закрытие сервера, а имя кнопки меняется на Start server. Если же была нажата кнопка Start server, то создается объект типа QHostAddress, осуществляется проверка адреса, если с адресом что-то не так, то при помощи функции addToLog() выводится сообщение о неверном адресе. Если же с адресом все в порядке, то осуществляется следующая проверка, происходит попытка запуска сервера с этим адресом при помощи \_serv->doStartServer(), если все хорошо, то выводится сообщение об успешном запуске и изменяется имя кнопки на Stop server. Если же что-то произошло не так, то выводится сообщение об ошибке.

Функция addToLog() осуществляет вывод сообщения в Поле сообщений. В функцию передаются параметры типа QString и QColor, которые предназначены для передачи сообщения и модификатора цвета для этого текста.

### **3.1.3** Класс клиента MyClient.

В сервере предусмотрено наличие класса, который обеспечивает работу клиента. Данный класс называется MyClient. Этот класс наследуется класс QObject и содержит в себе команды для взаимодействия клиента с сервером. Присутствует конструктор и деструктор, также содержит следующие функции:

* конструкторexplicit MyClient(int desc, MyServer \*serv, QObject \*parent = 0);
* деструктор~MyClient*;*
* *void setName(QString name) {\_name = name;};*
* *QString getName() const {return \_name;};*
* *bool getAutched() const {return \_isAutched;};*
* *void doSendCommand(quint8 comm) const;*
* *void doSendUsersOnline() const;*
* *void addUserToGui(QString name);*
* *void removeUserFromGui(QString name);*
* *void removeUser(MyClient \*client);*
* *void messageToGui(QString message, QString from, const QStringList &users);*
* *void onConnect();*
* *void onDisconnect();*
* *void onReadyRead();*
* *void onError(QAbstractSocket::SocketError socketError) const;*

В данном классе объявляются объекты \*\_sok типа QTcpSocket, объект \*\_serv типа MyServer, \_name типа QString, \_isAutched булевого типа и \_blockSize типа quint16*.*

Рассмотрим эти функции подробнее. Конструктор класса MyClientпринимает 3 значения, desc типа int – этот тип данных необходим для установки дескриптора из функции incomingConnection(), который будет рассмотрен ниже, также конструктор принимает данные типа MyServer и QObject эти данные также необходимы для приема параметров сервера. В самом конструкторе в переменную \_serv передаются данные с MyServer, в нем хранится указатель на объект-сервер. \_isAutched принимает значение по умолчанию false, ведь изначально никаких пользователей к серверу не подключено, а переменная \_name принимает значение Unknown. Размер принимаемого блока \_blockSize также получает значение 0. Создается сокет \_sok типа QTcpSocket с передачей данных MyClient. В сокет устанавливается дескриптор из incomingConnection(). Следом идет подключение сокета к сигналам и функциям. Сокет подключается к 4-м видам сигналов connected(), disconnected(), readyRead(), error().

Дескриптор не содержит никаких дополнительных команд.

При срабатывании функции onDisconnect(), на экран выводится сообщение Client disconnected, затем осуществляется проверка, если пользователь авторизован, то удаляем его из интерфейса при помощи функции removeuserFromGui() и сообщаем всем, что клиент вышел при помощи *\_serv->doSendToAllUserLeft()*, и убираем пользователя из списка при помощи removeUser()*.*

Функция onError() предназначена для вывода сообщений о ошибках, в нее передается параметр, содержащий ошибку. Могут поступать сообщения об ошибках следующего типа: the host was not found; the connection was refused by the peer; the following error occurred.

Функция onReadyRead() предназначена для выполнения набора инструкций при поступлении сигнала readyRead(). Создается объект in класса QDataStream, этот объект необходим для передачи команд из клиента. Считывается блок данных. В переменную \_blockSize помещается размер переданного сообщения. Далее в переменную command поступает команда серверу в размере 3-х байт. Если же поступила команда от пользователей, которые не были авторизованы, то действие не выполняется. Идет сравнения команды с имеющимися в списке команд.

При поступлении команды comAutchReq создается переменная name типа QString и в нее помещается имя пользователя. Идет проверка имени по доступным символам и размеру при помощи \_serv->isNameValid(name), если же имя не проходит проверку, то в функцию doSendCommand() передается команда comErrNameInvalid. Далее идет проверка имени на повтор, это осуществляется при помощи проверки возвратного значения \_serv->isNameUsed(name), если же имя было использовано, то в функцию doSendCommand() передается команда comErrNameUsed. Если все проверки пройдены, то в *\_name* записывается имя пользователя, *\_isAutched* приобретает значение true, происходит отправка новому клиенту список активных пользователей при помощи doSendUsersOnline() и добавляется этот пользователь в интерфейс при помощи addUserToGui(name). Затем происходит уведомление всех пользователей о подключении нового пользователя при помощи \_serv->doSendToAllUserJoin(\_name).

При поступлении команды comMessageToAll создается переменная типа QString, в нее записывается сообщение и оно отправляется всем пользователям при помощи \_serv->doSendToAllMessage() и передается сообщения в интерфейс при помощи messageToGui()*.*

При поступлении команда comMessageToUsers создается переменная users\_in типа QString и присваивает данные пользователя, который отправил сообщение. Затем создается переменная message типа QString, которая принимает сообщение от клиента. Строку разбиваем на имена при помощи объявления объекта users типа QStringList и присваиванием ему результата значения функции users*\_in.split(“,”)*. Сообщение отправляется нужным людям при помощи \_serv->doSendMessageToUsers() и интерфейс обновляется при помощи messageToGui().

Функция doSendCommand() предназначена для обработки команд и отправки их в клиент пользователей, в нем создается объект out типа QDataStream, в первые 2 байта помещается цифра 0, т.к. первые 2 байта выделяются под размер сообщения. Следом помещается код команды. Путем вычисления размера сообщения, указатель перемещается в начало и туда помещается размер сообщения и отправляется в клиент.

При помощи функции doSendUsersOnline() реализуется способ доступа к списку онлайна всем пользователям. В ней создается объект out типа QDataStream, в первые 2 байта помещается цифра 0, т.к. первые 2 байта выделяются под размер сообщения после помещается код команды *comUsersOnline*. Следом помещается код команды. Создается объект l типа *QStringList* и получает значения \_serv->getUsersOnline(). Создается объект типа QString. Через цикл проходим по всему списку пользователей и результат помещается в объект QString. Затем результат помещается в поток. Путем вычисления размера сообщения, указатель перемещается в начало и туда помещается размер сообщения и отправляется в клиент.

