ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ПРОФЕССИОНАЛЬНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ИРКУТСКОЙ ОБЛАСТИ
«АНГАРСКИЙ ТЕХНИКУМ СТРОИТЕЛЬНЫХ ТЕХНОЛОГИЙ»

Курсовая работа

Тема: Разработка браузерной игры "Морской бой"

Разработал _____Каталеев В.А.

Руководитель _____Денисюк А.В.

Ангарск, 2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

С каждым годом темпы развития игровой индустрии становятся все выше и выше. Из-за этого старые игры, по типу «Морского боя» становится менее популярнее с каждым годом, а ведь она могла получить своё развитие и стать популярной игрой.

Данная работа посвящена разработке Web-игры «Морской бой», в состоящей из клиентской части, веб-сервера, минимальной базой данных.
Клиентское часть будет включать в себя следующие функции:

- Форма регистрации пользователя.
- Форма авторизации пользователя.
- Просмотр своих результатов игры (количество побед и поражений)
- Рабочей связи с сервером.

Серверная часть будет включать в себя следующие функции:

- Принимать запросы пользователь.
- Обрабатывать данные.
- Выдавать результат.

База данных состоит из одной таблицы со столбцами:

- Имя пользователя (псевдоним, ник)
- Логина и пароль
- Количество побед и поражений.

Актуальность темы в том, что Flash-игры пользуются популярностью среди детей или студентов, которые не хотят слушать лекции, от чего они обращаются в интернет, в частности Web-играм.

Цель работы является разработать Веб-игру «Морской бой» как локально, как и по сети.

Для реализации этой цели были поставлены следующие задачи:

1. Изучить стандартные правила игры «Морского боя».
2. По этим правилам создать рабочий схему игры.
3. Выбрать подходящие инструменты по создание Веб-сервера, Веб-приложения.
4. Выбрать подходящие инструменты для соединения с пользователем и дальнейшие с ним взаимодействие(комнаты).
5. Выбрать подходящие инструменты для хранения данных пользователь как клиентской части, так же и на серверной части.

I. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

1.1 История появления морского боя.

Предположительно, история этой игры началась в начале прошлого века с появлением больших кораблей, броненосцев и линкоров, ведущих в тот период активные боевые действия. У некоторых людей появилось желание воспроизвести все эти боевые баталии на бумаге в клетку, тем более что она в то время стала доступной.

Согласно другой версии, «Морской бой» был изобретен в 1870-х годах бурлаком Петром Кондратьевым, решившим таким образом отвлечь себя и товарищей от изнуряющего, тяжкого труда.

Судя по всему, игра не только понравилась, но получила широкое распространение среди других слоев населения и успешно дошла до наших дней. В 1930-х фирма Starex в США выпускает специально для игры в морской бой разграфленные блокноты.

В 1960-е компания Милтона Брэдли выпустила новую версию игры с боевыми корабликами и фишками на пластмассовой доске. В 1982 г. эта игра появилась в виде пазлов. Вскоре появилась и электронная версия игры. Сегодня игра доступна для различных игровых платформ. И все же, самой лучшей, классической версией остается игра в «Морской бой» на бумаге.

1.2 Правила игры «Морской бой».

В игре участвуют два игрока. Цель игрока — уничтожить корабли соперника быстрее, чем он потопит ваши. Каждому игроку понадобится листок бумаги, желательно в клеточку, ручка, можно пользоваться карандашом. Игрок рисует на листке два квадрата, размер каждого из них 10×10, т.е по 10 клеток по вертикали и горизонтали.

Вертикаль нумеруем цифрами сверху вниз. По горизонтали под каждой клеткой записываем буквы алфавита от «а» до «к» («ё» и «й» пропускаем, об этом соперники договариваются заранее). Можно также

написать по горизонтали «снегурочка» или «республика» — в этих словах по 10 неповторяющихся букв. В одном квадрате игрок, скрытно от противника, расставляет свои корабли. Другое поле — пустое, здесь игрок будет отмечать подбитые корабли противника.

Игроки рассаживаются так, чтобы не видеть поля друг друга. Ну, или не показывать — шпионаж неспортивен, но каждый капитан сам должен заботиться о своих военных тайнах.

1.2.1 Правила расстановки кораблей.

Как правильно расположить флот:

- Корабли могут быть в любом месте, даже в углу и на пограничных клетках поля.
- Нарушать личное пространство судов нельзя — они не должны соприкасаться друг с другом ни носами, ни бортами, ни кормами. Даже углами их ставить запрещено. Минимальное расстояние между кораблями — одна клетка.
- Все палубники должны располагаться строго вертикально или горизонтально. Их нельзя ставить диагонально или сгибать в буквы «Г» или подобные.
- Играют только указанные десять судов. Никаких подмог не предусмотрено.

Впрочем, это правила классического морского боя. Игроки могут договориться об исключениях или добавить свои условия. Например, свободно расположить корабли или разрешить касаться углами. Это освежит сражение, если после нескольких боев оно начинает наскучивать.

Бывалые игроки рекомендуют использовать для расстановки судов разные ручки — нарисовав флот, следует обменяться пишущими

средствами. Это помешает под конец игры дорисовать корабль в свободное пространство.

1.2.2 Ход игры.

Так как ходят по очереди, капитаны перед началом дуэли определяют, кто будет делать первый выстрел. Чаще всего бросают монетку. После этого победивший капитан стреляет. При этом право первого хода практически не влияет на шанс победы в игре.

После жребия первый капитан стреляет — называет координату цели. Например, «А1» или «Д5». Оппонент сверяет полученную координату со своим расположением и дает ответ:

- если на указанной точке не было кораблей, то сообщает «мимо» или «промазал», чтобы поехиднее звучало;
- если стоял многопалубный корабль, то — «ранил» либо «подбил»;
- если там был однопалубный, то «убил». Также этот ответ дается, когда уже подбитый корабль добивают.

Количество палуб (клеток) — фактически уровень жизни корабля, то есть сколько выстрелов он может выдержать, прежде чем с честью пойдет ко дну. Катер тонет после первого же попадания, эсминец после двух, крейсер после трех. Линкор выдерживает три выстрела, а после четвертого тонет.

Получив ответ, стреляющий делает отметку у себя на поле. Чаще ставят крестик при удачном выстреле и точку, если снаряд ушел мимо. Если игрок промахнулся, право хода передается сопернику. Если ранил или потопил, то стреляющий делает еще один залп.
Игра заканчивается, когда у одного из игроков потонут все корабли. Он

считается проигравшим. По окончании боя капитаны обмениваются своими листами, чтобы свериться с выстрелами. Если есть нарушения со стороны победителя, то он с позором объявляется проигравшим.

Кроме того, каждый из игроков в любой момент может сдаться, если считает, что шансов выиграть нет.

Примеры нарушений:

- на поле расположены не все суда или есть лишние;
- корабли соприкасаются друг с другом;
- многопалубники не прямые, а в форме буквы «Г»;
- игрок подглядывал расстановку кораблей противника.

1.3 Серверная часть.

Для создания Веб-приложения и Веб-сервера будет использоваться Node.js, так как мы в техникуме изучали JavaScript (js).

JavaScript (js) — это язык программирования, который в первую очередь применяют в вебе. С его помощью сайты делают интерактивными: добавляют всплывающие окна, анимацию, кнопки лайков и формы для отправки информации. Его ещё называют главным языком фронтенда — «лицевой» стороны сайта, с которой взаимодействуют пользователи.

Node.js (Node) — это платформа с открытым исходным кодом для работы с языком **JavaScript**, построенная на движке Chrome V8. Она позволяет писать серверный код для веб-приложений и динамических веб-страниц, а также программ командной строки. В основе платформы — событийно-управляемая модель с неблокирующими операциями ввода-вывода, что делает ее эффективной и легкой.

Ещё чем меня привлёк Node.js:

- **Высокая скорость.** JavaScript-код, который выполняется в среде Node.js, может быть в несколько раз быстрее, чем написанный на языках вроде Ruby или Python. В Node.js используется модель асинхронного программирования. Модель позволяет продолжить обработку других задач, не дожидаясь завершения передачи данных. Когда требуется выполнить операцию ввода-вывода вроде доступа к файловой системе или базе данных, Node.js не блокирует главный поток ожиданием результатов. Платформа инициирует ее выполнение и продолжает выполнять другие задачи, пока результаты предыдущей операции не будут получены.
- **Универсальность и гибкость.** В Node.js выполняется код, который написан на JavaScript. Это означает, что frontend-разработчики, которые уже используют JavaScript в браузере, могут писать и клиентский, и серверный код на привычном языке программирования, не изучая инструмент с нуля. В Node.js можно быстро переходить на новые стандарты ECMAScript по мере их реализации. Новые возможности языка становятся доступны сразу после установки поддерживающей их версии Node.js.
- **Большое количество модулей и библиотек.** Экосистема Node.js стремительно развивается благодаря менеджеру пакетов NPM. Он содержит более 500 000 модулей и библиотек open-source, которые находятся в свободном доступе. Также постоянно появляются новые.

