Министерство науки и высшего образования РФ

Пензенский государственный университет

Кафедра «Вычислительная техника»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

по дисциплине «Программирование на языке Java»

на тему «Разработка многомодульного приложения на языке Java»

Выполнила студентка группы 21ВВП1:

Алёшина А.В.

Приняли:

Юрова О.В.

Карамышева Н.С.

Пенза 2024

**ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**Факультет Вычислительной техники

Кафедра “Вычислительная техника”

“УТВЕРЖДАЮ”

Зав. кафедрой ВТ

профессор М. А. Митрохин

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_20\_\_г

**ЗАДАНИЕ**

**на курсовое проектирование по курсу**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**Студенту\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Группа\_\_\_\_\_\_\_\_\_\_\_  
Тема\_проекта\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Исходные данные (технические требования) на проектирование

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Объем работы по курсу**

1. Расчетная часть

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Графическая часть

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Экспериментальная часть

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Срок выполнения проекта по разделам

1\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_3\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_4\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_5\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_6\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_7\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_8\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата выдачи задания “\_\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата защиты проекта “\_\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Руководитель\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Задание получил**      “\_\_\_\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_г.

**Студент\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Оглавление**

[Введение 5](#_Toc167698943)

[1. Постановка задачи 6](#_Toc167698944)

[2. Выбор решения 7](#_Toc167698945)

[3. Описание программы 9](#_Toc167698946)

[4. Описание способа огранизации пользовательского интерфейса 12](#_Toc167698947)

[5. Описание работы программы 13](#_Toc167698948)

[Заключение 19](#_Toc167698949)

[Литература 20](#_Toc167698950)

[Приложение А. Листинг программы 21](#_Toc167698951)

[Приложение А.1. Мейн файл «BattleSea.java». 21](#_Toc167698952)

[Приложение А.2. Раздел «panels». 22](#_Toc167698953)

[Приложение А.3. Комплектующие «panels». 28](#_Toc167698954)

[Приложение А.4. Раздел «Connection» Server. 45](#_Toc167698955)

[Приложение B. UML диаграммы 50](#_Toc167698956)

[Приложение B.1 UML – диаграмма последовательности действий 50](#_Toc167698957)

[Приложение B.2 UML – диаграмма протокола взаимодействия клиента и сервера. 51](#_Toc167698958)

[Приложение B.3 UML – диаграмма взаимодействия клиентов. 51](#_Toc167698959)

Введение

«Клиент - сервер» — вычислительная или сетевая архитектура, в которой задания или сетевая нагрузка распределены между поставщиками услуг, называемыми серверами, и заказчиками услуг, называемыми клиентами. Фактически клиент и сервер — это программное обеспечение. Обычно эти программы расположены на разных вычислительных машинах и взаимодействуют между собой через вычислительную сеть посредством сетевых протоколов, но они могут быть расположены также и на одной машине. Программы-серверы ожидают от клиентских программ запросы и предоставляют им свои ресурсы в виде данных или в виде сервисных функций (например, работа с электронной почтой, общение посредством систем мгновенного обмена сообщениями или просмотр web-страниц во всемирной паутине). Поскольку одна программа-сервер может выполнять запросы от множества программ-клиентов, её размещают на специально выделенной вычислительной машине, настроенной особым образом, как правило, совместно с другими программами-серверами, поэтому производительность этой машины должна быть высокой. Из-за особой роли такой машины в сети, специфики её оборудования и программного обеспечения, её также называют сервером, а машины, выполняющие клиентские программы, соответственно, клиентами.

В данном курсовом проекте разработана программа клиент-серверного приложения, которое позволяет играть пользователям в игру «Морской Бой».

1. Постановка задачи

Разработать программу, реализующую игру «Морской бой» между пользователями. Приложение должно обладать графическим интерфейсом и использовать следующие технологии:

1. Java Collection Framework
2. Механизм обработки исключительных ситуаций
3. Java Stream API
4. Java Multithreading
5. Сетевое взаимодействие.

Для разработки данного проекта был выбран много платформенный объектно-ориентированный язык Java. Это быстрый, безопасный и надёжный язык программирования для всего: от мобильных приложений до приложений для работы с большими данными и серверных технологий. Язык Java содержит в себе развивающейся API передачи данных по всемирной сети, так же придает особое значение безопасности, включая безопасность языка, безопасную передачу данных и управление доступом. Эти качества очень важны для построения клиент-серверных приложений.

В качестве среды для разработки был использован Apache NetBeans. Ключевой компонент NetBeans IDE - это бесплатная интегрированная среда разработки с открытым исходным кодом для создания программного обеспечения. Она позволяет создавать веб-приложения, корпоративное, десктопное и мобильное программное обеспечение. NetBeans предлагает инструменты и функции, которые помогают ускорить и облегчить процесс разработки. Он обладает интуитивным интерфейсом, автозаполнением кода, функцией перетаскивания элементов и другими возможностями, которые сокращают время, затрачиваемое на написание кода.

1. Выбор решения

Предметная область данного проекта - игра «Морской бой».

Для реализации проекта используются:

* Объектно-ориентированный язык Java
* Среда разработки Apache NetBeans
* Java Stream API
* Библиотека Swing
* И другие различные пакеты

Основной задачей данного курсового проекта является реализация взаимодействия клиента и сервера. Для выполнения задачи использовался протокол TCP. Протокол TCP – это сетевой протокол, который «заточен» под соединение. Иными словами, прежде, чем начать обмен данными, данному протоколу требуется установить соединение между двумя хостами. Данный протокол имеет высокую надежность, поскольку позволяет не терять данные при передаче, запрашивает подтверждения о получении от принимающей стороны и в случае необходимости отправляет данные повторно. При этом отправляемые пакеты данных сохраняют порядок отправки, то есть можно сказать, что передача данных упорядочена.

Механизм TCP предоставляет поток данных с предварительной установкой соединения, осуществляет повторный запрос данных в случае потери данных и устраняет дублирование при получении двух копий одного пакета, гарантируя тем самым целостность передаваемых данных и уведомление отправителя о результатах передачи.

Также была использована библиотека Swing для создания графического интерфейса на языке Java.

Игра реализуется с помощью нажатий на клавиши мыши (размещение кораблей и выстрелы по кораблям противника).

Правила игры следующие: в поле 10х10 пользователь расставляет свои корабли (корабли не должны лежать друг на друге, стоять рядом, пересекаться). Точно так же второй игрок на своем игровом поле расставляет свои корабли. Количество кораблей 10, а именно: один четырехпалубный (одна палуба соответствует одной клетке игрового поля), два трехпалубных, три двухпалубный и четыре однопалубных.

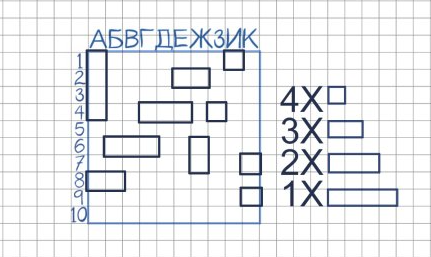


Рисунок 1. Расстановка кораблей на поле.

После расстановки кораблей пользователи поочередно «стреляют» по клеткам игрового поля противника. Победа достается тому, кто первым уничтожит все корабли противника. Ниже наглядно показан процесс игры и варианты действий для пользователей.

Для подключения игроков, создаётся сервер. Для этого один из игроков создаёт комнату и задаётся пароль к ней. Второй же подключается к уже созданному серверу.

1. Описание программы

Программа сервера является консольным приложением. Основной класс сервера находится в Server.java и содержит описание нити, которая создается при появлении нового клиента. Для взаимодействия с клиентами используется класс Connection.

Программа клиента разделена на несколько классов: View, Ship, Box, Model и Controller. Они обеспечивают логику программы и отрисовку на игровом поле.

Ship.java – это класс, который отвечает за создание и хранение кораблей. В нём содержатся методы createHorizontalShip и createVerticalShip. которыесоздают корабли горизонтальной или вертикальной ориентации на поле и записывают их в класс Box.

Box.java – это класс, который создаёт массив для хранения кораблей и управления их состоянием (подбиты или нет).

View.java – это класс, который отвечает за графический интерфейс игры, перерисовку игровых полей благодаря методам repaintMyField, repaintEnemyField и отправку выстрелов на сервер за счёт метода sendShot.

