МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

«Забайкальский государственный университет»

(ФГБОУ ВПО «ЗабГУ»)

Факультет: Энергетический

Кафедра: Информатики, вычислительной техники и прикладной математики

**КУРСОВАЯ РАБОТА**

По дисциплине: Протоколы вычислительных сетей

На тему: «Разработка клиент-серверного приложения для межсетевого взаимодействия»

Выполнил студент группы ВМК–21, Рычков Родион Викторович

Руководитель работы: старший преподаватель кафедры ИВТ и ПМ, Забелин Вячеслав Олегович

Чита

2024

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

«Забайкальский государственный университет»

(ФГБОУ ВПО «ЗабГУ»)

Факультет: Энергетический

Кафедра: Информатики, вычислительной техники и прикладной математики

**ЗАДАНИЕ**

на курсовую работу

По дисциплине: Протоколы вычислительных сетей

Студенту: Рычкову Родиону Викторовичу

Специальности (направления подготовки): Вычислительные машины и комплексы

1 Тема курсовой работы: «Сетевая игра “крестики-нолики”»

2 Срок подачи студентом законченной работы: 20.06.2024

3 Исходные данные к работе: описание предметной области

Дата выдачи задания: 15.02.2024

Руководитель курсовой работы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Забелин В.О./

(подпись, расшифровка подписи)

Задание принял к исполнению

«15» февраля 2024 г.

Подпись студента\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /Рычков Р.В. /

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

«Забайкальский государственный университет»

(ФГБОУ ВПО «ЗабГУ»)

Факультет: Энергетический

Кафедра: Информатики, вычислительной техники и прикладной математики

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовой работе

По дисциплине: Протоколы вычислительных сетей

На тему: «Сетевая игра “крестики-нолики”»

Выполнил студент группы ВМК-21, Рычков Родион Викторович

Руководитель работы: старший преподаватель кафедры ИВТ и ПМ, Забелин Вячеслав Олегович

Чита

2024

КАЛЕНДАРНЫЙ ГРАФИК

выполнения курсовой работы

УТВЕРЖДАЮ

    Зав.кафедрой\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2024 г.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Этапы выполнения курсовой работы | Месяцы и недели | | | | | | | | | | | | | | |
| Февраль | | | Март | | | | Апрель | | | | Май | | | |
| 1. Получение задания на курсовую работу | ++ | + |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 1. Анализ задачи | + | ++ | + |  |  |  |  |  |  |  |  |  |  |  |  |
| 1. Анализ данных |  | + | ++ |  | + | + |  |  |  |  |  |  |  |  |  |
| 1. Программная реализация |  |  | + | ++ | ++ | ++ | ++ | + | + | + | + |  |  |  |  |
| 1. Тестирование |  |  |  | + |  |  |  | ++ | ++ | 3 | + | + |  |  |  |
| 1. Документирование |  |  |  |  |  |  |  |  |  | ++ | ++ | + | + |  | + |
| 1. Представление руководителю чернового варианта работы |  |  |  |  |  |  |  |  |  |  |  | ++ | + |  | + |
| 1. Корректировка работы в соответствии с замечаниями руководителя |  |  |  |  |  |  |  |  |  |  |  |  | ++ | ++ | + |
| 1. Защита работы |  |  |  |  |  |  |  |  |  |  |  |  |  |  | ++ |

План выполнен: руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись, расшифровка подписи)

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2024 г

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 7](#_Toc169447186)

[1. Теоретическая часть 8](#_Toc169447187)

[1.1 Описание предметной области 8](#_Toc169447188)

[1.2 Обзор технологии HTTP 10](#_Toc169447189)

[1.3 Обзор Node.js 12](#_Toc169447190)

[2. Практическая часть 13](#_Toc169447191)

[2.1 Описание проекта 13](#_Toc169447192)

[2.2 Руководство пользователя 14](#_Toc169447193)

[ЗАКЛЮЧЕНИЕ 19](#_Toc169447194)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 20](#_Toc169447195)

[ПРИЛОЖЕНИЕ 22](#_Toc169447196)

# РЕФЕРАТ

Пояснительная записка ­­­­­–­ 39 с, 7 рис., 5 источников.

ЯЗЫК JAVASCRIPT, HTTP, КЛИЕНТ-СЕРВЕР

В данной работе рассматривается процесс создания клиент-серверного приложения, представляющего собой сетевую игру "Крестики-Нолики" с использованием технологии HTTP, реализованного на HTML, CSS, JavaScript и Node.js.