Для разработки веб-приложения будет использован EJS - это простой и популярный шаблонизатор для Node.js, который позволяет встраивать JavaScript код прямо в HTML. Он идеально подходит для быстрого и эффективного создания динамических страниц.

Так же мне понадобится Express**. Express** в Node.js нужен для упрощения разработки веб-приложений на базе этой среды. Он предоставляет набор функций для создания веб- и мобильных приложений.

Некоторые возможности Express:

- Настройка посредников для ответа на запросы HTTP.
- Определение таблицы маршрутизации для выполнения различных действий на основе метода HTTP и URL-адреса.
- Динамическое создание HTML-страниц на основе передачи аргументов шаблонам.
- Интеграция решений, таких как аутентификация, обработка ошибок и работа с базами данных.

Для хранения данных пользователя на сервере будет использоваться Express-session. **Express-session** в Node.js нужен для управления сессиями — хранения данных пользователя на сервере, которые не теряются при переходе на другую страницу.

Некоторые задачи, для которых используется express-session:

- Безопасное хранение чувствительных данных. Например, идентификатора пользователя.
- Эффективное и безопасное хранение глобальных переменных. Это полезно, например, при создании приложений, которые нуждаются во временном хранении данных объёмом более 4 КБ.
- Замена cookies на платформах, которые их не поддерживают.
- Реализация таких функций, как системы входа пользователей, корзины покупок и другие, где серверу нужно помнить информацию, специфичную для пользователя.

Почему выбрал Express-session, а не Cookie файлы?

> Express-session хранит данные сеанса на сервере, при этом в файле cookie на стороне клиента сохраняется только идентификатор сеанса, а не сами

данные. Express-session может использовать настраиваемые бэкенды, начиная от памяти и заканчивая любым количеством баз данных, что обеспечивает практически неограниченное хранение сеанса.

> Cookies хранят данные сеанса на стороне клиента в файле cookie. Объём памяти сеанса ограничен размером файла cookie, обычно чуть менее 4 тыс. байт. Cookies полезны в приложениях, где серверная часть не использует базу данных.

Ещё несколько их отличий:

Производительность: session работает медленнее из-за взаимодействия с сервером, cookies — быстрее, так как данные хранятся локально.

Безопасность: session более безопасен, так как данные хранятся на стороне сервера, cookies — менее безопасен, так как данные хранятся на стороне клиента.

Использование: session используется для хранения данных, специфичных для пользователя, cookies — для хранения данных предпочтений пользователя.

Для работы с базой данных буду использовать MySQL2. **MySQL2** — это быстрый, эффективный и богатый функциями клиент MySQL для Node.js. Он обеспечивает высокопроизводительный, неблокирующий и масштабируемый интерфейс для работы с базами данных MySQL в приложениях Node.js.

Некоторые возможности MySQL2:

- Улучшенная производительность. Благодаря асинхронным операциям приложение Node.js может продолжать выполнять другие задачи, ожидая завершения запросов к базе данных, что приводит к более быстрому ответу.
- Повышенная безопасность. MySQL2 предоставляет такие функции, как подготовленные заявления и параметризованные запросы, чтобы

помочь предотвратить атаки SQL-инъекций, обеспечивая безопасность данных.

- Поддержка Promises. MySQL2 поддерживает Promises, что позволяет работать с более элегантным и читаемым асинхронным кодом.
- Поддержка потоков. MySQL2 позволяет передавать большие наборы данных напрямую из базы данных, что снижает потребление памяти и улучшает общую производительность при работе с большими объёмами данных.
- Управление несколькими подключениями. С помощью MySQL2 можно легко управлять несколькими подключениями к базам данных, что делает его подходящим для приложений, требующих одновременного доступа к базам данных.

Для обеспечивающая двустороннюю связь между клиентами и серверами в режиме реального времени буду использовать Soket.io. **Socket.io** в Node.js — библиотека JavaScript, обеспечивающая двустороннюю связь между клиентами и серверами в режиме реального времени.

Она построена на основе протокола WebSocket. С помощью Socket.io можно создавать интерактивные веб-приложения, где сервер может отправлять данные клиенту и наоборот, без необходимости постоянно запрашивать сервер для обновлений страницы.

Некоторые особенности Socket.io:

- Коммуникация посредством передачи событий. Клиент и сервер могут отправлять и принимать события, которые могут быть обработаны соответствующим образом для выполнения определённых действий.
- Создание пространств имён и комнат. Это позволяет организовать структуру взаимодействия между клиентами и сервером.

- Надежность соединения. Соединение устанавливается независимо от присутствия прокси, балансировщиков нагрузки, «файервола» и антивируса.

От других аналогов, например, Web-Soket, Socket.io отличается тем, что:

- Позволяет отправлять сообщения всем подключённым клиентам. При использовании веб-сокетов для реализации подобной задачи потребуется список подключённых клиентов и отправка сообщений по одному.
- Поддерживает проксирование и балансировщики нагрузки. В веб-сокетах сложно использовать эти технологии, а Socket.IO поддерживает их «из коробки».
- Решает проблему с маршрутизацией соединений. Socket.IO решает её с помощью cookie или маршрутизации соединений на основе исходных адресов, в то время как у веб-сокетов не существует подобного механизма.

Для шифрования пароля будет использоваться BCrypt. **BCrypt** в Node.js — это библиотека для хэширования и шифрования паролей. Она позволяет создавать хэш из строки пароля и сохранять его в базе данных.

Когда пользователь отправляет пароль, он хэшируется, а приложение Node.js сохраняет хэш. Позже, когда пользователь хочет аутентифицировать свою учётную запись, нужно сравнить ввод пароля с сохранённым хэшем.

1.4 Клиентская часть.

Для страниц я будет выступать HTML с EJS представлением. Для работоспособности сайта будет использоваться JavaScript. Для стилизации и оформления страницы будет использоваться CSS.

HTML (HyperText Markup Language) — это язык гипертекстовой разметки, который используется для создания и структурирования веб-страниц.

Он помогает определять, как содержимое страницы должно отображаться в браузерах. Иными словами, HTML — это своеобразный каркас, на котором строится страница, включая текст, изображения, ссылки и другие элементы.

Основная цель HTML — структурировать и оформлять контент на сайте. HTML создаёт иерархическую структуру веб-страницы, используя заголовки, абзацы, списки и таблицы. Такая структура помогает пользователю легче ориентироваться на сайте.

С помощью HTML можно:

- делать текстовую разметку — форматировать текст, выделять фрагменты, создавать списки, добавлять сноски;
- встраивать медиа — размещать на сайте изображения, аудио, видео, карты;
- создавать ссылки и навигацию — гиперссылки и списки меню помогают быстрее найти информацию и сориентироваться на странице;
- создавать таблицы — нередко информацию удобно представить в табличном виде;
- создавать формы — формы нужны для регистрации посетителей сайта по телефону и электронной почте, оформления заказов, опросов и сбора обратной связи — отзывов, комментариев, предложений.

CSS — это код, который позволяет визуально оформить страницу: раскрасить подзаголовки, поменять фон или отформатировать изображение. CSS — важная часть фронтенд-разработки и один из ключевых навыков для веб-разработчика.

Некоторые задачи, которые решает CSS:

- Оформление текста. С помощью CSS можно задать цвет, размер шрифта, выравнивание, высоту строки и т. д.

- Оформление HTML-элементов. Можно задать размеры, отступы, цвет фона, тени, скругления.
- Построение сеток для расположения контента. Система сеток позволяет контенту быть уложенным как по вертикали, так и по горизонтали.
- Создание анимации. С CSS можно создавать красивые анимации без использования JavaScript.
- Создание адаптивного дизайна. Можно менять отображение в зависимости от размеров устройства, на котором просматривают сайт.

Всё это понадобится для того, чтобы создать следующие странички:

- Главная страница – что видят пользователи при первом заходе, в ней будут также находится кнопки для выбора игры, а также кнопки «Регистрации» и «Авторизоваться».
- Страница, где будет форма регистрации нового пользователя.
- Страница, где будет форма авторизации созданного или уже существующего пользователя.
- Страница, где будет происходить игра с ИИ(ботом).
- Страница, где будет происходить игра с другом на одном компьютере.
- Страница, где будет происходить игра по сети.

Для самой игры нужно будет сделать:

- Два поля 10 на 10 клеток. (Общин)
- Добавить кнопки, которые будут выполнять ту или иную функцию. (Общие)
- Функции для расположения кораблей по методу расстановки «Рандом» или расстановки «Вручную». (Клиент/Сервер)
- Функции для обработки выстрела игрока. (Клиент/Сервер)
- Функции для проверки уничтожение кораблей. (Клиент/Сервер)

- Создать рабочий ИИ, который может конкурировать с игроком. (Клиент)
- Функции для очищения поля после игры.
- И другие функции.