**Клиент – серверное взаимодействие**

Создается объект класса Connection с сокетом из класса Server. Server получает матрицу игрового поля и списки кораблей игроков. Затем объект класса ThreadConnection запускает или прерывает нить в зависимости от флага stopCicle. В этом потоке сервер обменивается с пользователями сообщением о состоянии их игровых полей.

При достижении условий поражения DEFEAT или прерывания связи DISCONNECT, флаг stopCicle устанавливается true, нить прерывается, и игра завершается.

UML – диаграмма последовательности действий

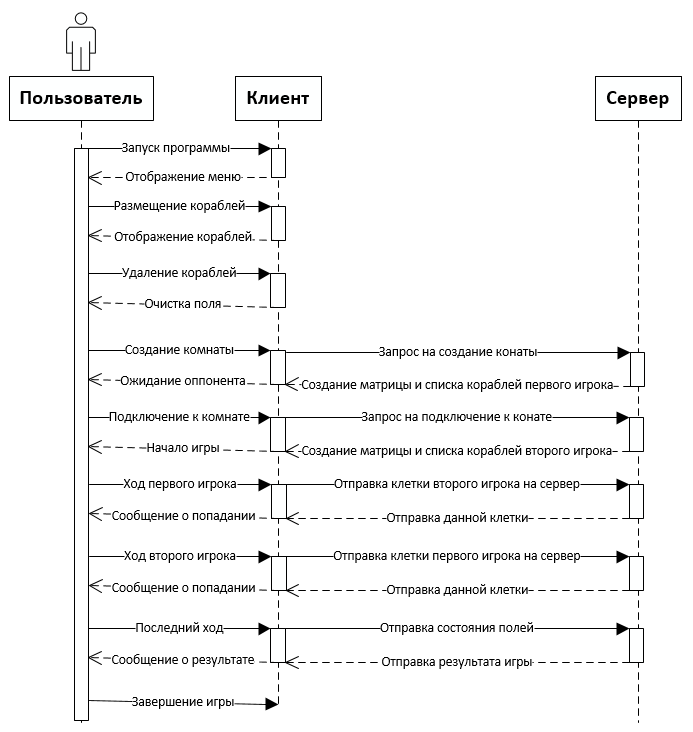


Рисунок2. UML - диаграмма последовательности действий

UML – диаграмма протокола взаимодействия клиента и сервера.

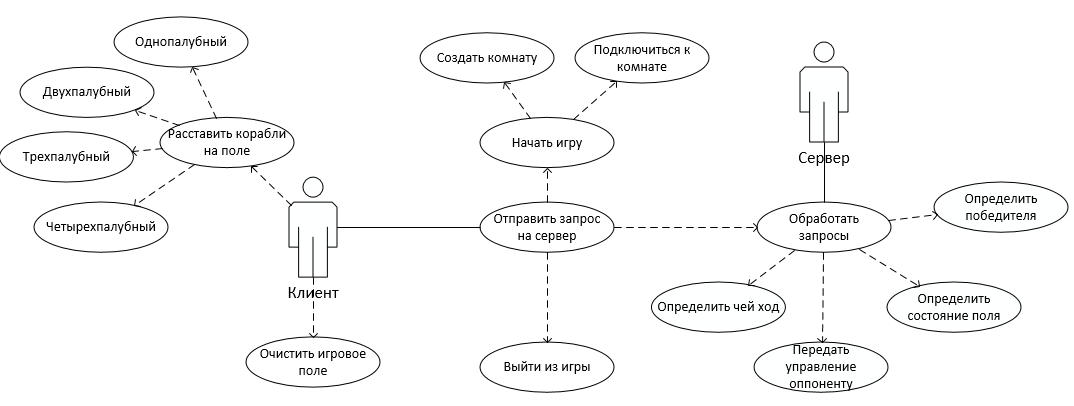


Рисунок 3. UML - диаграмма протокола взаимодействия клиента и сервера.

UML – диаграмма взаимодействия клиентов.



Рисунок4. UML - диаграмма взаимодействия клиентов.

1. Описание способа огранизации пользовательского интерфейса

Игровое поле представляет собой JPanel размером 400x400 пикселов. На этой панели располагается решетка из черных квадратов размером 40x40 пикселей. Каждый квадрат представляет собой ячейку игрового поля. Таким же является и поле противника. На этой панели также располагаются картинки кораблей и подбитых кораблей. Каждый корабль представлен списком ячеек, которые хранят информацию о координатах, статусе (подбит или нет) и отображаемой картинке. Каждый корабль имеет ArrayList, хранящий координаты *x* и *y*, картинки палуб и флаг подбита ли данная палуба или нет.

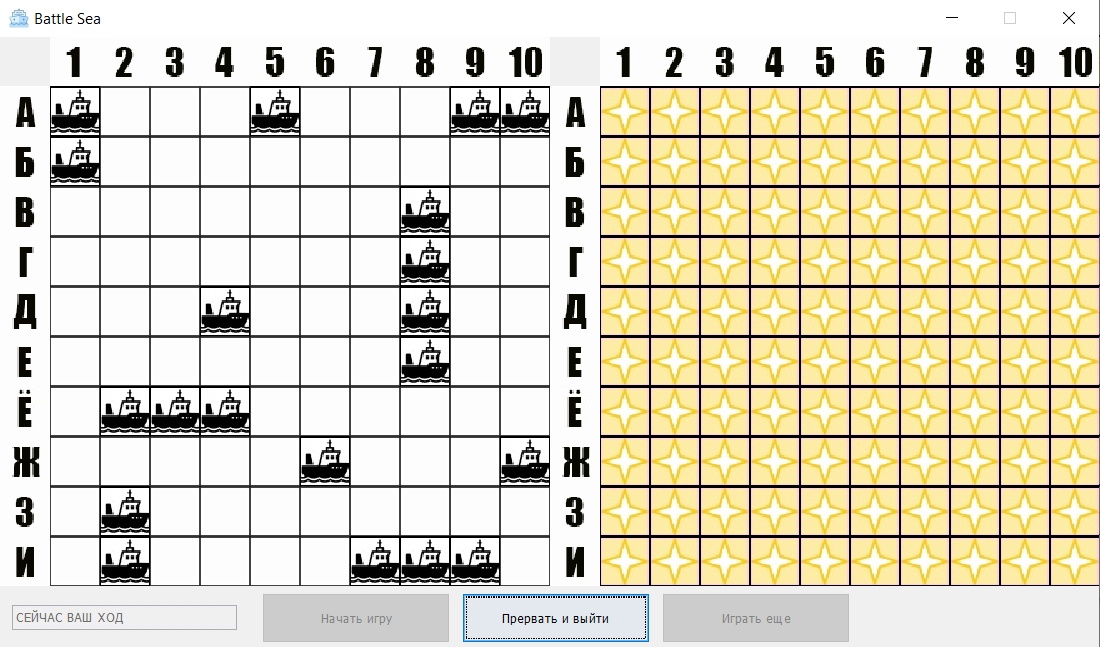


Рисунок 5. Окно ожидания подключения.

Когда игрок кликает левой кнопкой мыши на поле противника, проверяется, совпадает ли координата клика с координатами кораблей противника из списка. В случае совпадения, соответствующая ячейка открывается, отображая картинку подбитой части корабля. Если не совпадает, то поле заменяется на соответствующую картинку промаха. После этого ход передается противнику, и установленный ранее обработчик событий мыши mouseListener удаляется с нашего поля и добавляется на поле противника.

1. Описание работы программы

Ниже представлена работа приложения. Для начала необходимо запустить стартовое окно программы, в котором первый игрок будет расставлять корабли:



Рисунок 6. Окно расстановки кораблей.

После расстановки всех кораблей, необходимо начать игру. Для этого пользователь должен придумать четырёхзначный код комнаты, либо подключиться к уже созданной комнате.

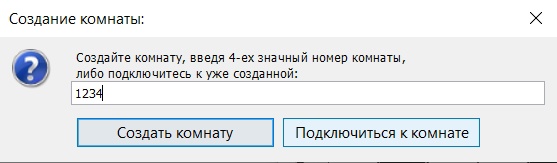


Рисунок 7. Окно создания сервера.