# ВВЕДЕНИЕ

Цель данной работы – реализовать сетевую игру «Крестики-нолики» с клиент-серверной архитектурой, используя технологию HTTP для взаимодействия между клиентом и серверов

Задачи:

1. Изучить основы создания клиент-серверных приложений с использованием технологии HTTP
2. Разработать сервер на Node.js для обработки запросов клиентов.
3. Реализовать клиентскую часть игры на HTML, CSS и JavaScript
4. Обеспечить взаимодействие между клиентами через сервер для проведения игровых сессий

## 1. Теоретическая часть

### 1.1 Описание предметной области

Клиент-серверные приложения являются одним из наиболее распространенных типов приложений в современной веб-разработке. В таких приложениях клиент отправляет запросы серверу, который обрабатывает эти запросы и отправляет ответы обратно клиенту. Для создания клиент-серверного приложения можно использовать различные протоколы и технологии. В данной работе выбрана технология HTTP, а для реализации серверной части используется Node.js, а для клиентской части - HTML, CSS и JavaScript.

Клиент-серверная архитектура разделяет функции приложения на две части: клиентскую и серверную. Клиент выполняет задачи, связанные с пользовательским интерфейсом и взаимодействием с пользователем, тогда как сервер обрабатывает бизнес-логику, управляет базой данных и выполняет другие задачи, требующие централизованного управления. Такой подход позволяет:

1. Обеспечить централизованный контроль над логикой и данными приложения.
2. Облегчить обновление и поддержку приложения, так как большая часть изменений может быть внесена на сервере без необходимости обновления клиентских приложений.
3. Повысить безопасность, так как данные обрабатываются и хранятся на сервере.

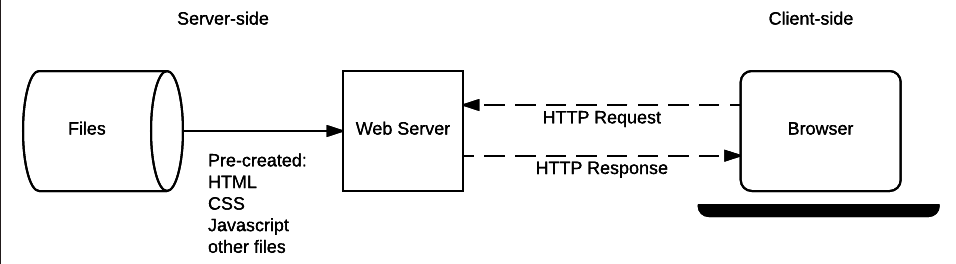


Рис.1 Работа клиент-сервера

### 1.2 Обзор технологии HTTP

HyperText Transfer Protocol (HTTP) — это прикладной протокол, который лежит в основе Всемирной паутины. HTTP определяет правила и стандарты для передачи данных между клиентом (обычно веб-браузером) и сервером. Это статeless протокол, что означает, что каждое сообщение HTTP обрабатывается независимо, без сохранения состояния между соединениями.

HTTP использует клиент-серверную модель взаимодействия. Клиент отправляет HTTP-запрос серверу, который затем возвращает HTTP-ответ. Запросы и ответы содержат три основных компонента:

1. **Стартовая строка**: Определяет метод запроса (например, GET, POST), URL-адрес и версию HTTP.
2. **Заголовки**: Несут метаданные о запросе или ответе, такие как тип контента, длина содержимого и информация о сессии.
3. **Тело**: Содержит данные, передаваемые клиентом или сервером, например, HTML-код, JSON-данные и т.д.

Веб-программирование включает создание клиентских и серверных компонентов для веб-приложений. Основными технологиями для разработки клиентской части являются HTML, CSS и JavaScript:

1. **HTML (HyperText Markup Language)**: Используется для создания структуры веб-страниц. HTML определяет элементы и их иерархию на странице, такие как заголовки, абзацы, изображения, ссылки и формы.
2. **CSS (Cascading Style Sheets)**: Применяется для стилизации HTML-элементов. С помощью CSS можно изменять цвета, шрифты, размеры, отступы и расположение элементов на странице. CSS позволяет отделить представление данных от их структуры.
3. **JavaScript**: Язык программирования, который позволяет добавлять интерактивность на веб-страницы. JavaScript используется для динамического обновления контента, обработки событий и взаимодействия с сервером без перезагрузки страницы через технологии, такие как AJAX (Asynchronous JavaScript and XML).

### 1.3 Обзор Node.js

**Node.js** — это серверная платформа, построенная на движке V8 от Google, которая позволяет запускать JavaScript на стороне сервера. Node.js поддерживает асинхронное программирование, что делает его отличным выбором для разработки масштабируемых сетевых приложений.

Node.js основывается на неблокирующей модели ввода-вывода (I/O), которая позволяет обрабатывать множество одновременных соединений без создания новых потоков для каждого из них. Это достигается за счёт использования событийно-ориентированной архитектуры и цикла событий (event loop), который обрабатывает все асинхронные операции.