1.5 Графическая составляющая.

А также для моей работы будет использоваться программы Krita и Figma для создания графики.

> **Krita** — это бесплатный редактор растровой графики с открытым исходным кодом, предназначенный в первую очередь для цифрового искусства и 2D-анимации.

Программа предоставляет широкий набор инструментов для создания цифрового искусства, иллюстраций, концепт-артов, комиксов и многого другого. Krita поддерживает как растровую, так и векторную графику, предлагает настраиваемый интерфейс и включает продвинутые функции, такие как движки кистей, управление слоями и инструменты управления цветом.

Будет использоваться для создания астровой графики.

> Figma — это инструмент для дизайна интерфейсов, который позволяет создавать, редактировать и совместно работать над проектами в режиме реального времени.

В нём можно разрабатывать различные интерфейсы и прототипы цифровых продуктов, создавать векторные объекты, двух- и трёхмерные иллюстрации.

Будет использоваться для создания векторной графики.

II. РАЗРАБОТКА ПРОГРАММНОГО ПРОДУКТА

Прежде чем приступать к разработке, стоит отметить, что необходимо иметь установленную программную платформу Node.js, и к ней нужные библиотеки (которые были упомянуты выше), чтобы можно было работать со сервером, а также какую-либо среду разработки, например, Visual Studio Code.

2.1 Разработка рабочей схемы игры.

В первую очередь для работы с Java Script нужно подготовить поля и окна, чтобы можно было работать.

HTML код полей:

```
<div id="box">
    <div id="block1">
        <div id="wrapperBlock" class="Field"></div><!--Поле1-->
    </div>
    <div id="block2">
        <div id="wrapperBlock2" class="Field"></div><!--Поле2-->
    </div>

</div>
```

HTML код выборки:

```
<div id="Block_button" class="block_btn">
    <center>
        <div><h2 id="User">Игрок</h2></div>
    </center>
    <div class="block_box">
        <div id="Inf">ПКМ - сменить положение корабля.</div>
        <center>
            <div id="Random" class="Clk">Рандом</div>
            <div id="Position" class="Clk">Вручную</div>
            <div id="Play" class="Clk">Играть</div>
        </center>
    </div>
</div>
```

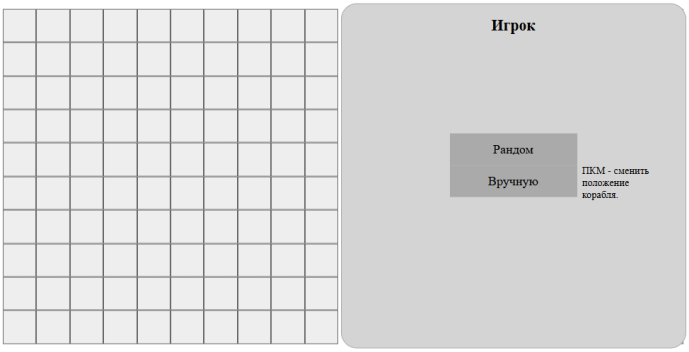
HTML код планки победы/поражение:

```
<div id="Block_Win_Lozz" class="block_wind2"><!--Победа/поржание-->
    <div id="box_bt">
        <center>
            <div><h2 id="Win_Lozz"></h2></div>
        </center>
        <div id="box">
            <center>
                <a href="/"><div id="Log_out" class="Clk">Выйти</div></a>
                <div id="Again" class="Clk">Заново</div>
            </center>
        </div>
    </div>
</div>
```

К ним также стили (CSS):

```
#wrapperBlock{
    padding: 0;
    display: flex;
    flex-flow: row wrap;
    /*width: calc(100vmin - 10px); //Соотношение 1-поле,2-промежуток
    height: calc(100vmin - 10px);*/
    width: calc(50vmin - 10px); /**/
    /*height: calc(10ыvmin - 10px);*/
}

#wrapperBlock2 {
    padding: 0;
    display: flex;
    flex-flow: row wrap;
    /*width: calc(100vmin - 10px); //Соотношение 1-поле,2-промежуток
    height: calc(100vmin - 10px);*/
    width: calc(50vmin - 10px); /**/
    /*height: calc(10ыvmin - 10px);*/
}


.square{
    margin: 0;
    padding: 0;
    width: calc(5vmin - 1px); /* Размер клетки*/
    height: calc(5vmin - 1px);
    border: 1px solid #686868;
    background-color: #eee;
    box-sizing: border-box;
}
```

Таким способом подготовили поля для игры.



Этого уже достаточно, чтобы приступить к созданию функций игры. И первая загвоздкой, которой столкнуться можно, это то, что полей нет. Так как нужно их сделать, и на это есть два способа:

1. Создать через JS.

```
for(i=0;i<10;i++){//С ботом.

    // Поле 1
    for(j=0;j<10;j++){
        var square = document.createElement("div");
        square.classList.add("square");
        square.id=i+";"+j;
        square.innerText = " ";
        wrapperBlock.appendChild(square)
    }
    // Поле 2
    for(j=0;j<10;j++){
        var square2 = document.createElement("div");
        square2.classList.add("square2");
        square2.addEventListener("mouseover", Hover_maus_on)
        square2.addEventListener("mouseout", Hover_maus_off)
        square2.id=i+";;"+j;
        square2.addEventListener('click',handleClick);
        square2.innerText = " ";
        wrapperBlock2.appendChild(square2)
    }
}
```

2. Создать с помощью EJS представлением.

```
<div id="box">
    <div id="block1">
        <div id="wrapperBlock" class="Field">
            <%for(let i=0;i<10;i++){%>
                <% for(let j=0;j<10;j++){%>
                    <div id="<%=i%>:<%=j%>:0" class="square"></div>
                <%}%>
            <%}%>
        </div><!--Поле1-->
    </div>
    <div id="block2">
        <div id="wrapperBlock2" class="Field">
            <%for(let i=0;i<10;i++){%>
                <% for(let j=0;j<10;j++){%>
                    <div id="<%=i%>:<%=j%>:1" class="square2"></div>
                <%}%>
            <%}%>
        </div><!--Поле2-->
    </div>
</div>
```

Теперь у нас есть 100% поля с клетками, и окно. Теперь нам нужны данные, от которые нужно будет отталкиваться. А именно нужны данные о кораблях (data_ship), пустые поля (field_xy), а также состояние полей (field_canShot).

```
let field_xy = [//Поле игрока
    [0,0,0,0,0,0,0,0,0,0],
    [0,0,0,0,0,0,0,0,0,0],
    [0,0,0,0,0,0,0,0,0,0],
    [0,0,0,0,0,0,0,0,0,0],
    [0,0,0,0,0,0,0,0,0,0],
    [0,0,0,0,0,0,0,0,0,0],
    [0,0,0,0,0,0,0,0,0,0],
    [0,0,0,0,0,0,0,0,0,0],
    [0,0,0,0,0,0,0,0,0,0],
    [0,0,0,0,0,0,0,0,0,0]
];

let field_CanShot= [//Поле игрока для проверки выстрела.
    [true,true,true,true,true,true,true,true,true,true,],
    [true,true,true,true,true,true,true,true,true,true,],
    [true,true,true,true,true,true,true,true,true,true,],
    [true,true,true,true,true,true,true,true,true,true,],
    [true,true,true,true,true,true,true,true,true,true,],
    [true,true,true,true,true,true,true,true,true,true,],
    [true,true,true,true,true,true,true,true,true,true,],
    [true,true,true,true,true,true,true,true,true,true,],
    [true,true,true,true,true,true,true,true,true,true,],
    [true,true,true,true,true,true,true,true,true,true,]
]

let data_ship = [//Корабли игрока
    ["Ch",4],//Четырёхпалубники
    ["Tr1",3],["Tr2",3],//Трёхпалубники
    ["db1",2],["db2",2],["db3",2], //Двухпалубники
    ["on1",1],["on2",1],["on3",1],["on4",1] //Однопалубники
]
```

Как уже заметно все они заключены в массив.

> data_ship – данные о кораблях.

data_ship[0-9][0-1] => [0-9] – тип корабля. [0-1] – «0» название корабля, «1» количество палуб.

> field_xy – поле куда будут помещены корабли. X – является рябом (с верху в низ), У – является самой клеткой ряда (С лева на право). Как и field_canShot.

> field_canShot – куда можно стрелять.

Приступим к простому, к расстановке кораблей методом «Рандом».