### **3.1.4** Класс MyServer.

Класс MyServer предназначен для работы функций сервера. Данный класс наследует параметры QTcpServer для передачи данных с клиента. Класс содержит в себе конструктор и методы для работы с сервером. Функции сервера:

* конструкторexplicit MyServer(QWidget \*widget = 0, QObject \*parent = 0);
* bool doStartServer(QHostAddress addr, qint16 port);
* void doCloseServer();
* void doSendToAllUserJoin(QString name);
* void doSendToAllUserLeft(QString name);
* void doSendToAllMessage(QString message, QString fromUsername);
* void doSendToAllServerMessage(QString message);
* void doSendServerMessageToUsers(QString message, const QStringList &users);
* void doSendMessageToUsers(QString message, const QStringList &users, QString fromUsername);
* QStringList getUsersOnline() const;
* bool isNameValid(QString name) const;
* bool isNameUsed(QString name) const;
* void addLogToGui(QString string, QColor color = Qt::black);
* void onMessageFromGui(QString message, const QStringList &users);
* void onRemoveUser(MyClient \*client);
* void incomingConnection(qintptr handle);

В классе объявляется объект \_clients типа QList<MyClient\*> и объект \*\_widget типа QWidget.

Конструктор MyServer принимает объекты типа QWidget и QObject. Внутри самого конструктора объект \_widget получает передаваемые параметры объекта QWidget. Это необходимо для прямой работы с интерфейсом сервера.

Функция doStartServer(*)* получает хост адрес и порт. Запускается сервер при помощи listen(addr, port), если сервер не было запущен, то выводится сообщения об ошибке с указанием адреса и порта, если же все хорошо, то сервер начинает свою работу.

Функция doCloseServer() предназначена для отправки клиенту сообщения об отключении сервера и его отключения от него. Реализуется закрытие сервера при помощи команды close(), которая берется из наследуемого класса QTcpServer. Затем, как и с отправкой любых команд создается объект out типа QDataStream, в этот объект помещается первоначальный размер на 2 байта равный 0 и помещается команда закрытия сервера. После указатель перемещается в начало и высчитывается размер сообщения, в первые 2 байта помещается размер сообщения и отправляется всем клиентам.

Функция doSendToAllUserJoin() предназначена для отправки команды о подключенном пользователе и его имени. Создается объект out типа QDataStream, в этот объект помещается первоначальный размер на 2 байта равный 0 и помещается команда comUserJoin. После указатель перемещается в начало и высчитывается размер сообщения, в первые 2 байта помещается размер сообщения и отправляется всем клиентам.

Функция doSendToAllUserLeft() предназначена для отправки команды о отключенном пользователе и его имени. Создается объект out типа QDataStream, в этот объект помещается первоначальный размер на 2 байта равный 0 и помещается команда comUserLeft. После указатель перемещается в начало и высчитывается размер сообщения, в первые 2 байта помещается размер сообщения и отправляется всем клиентам.

Функция doSendToAllMessage() предназначена для отправки команды отправки сообщения для всех пользователей. Создается объект out типа QDataStream, в этот объект помещается первоначальный размер на 2 байта равный 0 и помещается команда comMessageToAll, после помещается имя пользователя, отправившего сообщение, и само сообщение. После указатель перемещается в начало и высчитывается размер сообщения, в первые 2 байта помещается размер сообщения и отправляется всем клиентам.

Функция doSendToAllServerMessage() предназначена для отправки команды отправки сообщения для всех пользователей от имени сервера. Создается объект out типа QDataStream, в этот объект помещается первоначальный размер на 2 байта равный 0 и помещается команда comPublicServerMessage, после помещается само сообщение. После указатель перемещается в начало и высчитывается размер сообщения, в первые 2 байта помещается размер сообщения и отправляется всем клиентам.

Функция doSendServerMessageToUsers() предназначена для отправки команды отправки приватных сообщений для пользователей от сервера. Создается объект out типа QDataStream, в этот объект помещается первоначальный размер на 2 байта равный 0 и помещается команда comPrivateServerMessage, после помещается само сообщение. После указатель перемещается в начало и высчитывается размер сообщения, в первые 2 байта помещается размер сообщения и отправляется всем клиентам.

Функция doSendMessageToUsers() предназначена для отправки команды отправки приватных сообщений пользователями. Создается объект out типа QDataStream, в этот объект помещается первоначальный размер на 2 байта равный 0 и помещается команда comMessageToUsers, указывается пользователь, что отправил сообщение и само сообщение, после помещается само сообщение. После указатель перемещается в начало и высчитывается размер сообщения, в первые 2 байта помещается размер сообщения и отправляется всем клиентам. Подобный цикл повторяется с новым объектом outToSender типа QDataStream. Затем отправляется команда comMessageToUsers, помещает список пользователей и само сообщение.

Функция getUsersOnline() предназначена для передачи списка пользователей, которые сейчас являются онлайн. Создается объект l типа QStringList. Осуществляется считывание всех пользователей и если они авторизованы, то помещаются в объект l и возвращается.

Функция isNameValid() предназначена для проверки имени на установленные правила.

Функция isNameUsed() отвечает за то, было ли использован “никнейм” или нет. Осуществляется обычным сравнением поступивших данных с таблицей имен.

Функция incomingConnection() принимает сигналы от клиента на сервер, которые потом передаются основным функциям и затем обрабатываются. Создается объект типа MyClient и выделяется под него память, передавая при этом конструктору параметры handle, которые поступают на функцию и данные класса MyServer. Идет проверка на наличие указателя интерфейса сервера и подключается объект client к сигналам addUserToGui(), removeUserFromGui(), messageToGui(). Все эти сигналы вызывают функции onAddUserToGui(), onRemoveUserFroGui(), onMessageToGui(), которые вызываются из класса Dialog с помощью указателя \_widget. После объект client подключается к сигналу removeUser(), который через объект класса MyServer вызывает функцию onRemoveUser().

Функция onMessageFromGui() предназначена для отправки сообщения из клиента сервера и принимает параметры типа QString и QStringList. Если же не было выбрано никакого пользователя, то вызывается функция doSendToAllServerMessage(), если же пользователь был выбран, то вызывается функция doSendServerMessageToUsers(), передавая при этом параметры сообщения и имени пользователя.

Диаграмма классов изображена на рисунке 3.1.