Основные особенности Node.js:

1. **Асинхронность**: Node.js использует неблокирующие вызовы I/O, что позволяет выполнять другие операции, пока происходит ввод-вывод. Это существенно повышает производительность и масштабируемость приложений.
2. **Однопоточность**: Хотя Node.js использует однопоточный цикл событий, он может управлять множеством соединений одновременно благодаря своей асинхронной природе.
3. **Богатая экосистема**: Node.js обладает огромным количеством модулей и библиотек, доступных через npm (Node Package Manager), что ускоряет процесс разработки и добавляет множество готовых решений для различных задач.

## 2. Практическая часть

### 2.1 Описание проекта

В этом разделе рассматривается процесс разработки клиент-серверного приложения для сетевой игры "Крестики-Нолики" на основе протокола HTTP. Приложение состоит из серверной части, реализованной на Node.js, и клиентской части, разработанной с использованием HTML, CSS и JavaScript. Основная цель проекта — обеспечить взаимодействие между двумя игроками, которые могут играть друг с другом через интернет.

Архитектура приложения включает два основных компонента: серверную часть и клиентскую часть. Серверная часть отвечает за управление состоянием игры, обработку ходов игроков и синхронизацию данных между клиентами. Клиентская часть предоставляет пользовательский интерфейс для взаимодействия с игрой и отправки запросов на сервер.

1. **Серверная часть**:
   * **Node.js**: Используется в качестве серверной платформы, обеспечивающей высокую производительность и асинхронную обработку запросов.
   * **HTTP**: Протокол для передачи данных между клиентом и сервером.
   * **WebSocket**: Протокол для двусторонней связи в реальном времени между клиентом и сервером, используемый для мгновенного обновления состояния игры.
2. **Клиентская часть**:
   * **HTML/CSS**: Язык разметки и стилизации, используемые для создания интерфейса игры.
   * **JavaScript**: Язык программирования, обеспечивающий динамическое взаимодействие с сервером и обновление пользовательского интерфейса.
   * **WebSocket**: Используется для получения обновлений от сервера в реальном времени.

### 2.2 Руководство пользователя

При первом запуске любой пользователь видит стартовый экран (рис. 2) с кнопкой Ready, смена иконок крестика и нолика Change Igame, выбором иконок и сменой размера поля. Пользователь нажимает на кнопку Ready и игра запускается с выбранными настройками. На странице клиента игрок ожидает когда игру запустит сервер.