Для начала дадим обработчик к кнопке «Рандом», а при клике на неё, будет вызываться функция Random.

```
//Вызвать функцию "рандом", чтобы рандомно раставить корабли.
let Rand = document.getElementById("Random")
Rand.addEventListener('click', Random)
```

В самой функции:

Первым очередь будет функция, которая будет очищать поля, если они были ли заполнены, по сути, если элемент массива не равняется ноль, то она приравняет к нулю.

```
if(check_Fl()==true){}//Проверяем, был ли поле уже запонино.
```

После чего запускается цикл (for) с 10 повторением, который начинается с 0.

Число повторений (0-9), они будут определять, какой первым будет располагаться корабль (От большого корабля к маленьким). От кораблей мы берём их название и кол-во палуб, и создаём временный массив.

```
Name_Ship = data_ship[r][0]//Название корабля.
dask = data_ship[r][1]//Кол-во палуб.
let vr_coord =[]//Временый массив, мнимых координат корабля на поле.
Wh_tf=true
```

После чего запускаем ещё один цикл (while), но уже он повторятся до тех пор, пока он поставит корабль.

Теперь мы определим по какой оси будет стоять корабль: 0 – горизонтальное, 1 – вертикальное положение.

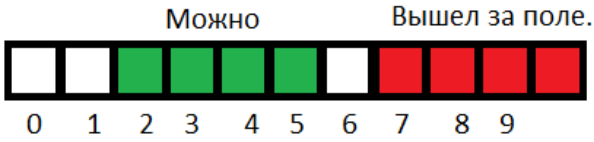
```
var xy = Math.round(Math.random()) //Рандомное число от 0 до 1
```

В зависимости от расположения, будет зависеть, какие координаты будут браться в дальнейшем. Рассмотрим горизонтальное расположение:

```
min = 0
max = 9
x = Math.floor(Math.random() * (max - min + 1)) + min
y = Math.floor(Math.random() * (max - dask - min + 2)) + min;//Диапозон с учётом 
absl = max - dask+1 //Максимум по координатам, котором можно поставить корабль,
                    //чтобы не зайти за поле.
alf = dask//Временный массив координат кораблей.

/*-----------------------------------------------------------------------------

//Из первой координаты создаём уже сам корабль.
for (i=0;i<alf;i++){

    vr_coord[i] = [x,y]//Координаты временного массива.
    y= y+1
}
```

Берём сначала минимальные координаты и максимальные, после чего подставляем в формулу для рандомной генерации позиции корабля. Как вы заметили в формуле x нет dask, а y - есть.

Нужен dask для того, чтобы огранить возможность выбрать поля.



Записываем во временный массив.

```
if(checkShipBoard(vr_coord)==true){

    for(r1=0;r1<vr_coord.length;r1++){
        x5=vr_coord[r1][0]
        y5=vr_coord[r1][1]
        document.getElementById(x5+";"+y5).classList.add("Ship_shadow")
        field_xy[x5][y5] = Name_Ship
    }
    for(c=0;c<vr_coord.length;c++){ //Обводка вокруг корабля.
        let vr_Check_Ship =[]
        //Начальные координаты палубы корабля.
        x1=vr_coord[c][0]
        y1=vr_coord[c][1]
        y1=y1+1 //Сдвиг вправо 1 раз.
        vr_Check_Ship.push([x1,y1])
        x1=x1+1//Сдвиг вниз 1 раз.
        vr_Check_Ship.push([x1,y1])
        for(t=0;t<2;t++){//Сдвиг влево 2 раза.
            y1=y1-1
            vr_Check_Ship.push([x1,y1])
        }
        for(tg=0;tg<2;tg++){//Сдвиг вверх 2 раза
            x1=x1-1
            vr_Check_Ship.push([x1,y1])
        }
        for(th=0;th<2;th++){//Сдвиг право 2 раза
            y1=y1+1
            vr_Check_Ship.push([x1,y1])
        }

        //Проверка координат, чтобы пометить поле возле корабля.
        for(chk=0;chk<vr_Check_Ship.length;chk++){
            x2= vr_Check_Ship[chk][0]
            y2= vr_Check_Ship[chk][1]
            if (x2>-1 && x2<10){ //9>=x2>=0
                if (y2>-1 && y2<10){
                    if(field_xy[x2][y2]==0){
                        field_xy[x2][y2] = 1
                    }
                }
```

После чего мы в первую очередь проверим, можно ли поставить на этих координатах корабль.

По сути она берёт поле и проверяет, занято ли место по этим координатам. Если нет, то цикл продолжит работать до того момента, пока не будет возможность поставить корабль. А если можно, мы записываем в массив название корабля в каждую ячейку по сгенерированными координатам. После чего берём координаты вокруг клетки, и если эти клетки не заняты (то есть равно «0»), то помечаем, что уже заняты они (1). Нужно для того, чтобы следующий корабль не сгенерировался в упор к кораблю (так как по стандартным правилам нельзя ставить корабли рядом, только через одну клетку).

Вы уже заметили по скриншоту:

```
document.getElementById(x5+";"+y5).classList.add("Ship_shadow")
```

При генерации корабля, нам нужно отобразить на поле пользователя. Этой строчкой кода мы берём элемент с этими ID и даём класс. При создании поля мы также к каждой клетке давали свой ID (используя два цикла), нужно для того, чтобы каждая клетка была уникальная, а в будущем для выстрела разобрать координаты для определения в системе, можно по не стрелять и есть ли по этим координатам корабль.

Когда сгенерируются все корабли, появится кнопка «Играть».

До этого момента она была скрыта, чтобы игрок не мог начать игру до расстановки своих кораблей.

```
Play.style.display="block"
```
Отображаем кнопку у пользователя.

При нажатии на неё, окно с выбором расположением кораблей, скроется, чтобы увидеть второе поле. Таким же способом, реализуется рандом для второго игрока, но уже место скрытия, оно передвинется, но уже влево, чтобы можно было расстановить корабли, но уже для второго игрока.

Для игры с ботом, рандом работает также, как и у игрока, но уже не отображается на клиенте, только в системе.

В приложение А будет размещён часть код для расстановки кораблей «Вручную».

Теперь рассмотрим, как будет происходить выстрел. Для начала сделаем так, чтобы при наведении на клетку соперника. Сначала даём следующие обработчики:

```
square2.addEventListener("mouseover", Hover_maus_on)
square2.addEventListener("mouseout", Hover_maus_off)
```

Первый работает при наведении, другой, когда отвели от этой клетки. Это в принципе можно сделать в CSS (hover), но столкнёмся с проблемой, что после очистки поля, больше не будет наводиться на поле, клетка не изменит цвет.

Функции для ник. (add – дать класс, remove – удалить класс с клетки)

```
function Hover_maus_on(event){
    document.getElementById(event.srcElement.id).classList.add("hover_on2")
}
function Hover_maus_off(event){
    document.getElementById(event.srcElement.id).classList.remove("hover_on2")
}
```

Во время генерации полей, мы также давали каждой клетке обработчик, чтобы можно было обрабатывать событие «click».

По нажатии, выполняется следующая функции:

```
if(Ship_Sum > dead_ship){
    if(move_user==true){
        coords=event.srcElement.id.split(";;");
        x_coord=coords[0];
        y_coord=coords[1];
        // Берём id div-ва
        let att= document.getElementById(event.srcElement.id)

        if(field_CanShot2[x_coord][y_coord]==true){//Проверка на выстрел
            field_CanShot2[x_coord][y_coord] = false

            //Проверка на попадания по карабл, и если он попал, то данные изменяются и перен

            for(g = 0;g<10;g++){
                if(field_xy2[x_coord][y_coord]==data_ship2[g][0]){//Если попал
                    hp = data_ship2[g][1]
                    hp = hp-1
                    data_ship2[g][1] = hp
                    data_ship2[g].push(att)
                    att.classList.add("hit") //Цвет, что ты попал.

                }else if(field_xy2[x_coord][y_coord]==0 || field_xy2[x_coord][y_coord]==1){
                    move_label.innerText= "<"
                    att.classList.add("miss") //Цвет, что ты промазал.
                    move_user=false
                    move_pc = true
                    move_pc_cycle = true
                    move_bot()
                    break
                }
                //Мертвяк.
                if (data_ship2[g][1]==0){
                    for(k=2;k<data_ship2[g].length;k++){
                        stl= data_ship2[g][k]
                        stl.classList.add("dead")
                    }
                    dead_ship = dead_ship + 1
                    dead_ship_check2()
                    data_ship2[g][1] = "dead"
                    Result()
                }
            }
```

Сначала проверяем, остались ли корабли в живых у игрока/бота.

После чего проверяем, ход ли игрока, если нет, то функция выполнится, но не обработает выстрел игрока. Если же ход игрока, то с начал берём ID клетки и «Дробим» на координаты. Отдельно помещаем ID в переменную.

После чего проверяем, если ли смысл стрелять по этим координатам, то есть, если по этим координатам уже стреляли, или уже потоплен корабль, то не смысла стрелять по этим клеткам.