После запуска придёт уведомление о успешном создании комнаты и ожидании подключения оппонента.

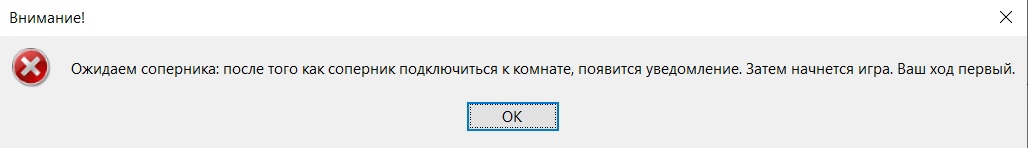


Рисунок 8. Окно ожидания подключения.

При подключении второго игрока в окне приложения отображается сообщение «Вы успешно подключились к комнате. Ваш соперник ходит первым.».

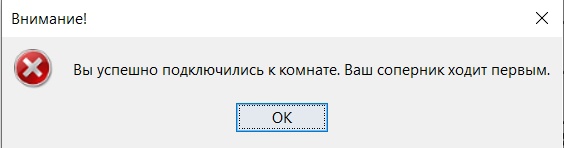


Рисунок 9. Подключение второго игрока.

У первого же игрока появится уведомление о том, что второй игрок подключился.

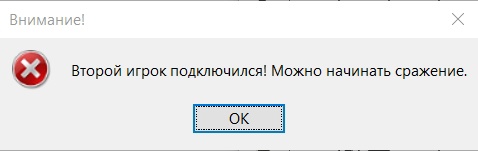


Рисунок 10. Подключение первого игрока.

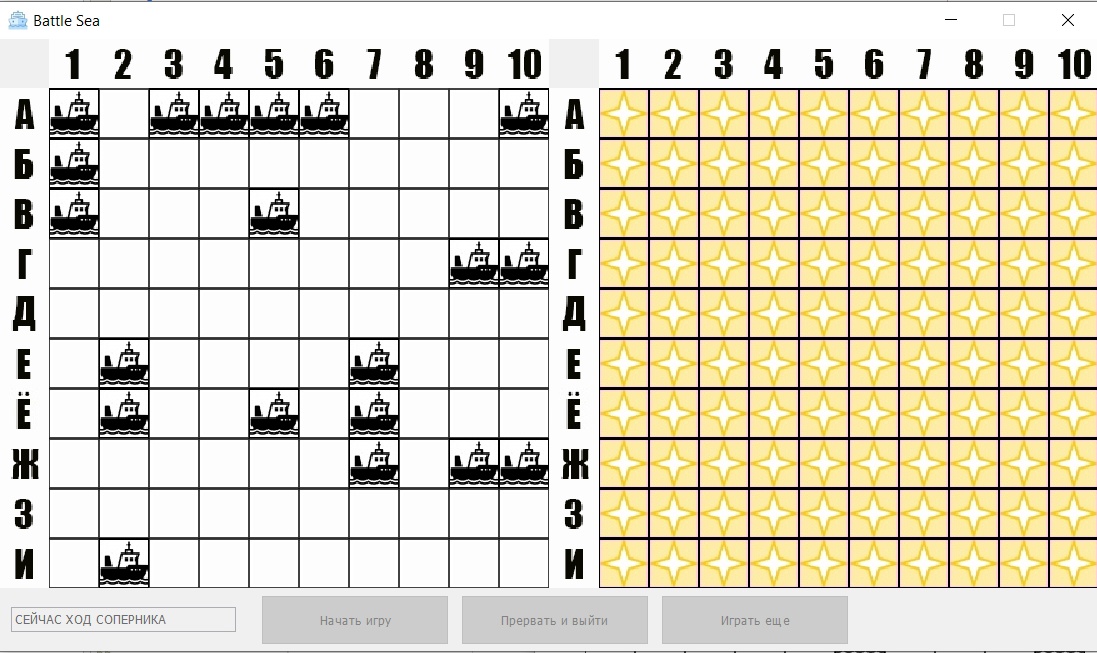


Рисунок 11. Начало игры.

По ходу боя игроки должны по очереди отстреливать поле оппонента. Если пользователь не попадает, то данная клетка будет отмечаться точкой.

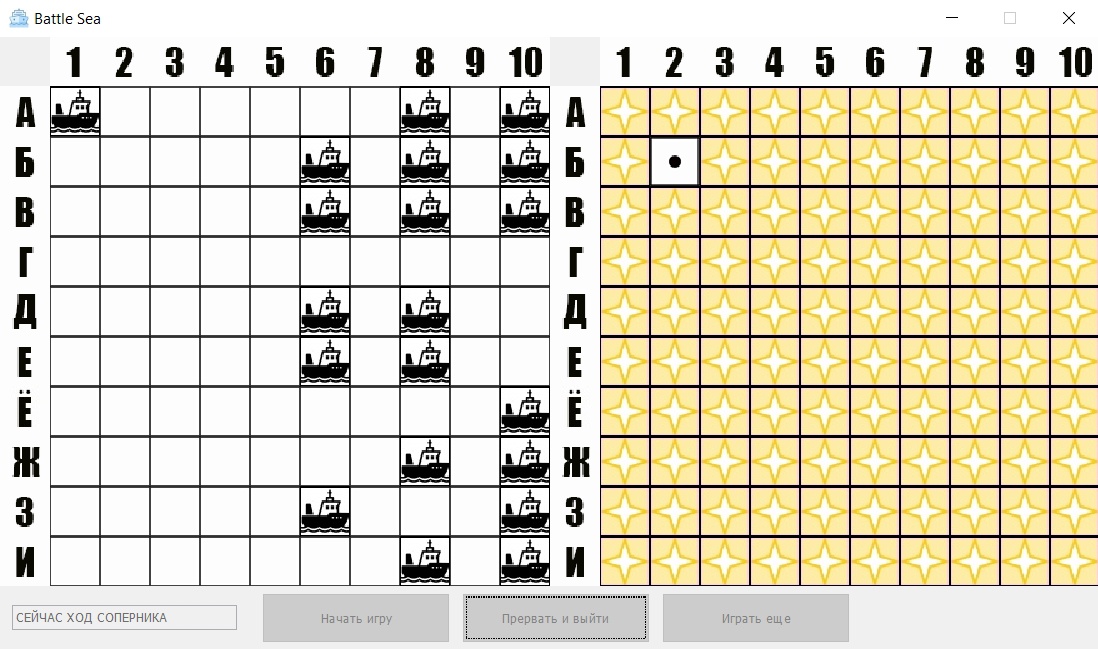


Рисунок 12. Промах игрока.

В случае попадания по кораблю, он станет перечёркнут на полях обоих игроков.

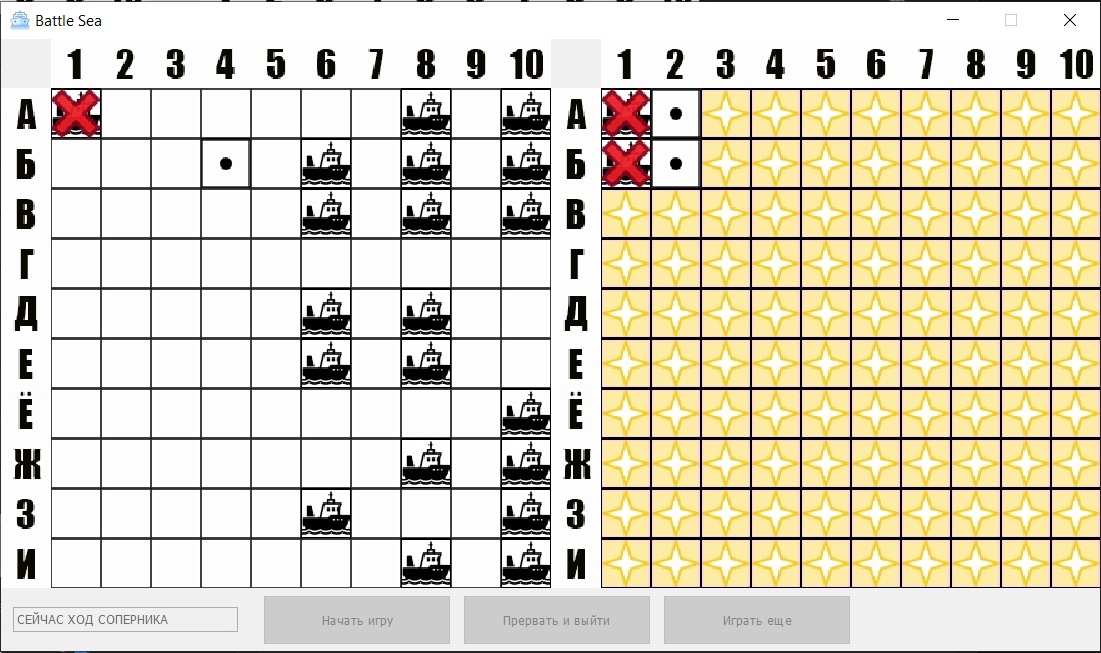


Рисунок 13. Попадание по кораблю оппонента.