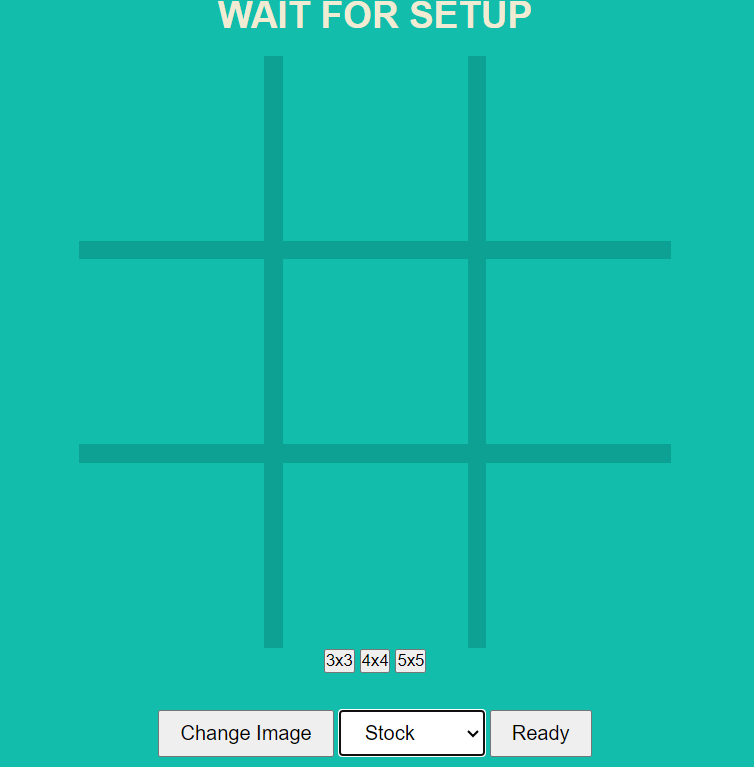


Рис.2 Меню главного игрока

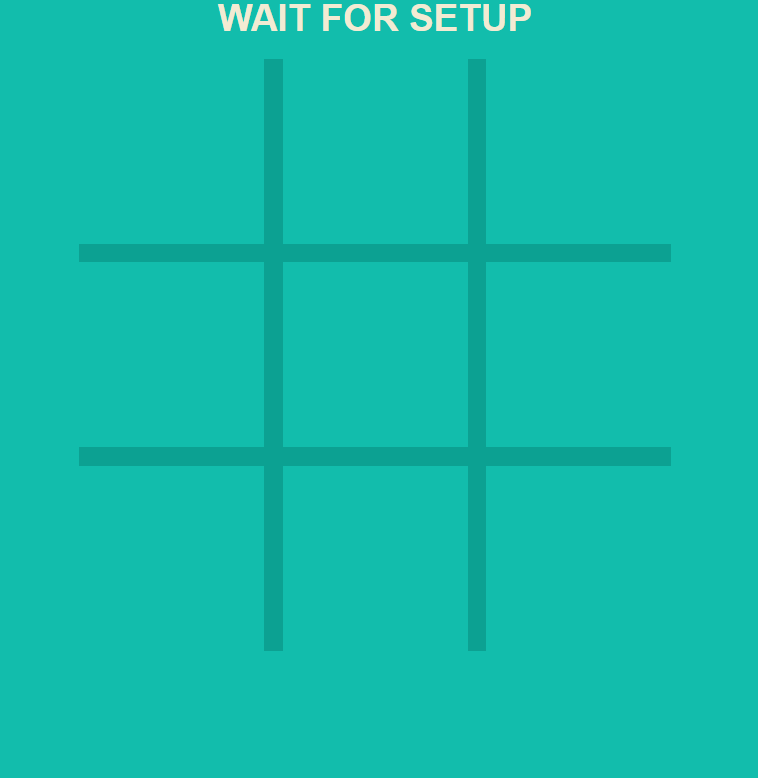


Рис.3. Меню клиента

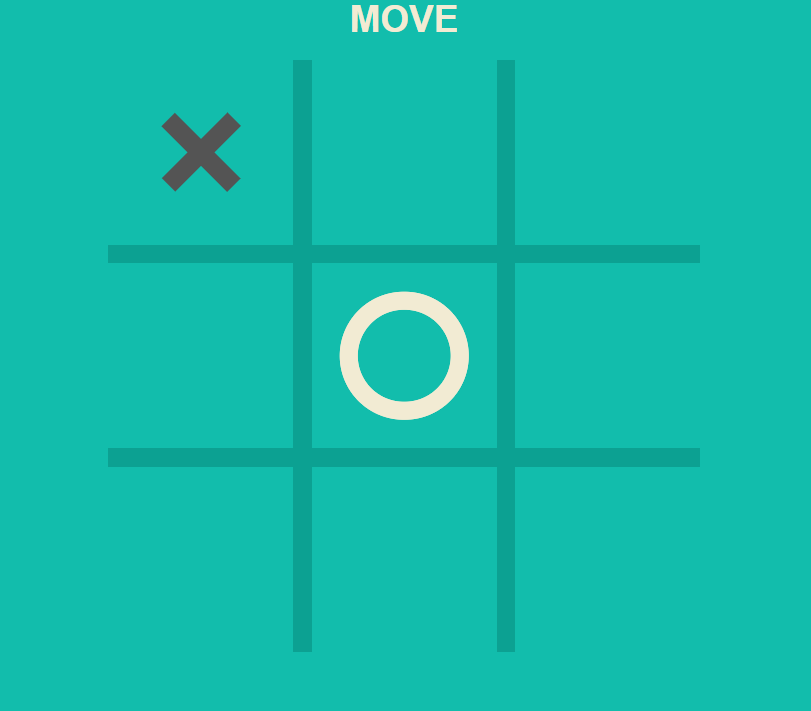


Рис 4. Ход игроков

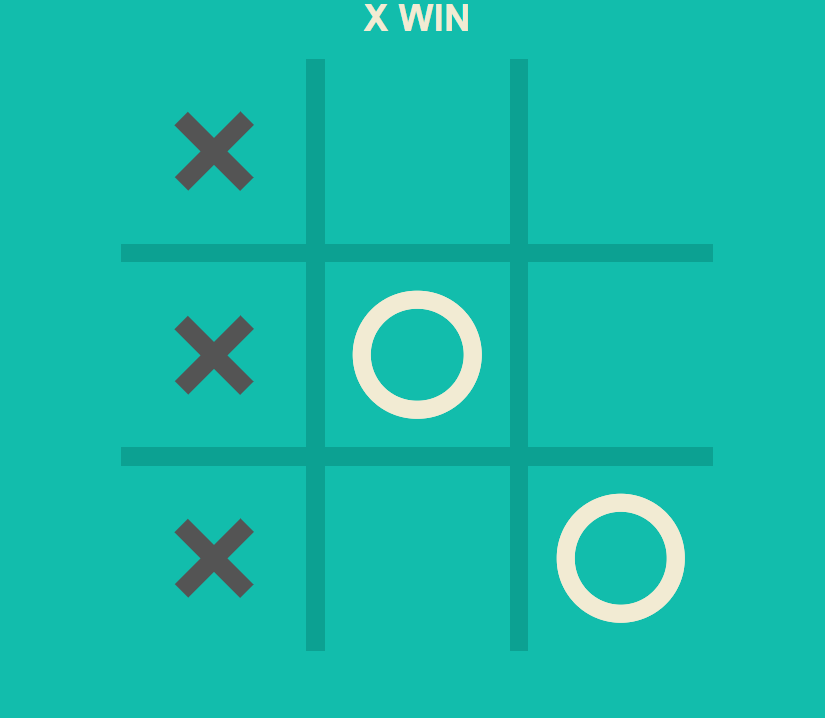


Рис. 5. Победа главного игрока

Чтобы перезапустить игру, необходимо обоим пользователям перезапустить страницу

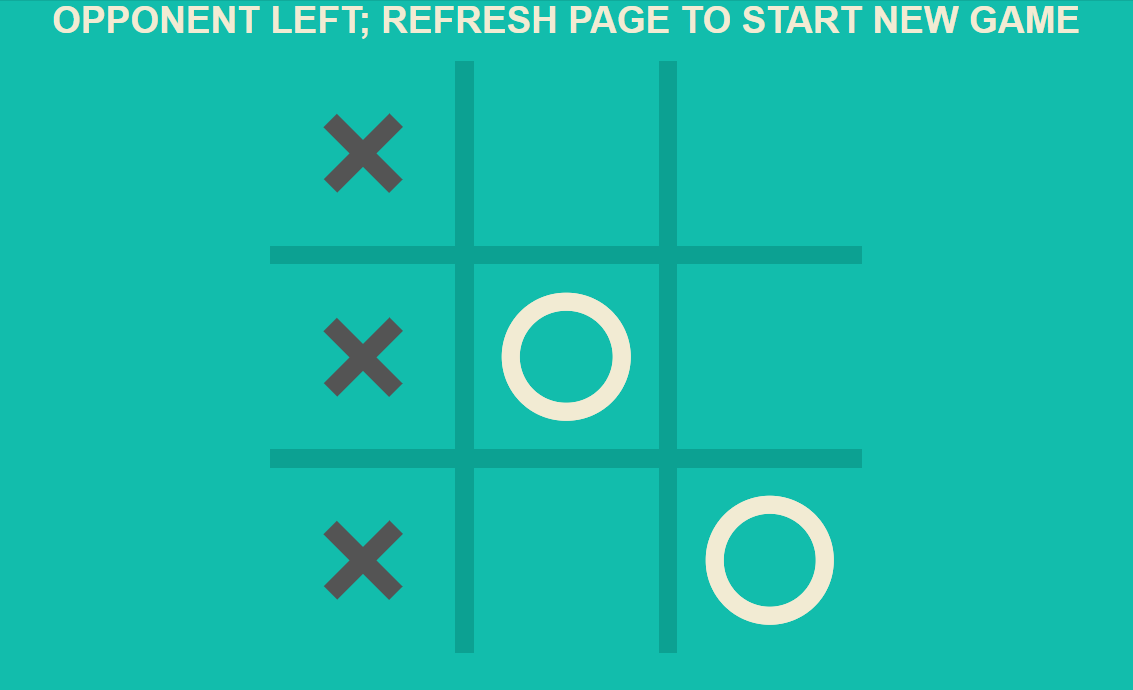


Рис.6. Уведомление о необходимости перезагрузки

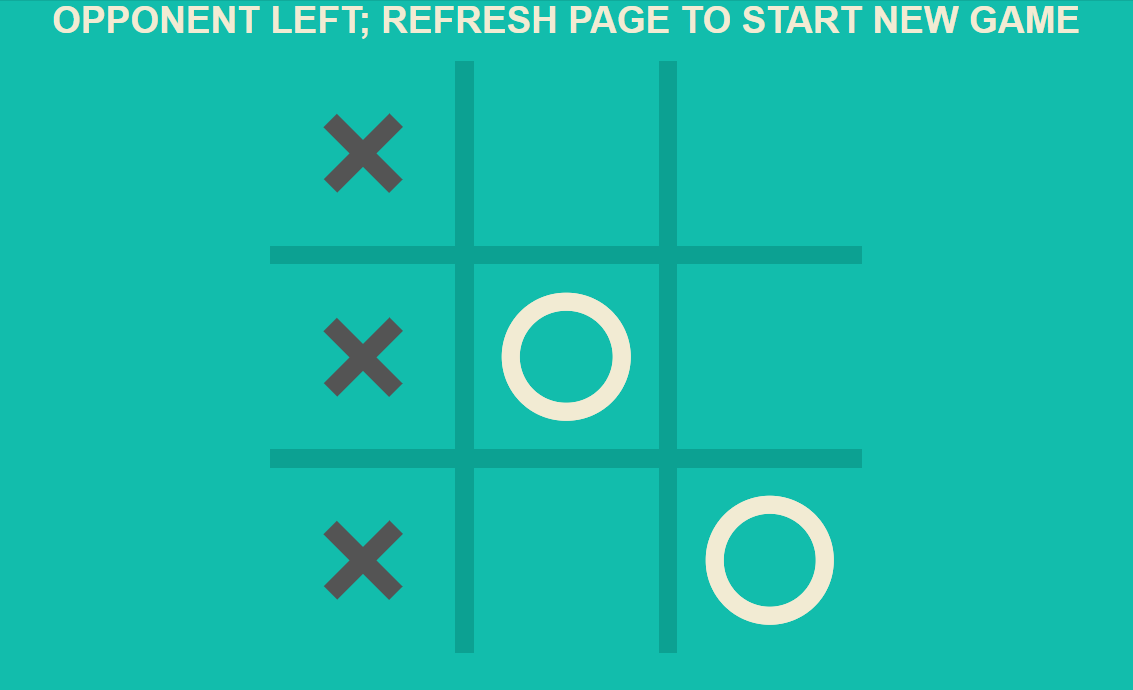


Рис.7. Поле 4x4

# ЗАКЛЮЧЕНИЕ

В данной работе был представлен процесс разработки клиент-серверного приложения для сетевой игры "Крестики-Нолики" на основе протокола HTTP. Реализация проекта включала создание серверной части на платформе Node.js и клиентской части с использованием HTML, CSS и JavaScript. Приложение предоставляет возможность пользователям взаимодействовать в реальном времени, играя друг с другом через интернет, что достигается благодаря использованию WebSocket для двусторонней связи.

**Основные результаты**

1. **Архитектура приложения**: Были определены ключевые компоненты системы, включая серверную и клиентскую части. Разработана структура, обеспечивающая эффективное взаимодействие и обмен данными между ними.
2. **Функциональные и нефункциональные требования**: Приложение удовлетворяет основным функциональным требованиям, таким как, создание к игре, обработка ходов и синхронизация состояния игры
3. **Технологический стек**: В ходе работы использовались современные технологии, такие как Node.js, HTML, CSS и JavaScript. Это позволило создать приложение, способное удовлетворить потребности пользователей.