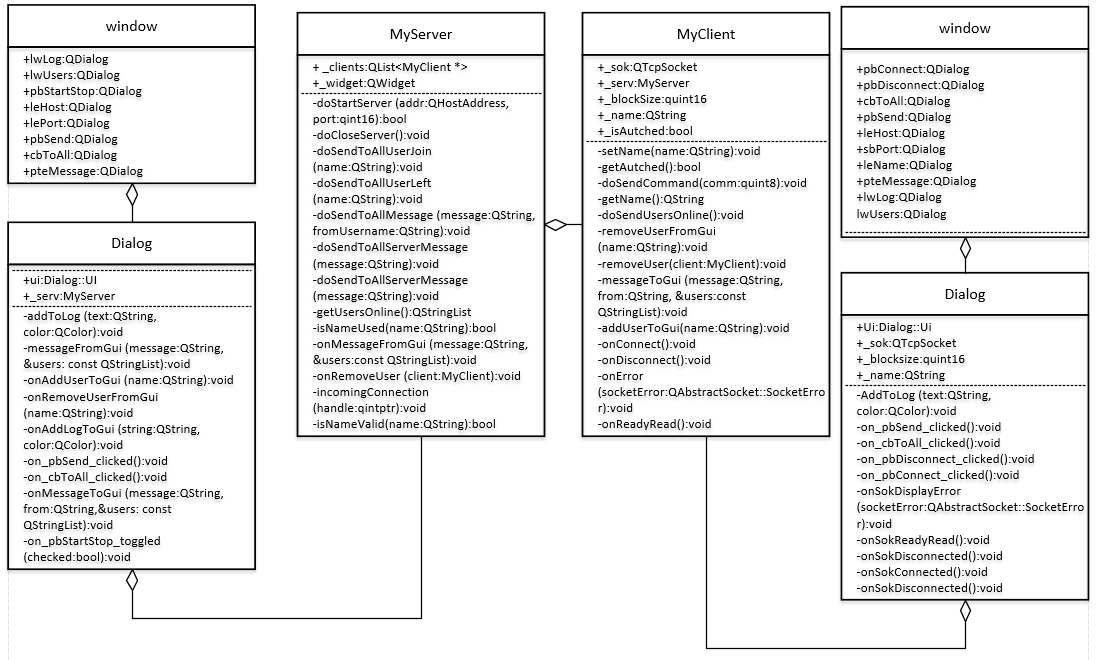


Рисунок 3.1 – Диаграмма классов программного обеспечения.

# **4** РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

Реализация передачи данных между клиентом и сервером была осуществлена при помощи класса QDataStream. При помощи данного класса была осуществлена передача команд, имен, сообщений и тд. Рассмотрим примеры использования QDataStream в клиенте и сервере.

**4.1** Клиент чата

Приведем пример кода, которые реализовывает прием и обработку команд поступивших на сервер:

void Dialog::onSokReadyRead()

{

QDataStream in(\_sok);

if (\_blockSize == 0) {

if (\_sok->bytesAvailable() < (int)sizeof(quint16))

return;

in >> \_blockSize;

qDebug() << "\_blockSize now " << \_blockSize;

}

if (\_sok->bytesAvailable() < \_blockSize)

return;

else

\_blockSize = 0;

quint8 command;

in >> command;

qDebug() << "Received command " << command;

В данном фрагменте кода создается объект для приема сообщений. Осуществляется считывание блока на первые 2 байта. Если пришло меньше 2-х байт, то ожидаем его прихода. Как только блок данных поступает и становится известен его размер, получаем команду, которую необходимо выполнить.

switch (command)

{

case MyClient::comAutchSuccess:

{

ui->pbSend->setEnabled(true);

AddToLog("Enter as "+\_name,Qt::green);

}

break;

В этом кейсе выполняется подтверждения подключения.

case MyClient::comUsersOnline:

{

AddToLog("Received user list "+\_name,Qt::green);

ui->pbSend->setEnabled(true);

QString users;

in >> users;

if (users == "")

return;

QStringList l = users.split(",");

ui->lwUsers->addItems(l);

}

break;

В этом кейсе осуществляется вывод списка пользователей, находящихся в сети.

case MyClient::comPublicServerMessage:

{

QString message;

in >> message;

AddToLog("[PublicServerMessage]: "+message, Qt::red);

}

break;

В этом кейсе осуществляется вывод публичного серверного соощения.

case MyClient::comMessageToAll:

{

QString user;

in >> user;

QString message;

in >> message;

AddToLog("["+user+"]: "+message);

}

break;

В этом кейсе осуществляется вывод сообщения для всех пользователей от конкретного пользователя.

case MyClient::comMessageToUsers:

{

QString user;

in >> user;

QString message;

in >> message;

AddToLog("["+user+"](private): "+message, Qt::blue);

}

break;

В этом кейсе осуществляется прием приватного сообщения от пользователя к пользователю(-лям).

case MyClient::comPrivateServerMessage:

{

QString message;

in >> message;

AddToLog("[PrivateServerMessage]: "+message, Qt::red);

}

break;

В этом кейсе осуществляется прием приватного сообщения от сервера к пользователю

case MyClient::comUserJoin:

{

QString name;

in >> name;

ui->lwUsers->addItem(name);

AddToLog(name+" joined", Qt::green);

}

break;

В этом кейсе осуществляется вывод сообщения о подключенном к серверу пользователе.

case MyClient::comUserLeft:

{

QString name;

in >> name;

for (int i = 0; i < ui->lwUsers->count(); ++i)

if (ui->lwUsers->item(i)->text() == name)

{

ui->lwUsers->takeItem(i);

AddToLog(name+" left", Qt::green);

break;

}

}

break;

В этом кейсе осуществляется вывод сообщения об отключении пользователя и удалении его из Поля пользователей.

case MyClient::comErrNameInvalid:

{

QMessageBox::information(this, "Error", "This name is invalid.");

\_sok->disconnectFromHost();

}

break;

В этом кейсе осуществляется вывод сообщения об ошибке ввода неверного имени.

case MyClient::comErrNameUsed:

{

QMessageBox::information(this, "Error", "This name is already used.");

\_sok->disconnectFromHost();

}

break;

В этом кейсе осуществляется вывод ошибки ввода имени “Имя уже занято.”

case MyClient::comServerClose:

{

QMessageBox::information(this, "Error", "Server is closed.");

\_sok->disconnectFromHost();

}

break;

}

}

В этом кейсе выводится сообщение об ошибке в случае отключения сервера.

Рассмотри также в клиенте функции передачи данных об подключенном пользователе:

void Dialog::onSokConnected()

{

ui->pbConnect->setEnabled(false);

ui->pbDisconnect->setEnabled(true);

\_blockSize = 0;

AddToLog("Connected to"+\_sok->peerAddress().toString()+":"+QString::number(\_sok->peerPort()),Qt::green);

QByteArray block;

QDataStream out(&block, QIODevice::WriteOnly);

out << (quint16)0;

out << (quint8)MyClient::comAutchReq;

out << ui->leName->text();

\_name = ui->leName->text();

out.device()->seek(0);

out << (quint16)(block.size() - sizeof(quint16));

\_sok->write(block);

}

В этом фрагменте кода осуществляется смена состояния кнопок подключения и отключения после реакции на сигнал connected(). \_blockSize равен нулю. На экран в Поле сообщений выводится сообщение об подключении к серверу, его порту и адресу. Создается объект out типа QDataStream, в этот объект помещается первоначальный размер на 2 байта равный 0 и помещается команда comAutchReq, имя пользователя. После указатель перемещается в начало и высчитывается размер сообщения, в первые 2 байта помещается размер сообщения и отправляется всем клиентам.

**4.2** Серверная часть

Принцип работы сервера схож по работе с его клиентом. С помощью тех же алгоритмов будет осуществляться прием и передача сообщений между клиентом и сервером. Точно также осуществляется обработка команд.

Рассмотрим алгоритм приема сообщений и работы сервера с его интерфейсом.