В случае полного уничтожения корабля, зона вокруг него станет недоступна для выстрелов. Потому что корабли в этой области не могут находиться.

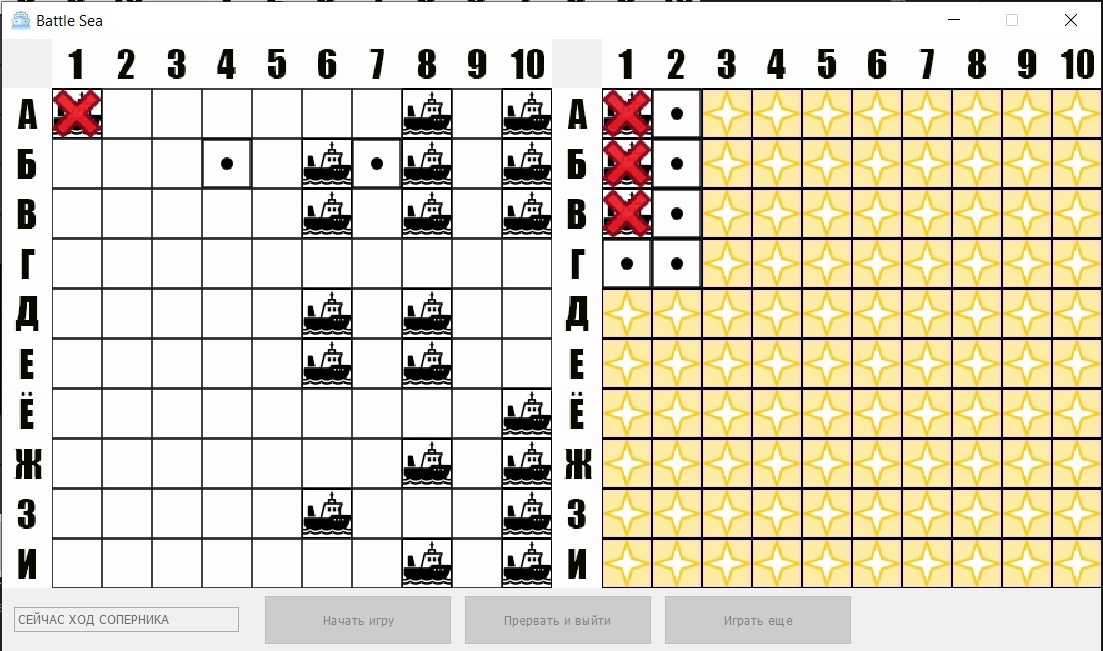


Рисунок 14. Уничтожение корабля оппонента.

Уничтожив все корабли противника, игра закончится. Победителю придёт уведомление «Все корабли противника уничтожены. Вы одержали победу!».

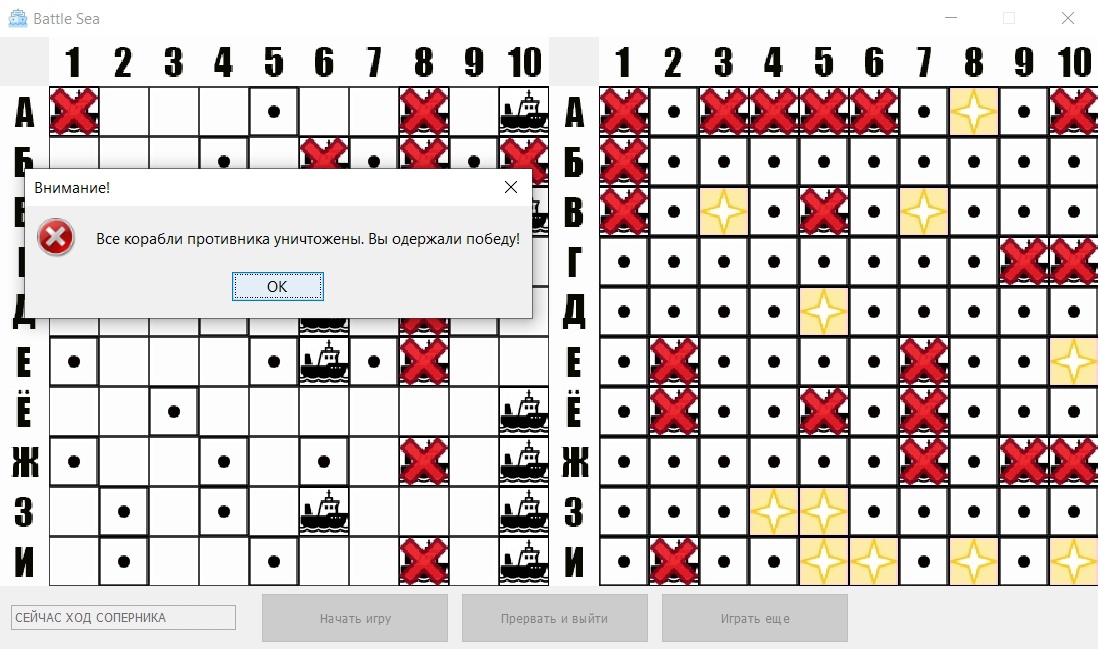


Рисунок 15. Победа.

Проигравшему игроку придёт уведомление его о поражении.

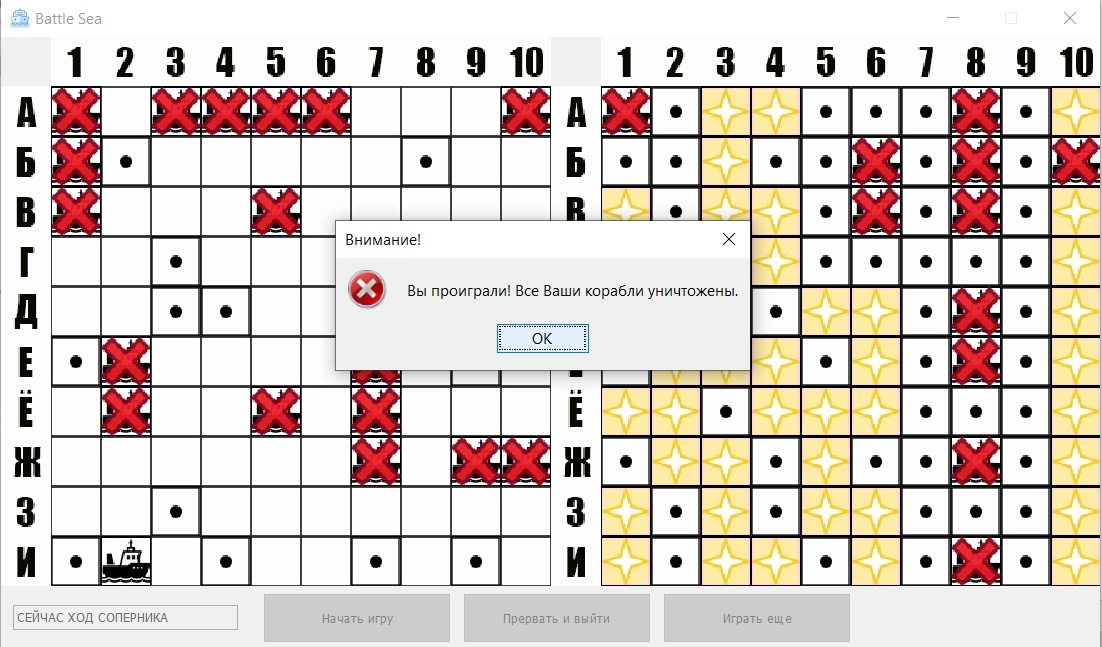


Рисунок 16. Поражение.