Если же можно, то помечаем, что уже по этой координате уже нет смысла стрелять.

После чего обрабатываем его, если попал, то на этой клетке появится крестик, что попал. Если мимо, то точка и передаётся ход либо второму игроку, либо ИИ.

После чего проверяем, если корабль, по которому попали убит, то сработает функции для пометки того, что этот корабль убит на клиенте и в системе. После чего вокруг него появится обводка из точек, что во круг нет смысла стрелять.

И после каждого убитого корабля, будет проверка, есть ли уцелевшие корабли, если да, то ничего не сработает, а если уже убиты, то запустится таймер, и через несколько секунд появится плашка, что вы либо победи, либо проиграли.

Мы рассмотрели выстрел игрока, который поможет для разработки уже игры на двоих и для игры по сети.

Теперь рассмотрим, выстрел ИИ.

Я выбрал бота, который будет стрелять в рандомные координаты. Выбор на этом, так как бот будет не предсказуемым, ведь он либо может уничтожить несколько кораблей подряд, либо долго не попадать по ним. По этой методики он будет по силе приближён к игроку, а также он будет давать шанс обыграть себя. По определению, это золотая середина, между сложным ботом и лёгким.

Для начала я выбрал методику, что ИИ будет сначала рандомно стрелять по сгенерированным координатам, после чего если попал, будет обстрел вокруг координаты, а именно крестом (+).

И когда он найдёт, корабль, будет обстрел по оси. Если он попал куда в центр, и по выстрелам промажет, то следующим, он будет стрелять уже на противоположную сторону. Также он перед выстрелом проверяет, если смысл стрелять по этой координате. Когда он уничтожил корабль, то он заново будет стрелять в рандомные координаты. Когда остаются только однопалубные корабли, он сохраняет координаты оставшихся свободных

полей, и также будет обстреливать их по координатам, но не генерируемых случайным образом, а выбирая их из заранее созданного списка, который сохранен в массиве. При каждом ходе координаты будут удаляться из списка. Игра закончится, если бот уничтожит все корабли пользователя.

Код ИИ, общий(рис. 1):

```
if(Ship_Sum > dead_ship2){
    /*Проверка на наличие кораблей на поле, если остались только однопалубники,
    то чтобы программа не продолжала рандомно обстреливать, а на уже свободные координаты начать обстреливать.
    */
    if(dead_ship2==6||6<dead_ship2 && trig==true){
        check=0
        for(i=0;i<6;i++){
            if(data_ship[i][1]=="dead"){
                check++
            }
        }
        if(check==6){
            hit=false
            trig=false
            Shelling_of_field= true
            for(i=0;i<10;i++){
                for(j=0;j<10;j++){
                    if(field_CanShot[i][j]==true){
                        shell_field.push([i,j])
                    }
                }
            }
        }
    }
    if(Shelling_of_field == true && move_pc==true){
        setTimeout(Sof,1200)
    }
    if(hit==0 && move_pc==true){//Новый выстрел.
        setTimeout(Naw_hit,1200)
    }

    if(hit==1 && move_pc==true){//Обстрел первого попадания крестом"+"
        setTimeout(Hit_crest, 1200)
    }

    if(hit>1 && move_pc==true){//Добить корабль.
        setTimeout(Hit_Finishing, 1200)
    }
    //Result() <-------------------------------------------------
}
if(Ship_Sum == dead_ship2){
    move_pc= false
    move_user = false
    Result()
}
```

Рисунок 1 – Код ИИ.

В целом, всё просто, обстрел клетки будет происходить сначала справа, и если мимо, то снизу, влево, а после вверх.

Почти всё тоже самое как с рандомным выстрелом.

Это основные функции, которая понадобятся для других режимов.

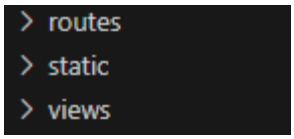
2.2 Серверная часть.

Теперь же разберёмся со сервером. Для начала создадим сам сервер. Нужно создать отдельный файлом JS, чтобы поместить сам сервер.

```
//Импорт маршрутов.
var indexRouter = require('./routes/index');
```

В приложение Г размещён код структуры сервера.

В папке, где хранится сервер нужно будет создать ещё ряд папок.



В них будет храниться Маршруты, JS, картинки и CSS для клиента, а также EJS представления.

Теперь можно работать со сервером. Теперь же на нужно создать следующие файлы views:

- Главный сайт, на который пользователя зайдёт первые.
- Страницы для Авторизации и Регистрации.
- Страницы с Соло и Дуо игрой.
- Страница, где будут происходить мультиплеер.

В общем 6 страниц, но уже не HTML формат файла, а EJS.

А чтобы начать с этим работать, нам нужно сначала подключить базу данных. А для этого мы создаём новый файл рядом с сервером, но чтобы работать с ней, нам понадобится таблицу в любом редакторе баз данных (Workbench, DBeaver), База данных будет называться battle_ship_user. После чего нужно сделать примерно такой скрипт:

```
CREATE TABLE `user` (
  `id` int NOT NULL AUTO_INCREMENT,
  `Nick` varchar(20) NOT NULL,
  `Login` varchar(20) NOT NULL,
  `Password` varchar(255) NOT NULL,
  `Win` int NOT NULL DEFAULT '0',
  `Loss` int NOT NULL DEFAULT '0',
  PRIMARY KEY (`id`)
```

id – ID пользователя

Nick – Имя пользователя.

Login – Логин

Password – Пароль

Win и Loss – Победы и поражения

С этой таблицы будем работать, теперь перейдём к самому подключению. Файл, который был создан рядом со сервером (Допустим: data_base.js)

```
const mysql = require('mysql2');

const connection_db = mysql.createConnection({
    //Моя
    host: 'localhost',
    user: 'root',
    password: 'qwer',
    database: 'battle_ship_user'
    //Для курсовой.
    //host: '192.168.88.188',
    //user: 'student11',
    //password: '04bti9',
    //database: 'battle_ship_user'
});

connection_db.connect((err)=>{
    if(err) throw err;
    console.log("Connect data base")
})
```

Подключаем модуль mysql2.

После чего прописываем коннект к базе данных.

host - хост, на котором запущен сервер mysql. По умолчанию имеет значение "localhost".

User – Имя пользователя

Password – пароль.

Database – Имя базы данных.

После чего проверяем подключение, если правильно, то выдаст в консоль, что подключён к базе данных.

Теперь пропишем функции (рис. 3):

1. Добавление нового пользователя (Регистрация).
2. Авторизация.
3. Добавлять единицы к победе.
4. Добавление единицы к поражению.
5. Берём человека по id.

После чего экспортируем функции.

```
function createUser(nick,login, hashedPassword, callback) {

    const query = 'INSERT INTO user (Nick, Login, Password) VALUES (?, ?, ?)';
    connection_db.query(query, [nick,login, hashedPassword], callback);
}

function findUserByUsername(login, callback) {
    const query = 'SELECT * FROM user WHERE Login = ?';
    connection_db.query(query, [login], callback);
}

function addwin(id,callback){
    const query = 'UPDATE user SET Win = Win+1 WHERE id = ?'
    connection_db.query(query, [id], callback);
}
function addloss(id,callback){
    const query = 'UPDATE user SET Loss = Loss+1 WHERE id = ?'
    connection_db.query(query, [id], callback);
}

function updatasession(id,callback){
    const query = 'SELECT * FROM user WHERE id = ?'
    connection_db.query(query, [id], callback)
}

module.exports={
    connection_db,
    createUser,
    findUserByUsername,
    addwin,
    addloss,
    updatasession
}
```

Рисунок 3 – функции запросов в базу данных.

Теперь можно работать с базой данных, и брать/вносить данные.

Теперь пропишем строку, чтобы можно было импортировать функции. Импортируем на сервер и в маршруты.

```
const sample = require('./modul/sample')
```

Теперь к регистрации пользователя. По заранее заготовленному макету, куда будут вносить данные.

```
<div>
    <form action="/Register" method="post" id="Forma">
        <div id="frm2">
            <div class="lb">Ник</div>
            <input type="text" name="nick" class="frm">
        </div>

        <div id="frm2">
            <div class="lb">Логин</div>
            <input type="text" name="login" class="frm" placeholder="От 4 синволов">
        </div>

        <div id="frm2">
            <div class="lb">Пароль</div>
            <input type="text" name="password" class="frm"
            placeholder="От 4 до 8 синволов">
        </div>

        <button type="submit" class="But_log">Войти</button>
    </form>
    <div id="info">Когда регистрация произошла верно, вы автоматически вернётесь на главную страницу.</div>
    <% if(typeof error !== 'undefined') {%>
        <div id="er"><%= error %></div>
    <% } %>
</div>
```

Вся обработка будет происходить в маршрутах. Также будут операции с Авторизации и Обновление данных.