4. **Реализация и тестирование**: Приложение было успешно реализовано и протестировано.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Клиент-серверная архитектура [Электронный ресурс] –  
   URL: https://vladislaveremeev.gitbook.io/qa\_bible/seti-i-okolo-nikh/klient-servernaya-arkhitektura-client-server-architecture (Дата обращения: 10.06.2024)
2. Технология HTTP [Электронный ресурс] –  
   URL: https://skillbox.ru/media/code/chto-takoe-http-i-zachem-on-nuzhen/?ysclid=lxh6i6obgs600977361 (Дата обращения: 03.03.2024)
3. Документация JavaScript [Электронный ресурс] –   
   URL: <https://devdocs.io/javascript/> (Дата обращения: 04.03.2024)
4. Компьютерные сети. Принципы, технологии, протоколы: Юбилейное издание [Текст]. Учебник для вузов. / Олифер Виктор, Олифер Наталья – СПб.: Питер, 2020 ­­­– 1008. (Дата обращения 01.04.2024)
5. Руководство Node.js [Электронный ресурс] –  
   URL: https://nodejsdev.ru/guides/freecodecamp/ (Дата обращения: 01.05.2024)

# ПРИЛОЖЕНИЕ

Код html страницы игры

<!-- index.html -->

<!DOCTYPE html>

<html lang="en">

  <head>

    <meta charset="UTF-8" />

    <meta http-equiv="X-UA-Compatible" content="IE=edge" />

    <meta name="viewport" content="width=device-width, initial-scale=1.0" />

    <link rel="stylesheet" href="./style.css" />

    <script type="module" src="./script.js" defer></script>

    <title>Tic Tac Toe</title>

  </head>

  <body>

    <div class="message">searching...</div>

    <div class="board">

      <!-- Cells will be dynamically generated based on the selected size -->

    </div>

    <!-- Add buttons for different board sizes -->

    <div id="sizeButtons" style="opacity: 0;">