Чтобы покинуть игру досрочно, необходимо нажать на кнопку «Прервать и выйти». В этом случае игроку, покинувшему игру будет засчитано техническое поражение, а оппоненту будет засчитана техническая победа.

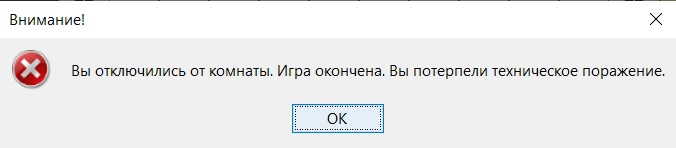


Рисунок 17. Техническое поражение.

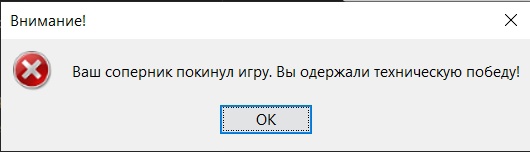


Рисунок 18. Техническое победа.

Заключение

В ходе выполнения курсового проектирования были изучены принципы работы протоколов TCP/IP. Были получены навыки разработки многомодульного приложения с пользовательским интерфейсом. Получены навыки разработки клиент-серверных приложений.

На данный период времени разработка является актуальной и имеет перспективы для улучшения. Актуальность темы разработки обусловлена малым количеством аналогов данного проекта представленных в данный период времени. В итоге была разработана игра «Морской бой» на языке Java среде Apache NetBeans.

Литература

1. «Как программировать на Java» Дейтел, Х.М.
2. «Язык программирования Java» А.Н. Семочкин
3. «Язык программирования Java и среда NetBeans» Монахов В.В.
4. «Java. Полное руководство» Герберт Шилдт

Приложение А. Листинг программы

Приложение А.1. Мейн файл «BattleSea.java».

package BattleSea;

import mvc.\*;

public class BattleSea {

public static void main(String[] args) {

View view = new View();

Model model = new Model();

Controller controller = new Controller(view, model);

view.setController(controller);

view.setModel(model);

view.init();

}

}

Приложение А.2. Раздел «panels».

Файл ChoosePanel.java

package mvc.panels;

import mvc.\*;

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

public class ChoosePanel extends JPanel {

private View view;

private JPanel panelRadio;

private JPanel panelPlacement;

private JRadioButton oneDeck;

private JRadioButton twoDeck;

private JRadioButton threeDeck;

private JRadioButton fourDeck;

private JRadioButton vertical;

private JRadioButton horizontal;

private JButton clearField;

private ButtonGroup groupDeck;

private ButtonGroup groupPlacement;

public ChoosePanel(View view) {

this.view = view;

setLayout(null);

this.setPreferredSize(new Dimension(255, 400));

panelRadio = new JPanel();

panelRadio.setLayout(new BoxLayout(panelRadio, BoxLayout.Y\_AXIS));

panelRadio.setBounds(13, 190, 230, 130);

panelPlacement = new JPanel();

panelPlacement.setLayout(new BoxLayout(panelPlacement, BoxLayout.Y\_AXIS));

panelPlacement.setBounds(13, 330, 230, 80);

clearField = new JButton("Убрать все корабли");

clearField.setBounds(13, 410, 230, 30);

clearField.addActionListener(new ActionClearField());

panelRadio.setBorder(BorderFactory.createTitledBorder("Палубность"));

panelPlacement.setBorder(BorderFactory.createTitledBorder("Ориентация корабля"));

oneDeck = new JRadioButton();

setNameOneDeck(4);

twoDeck = new JRadioButton();

setNameTwoDeck(3);

threeDeck = new JRadioButton();

setNameThreeDeck(2);

fourDeck = new JRadioButton();

setNameFourDeck(1);

vertical = new JRadioButton("Вертикальная");

horizontal = new JRadioButton("Горизонтальная");

groupDeck = new ButtonGroup();

groupPlacement = new ButtonGroup();

panelRadio.add(oneDeck);

panelRadio.add(twoDeck);

panelRadio.add(threeDeck);

panelRadio.add(fourDeck);

panelPlacement.add(vertical);

panelPlacement.add(horizontal);

add(panelRadio);

add(panelPlacement);

add(clearField);

groupDeck.add(oneDeck);

groupDeck.add(twoDeck);

groupDeck.add(threeDeck);

groupDeck.add(fourDeck);

groupPlacement.add(vertical);

groupPlacement.add(horizontal);

}

Файл EnemyField.java

package mvc.panels;

import mvc.Picture;

import mvc.View;

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.MouseAdapter;

import java.awt.event.MouseEvent;

import java.awt.event.MouseListener;

public class EnemyField extends JPanel {

private View view;

public EnemyField(View view) {

this.view = view;

this.setPreferredSize(new Dimension(Picture.COLUMNS \* Picture.IMAGE\_SIZE, Picture.ROWS \* Picture.IMAGE\_SIZE));

}

@Override

public void paintComponent(Graphics g) {

super.paintComponent(g);

view.repaintEnemyField(g);

}

//добавляет слушателя к панели поля соперника

public void addListener() {

addMouseListener(new ActionMouse());

}

Файл MyField.java

package mvc.panels;

import mvc.\*;

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.MouseAdapter;

import java.awt.event.MouseEvent;

public class MyField extends JPanel {

private View view;

private ChoosePanel choosePanel;

public void setChoosePanel(ChoosePanel choosePanel) {

this.choosePanel = choosePanel;

}

public MyField(View view) {

this.view = view;

this.setPreferredSize(new Dimension(Picture.COLUMNS \* Picture.IMAGE\_SIZE, Picture.ROWS \* Picture.IMAGE\_SIZE));

this.addMouseListener(new ActionMouse());

}

Файл PanelButtons.java

package mvc.panels;

import mvc.View;

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

public class PanelButtons extends JPanel {

private View view;

private JTextField infoField;

private JButton startGameButton;

private JButton exitButton;

private JButton restartGameButton;

public PanelButtons(View view) {

this.view = view;

setLayout(null);

setPreferredSize(new Dimension(400, 50));

infoField = new JTextField();

setTextInfo("РАССТАВЛЯЕМ КОРАБЛИ");

infoField.setEnabled(false);

infoField.setBounds(10, 15, 180, 20);

startGameButton = new JButton("Начать игру");

startGameButton.setBounds(210, 5, 150, 40);

startGameButton.addActionListener(new ActionButtonStartClass());

exitButton = new JButton("Прервать и выйти");

exitButton.setBounds(370, 5, 150, 40);

exitButton.addActionListener(new ActionButtonDisconnect());

exitButton.setEnabled(false);

restartGameButton = new JButton("Играть еще");

restartGameButton.setBounds(530, 5, 150, 40);

restartGameButton.setEnabled(false);

restartGameButton.addActionListener(new ActionButtonRestartGame());

add(infoField);

add(startGameButton);

add(exitButton);

add(restartGameButton);

}

public JButton getRestartGameButton() {

return restartGameButton;

}

public JButton getStartGameButton() {

return startGameButton;

}

public JButton getExitButton() {

return exitButton;

}

public void setTextInfo(String text) {

infoField.setText(text.toUpperCase());

}

//класс слушатель для кнопки "Начать игру"

private class ActionButtonStartClass implements ActionListener {

Приложение А.3. Комплектующие «panels».