Выполнение приема и передачи входящих сообщений:

void MyServer::incomingConnection(qintptr handle)

{

MyClient \*client = new MyClient(handle, this, this);

if (\_widget != 0)

{

connect(client, SIGNAL(addUserToGui(QString)), \_widget, SLOT(onAddUserToGui(QString)));

connect(client, SIGNAL(removeUserFromGui(QString)), \_widget, SLOT(onRemoveUserFromGui(QString)));

connect(client, SIGNAL(messageToGui(QString,QString,QStringList)), \_widget, SLOT(onMessageToGui(QString,QString,QStringList)));

}

connect(client, SIGNAL(removeUser(MyClient\*)), this, SLOT(onRemoveUser(MyClient\*)));

\_clients.append(client);

}

Через qintptr handle передается входящее сообщение подключенного пользователя. Создается объект \*client типа MyClient и выделяется под этот объект память с передачей параметров handle (для установки в сокет дескриптора) и MyServer. Идет проверка на состояние \_widget, если указатель есть, то client подключается к сигналам и использует функции интерфейса сервера. Таким образов вся информация о подключенных пользователях, об отправленных сообщениях и т.д. отображается на экране клиента сервера.

Остальные функции похожи. Передача и обработка команд, передача сообщений и информации о пользователях, все это выполнено с помощью передачи информации через объект типа QDataSteam.

# **5** ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ

Тестирование программного обеспечения – процесс исследования программного обеспечения с целью получения информации о качестве продукта. Тестирование является одним из важных этапов разработки, поскольку при написании программного кода невозможно предусмотреть все случаи, и ошибки в работе программы неизбежны. Кроме того, возможно появление ошибок в случае внесения дополнительного функционала в программу даже в уже отлаженном рабочем коде.

Тестирование программы направлено на доказательства её соответствия заявленным требованиям. Таким образом, основной целью тестирования является выделение несоответствия функционирования программы заявленным требованиям и устранение их, для повышения качества программного продукта.

Тестирование программы проводилось в два этапа:

* поэтапное тестирование отдельно каждого модуля в процессе написания программного кода;
* полное тестирование программы после окончания процесса написания программного кода.

Оба этапа являются достаточно важными, ни один из них невозможно исключить. Например, без модульного тестирования, при анализе работы программы в целом, будет происходить достаточное количество сбоев, выявить и локализовать которые может оказаться достаточно сложным заданием, в то время как при анализе работы одного модуля неисправность оказывается достаточно очевидной. И обратный случай, работоспособность каждого компонента в отдельности не гарантирует корректное поведение всей программы в целом.

Тестирование программы проводилось на следующих компьютерах:

* Intel Core i5 7600K 4,5 GHz, 16GB RAM, 2TB HDD, ASUS PRIME Z270-A, блок питания 750Вт, видеокарта Nvidia GTX 1060 3GB, операционная система Windows 10
* Intel Core i3 4010 1,7GHz, 4GB RAM, 750GB HDD, видеокарта Nvidia GeForce 840M 2GB, операционная система Windows 8.1

В программе предусмотрена проверка имен, вводимых пользователями, на совпадение и правильность написания. Проверка осуществляется по количеству символов и типу символов.

На рисунке 5.1 показана ошибка ввода неверного имени.

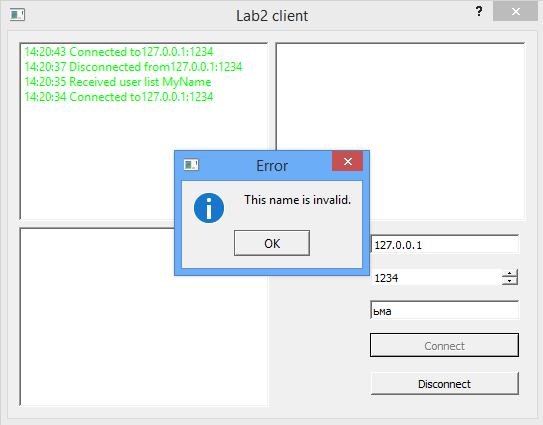


Рисунок 5.1 – Ошибка при вводе не валидных данных студента

Для модульного тестирования была использована библиотека Boost. Модульное тестирование помогает проверить правильность программы путем проверки запланированного поведения кода. В модульном тестировании анализируется функциональность программы на обнаружение отдельного тестируемого поведения, которое можно протестировать в качестве индивидуального модуля.

Модульное тестирование наиболее эффективно, когда оно является неотъемлемой частью рабочего процесса разработки ПО. После написания функции или другого блока кода приложения создаются модульные тесты, которые проверяют поведение кода в ответ на стандартные, граничные и некорректные случаи ввода данных, также проверяются любые явные или предполагаемые допущения, сделанные кодом. В практике разработки программного обеспечения, известной как разработка, управляемая тестом, создается модуль тестов перед написанием кода, поэтому модульные тесты используются в качестве технической документации и спецификации функциональности. После того, как модульные тесты были готовы для большей части функциональности и реализованные методы прошли все необходимые пункты тестирования, началось функциональное тестирование.

Тесты, проведенные над программой другими пользователями, представлены в таблице 5.1.

Таблица 5.1 – Тестирование программы

|  |  |  |  |
| --- | --- | --- | --- |
| Меню программы | Содержание теста | Ожидаемый результат | Тест пройден |
| Пользователь | Попытка ввести имя и подключиться к серверу. | Если имя соответствует требованиям, то клиент подключается к серверу. | да |
| Пользователь | Ввод IP адреса сервера и попытка подключения. | Если адрес совпадает с адресом сервера, то клиент подключается к серверу | да |
| Пользователь | Попытка ввода сообщения и его отправка | Сообщение отправляется в публичный чат и отображается в Поле сообщений. | да |
| Пользователь | Отправка приватного сообщения | Приватное сообщение было отправлено пользователю(-лям) выбранным в Поле пользователей. | да |
| Администратор | Ввод IP адреса для сервера и включение сервера | Сервер начинает свою работу | да |
| Администратор | Отправка публичного сообщения | Сообщение отправляется пользователям и отображается в Поле сообщений | да |
| Администратор | Отправка приватного сообщения | Сообщение отправляется приватно пользователю(-лям) выбранному в Поле пользователей | да |
| Администратор | Отключение сервера | Сервер прекращает свою работу, следовательно, пользователи отключаются от сервера | да |

Как видно из таблицы, приложение хорошо справилось с тестами, что говорит о высокой его работоспособности.

# **6** РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

## **6.1** Требования к аппаратному и программному обеспечению

Минимальные требования для работы серверной части системы:

* Процессор Intel Core i3 или старше;
* 512 мегабайт оперативной памяти;
* 10МB свободного места на жестком диске;
* операционная система Windows;
* наличие подключения по локальной сети;

Минимальные требования для работы клиентской части системы:

* Процессор Intel Core i3 или старше;
* 512 мегабайт оперативной памяти;
* операционная система Windows;
* наличие подключения по локальной сети;
* 250МB свободного места на жестком диске;
* поддержка OpenGL 3.3;

## **6.2** Руководство по использованию программного средства.

Данный проект является мультиплеерной игрой, следовательно для игры требуется наличие локальной сети и запущенный сервер. Для включения сервера необходимо просто запустить “Server.exe”. После запуска на экране появится пустая консоль. Теперь сервер запущен и ожидает подключения клиентов.