      <button id="size3Button">3x3</button>

      <button id="size4Button">4x4</button>

      <button id="size5Button">5x5</button>

    </div>

    <div id="changeImg" style="opacity: 0;">

    <button id="changeImageBtn">Change Image</button>

    <select id="imageDropdown">

      <option value="zemelya">Zemelya</option>

      <option value="cpu">CPU</option>

      <option value="stock">Stock</option>

    </select>

    <button id="readyButton" disabled>Ready</button>

  </div>

  </body>

</html>

Код скрипта

// script.js

let size = 0; // Инициализируем размер по умолчанию

let cellElements = [];

const messageElement = document.querySelector(".message");

let field = Array(size \* size).fill(""); // Используем fill для инициализации массива пустыми значениями

let isGameActive = false; // флаг "Активна игра или нет"

let symbol = null; // текущий символ

let turn = null; // ход

let ws = new WebSocket("ws://localhost:8080"); // адрес, куда идут запросы WebSocket

const buttonContainer = document.getElementById("sizeButtons");

const buttonImg = document.getElementById("changeImg");

buttonContainer.style.display = "none";

buttonImg.style.display = "none";

let isCustom = false; // флаг режима "Кастомные иконки"

const board = document.querySelector(".board");

const tictacImg = {

  zemelya: ["./custom/osipov1.svg", "./custom/petan1.svg"],

  cpu: ["./custom/intel.svg", "./custom/amd.svg"],

}; // категории картинок и их соответствие

let selectedCategory; // выбранная категория картинок

let isHostReady = false; // флаг готовности хоста

let isHost = false; // флаг "хост игрок или нет"