Файл Box.java

package mvc;

import java.io.Serializable;

public class Box implements Serializable {

//координаты бокса х и y на поле

private int x;

private int y;

private Picture picture; //значение соответствующей картинки для даннго бокса

private boolean isOpen = false; //флаг открыт ли данный бокс (== стрелял ли соперник в данный бокс)

public boolean isOpen() {

return isOpen;

}

public void setOpen(boolean open) {

isOpen = open;

}

public Picture getPicture() {

return picture;

}

public void setPicture(Picture picture) {

this.picture = picture;

}

public Box(Picture picture, int x, int y) {

this.picture = picture;

this.x = x;

this.y = y;

}

public int getX() {

return x;

}

public int getY() {

return y;

}

}

Файл Controller.java

package mvc;

import mvc.connect.Connection;

import mvc.connect.Message;

import mvc.connect.MessageType;

import mvc.connect.Server;

import java.io.IOException;

import java.net.Socket;

import java.util.List;

public class Controller {

private View view;

private Model model;

private Connection connection;

public Controller(View view, Model model) {

this.view = view;

this.model = model;

}

//загружает наше пустое игровое поле

public void loadEmptyMyField() {

//очистка списков всех типов кораблей

model.getShipsOneDeck().clear();

model.getShipsTwoDeck().clear();

model.getShipsThreeDeck().clear();

model.getShipsFourDeck().clear();

model.setMyField(new Box[Picture.COLUMNS][Picture.ROWS]);

//присваение элементам матрицы нашего игрового поля значений

for (int i = 0; i < Picture.ROWS; i++) {

for (int j = 0; j < Picture.COLUMNS; j++) {

if (i == 0 && j == 0) continue;

else if (i == 0 && j != 0) { //если это первый столбец то присваиваем значение картинок с буквами

model.addBoxInField(model.getMyField(), new Box(Picture.valueOf("SYM" + j), Picture.IMAGE\_SIZE \* i, Picture.IMAGE\_SIZE \* j));

} else if (i != 0 && j == 0) { //если это первая строка присваиваем значение картинок с цифрами

model.addBoxInField(model.getMyField(), new Box(Picture.valueOf("NUM" + i), Picture.IMAGE\_SIZE \* i, Picture.IMAGE\_SIZE \* j));

} else { //в остальных случаях знаяение картинки с пустой клеткой

model.addBoxInField(model.getMyField(), new Box(Picture.EMPTY, Picture.IMAGE\_SIZE \* i, Picture.IMAGE\_SIZE \* j));

}

}

}

}

//добавляет корабль предварительно проверяя - не пересекается ли новый корабль с уже добавленным ораблем

public void addShip(Ship ship) {

List<Box> boxesOfShip = ship.getBoxesOfShip();

for (Box boxShip : boxesOfShip) {

if (checkAround(boxShip, boxesOfShip)) {

boxesOfShip.clear();

return;

}

}

if (boxesOfShip.size() != 0) model.addShip(ship);

}

//удаляет корабль по заданным координатам

public Ship removeShip(int x, int y) {

List<Ship> allShips = model.getAllShips(); //получаем список всех добавленных кораблей

for (Ship ship : allShips) {

for (Box box : ship.getBoxesOfShip()) {

if (x == box.getX() && y == box.getY()) { //перебираем корабли, затем их боксы,

// если координаты бокса совпадают с заданными - удаляем корабль

model.removeShip(ship);

return ship;

}

}

}

return null;

}

//метод проверки на пересечение с другими короблями

private boolean checkAround(Box box, List<Box> boxesOfShip) {

int myX = box.getX();

int myY = box.getY();

for (int i = myX - Picture.IMAGE\_SIZE; i <= myX + Picture.IMAGE\_SIZE; i += Picture.IMAGE\_SIZE) {

for (int j = myY - Picture.IMAGE\_SIZE; j <= myY + Picture.IMAGE\_SIZE; j += Picture.IMAGE\_SIZE) {

Box boxFromMatrix = model.getBox(model.getMyField(), i, j);

if (boxFromMatrix != null && boxFromMatrix.getPicture() == Picture.SHIP && !boxesOfShip.contains(boxFromMatrix)) {

View.callInformationWindow("Сюда нельзя добавлять корабль - пересечение с другим");

boxesOfShip.clear();

return true;

}

}

}

return false;

}

//метод открывающий путсые клетки вокруг подбитого корабля на поле противника

public void openBoxesAround(Box boxShot) {

Ship ship = model.getShipOfEnemy(boxShot); //по боксу в который выстрелили получаем корабль

if (ship != null) {

//если число палуб == числу подбитых (открытых) боксов корабля - то открываем все пустые клетки вокруг

if (ship.getCountDeck() == getCountOpenBoxOfShip(ship)) model.openAllBoxesAroundShip(ship);

//иначе если подбита (отрыт бокс) только одна палуба - ничего не делаем

else if (getCountOpenBoxOfShip(ship) == 1) return;

// в остальных случаях открываем пустые клетки вокруг подбитых палуб

else {

for (Box box : ship.getBoxesOfShip()) {

if (box.isOpen())

model.openBoxAroundBoxOfShipEnemy(box.getX(), box.getY(), ship.isHorizontalPlacement());

}

}

}

}

//метод возвращает количество открытых боксов (подбитых палуб) корабля

public int getCountOpenBoxOfShip(Ship ship) {

int count = 0;

for (Box box : ship.getBoxesOfShip()) {

if (box.isOpen()) count++;

}

return count;

}

//проверка на окончание игры

private boolean checkEndGame() {

List<Box> allBoxesOfShip = model.getAllBoxesOfShips(); //получаем список всех своих кораблей

for (Box box : allBoxesOfShip) {

//проверяем сли хотя бы один корабль имеет значение картини SHIP то игра не окончена

if (box.getPicture() == Picture.SHIP) return false;

}

return true;

}

//проверка на полный комплект добавленный кораблей перед стартом игры

public boolean checkFullSetShips() {

//просто проверяем количство корабля в соответствующих списках по каждому типу кораблей

if (model.getShipsOneDeck().size() == 4 &&

model.getShipsTwoDeck().size() == 3 &&

model.getShipsThreeDeck().size() == 2 &&

model.getShipsFourDeck().size() == 1) return true;

else return false;

}

Файл Model.java

package mvc;

import java.util.ArrayList;

import java.util.List;

public class Model {

private Box[][] myField = new Box[Picture.COLUMNS][Picture.ROWS]; //матрица с боксами нашего игрового поля

private Box[][] enemyField = new Box[Picture.COLUMNS][Picture.ROWS]; //матрица с боксами ирового поля соперника

private List<Ship> shipsOneDeck = new ArrayList<>(); //список всех наших однопалубных кораблей

private List<Ship> shipsTwoDeck = new ArrayList<>(); //список всех наших двухпалубных кораблей

private List<Ship> shipsThreeDeck = new ArrayList<>(); //список всех наших трехпалубных кораблей

private List<Ship> shipsFourDeck = new ArrayList<>(); //список всех наших четырехпалубных кораблей

private List<Ship> allShipsOfEnemy = new ArrayList<>(); //список всех кораблей соперниа

public void setAllShipsOfEnemy(List<Ship> allShipsOfEnemy) {

this.allShipsOfEnemy = allShipsOfEnemy;

}

//метод, который возвращает список ВСЕХ наших ораблей

public List<Ship> getAllShips() {

List<Ship> allBoxesOfShips = new ArrayList<>();

allBoxesOfShips.addAll(shipsFourDeck);

allBoxesOfShips.addAll(shipsThreeDeck);

allBoxesOfShips.addAll(shipsTwoDeck);

allBoxesOfShips.addAll(shipsOneDeck);

return allBoxesOfShips;

}

//метод, который возвращает ВСЕ боксы из ВСЕХ наших кораблей

public List<Box> getAllBoxesOfShips() {

List<Box> allBoxes = new ArrayList<>();

List<Ship> allShips = getAllShips();

for (Ship ship : allShips) {

allBoxes.addAll(ship.getBoxesOfShip());

}

return allBoxes;

}

public List<Ship> getShipsOneDeck() {

return shipsOneDeck;

}

public List<Ship> getShipsTwoDeck() {

return shipsTwoDeck;

}

public List<Ship> getShipsThreeDeck() {

return shipsThreeDeck;

}

public List<Ship> getShipsFourDeck() {

return shipsFourDeck;

}

public Box[][] getMyField() {

return myField;

}

public void setMyField(Box[][] myField) {

this.myField = myField;

}

public Box[][] getEnemyField() {

return enemyField;

}

public void setEnemyField(Box[][] enemyField) {

this.enemyField = enemyField;

}

//метод, устанавливающий значение указанного бокс в указанную матрицу (игрового поля)

public void addBoxInField(Box[][] fieldBox, Box box) {

//по координатам бокса вычисляем индексы соответствующего места в матрице

int i = box.getX() / Picture.IMAGE\_SIZE;

int j = box.getY() / Picture.IMAGE\_SIZE;

fieldBox[i][j] = box;