В приложение Б размещён код, где происходит вся регистрация. Когда пользователь отправит данные, сначала, он пройдёт проверку (валидацию), после чего пойдёт проверка, есть ли такой логин вообще, и если есть, то сервер отправит пользователю, что такой логин уже занят, а если нет таких, следовательно, создастся новый пользователь, пароль, который придумал пользователь, будет закэширует его.

Также будет происходить Авторизация, но уже, когда пользователь введёт логин и пароль, и если он окажется верным, то данные будут сохранены в сессию.

Теперь у нас есть сессия, теперь можно брать данные пользователя. Чтобы с главного сайта убрать кнопки Регистрации и Авторизации, в представлении пропишем следующее:

```
<% if(typeof error !== 'undefined') {%>
    <div id="er"><%= error %></div>
<% } %>
<div class="block_profil">
    <% if(session.USER) {%>
        <div id="profil" class="block_log_reg">
            <center><div><h2>Профиль</h2></div></center>
            <div class="statys_prof">Ваш ник: <%- session.USER.Nick %></div>
            <div class="statys_prof">Победы: <%- session.USER.Win %></div>
            <div class="statys_prof">Поражений: <%-session.USER.Loss %></div>
            <div class="statys_prof">
                <form action="/go_out" method="post">
                    <button type="submit" class="But_out">Выйти</button>
                </form>
            </div>
        </div>
    <%}else{%>
        <div id="Reg_log">
            <center>
                <a href="/Register"><div class="Reg_log"><p>Зарегестрироваться</p></div></a>
                <a href="/Login"><div class="Reg_log"><p>Вход</p></div></a>
            </center>

        </div>
    <%}%>
</div>
```

Если сессия существует, то появится окно пользователя, если её нет – высветятся кнопки. Таким же способом, я сделал ограничение, если нет сессии, то игра по сети будет невозможна, и будет выходить плашка, что сначала зарегистрируйтесь.

Так как мы уже рассмотрели схему игры, теперь можно перейти игре по сети. Сделаю сразу пометку:

Принципы работы схемы игры будут похожими, но что-то категорично отличаться.

Переходи на страницу, где проходят игра по сети, в первую очередь, для при входе на страницу будет выполнятся следующая функция:

```
async function take_session() {
    const response = await fetch ('/take_data')
    const data = await response.json();// data_o_user // Da
    if (data.error) {
        console.log("Ошибка:", data.error);
        return;
    }else{
        data_o_user= data

        socket = io();
        socket.emit('Id_identified', data.Data_session.Id)
```

Нужна для того, чтобы взять данные сессии и их сохраняем на клиенте. После чего запускаем сокет.

Пометка:

Чтобы заработали сокеты, нужно будет в клиентской части прописать в head.

```
<script src="/socket.io/socket.io.js"></script>
```

Как работают запросы Soket.io (основное):

Socket.emit – отправить запрос.

Socket.on – принять запрос.

Socket.join(name_room) – Создать комнаты.

Мы отправляем на сервер запрос, и передаём данные сессии.

```
io.on('connection', (socket) => {
    //Тут должны создаваться кастомные ID

    socket.on('Id_identified',(id)=>{//Создаём кастомный id.
        console.log("------------------------------")
        console.log('Коннект пользователя');
        user = socket.id
        console.log('Id-пользователя:',user);
        ID_user=id
        userSockets[ID_user] = socket
        console.log('Кастомный-Id:',ID_user);
        console.log("------------------------------")
        for(i=0;i<room_using.length;i++){···
        }
    })
```

Сначала происходит коннект пользователя, после чего создаётся кастомный ID, а нужен, чтобы 100% идентифицировать пользователя. Потом сохраняем на сервере, что тот в сети.

В приложение В размещён код, где происходит обработка пользователя, когда тот выходит из сайта, переходит на главную страницу, а также обработка выхода пользователя в момент игры, при этом обрабатывается те случаи, когда оба пользователя выходят из игры до окончания её.

Теперь перейдём к созданию комнат (рис. 4). При нажатии «Быстрая игра», мы передаём в сокет ID пользователя и Ник. После чего определяю, были ли создана комната или нет. Если нет, то Socket создаёт комнату.

```
socket.on('cre_con_room', (data)=>{//Создаём или присоеденяемся к комнате.
    console.log("-----------------------------")
    Nick=data.Nick
    ID_user=data.ID_user
    key_click++
    client++
    rooms=Math.round(client/2)
    name_room="Room:"+rooms
    if(key_click==1){

        socket.join(name_room)

        console.log("Создана комната:",name_room)
        console.log("Ждём второго игрока.")

        room_using.push([name_room,"expec", 1 ,{Nick:Nick,ID_user:ID_user}])

        for(j=0;j<room_using.length;j++){
            console.log(room_using[j])
        }
        console.log("-----------------------------")
        socket.emit('serverMsg', {Room:name_room, fd: 0})//Комната, поле
```

Рисунок 4 – Код для создания комнаты.

Как работает комнаты:

Когда происходит запрос, переменная client увеличивается на единицу, после чего происходит математическая формула, допустим «1».

1/2=0,5 => После чего оно округляет до целого, то есть до 1.

Следующий игрок.

2/2= 1 => Следовательно он подключится к комнате 1.

3/2=1,5 => Комната будет 2. Так мы точно будем уверены тому, что в комнате будет только 2 игрока.

После чего отправляем человеку имя комнаты, в которой он находится и его поле, на котором будет пользователь. А также мы помещаем созданную комнату в массив.

Комната состоит (1/2 часть комнаты):

1. Имя комнаты.
2. Статус комнаты.
3. Кол-во игроков.
4. Первый пользователь.

Когда второй пользователь присоединяется к комнате, после чего в комнату помещается остальные данные (2/2 часть комнаты:

1. Имя второго пользователя
2. Поле первого пользователя
3. Поле второго пользователя,
4. Видимые поля.
5. Чей ход.
6. Статус комнаты меняется.

После чего отправляется данные: Ник пользователя, и триггер тому, чтобы начать игры, а именно дать возможность расстановить корабли.

Важное примечание:

```
field_xy: JSON.parse(JSON.stringify(sample.field_xy)),
field_CanShot: JSON.parse(JSON.stringify(sample.field_CanShot)),
data_ship: JSON.parse(JSON.stringify(sample.data_ship))
```

До этого я сделал модуль, который содержит шаблон с готовыми пустыми полями, и данными кораблей. Но когда копируешь массивы, то они только копируют образно «ссылку» к массиву, и, если изменить в копии будет изменён и в шаблоне. А эта конструкция

```
JSON.parse(JSON.stringify())
```

поможет сделать независимую копию массива, чтобы не изменять сам шаблон.

После чего пользователь выбирает как расположить корабли (по методу рандома и вручную). Суть такая же как в схеме, но есть важные моменты.

Очищение поля на сервере происходит по замене на пустой шаблон. А очищение поля на клиенте происходит удаление классов.

Когда игрок закончил расстановку, посылается на сервер, что тот готов игре, идёт ожидание игрока. Когда второй пользователь закончил, на сервере выполняется следующие:

```
if(rd==2){
    room_using[i][1] = "play"
    room_u = room_using[i][0]
    for(j=0;j<vr_field_pos.length;j++){
        if(vr_field_pos[j][0]==room_u){
            vr_field_pos.splice(j,1)
        }
    }
    var rd = Math.round(Math.random()) //Рандомное число от 0 до 1
    room_using[i][room_using[i].length - 1] = rd
    io.to(room_using[i][0]).emit('Play_game_ship', rd)
    console.log(rd)
```

Посылается запрос в комнату, а отдельно пользователю.

Статус комнаты изменяется на «Play», переменная rd – чей ход.

Теперь рассмотрим более подробно выстрел игрока:

```
function Click(event){
    coord_xy_p = event.srcElement.id.split(":")
    if(moving==field_user){
        socket.emit('handleClick',{
            ID_user:data_o_user.Data_session.Id,
            coord_click:coord_xy_p
        })
    }
}
```

Для игры по сети ID клеток было изменено на x:y:p

X и У уже вам известны, P – поле (0, 1), или протокол. Нужен для того, чтобы можно было отделить первое поле от второго.

Происходит событие клик, после чего на сервер отправляется координаты поля. Происходит проверка «Чей ход», и если не его ход, то дальнейшая операция не пройдёт. Если его ход, то по ID пользователя будет искаться комната.

```
room_using[i][3].ID_user==data.ID_user||room_using[i][4].ID_user==data.ID_user
```

После чего по протоколу определяем на какой поле происходит событие, после чего по координатам X и У, определяем в массиве поля противника, проверяем есть ли смысл стрелять по этой клетке, проверяем ли есть там корабль или нет.