// событие переключения с картинок на сток

changeImageBtn.addEventListener("click", function () {

  handleDropDownChange();

});

function handleDropDownChange() {

  const imageDropdown = document.getElementById("imageDropdown");

  selectedCategory = imageDropdown.value.toLowerCase();

  handleImageChange();

}

document.getElementById("readyButton").addEventListener("click", function () {

  selectedCategory = isCustom === false ? "stock" : selectedCategory;

  //console.log("sending to server: ", size);

  ws.send(

    JSON.stringify({

      method: "resize",

      size: size,

    })

  );

  // Отправка сообщения о готовности серверу

  ws.send(

    JSON.stringify({

      method: "hostReady",

      category: selectedCategory,

    })

  );

  isHostReady = true;

  updateMessage();

});

// переключатель картинка/стоковые tic-tac

function handleImageChange() {

  //console.log(selectedCategory);

  if (selectedCategory != "stock") isCustom = true;

  else isCustom = false;

  //console.log("custom flag:", isCustom);

  updateBoard();

}

// Функция для обновления изображений с использованием значений из объекта

function updateBackgroundImages() {

  cellElements.forEach((cell, index) => {

    const xIndex = 0; // Индекс для X-img

    const oIndex = 1; // Индекс для img-O

    // Выбираем изображение в зависимости от символа в поле (X или O)

    const backgroundImage =

      field[index] === "X"

        ? `url('${tictacImg[selectedCategory][xIndex]}')`

        : `url('${tictacImg[selectedCategory][oIndex]}')`;

    // Применяем выбранное изображение к фону ячейки

    cell.style.setProperty("--bg-img-before", backgroundImage);

    cell.style.setProperty("--bg-img-after", backgroundImage);

  });

}

// генерация полей

function generateField() {

  field = Array(size \* size).fill(""); // Пересоздаем поле при изменении размера

  generateBoard(size);

  updateBoard();

}

// кнопки изменения размера

document.getElementById("size3Button").addEventListener("click", function () {

  changeBoardSize(3);

});

document.getElementById("size4Button").addEventListener("click", function () {

  changeBoardSize(4);

});

document.getElementById("size5Button").addEventListener("click", function () {

  changeBoardSize(5);

});

// генерация игрового поля

function generateBoard(size) {

  const board = document.querySelector(".board");

  board.innerHTML = ""; // Очищаем существующее поле

  let columns;

  switch (size) {

    case 3:

      columns = 3;

      break;

    case 4:

      columns = 4;

      break;

    case 5:

      columns = 5;

      break;

    default:

      columns = 3;

      break;

  }

  // Создаем сетку с заданным количеством колонок

  board.style.gridTemplateColumns = `repeat(${columns}, 20vmin)`;

  for (let i = 0; i < size \* size; i++) {

    const cell = document.createElement("div");

    cell.classList.add("cell");

    board.appendChild(cell);

  }

}

// обновление размера поля

function changeBoardSize(newSize) {

  size = newSize;

  generateField();

}

ws.onmessage = (message) => {

  const response = JSON.parse(message.data);

  //.log("From server: ", response);

  // Обработка входа игрока

  if (response.method === "join") {

    symbol = response.symbol;

    turn = response.turn;

    size = response.size === 0 ? 3 : response.size;

    isGameActive = symbol === turn;

    changeBoardSize(size);

    updateMessage();

  }

  // Выведем сообщение о состоянии хода (кто ходит?)

  if (response.method === "update") {

    field = response.field;

    turn = response.turn;

    isGameActive = symbol === turn;

    updateBoard();

    updateMessage();

  }

  // Выведем сообщение об исходе схватки

  if (response.method === "result") {

    field = response.field;

    updateBoard();

    isGameActive = false;

    setTimeout(() => {

      messageElement.textContent = response.message;

    }, 100);

    removeCellEvent();

  }

  // Выведем сообщение о выходе оппонента

  if (response.method === "left") {

    isGameActive = false;

    messageElement.textContent =

      response.message + "; Refresh page to start new game";

    removeCellEvent();