}

//метод, возвращающий бокс из уазанной матрицы (игрового поля) по координатам панели отрисовки игрового поля

public Box getBox(Box[][] field, int x, int y) {

int i = x / Picture.IMAGE\_SIZE;

int j = y / Picture.IMAGE\_SIZE;

int lenght = field.length - 1;

//если координаты указывают на элемент индес которого больше размерности матрицы, то возвращаем null

if (!(i > lenght || j > lenght)) {

return field[i][j];

}

return null;

}

//метод, который устанавливает значение isOpen в true (открывает боксы после попадания в корабль) боксов, находящихся рядом с боксом корабля, определенного входными координатами

public void openBoxAroundBoxOfShipEnemy(int x, int y, boolean isHorizontalPlacement) {

//для горизонтально ориентированного корабля

if (isHorizontalPlacement) {

Box boxUp = getBox(enemyField, x, y - Picture.IMAGE\_SIZE);

if (boxUp != null) boxUp.setOpen(true);

Box boxDown = getBox(enemyField, x, y + Picture.IMAGE\_SIZE);

if (boxDown != null) boxDown.setOpen(true);

}

//для вертиально ориентированного корабля

else {

Box boxLeft = getBox(enemyField, x - Picture.IMAGE\_SIZE, y);

if (boxLeft != null) boxLeft.setOpen(true);

Box boxRight = getBox(enemyField, x + Picture.IMAGE\_SIZE, y);

if (boxRight != null) boxRight.setOpen(true);

}

}

//возвращает по боксу, в который произвели выстрел, корабль противника, если координаты боксШота равный координатам

//бокса одного из корабля противника, иначе возвращаем null

public Ship getShipOfEnemy(Box boxShot) {

for (Ship ship : allShipsOfEnemy) {

for (Box box : ship.getBoxesOfShip()) {

if (boxShot.getX() == box.getX() && boxShot.getY() == box.getY()) {

return ship;

}

}

}

return null;

}

Файл Picture.java

package mvc;

import javax.swing.\*;

import java.awt.\*;

public enum Picture {

CLOSED, //значение для картинки закрытого бокса

DESTROY\_SHIP, //для угичтоженной палубы корабля

EMPTY, //для пустой клети

NUM1, NUM2, NUM3, NUM4, NUM5, NUM6, NUM7, NUM8, NUM9, NUM10, //для нумерации стро игрового поля

POINT, //для пустого отстрелянного бокса

SHIP, //для палубы корабля

INFO, //для информации на панели выбора настроек размещения кораблей

SYM1, SYM2, SYM3, SYM4, SYM5, SYM6, SYM7, SYM8, SYM9, SYM10; //для обозначения буквами столбцов

public static final int COLUMNS = 11; //кол-во столбцов

public static final int ROWS = 11; //ол-во строк

public static final int IMAGE\_SIZE = 40; //размер стороны картинки в писелах

public static Image getImage(String nameImage) {

String fileName = "C:/img/" + nameImage.toLowerCase() + ".png"; //для подгрузки имейджа из ресурсов

ImageIcon icon = new ImageIcon(fileName);

return icon.getImage();

}

}

Файл Ship.java

package mvc;

import java.io.Serializable;

import java.util.ArrayList;

import java.util.List;

public class Ship implements Serializable {

private final int countDeck; //количество палуб

private List<Box> boxesOfShip; //список всех босов данногго корабля

private final boolean isHorizontalPlacement; //ориентация корабля

public List<Box> getBoxesOfShip() {

return boxesOfShip;

}

public boolean isHorizontalPlacement() {

return isHorizontalPlacement;

}

public int getCountDeck() {

return countDeck;

}

public Ship(int countDeck, boolean isHorizontalPlacement) {

this.countDeck = countDeck;

this.isHorizontalPlacement = isHorizontalPlacement;

boxesOfShip = new ArrayList<>(countDeck);

}

//метод создания корабля горизонтальной ориентации по заданным кординатам x и y

public void createHorizontalShip(int x, int y) {

//для того чтобы корабль не отрисовался за пределами игрового поля, высчитываем максимально возможную координату по X для отрисовки с учетом количества палуб

int pointLimitValueForPaint = (Picture.COLUMNS - countDeck) \* Picture.IMAGE\_SIZE;

for (int i = 0; i < countDeck; i++) {

Box newBox;

//если Х больше максимально допустимой точки отрисовки, то координата по Х для бокса = pointLimitValueForPaint + i \* Picture.IMAGE\_SIZE

if (x > pointLimitValueForPaint) {

newBox = new Box(Picture.SHIP, pointLimitValueForPaint + i \* Picture.IMAGE\_SIZE, y);

boxesOfShip.add(newBox);

} else {

newBox = new Box(Picture.SHIP, (x + i \* Picture.IMAGE\_SIZE), y);

boxesOfShip.add(newBox);

}

}

}

//аналогичный метод для создания корабля с горизонтальной ориентацией

public void createVerticalShip(int x, int y) {

int pointStartPaint = (Picture.ROWS - countDeck) \* Picture.IMAGE\_SIZE;

for (int i = 0; i < countDeck; i++) {

Box newBox;

if (pointStartPaint < y) {

newBox = new Box(Picture.SHIP, x, pointStartPaint + i \* Picture.IMAGE\_SIZE);

boxesOfShip.add(newBox);

} else {

newBox = new Box(Picture.SHIP, x, (y + i \* Picture.IMAGE\_SIZE));

boxesOfShip.add(newBox);

}

}

}

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (o == null || o.getClass() != this.getClass()) return false;

Ship ship = (Ship) o;

return countDeck == ship.getCountDeck() && (boxesOfShip != null && ship.getBoxesOfShip() != null &&

boxesOfShip.hashCode() == ship.getBoxesOfShip().hashCode());

}

}

Файл View.java

package mvc;

import mvc.panels.ChoosePanel;

import mvc.panels.EnemyField;

import mvc.panels.MyField;

import mvc.panels.PanelButtons;

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.MouseListener;

import java.io.IOException;

public class View extends JFrame {

private Controller controller;

private Model model;

private MyField myField; //панель нашего игрового поля

private EnemyField enemyField; //панель игрового поля соперника

private ChoosePanel choosePanel; //панель выбора настроек при добавлении корабля

private PanelButtons panelButtons; //панель кнопок

public View() {

try {

UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());

} catch (Exception e) {

e.printStackTrace();

}

setTitle("Battle Sea");

setDefaultCloseOperation(WindowConstants.EXIT\_ON\_CLOSE);

setLocationRelativeTo(null);

setResizable(false);

setIconImage(Picture.getImage("icon"));

}

public void setController(Controller controller) {

this.controller = controller;

}

public void setModel(Model model) {

this.model = model;

}

//инициализация графического интерфейса

public void init() {

if (enemyField != null) {

remove(enemyField);

remove(myField);

remove(panelButtons);

}

controller.loadEmptyMyField();

add(choosePanel = new ChoosePanel(this), BorderLayout.EAST);

add(myField = new MyField(this), BorderLayout.WEST);

add(panelButtons = new PanelButtons(this), BorderLayout.SOUTH);

myField.setChoosePanel(choosePanel);

pack();

revalidate();

setVisible(true);

}

//метод для вызова информационного диалогового окна с заданныйм текстом

public static void callInformationWindow(String message) {

JOptionPane.showMessageDialog(

null, message,

"Внимание!", JOptionPane.ERROR\_MESSAGE

);

}

//метод для загрузки нашего пустого игровоо поля

public void loadEmptyMyField() {

controller.loadEmptyMyField();

myField.repaint(); //переотрисовка нашего игрового поля

//установка имени радиоБаттонов на панели выбора настроек добавления корабля

choosePanel.setNameOneDeck(4);

choosePanel.setNameTwoDeck(3);

choosePanel.setNameThreeDeck(2);

choosePanel.setNameFourDeck(1);

}

//добавление корабля

public void addShip(Ship ship) {

controller.addShip(ship);

}

//удаление корабля с нашего поля по координатам

public Ship removeShip(int x, int y) {

return controller.removeShip(x, y);

}

//обновляет интерфейс клиента после подключения обоих игроков

public void refreshGuiAfterConnect() {

MouseListener[] listeners = myField.getMouseListeners();

for (MouseListener lis : listeners) {

myField.removeMouseListener(lis); //удаление слушателя у панели нашего игрового поля

}

choosePanel.setVisible(false);

remove(choosePanel); //удаление панели настроек добавления корабля

add(enemyField = new EnemyField(this), BorderLayout.EAST);

Приложение А.4. Раздел «Connection» Server.