Для запуска клиента игры, необходимо запустить файл “Space War: Arena.exe”. В процессе запуска, будет осуществлено подключение к серверу (рис 6.1).

# ЗАКЛЮЧЕНИЕ

Данное программное обеспечение предназначено для обмена сообщениями между разными пользователями. Оно позволяет отправлять сообщения как публично, данное сообщение видно всем пользователям, в том числе и администратору, так и приватно, отправляя сообщение как одному, так и нескольким пользователям одновременно. Клиент чата позволяет подключатся к серверу по указанному ip-адресу, если по этому адресу сервер был запущен.

Клиент сервера, предназначенный для администратора, обладает теми же функциями, что и клиент, предназначенный для пользователей.

Данное программное обеспечение работает исправно, были проведены все тесты. Функции работают исправно, весь функционал обмена сообщениями находится на сервере, поэтому клиент пользователя не сможет играть роль сервера, а должен подключаться непосредственно к серверу.

Данная чат-программа имеет интуитивно понятный интерфейс и предназначена для связи между сотрудниками малых или средних фирм.

# СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

[1] Шлее, М. – Qt 5.3. Профессиональное программирование на С++. – СПб.: БХВ-Петербург,2015.-928с.:ил. – (в подленнике)

[2] Шилтд, Г. – С++. Базовый курс. – РФ: Вильямс, 2015.-624с.:ил.

[3] Шилтд, Г. – Полный справочник по С++. – РФ: Вильямс, 2016.-800с.:ил.

[4] Википедия.орг [Электронный ресурс]. – Электронные данные – Режим доступа : wikipedia.org.

[5] Хабр.ком [Электронный ресурс]. – Электронные данные – Режим доступа : habr.com

[6] Реалкодинг.нет [Электронный ресурс]. – Электронные данные – Режим доступа : realcoding.net

# Приложение А

# Приложение Б

Листинг программы

//dialog.h

#ifndef DIALOG\_H

#define DIALOG\_H

#include <QDialog>

#include <QTcpSocket>

#include <QHostAddress>

#include "D:\Downloads\QSocket\Lab2\_server\myclient.h"

#include <QMessageBox>

class MyClient;

namespace Ui {

class Dialog;

}

class Dialog : public QDialog

{

Q\_OBJECT

public:

explicit Dialog(QWidget \*parent = 0);

~*Dialog*();

private slots:

void onSokConnected();

void onSokDisconnected();

void onSokReadyRead();

void onSokDisplayError(QAbstractSocket::SocketError socketError);

void on\_pbConnect\_clicked();

void on\_pbDisconnect\_clicked();

void on\_cbToAll\_clicked();

void on\_pbSend\_clicked();

private:

Ui::Dialog \*ui;

QTcpSocket \*\_sok;

quint16 \_blockSize;

QString \_name;

void AddToLog(QString text, QColor color = Qt::black);

};

#endif

//dialog.cpp

#include "dialog.h"

#include "ui\_dialog.h"

#include <QtGui>

#include <QDebug>

Dialog::Dialog(QWidget \*parent) :QDialog(parent),ui(new Ui::Dialog)

{

ui->setupUi(this);

\_name = "";

\_sok = new QTcpSocket(this);

connect(\_sok, SIGNAL(readyRead()), this, SLOT(onSokReadyRead()));

connect(\_sok, SIGNAL(connected()), this, SLOT(onSokConnected()));

connect(\_sok, SIGNAL(disconnected()), this, SLOT(onSokDisconnected()));

connect(\_sok, SIGNAL(error(QAbstractSocket::SocketError)),this, SLOT(onSokDisplayError(QAbstractSocket::SocketError)));

}

Dialog::~*Dialog*()

{

delete ui;

}

void Dialog::onSokDisplayError(QAbstractSocket::SocketError socketError)

{

switch (socketError) {

case QAbstractSocket::RemoteHostClosedError:

break;

case QAbstractSocket::HostNotFoundError:

QMessageBox::information(this, "Error", "The host was not found");

break;

case QAbstractSocket::ConnectionRefusedError:

QMessageBox::information(this, "Error", "The connection was refused by the peer.");

break;

default:

QMessageBox::information(this, "Error", "The following error occurred: "+\_sok->errorString());

}

}

void Dialog::onSokReadyRead()

{

QDataStream in(\_sok);

//если считываем новый блок первые 2 байта это его размер

if (\_blockSize == 0) {

//если пришло меньше 2 байт ждем пока будет 2 байта

if (\_sok->*bytesAvailable*() < (int)sizeof(quint16))

return;

//считываем размер (2 байта)

in >> \_blockSize;

qDebug() << "\_blockSize now " << \_blockSize;

}

//ждем пока блок прийдет полностью

if (\_sok->*bytesAvailable*() < \_blockSize)

return;

else

//можно принимать новый блок

\_blockSize = 0;

//3 байт - команда серверу

quint8 command;

in >> command;

qDebug() << "Received command " << command;

switch (command)

{

case MyClient::comUsersOnline:

{

AddToLog("Received user list "+\_name,Qt::green);

ui->pbSend->setEnabled(true);

QString users;

in >> users;

if (users == "")

return;

QStringList l = users.split(",");

ui->lwUsers->addItems(l);

}

break;

case MyClient::comPublicServerMessage:

{

QString message;

in >> message;

AddToLog("[PublicServerMessage]: "+message, Qt::red);

}

break;

case MyClient::comMessageToAll:

{

QString user;

in >> user;

QString message;

in >> message;

AddToLog("["+user+"]: "+message);

}

break;

case MyClient::comMessageToUsers:

{

QString user;

in >> user;

QString message;

in >> message;

AddToLog("["+user+"](private): "+message, Qt::blue);

}

break;

case MyClient::comPrivateServerMessage:

{

QString message;

in >> message;

AddToLog("[PrivateServerMessage]: "+message, Qt::red);

}

break;

case MyClient::comUserJoin:

{

QString name;

in >> name;

ui->lwUsers->addItem(name);

AddToLog(name+" joined", Qt::green);

}

break;

case MyClient::comUserLeft:

{

QString name;

in >> name;

for (int i = 0; i < ui->lwUsers->count(); ++i)

if (ui->lwUsers->item(i)->text() == name)

{

ui->lwUsers->takeItem(i);

AddToLog(name+" left", Qt::green);

break;

}

}

break;

case MyClient::comErrNameInvalid:

{

QMessageBox::information(this, "Error", "This name is invalid.");

\_sok->*disconnectFromHost*();

}

break;

case MyClient::comErrNameUsed:

{

QMessageBox::information(this, "Error", "This name is already used.");

\_sok->*disconnectFromHost*();

}

break;

case MyClient::comServerClose:

{

AddToLog("The server was closed",Qt::red);

\_sok->*disconnectFromHost*();

}

break;

}

}

void Dialog::onSokConnected()

{

ui->pbConnect->setEnabled(false);

ui->pbDisconnect->setEnabled(true);

\_blockSize = 0;

AddToLog("Connected to"+\_sok->peerAddress().toString()+":"+QString::number(\_sok->peerPort()),Qt::green);

//try autch

QByteArray block;

QDataStream out(&block, QIODevice::WriteOnly);

out << (quint16)0;

out << (quint8)MyClient::comAutchReq;

out << ui->leName->text();

\_name = ui->leName->text();

out.device()->*seek*(0);

out << (quint16)(block.size() - sizeof(quint16));

\_sok->write(block);

}

void Dialog::onSokDisconnected()

{

ui->pbConnect->setEnabled(true);

ui->pbDisconnect->setEnabled(false);

ui->pbSend->setEnabled(false);

ui->lwUsers->clear();

AddToLog("Disconnected from"+\_sok->peerAddress().toString()+":"+QString::number(\_sok->peerPort()), Qt::green);

}

void Dialog::on\_pbConnect\_clicked()

{

\_sok->*connectToHost*(ui->leHost->text(), ui->sbPort->value());

}

void Dialog::on\_pbDisconnect\_clicked()

{

\_sok->*disconnectFromHost*();

}

void Dialog::on\_cbToAll\_clicked()

{

if (ui->cbToAll->isChecked())

ui->pbSend->setText("Send To All");

else

ui->pbSend->setText("Send To Selected");

}

void Dialog::on\_pbSend\_clicked()

{

QByteArray block;

QDataStream out(&block, QIODevice::WriteOnly);

out << (quint16)0;

if (ui->cbToAll->isChecked())

out << (quint8)MyClient::comMessageToAll;

else

{

out << (quint8)MyClient::comMessageToUsers;

QString s;

foreach (QListWidgetItem \*i, ui->lwUsers->selectedItems())

s += i->text()+",";

s.remove(s.length()-1, 1);

out << s;

}

out << ui->pteMessage->document()->toPlainText();

out.device()->*seek*(0);

out << (quint16)(block.size() - sizeof(quint16));

\_sok->write(block);

ui->pteMessage->clear();

}

void Dialog::AddToLog(QString text, QColor color)

{

ui->lwLog->insertItem(0, QTime::currentTime().toString()+" "+text);

ui->lwLog->item(0)->setTextColor(color);

}

//main.cpp

#include <QApplication>

#include "dialog.h"

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

{

QApplication a(argc, argv);

Dialog w;

w.show();

return a.exec();

}

//dialog.h

#ifndef DIALOG\_H

#define DIALOG\_H

#include <QDialog>

#include <QDebug>

#include <QtGui>

#include <QtCore>

#include "myserver.h"

namespace Ui {

class Dialog;

}

class Dialog : public QDialog

{

Q\_OBJECT

public:

explicit Dialog(QWidget \*parent = 0);

~Dialog();

private:

Ui::Dialog \*ui;

MyServer \*\_serv;

void addToLog(QString text, QColor color);

signals:

void messageFromGui(QString message, const QStringList &users);

public slots:

void onAddUserToGui(QString name);

void onRemoveUserFromGui(QString name);

void onMessageToGui(QString message, QString from, const QStringList &users);

void onAddLogToGui(QString string, QColor color);

private slots:

void on\_pbSend\_clicked();

void on\_cbToAll\_clicked();

void on\_pbStartStop\_toggled(bool checked);

};

#endif

//myclient.h

#ifndef MYCLIENT\_H

#define MYCLIENT\_H

#include <QObject>

#include <QDebug>

#include <QTcpSocket>

#include <QThreadPool>

#include <QtGui>

#include <QRegExp>

#include "myserver.h"

class MyServer;

class MyClient : public QObject

{

friend class MyServer;

Q\_OBJECT

public:

static const QString constNameUnknown;

static const quint8 comAutchReq = 1;

static const quint8 comUsersOnline = 2;

static const quint8 comUserJoin = 3;

static const quint8 comUserLeft = 4;

static const quint8 comMessageToAll = 5;

static const quint8 comMessageToUsers = 6;

static const quint8 comPublicServerMessage = 7;

static const quint8 comPrivateServerMessage = 8;

static const quint8 comErrNameInvalid = 201;

static const quint8 comErrNameUsed = 202;

static const quint8 comServerClose = 10;

explicit MyClient(int desc, MyServer \*serv, QObject \*parent = 0);

~MyClient();

void setName(QString name) {\_name = name;}

QString getName() const {return \_name;}

bool getAutched() const {return \_isAutched;}

void doSendCommand(quint8 comm) const;

void doSendUsersOnline() const;

signals:

void addUserToGui(QString name);

void removeUserFromGui(QString name);

void removeUser(MyClient \*client);

void messageToGui(QString message, QString from, const QStringList &users);

private slots:

void onConnect();

void onDisconnect();

void onReadyRead();

void onError(QAbstractSocket::SocketError socketError) const;

private:

QTcpSocket \*\_sok;

MyServer \*\_serv;

quint16 \_blockSize;

QString \_name;

bool \_isAutched;

};

#endif

//myserver.h

#ifndef MYSERVER\_H

#define MYSERVER\_H

#include "myclient.h"

#include <QTcpServer>

#include <QDebug>

class MyClient;

class MyServer : public QTcpServer

{

Q\_OBJECT

public:

explicit MyServer(QWidget \*widget = 0, QObject \*parent = 0);

bool doStartServer(QHostAddress addr, qint16 port);

void doCloseServer();

void doSendToAllUserJoin(QString name);

void doSendToAllUserLeft(QString name);

void doSendToAllMessage(QString message, QString fromUsername);

void doSendToAllServerMessage(QString message);

void doSendServerMessageToUsers(QString message, const QStringList &users);

void doSendMessageToUsers(QString message, const QStringList &users, QString fromUsername);

QStringList getUsersOnline() const;

bool isNameValid(QString name) const;

bool isNameUsed(QString name) const;

signals:

void addLogToGui(QString string, QColor color = Qt::black);

public slots:

void onMessageFromGui(QString message, const QStringList &users);

void onRemoveUser(MyClient \*client);

protected:

void incomingConnection(qintptr handle);

private:

QList<MyClient \*> \_clients;

QWidget \*\_widget;

};

#endif

//dialog.cpp

#include "dialog.h"

#include "ui\_dialog.h"

#include <QMessageBox>

Dialog::Dialog(QWidget \*parent) :QDialog(parent), ui(new Ui::Dialog)

{

ui->setupUi(this);