Если он есть, то помечается во видимом массиве, что по этой клетке попали, в данных кораблях помечается, что на одну целую палубу стало меньше, а также записываем координаты попадания, также и с промахом.

Если корабль был уничтожен, то всё также помечается, но уже место попадания, будут помечаться, что корабль уничтожен, и вокруг корабля будет помечаться точками (m), также и в массиве, где можно узнать: есть смысл стрелять по этим клеткам.

Отправляется результат выстрела с такими данными:

```
Ship:room_using[i][6].data ship[j], Status:"dead/hit"
```

После чего идёт проверка, уничтожены ли все корабли, и если да, то произойдёт следующие:

```
id = room_using[i][3].ID_user
connection_db.addwin(id,(err)=>{
    if(err){
        console.log(err)
    }else{
        console.log(`Выйграл пользователь с id:${id}`)
    }
})
id = room_using[i][4].ID_user
connection_db.addloss(id,(err)=>{
    if(err){
        console.log(err)
    }else{
        console.log(`Проиграл пользователь с id:${id}`)
    }
})
```

Происходит запись изменений в базу данных. Одному игроку начисляется победа, другому поражения.

На клиентской части мы удаляем все обработчики, чтобы избежать не нужных действий, место, где показывает на кого идёт ход.

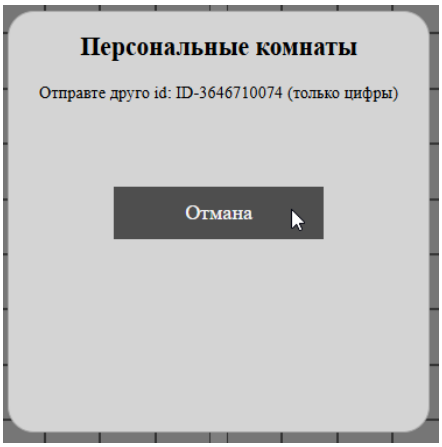
И через время появится окно, в котором будет показан победитель.

Рисунок 5 – Окно с победителем.

Перейдём к персональным комнатам. По названию понятно, что это комнаты, которые будут отдельно создаваться от других комнат, чтобы была возможность поиграть с другом или другим пользователем, не полагаясь на одновременное нажатие кнопки «Быстрая игра».

Для начала мы создадим окно, в котором будут происходить следующие операции:

- Создать комнату.
- Присоединиться к комнате.



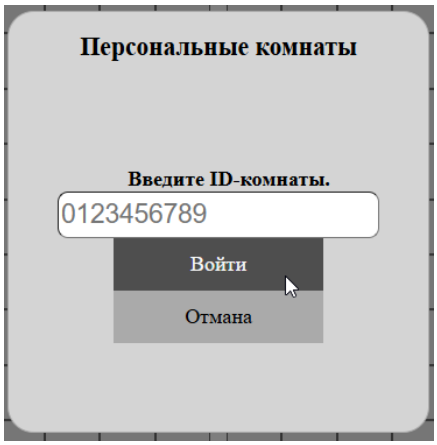


Рисунок 6 – Созданная комната

Рисунок 7 -Войти в комнату

Когда один из пользователей создаёт комнату, на сервер происходит запрос, в котором происходит цикл на 10 повторений, в котором происходит выбор цифры от 0 до 9, и так собирается ID комнаты (рис. 6).

После создании комнаты, пользователь, который создал комнату, нужно будет передать цифры ID своему другу, чтобы тот ввёл в поле ввода и смог играть с ним в созданной комнате (рис. 7).

В приложение Д размещён код, где происходит проверка комнаты на существование комнаты по такому ID, а также занята ли позиция второго игрока.

ЗАКЛЮЧЕНИЕ

Подводя итоги работы, стоит отметить, что в процессе разработки код были рассмотрены классический правила игры для игры морской, которые для всех были привычны.

Также в процессе разработки были выявлены недостатки, который можно исправить. К недостаткам могу отнести:

1. Не достаточно оптимизированный код с повторением одно и того же кода, структурированный он плохо, и другие недостатки.
2. Не до конца была сделана система так, чтобы обеспечить полную стабильность сервера. Так как клиент может влиять на некоторые параметры, которые могут серьёзно повреди саму логику игры или привести к поломки сервера.

На основе структуры игры был разработан интерфейс на EJS представление с HTML разметками, за функционал отвечал JavaScript со Soket.io, а стили и визуальным составляющим CSS, Krita и Figma.

В заключение хочется сказать, что у игры есть потенциал, а также сильные пути развития, который помогу игре расцвести новыми красками. Как и у многих проект имеет свои недостатки, но главная роль программиста, кроме разработки работоспособный код, является улучшение его, чтобы был проще, давал максимум пользы, а также эффективность его должна расти, чтобы его продуктом пользовались всё больше и больше. Это также относится и моя работа.

СПИСОК ЛИТЕРАТУРЫ

1. Флэнаган Д. — JavaScript. Подробное руководство (7-е изд.) — 2023.
2. Симпсон К. — Выразительный JavaScript (3-е изд.) — 2023.
3. Хавербеке М. — Изучаем программирование на JavaScript (2-е изд.) — 2022.
4. Азат Мардан — JavaScript и Node.js для начинающих — 2022.
5. Дакетт Дж. — HTML и CSS. Разработка и дизайн веб-сайтов — 2023.
6. Макфарланд Д. — Новая большая книга CSS — 2022.
7. Мейер Э. — CSS. Карманный справочник (5-е изд.) — 2023.
8. Кирупа П. — Node.js для начинающих — 2023.
9. Кантелон М., Хартер М. — Node.js в действии (2-е изд.) — 2022.
10. Бегер А. — Разработка серверов приложений на Node.js — 2023.
11. Алексей Орлов — Современный JavaScript для нетерпеливых — 2023.
12. Николас Закас — JavaScript для профессиональных веб-разработчиков (4-е изд.) — 2023.
13. Эрик Эллиотт — JavaScript с примерами кода — 2022.
14. Юрий Артюх — Паттерны проектирования в JavaScript — 2023.
15. Леа Веру — CSS Secrets — 2022.
16. Эстель Вейл — Flexbox и Grid Layout в CSS — 2023.
17. Джереми Кит — *HTML5 для веб-дизайнеров* — 2022.
18. Рэйчел Эндрю — The New CSS Layout — 2023.
19. Самир Бхобе — Node.js Design Patterns (3rd ed.) — 2023.
20. Лукас Да Коста — Node.js Best Practices — 2023.
21. Азад Боллук — Node.js Performance Optimization — 2022.