Файл Connection.java

package mvc.connect;

import java.io.\*;

import java.net.Socket;

public class Connection implements Closeable {

private final Socket socket;

private final ObjectOutputStream out;

private final ObjectInputStream in;

public Connection(Socket socket) throws IOException {

this.socket = socket;

this.out = new ObjectOutputStream(socket.getOutputStream());

this.in = new ObjectInputStream(socket.getInputStream());

}

public void send(Message message) throws IOException {

synchronized (this.out){

out.writeObject(message);

}

}

public Message receive() throws IOException, ClassNotFoundException {

synchronized (this.in){

Message message = (Message) in.readObject();

return message;

}

}

@Override

public void close() throws IOException {

in.close();

out.close();

socket.close();

}

}

Файл Message.java

package mvc.connect;

import mvc.Box;

import mvc.Ship;

import java.io.Serializable;

import java.util.List;

public class Message implements Serializable {

private int x;

private int y;

private MessageType messageType;

private Box[][] gameField;

private List<Ship> listOfAllShips;

public Message(MessageType messageType, Box[][] gameField, List<Ship> allShipsOfEnemy) {

this.messageType = messageType;

this.gameField = gameField;

this.listOfAllShips = allShipsOfEnemy;

}

public Message(MessageType messageType) {

this.messageType = messageType;

}

}

Файл MessageType.java

package mvc.connect;

public enum MessageType {

FIELD, //при отправке приеме матрицы игрового поля

SHOT, //при отправке приеме выстрела

DEFEAT, //при отправке приеме поражения одного из игроков

ACCEPTED, //при принятии сокета сервером

DISCONNECT, //при отключении клиента

MY\_DISCONNECT;

}

Файл Server.java

package mvc.connect;

import mvc.Box;

import mvc.Ship;

import mvc.View;

import java.io.IOException;

import java.net.ServerSocket;

import java.net.Socket;

import java.util.ArrayList;

import java.util.List;

public class Server extends Thread {

private ServerSocket serverSocket;

private Box[][] fieldPlayer1; //матрица игрового поля игрока 1

private Box[][] fieldPlayer2; //матрица игрового поля игрока 2

private List<Ship> allShipsPlayer1; //список кораблей игрока 1

private List<Ship> allShipsPlayer2; //список кораблей игрока 2

private volatile boolean allPlayersConnected = false; //флаг подключились ли все игроки

private List<Connection> listConnection = new ArrayList<>(); //список коннекшенов всех игроков

public Server(int port) throws IOException {

serverSocket = new ServerSocket(port);

}

@Override

public void run() {

startServer();

}

//запускает сервер

private void startServer() {

try {

while (!allPlayersConnected) { //не всели игроки подключились?

Socket socket = serverSocket.accept(); //принимаем подключение клиента

if (listConnection.size() == 0) { //если в списке еще нет подключений, то...

Connection connection = new Connection(socket);

listConnection.add(connection); //добавляем коннекшн в список

connection.send(new Message(MessageType.ACCEPTED)); //отправляем клиенту о принятии

Message message = connection.receive(); //ждем от клиента матрицу его поля и спислк кораблей

if (message.getMessageType() == MessageType.FIELD) {

//устанавливаем поле и корабли в соотствующие поля

fieldPlayer1 = message.getGameField();

allShipsPlayer1 = message.getListOfAllShips();

}

//запускаем нить основного цикла общения клиента и сервера

new ThreadConnection(connection).start();

}

//аналогично для игрока 2, только после того как он пришлет поле и корабли, отправляем клиентам поля и корабли соперника

else if (listConnection.size() == 1) {

Connection connection = new Connection(socket);

listConnection.add(connection);

connection.send(new Message(MessageType.ACCEPTED));

Message message = connection.receive();

if (message.getMessageType() == MessageType.FIELD) {

fieldPlayer2 = message.getGameField();

allShipsPlayer2 = message.getListOfAllShips();

connection.send(new Message(MessageType.FIELD, fieldPlayer1, allShipsPlayer1));

listConnection.get(0).send(new Message(MessageType.FIELD, fieldPlayer2, allShipsPlayer2));

}

new ThreadConnection(connection).start();

allPlayersConnected = true;

}

}

serverSocket.close();

} catch (Exception e) {

View.callInformationWindow("Возникла ошибка при запуске сервера игровой комнаты.");

}

}

Приложение B. UML диаграммы

Приложение B.1 UML – диаграмма последовательности действий

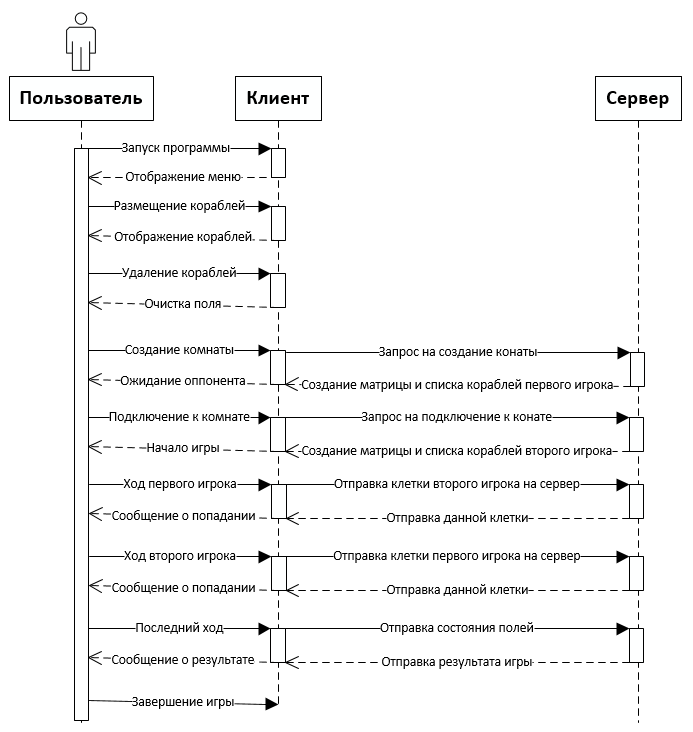


Рисунок19. UML - диаграмма последовательности действий

Приложение B.2 UML – диаграмма протокола взаимодействия клиента и сервера.

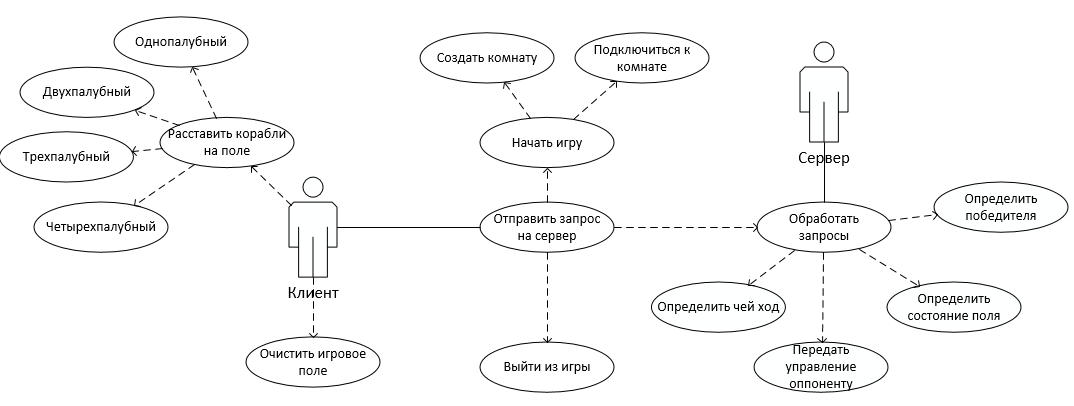


Рисунок 20. UML - диаграмма протокола взаимодействия клиента и сервера.

Приложение B.3 UML – диаграмма взаимодействия клиентов.



Рисунок21. UML - диаграмма взаимодействия клиентов.