\_serv = new MyServer(this, this);

connect(this, SIGNAL(messageFromGui(QString,QStringList)), \_serv, SLOT(onMessageFromGui(QString,QStringList)));

connect(\_serv, SIGNAL(addLogToGui(QString,QColor)), this, SLOT(onAddLogToGui(QString,QColor)));

if (\_serv->doStartServer(QHostAddress::LocalHost, 1234))

{

ui->lwLog->insertItem(0, QTime::currentTime().toString()+" server strated at "+\_serv->serverAddress().toString()+":"+QString::number(\_serv->serverPort()));

ui->lwLog->item(0)->setTextColor(Qt::green);

}

else

{

ui->lwLog->insertItem(0, QTime::currentTime().toString()+" server not strated at "+\_serv->serverAddress().toString()+":"+QString::number(\_serv->serverPort()));

ui->lwLog->item(0)->setTextColor(Qt::red);

ui->pbStartStop->setChecked(true);

}

}

Dialog::~*Dialog*()

{

delete ui;

}

void Dialog::onAddUserToGui(QString name)

{

ui->lwUsers->addItem(name);

ui->lwLog->insertItem(0, QTime::currentTime().toString()+" "+name+" joined");

ui->lwLog->item(0)->setTextColor(Qt::green);

}

void Dialog::onRemoveUserFromGui(QString name)

{

for (int i = 0; i < ui->lwUsers->count(); ++i)

if (ui->lwUsers->item(i)->text() == name)

{

ui->lwUsers->takeItem(i);

ui->lwLog->insertItem(0, QTime::currentTime().toString()+" "+name+" left");

ui->lwLog->item(0)->setTextColor(Qt::green);

break;

}

}

void Dialog::onMessageToGui(QString message, QString from, const QStringList &users)

{

if (users.isEmpty())

ui->lwLog->insertItem(0, QTime::currentTime().toString()+" message from "+from+": "+message+" to all");

else

{

ui->lwLog->insertItem(0, QTime::currentTime().toString()+" message from "+from+": "+message+" to "+users.join(","));

ui->lwLog->item(0)->setTextColor(Qt::blue);

}

}

void Dialog::onAddLogToGui(QString string, QColor color)

{

addToLog(string, color);

}

void Dialog::on\_pbSend\_clicked()

{

if (ui->lwUsers->count() == 0)

{

QMessageBox::information(this, "", "No clients connected");

return;

}

QStringList l;

if (!ui->cbToAll->isChecked())

foreach (QListWidgetItem \*i, ui->lwUsers->selectedItems())

l << i->text();

emit messageFromGui(ui->pteMessage->document()->toPlainText(), l);

ui->pteMessage->clear();

if (l.isEmpty())

addToLog("Sended public server message", Qt::black);

else

addToLog("Sended private server message to "+l.join(","), Qt::black);

}

void Dialog::on\_cbToAll\_clicked()

{

if (ui->cbToAll->isChecked())

ui->pbSend->setText("Send To All");

else

ui->pbSend->setText("Send To Selected");

}

void Dialog::on\_pbStartStop\_toggled(bool checked)

{

if (checked)

{

QHostAddress addr;

addToLog(" server stopped at "+\_serv->serverAddress().toString()+":"+QString::number(\_serv->serverPort()), Qt::green);

\_serv->doCloseServer();

ui->pbStartStop->setText("Start server");

}

else

{

QHostAddress addr;

if (!addr.setAddress(ui->leHost->text()))

{

addToLog(" invalid address "+ui->leHost->text(), Qt::red);

return;

}

if (\_serv->doStartServer(addr, ui->lePort->text().toInt()))

{

addToLog(" server strated at "+ui->leHost->text()+":"+ui->lePort->text(), Qt::green);

ui->pbStartStop->setText("Stop server");

}

else

{

addToLog(" server not strated at "+ui->leHost->text()+":"+ui->lePort->text(), Qt::red);

ui->pbStartStop->setChecked(true);

}

}

}

void Dialog::addToLog(QString text, QColor color)

{

ui->lwLog->insertItem(0, QTime::currentTime().toString()+text);

ui->lwLog->item(0)->setTextColor(color);

}

//myclient.cpp

#include "myclient.h"

#include <QWidget>

#include <QMessageBox>

const QString MyClient::constNameUnknown = QString(".Unknown");

MyClient::MyClient(int desc, MyServer \*serv, QObject \*parent) :QObject(parent)

{

\_serv = serv;

\_isAutched = false;

\_name = constNameUnknown;

\_blockSize = 0;

\_sok = new QTcpSocket(this);

\_sok->setSocketDescriptor(desc);

connect(\_sok, SIGNAL(connected()), this, SLOT(onConnect()));

connect(\_sok, SIGNAL(disconnected()), this, SLOT(onDisconnect()));

connect(\_sok, SIGNAL(readyRead()), this, SLOT(onReadyRead()));

connect(\_sok, SIGNAL(error(QAbstractSocket::SocketError)), this, SLOT(onError(QAbstractSocket::SocketError)));

qDebug() << "Client connected" << desc;

}

MyClient::~MyClient()

{

}

void MyClient::onConnect()

{

}

void MyClient::onDisconnect()

{

qDebug() << "Client disconnected";

if (\_isAutched)

{

emit removeUserFromGui(\_name);

\_serv->doSendToAllUserLeft(\_name);

emit removeUser(this);

}

deleteLater();

}

void MyClient::onError(QAbstractSocket::SocketError socketError) const

{

QWidget w;

switch (socketError) {

case QAbstractSocket::RemoteHostClosedError:

break;

case QAbstractSocket::HostNotFoundError:

QMessageBox::information(&w, "Error", "The host was not found");

break;

case QAbstractSocket::ConnectionRefusedError:

QMessageBox::information(&w, "Error", "The connection was refused by the peer.");

break;

default:

QMessageBox::information(&w, "Error", "The following error occurred: "+\_sok->errorString());

}

}

void MyClient::onReadyRead()

{

QDataStream in(\_sok);

if (\_blockSize == 0) {

if (\_sok->bytesAvailable() < (int)sizeof(quint16))

return;

in >> \_blockSize;

qDebug() << "\_blockSize now " << \_blockSize;

}

if (\_sok->bytesAvailable() < \_blockSize)

return;

else

\_blockSize = 0;

quint8 command;

in >> command;

qDebug() << "Received command " << command;

if (!\_isAutched && command != comAutchReq)

return;

switch(command)

{

case comAutchReq:

{

QString name;

in >> name;

if (!\_serv->isNameValid(name))

{

doSendCommand(comErrNameInvalid);

return;

}

if (\_serv->isNameUsed(name))

{

doSendCommand(comErrNameUsed);

return;

}

\_name = name;

\_isAutched = true;

doSendUsersOnline();

emit addUserToGui(name);

\_serv->doSendToAllUserJoin(\_name);

}

break;

case comMessageToAll:

{

QString message;

in >> message;

\_serv->doSendToAllMessage(message, \_name);

emit messageToGui(message, \_name, QStringList());

}

break;

case comMessageToUsers:

{

QString users\_in;

in >> users\_in;

QString message;

in >> message;

QStringList users = users\_in.split(",");

\_serv->doSendMessageToUsers(message, users, \_name);

emit messageToGui(message, \_name, users);

}

break;

}

}

void MyClient::doSendCommand(quint8 comm) const

{

QByteArray block;

QDataStream out(&block, QIODevice::WriteOnly);

out << (quint16)0;

out << comm;

out.device()->seek(0);

out << (quint16)(block.size() - sizeof(quint16));

\_sok->write(block);

qDebug() << "Send to" << \_name << "command:" << comm;

}

void MyClient::doSendUsersOnline() const

{

QByteArray block;

QDataStream out(&block, QIODevice::WriteOnly);

out << (quint16)0;

out << comUsersOnline;

QStringList l = \_serv->getUsersOnline();

QString s;

for (int i = 0; i < l.length(); ++i)

if (l.at(i) != \_name)

s += l.at(i)+(QString)",";

s.remove(s.length()-1, 1);

out << s;

out.device()->seek(0);

out << (quint16)(block.size() - sizeof(quint16));

\_sok->write(block);

qDebug() << "Send user list to" << \_name << ":" << s;

}

//myserver.cpp

#include "myserver.h"

#include "myclient.h"

#include "dialog.h"

MyServer::MyServer(QWidget \*widget, QObject \*parent) :QTcpServer(parent)

{

\_widget = widget;

}

bool MyServer::doStartServer(QHostAddress addr, qint16 port)

{

if (!listen(addr, port))

{

qDebug() << "Server not started at" << addr << ":" << port;

return false;

}

qDebug() << "Server started at" << addr << ":" << port;

return true;

}

void MyServer::doCloseServer(){

close();

QByteArray block;

QDataStream out(&block, QIODevice::WriteOnly);

out << (quint16)0 << MyClient::comServerClose;

out.device()->seek(0);

out << (quint16)(block.size() - sizeof(quint16));

for (int i = 0; i < \_clients.length(); ++i)

\_clients.at(i)->\_sok->write(block);

}

void MyServer::doSendToAllUserJoin(QString name)

{

QByteArray block;

QDataStream out(&block, QIODevice::WriteOnly);

out << (quint16)0 << MyClient::comUserJoin << name;

out.device()->seek(0);

out << (quint16)(block.size() - sizeof(quint16));

for (int i = 0; i < \_clients.length(); ++i)

if (\_clients.at(i)->getName() != name && \_clients.at(i)->getAutched())

\_clients.at(i)->\_sok->write(block);

}

void MyServer::doSendToAllUserLeft(QString name)

{

QByteArray block;

QDataStream out(&block, QIODevice::WriteOnly);

out << (quint16)0 << MyClient::comUserLeft << name;

out.device()->seek(0);

out << (quint16)(block.size() - sizeof(quint16));

for (int i = 0; i < \_clients.length(); ++i)

if (\_clients.at(i)->getName() != name && \_clients.at(i)->getAutched())

\_clients.at(i)->\_sok->write(block);

}

void MyServer::doSendToAllMessage(QString message, QString fromUsername)

{

QByteArray block;

QDataStream out(&block, QIODevice::WriteOnly);

out << (quint16)0 << MyClient::comMessageToAll << fromUsername << message;

out.device()->seek(0);

out << (quint16)(block.size() - sizeof(quint16));

for (int i = 0; i < \_clients.length(); ++i)

if (\_clients.at(i)->getAutched())

\_clients.at(i)->\_sok->write(block);

}

void MyServer::doSendToAllServerMessage(QString message)

{

QByteArray block;

QDataStream out(&block, QIODevice::WriteOnly);

out << (quint16)0 << MyClient::comPublicServerMessage << message;

out.device()->seek(0);

out << (quint16)(block.size() - sizeof(quint16));

for (int i = 0; i < \_clients.length(); ++i)

if (\_clients.at(i)->getAutched())

\_clients.at(i)->\_sok->write(block);

}

void MyServer::doSendServerMessageToUsers(QString message, const QStringList &users)

{

QByteArray block;

QDataStream out(&block, QIODevice::WriteOnly);

out << (quint16)0 << MyClient::comPrivateServerMessage << message;

out.device()->seek(0);

out << (quint16)(block.size() - sizeof(quint16));

for (int j = 0; j < \_clients.length(); ++j)

if (users.contains(\_clients.at(j)->getName()))

\_clients.at(j)->\_sok->write(block);

}

void MyServer::doSendMessageToUsers(QString message, const QStringList &users, QString fromUsername)

{

QByteArray block, blockToSender;

QDataStream out(&block, QIODevice::WriteOnly);

out << (quint16)0 << MyClient::comMessageToUsers << fromUsername << message;

out.device()->seek(0);

out << (quint16)(block.size() - sizeof(quint16));

QDataStream outToSender(&blockToSender, QIODevice::WriteOnly);

outToSender << (quint16)0 << MyClient::comMessageToUsers << users.join(",") << message;

outToSender.device()->seek(0);

outToSender << (quint16)(blockToSender.size() - sizeof(quint16));

for (int j = 0; j < \_clients.length(); ++j)

if (users.contains(\_clients.at(j)->getName()))

\_clients.at(j)->\_sok->write(block);

else if (\_clients.at(j)->getName() == fromUsername)

\_clients.at(j)->\_sok->write(blockToSender);

}

QStringList MyServer::getUsersOnline() const

{

QStringList l;

foreach (MyClient \* c, \_clients)

if (c->getAutched())

l << c->getName();

return l;

}

bool MyServer::isNameValid(QString name) const

{

if (name.length() > 20 || name.length() < 5)

return false;

QRegExp r("[A-Za-z0-9\_]+");

return r.exactMatch(name);

}

bool MyServer::isNameUsed(QString name) const

{

for (int i = 0; i < \_clients.length(); ++i)

if (\_clients.at(i)->getName() == name)

return true;

return false;

}

void MyServer::incomingConnection(qintptr handle)

{

MyClient \*client = new MyClient(handle, this, this);

if (\_widget != 0)

{

connect(client, SIGNAL(addUserToGui(QString)), \_widget, SLOT(onAddUserToGui(QString)));

connect(client, SIGNAL(removeUserFromGui(QString)), \_widget, SLOT(onRemoveUserFromGui(QString)));

connect(client, SIGNAL(messageToGui(QString,QString,QStringList)), \_widget, SLOT(onMessageToGui(QString,QString,QStringList)));

}

connect(client, SIGNAL(removeUser(MyClient\*)), this, SLOT(onRemoveUser(MyClient\*)));

\_clients.append(client);

}

void MyServer::onRemoveUser(MyClient \*client)

{

\_clients.removeAt(\_clients.indexOf(client));

}

void MyServer::onMessageFromGui(QString message, const QStringList &users)

{

if (users.isEmpty())

doSendToAllServerMessage(message);

else

doSendServerMessageToUsers(message, users);

}