ИНТЕРНЕТ РЕСУРСЫ

1. https://learn.javascript.ru
2. https://metanit.com/web/javascript/
3. https://nodejsdev.ru/guides/freecodecamp/
4. https://socket.io
5. https://my-js.org/docs/guide/socket/
6. https://www.youtube.com/watch?v=0zTY73khJPM&t=392s
7. https://www.youtube.com/watch?v=zw_ZUfiJNDU

```
function Pos_ship(event){
    if(Ship<10 && key_pos==true){
        coords2= event.srcElement.id.split(";")
        Name_Ship = data_ship[Ship][0]
        x_coord = Number(coords2[0])
        y_coord = Number(coords2[1])

        let vr_coord2 = []
        for(i=0;i<data_ship[Ship][1];i++){
            if(coord_xy==0){
                vr_coord2.push([x_coord,y_coord ])
                field_xy[x_coord][y_coord]=Name_Ship
                document.getElementById(x_coord+";"+y_coord).classList.add("Sh
ip_shadow")
                y_coord=y_coord+1
            } else{
                vr_coord2.push([x_coord,y_coord ])
                field_xy[x_coord][y_coord]=Name_Ship
                document.getElementById(x_coord+";"+y_coord).classList.add("Sh
ip_shadow")
                x_coord=x_coord+1
            }
        }
        for(c=0;c<vr_coord2.length;c++){ //Обводка вокруг корабля.
            let vr_Check_Ship =[]
            //Начальные координаты палубы корабля.
            x1=vr_coord2[c][0]
            y1=vr_coord2[c][1]
            y1=y1+1 //Сдвиг вправо 1 раз.
            vr_Check_Ship.push([x1,y1])
            x1=x1+1//Сдвиг вниз 1 раз.
            vr_Check_Ship.push([x1,y1])
            for(t=0;t<2;t++){//Сдвиг влево 2 раза.
                y1=y1-1
                vr_Check_Ship.push([x1,y1])
            }
            for(tg=0;tg<2;tg++){//Сдвиг вверх 2 раза
                x1=x1-1
                vr_Check_Ship.push([x1,y1])
            }
            for(th=0;th<2;th++){//Сдвиг право 2 раза
                y1=y1+1
                vr_Check_Ship.push([x1,y1])
            }

            //Проверка координат, чтобы пометить поле возле корабля.
            for(chk=0;chk<vr_Check_Ship.length;chk++){
```

```
                x2= vr_Check_Ship[chk][0]
                y2= vr_Check_Ship[chk][1]
                if (x2>-1 && x2<10){ //9>=x2>=0
                    if (y2>-1 && y2<10){
                        if(field_xy[x2][y2]==0){
                            field_xy[x2][y2] = 1
                        }
                    }

                }
            }

        }
        Ship++
    }
    if(Ship==10){
        Play.style.display="block"
    }
}


function Change_Pos(event){//ПКМ
    if(key_f_pos==true && Ship<11){
        event.preventDefault(); // Отменяем вызов стандартного контекстного
меню браузера
        if(coord_xy == 0){
            coord_xy = 1
        }else{
            coord_xy = 0
        }
    }
}
```

```
router.post('/Register', (req, res) => {
    const { nick,login, password} = req.body;
    const hashedPassword = bcrypt.hashSync(password, 10);
    // Проверяем, что username, password и email не пустые
    if (!nick || !password || !login) {
        return res.render('Register',{ error: "Заполните полностью форму." })
    };
    console.log(login)

    if(login.length < 4){
        return res.render('Register', { error: 'Ваш логин меньше 4-ёх синволов.'});
    };

    if(password.length < 4){
        return res.render('Register', { error: 'Ваш пароль меньше 4-ёх синволов.'});
    };
    if(login.length >8){
        return res.render('Register', { error: 'Ваш пароль больше 8-ми синволов.'});
    };
    if(nick==login){
        return res.render('Register', { error: 'Пожалуйста, с цели безопасности аккаунта, Логин и ваш Ник не должен совпадать.'});
    }

    connection_db.findUserByUsername(login, (err, results) => {
        if(Object.keys(results).length > 0){
            console.error(err);
            return res.render('Register', { error: 'Простите, такой логин уже существует.'});
        }
        else{
            connection_db.createUser(nick,login, hashedPassword, (err, results) => {
            if (err) {
                console.error(err);
                return res.redirect('/Register')
            }
            res.redirect('/');
            console.log("Пользователь зарегистрерировался.")
            });
        }
    })
```

```
socket.on("disconnect", () => {//Выход
        console.log('Пользователь отключился');
        // Удаляем отключённый сокет
        for (const [id, sock] of Object.entries(userSockets)) {
          if (sock === socket) {
            for(i=0;i<room_using.length;i++){//Проверяем, есть ли пользователь
в комнате.
                if
(room_using[i][3].ID_user==id||room_using[i][4].ID_user==id) {
                    console.log(`Пользователь ${id} вышел`);
                    console.log("---------------------------")
                    console.log("Статус:",room_using[i][1])
                    console.log("---------------------------")
                    if(room_using[i][1]=="expec"){// Пользователь вышел, не
начав игру в комнате.
                        console.log("expec")
                        console.log(`Комната ${room_using[i][0]} была удалина
из массиве.`)
                        socket.leave(room_using[i][0]);
                        room_using.splice(i,1)
                        key_click=0
                        client--
                        break
                    }
                    if(room_using[i][1]=="end"||room_using[i][1]=="---"){
                        console.log("end")
                        if(room_using[i][3].ID_user==id){
                            room_using[i][3].ID_user="---"
                        }if(room_using[i][4].ID_user==id){
                            room_using[i][4].ID_user="---"
                        }
                        socket.leave(room_using[i][0]);
                        us = room_using[i][2]
                        us--
                        room_using[i][2] = us
                        io.to(room_using[i][0]).emit('no_revang')
                        if(room_using[i][2]==0){
                            console.log(`Комната ${room_using[i][0]} была
удалина из массиве.`)
                            room_using.splice(i,1)
                        }
                        break
                    }
                    if(room_using[i][1]=="expec_play"){
                        if(room_using[i][3].ID_user==id){
                            room_using[i][3].ID_user="---"
```

```
                        }if(room_using[i][4].ID_user==id){
                            room_using[i][4].ID_user="---"
                        }

                        for(j=0;j<vr_field_pos.length;j++){
                            if(vr_field_pos[j][0]==room_using[i][0]){
                                vr_field_pos.splice(j,1)
                            }
                        }

                        room_using[i][1] = "end"
                        socket.leave(room_using[i][0]);
                        console.log("Vr_:",vr_field_pos)
                        us = room_using[i][2]
                        us--
                        room_using[i][2] = us
                        console.log(`Игра была отменина.`)
                        io.to(room_using[i][0]).emit('cancellation')
                    }
                    if(room_using[i][1]=="play"||room_using[i][1]=="t_stusy"){
                        console.log("play-t_stusy")
                        room_using[i][1] = "t_stusy"
                        console.log(`Пользователь ${id} вышел из игры.`);
                        socket.leave(room_using[i][0]);
                        us = room_using[i][2]
                        us--
                        room_using[i][2] = us
                        if(room_using[i][2]==0){
                            console.log(`Игра была отменина в комнате:
${room_using[i][0]}. `)
                            room_using.splice(i,1)
                        }
                        break
                    }
                }
            }
            delete userSockets[id];
            console.log("-------------------------")
            console.log(room_using)
            console.log(`Пользователь ${id} отключён`);
            console.log("-------------------------")
            break;
          }
        }
    });
```

```
const express = require('express');
const app = express();
const http = require('http').createServer(app);
const io = require('socket.io')(http);
const path = require('path');
const session = require('express-session')
app.use(session({
    secret:"Secret",
    resave: false,
    saveUninitialized: true,
    cookie: {maxAge:60000*24}
}))
```

```
//Импорт маршрутов.
var indexRouter = require('./routes/index');
```

```
app.use(express.urlencoded({ extended: true }));
app.use(express.json());

// Настройка EJS
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'ejs');

// Middleware для статических файлов
app.use(express.static(path.join(__dirname, 'static')));

app.use((req,res,next) =>{
    res.locals.sessionMiddleware= req.session
    next()
})
// Маршрут для главной страницы

app.use('/', indexRouter);
```

```
// Запуск сервера
const PORT = 3000;
http.listen(PORT, () => {
    console.log(`Сервер запущен на http://localhost:${PORT}`);
});
```

```
socket.on('Log_rooming_1', (data)=>{
        console.log("Я сработал")
        console.log(data)
        noting = 0
        room_prem="ID-"+`${data.ID_room}`
        for(i=0;i<room_using.length;i++){
            console.log("Ищем.")
            if(room_using[i][0]==room_prem){
                console.log(room_using[i][0])
                console.log(room_using[i].length)
                if(room_using[i].length<5){
                    console.log("Можно присоедениться")
                    socket.join(room_prem)
                    socket.emit('serverMsg', {Room:room_prem, fd: 1})
                    room_using[i].push(
                        {Nick:data.Nick,ID_user:data.ID_user},
                        {
                            field_xy: JSON.parse(JSON.stringify(sample.field_xy)),
                            field_CanShot: JSON.parse(JSON.stringify(sample.field_CanShot)),
                            data_ship: JSON.parse(JSON.stringify(sample.data_ship))
                        },
                        {
                            field_xy: JSON.parse(JSON.stringify(sample.field_xy)),
                            field_CanShot: JSON.parse(JSON.stringify(sample.field_CanShot)),
                            data_ship: JSON.parse(JSON.stringify(sample.data_ship))
                        },
                        {
                            See_field1: JSON.parse(JSON.stringify(sample.field_xy)),
                            See_field2: JSON.parse(JSON.stringify(sample.field_xy))
                        },
                        0
                    )
                    room_using[i][1] = "expec_play"
                    console.log("Статус:",room_using[i][1])
                    room_using[i][2] = 2

                    console.log("Отправка Ников.")

                    use1 = room_using[i][3].ID_user//id 1 пользователя комнаты
                    use2 = room_using[i][4].ID_user//id 2 пользователя комнаты

                    Room_Users = {User1_Nick:room_using[i][3].Nick,
User2_Nick:room_using[i][4].Nick}

                    socket = userSockets[use1];//Soket 1 пользователя комнаты
                    socket.emit('Conn_us', Room_Users)

                    socket = userSockets[use2];//Soket 2 пользователя комнаты
                    socket.emit('Conn_us', Room_Users)

                    io.to(room_prem).emit('Play_game')
                    break
                }else{
                    console.log("Она есть, но уже заполнена.")
                    Status = "Простите, эта комната уже занята."
```

```
                socket.emit('err_status',Status)
            }
        }else{
            noting++
            console.log(noting)
            if(noting==room_using.length){
                console.log("Нет таких комнат")
                Status = "Такой комнаты нет."
                socket.emit('err_status',Status)
            }
        }
    }
    if(room_using.length==0){
        Status = "Такой комнаты нет."
        socket.emit('err_status',Status)
    }
})
```