  }

  // Хосту отрисуем кнопки изменения визуала игры

  if (response.method === "isHost") {

    buttonContainer.style.display = "block";

    buttonImg.style.display = "block";

    isHost = true;

  }

  if (response.method === "hostReady") {

    isHostReady = true;

    selectedCategory = response.selectedCategory;

    handleImageChange();

    updateMessage();

  }

  if (response.method === "resize") {

    changeBoardSize(response.size);

  }

};

// обработка нажатия на клетку

function makeMove(index) {

  if (!isGameActive || field[index] !== "") {

    return;

  }

  isGameActive = false;

  field[index] = symbol;

  ws.send(

    JSON.stringify({

      method: "move",

      symbol: symbol,

      field: field,

      size: size,

    })

  );

}

// обновление доски

function updateBoard() {

  cellElements.forEach((cell, index) => {

    // если выбраны картинки вместо стока

    if (isCustom) {

      if (field[index] !== "") {

        cell.classList.remove(field[index] + "-img"); // Убираем текущий класс суффикса -img

        cell.classList.remove(field[index]); // Убираем текущий класс

        updateBackgroundImages();

        //console.log("Custom true", cell.classList);

        cell.classList.add(field[index] + "-img"); // Добавляем соответствующий класс суффикса -img

      }

    } else {

      if (field[index] !== "") {

        cell.classList.remove(field[index]); // Убираем текущий класс поля

        cell.classList.remove(field[index] + "-img"); // Убираем текущий класс суффикса -img

        // console.log("Custom false", cell.classList);

        cell.classList.add(field[index]); // Возвращаем сток

      }

    }

  });

}

// функция обновления состояния игры

function updateMessage() {

  //console.log(isHostReady);

  if (isHostReady) {

    addCellEvent();

    if (symbol === turn) {

      messageElement.textContent = "move";

      handleDisplayBlock();

    } else {

      messageElement.textContent = `waiting ${turn}...`;

      handleDisplayBlock();

    }

  } else {

    messageElement.textContent = "Wait for setup"; // Новое сообщение для ожидания готовности хоста

    buttonContainer.style.display = isHost === true ? "block" : "none";

    buttonImg.style.display = isHost === true ? "block" : "none";

    const readyButton = document.getElementById("readyButton");

    readyButton.disabled = false;

    buttonContainer.style.opacity = 1;

    buttonImg.style.opacity = 1;

    removeCellEvent();

  }

}

function handleDisplayBlock() {

  buttonContainer.style.display = "none";

  buttonImg.style.display = "none";

}

function removeCellEvent() {

  // Устанавливаем pointer-events: none для каждой ячейки

  cellElements.forEach((cell) => {

    cell.style.pointerEvents = "none";

  });

  cellElements.forEach((cell, index) =>

    cell.removeEventListener("click", () => {

      makeMove(index);

    })

  );

}

function addCellEvent() {

  cellElements = document.querySelectorAll(".cell");

  cellElements.forEach((cell, index) =>

    cell.removeEventListener("click", (event) => {

      makeMove(index);

    })

  );

  cellElements.forEach((cell, index) =>

    cell.addEventListener("click", (event) => {

      makeMove(index);

    })

  );

}

Код CSS файла

\* {

  padding: 0;

  margin: 0;

  -webkit-user-select: none;

  -moz-user-select: none;

  -ms-user-select: none;

  user-select: none;

  -webkit-tap-highlight-color: transparent;

}

body {

  display: flex;

  flex-direction: column;

  /\* justify-content: center; \*/

  align-items: center;

  height: 100vh;

  margin: 0;

  overflow: auto;

  background-color: #12bdac;

}

.board {

  display: grid;

  grid-template-columns: repeat(3, 20vmin);

  grid-template-rows: repeat(3, 20vmin);

  gap: 2vmin;

  background-color: #0ca192;

}

.cell {

  position: relative;

  cursor: pointer;

  background-color: #12bdac;

  width: 20vmin;  /\* Используем относительные единицы \*/

  height: 20vmin; /\* Используем относительные единицы \*/

}

.cell.X:before,

.cell.X:after {

  content: "";

  position: absolute;

  top: 50%;

  left: 50%;

  width: 10vmin;

  height: 2vmin;

  background: #545454;

}

.cell.X-img:before,

.cell.X-img:after {

  content: "";

  position: absolute;

  top: 50%;

  left: 50%;

  transform: translate(-50%, -50%);

  width: 17vmin;

  height: 17vmin;

  background: none; /\* Убираем фон, так как будем использовать background-image \*/

}

.cell.X-img:before {

  background-image:  var(--bg-img-before);

  background-size: cover; /\* Растягиваем изображение, чтобы оно занимало всю площадь \*/

}

.cell.X-img:after {

  background-image:  var(--bg-img-after);

  background-size: cover; /\* Растягиваем изображение, чтобы оно занимало всю площадь \*/

}

.cell.X:before {

  transform: translate(-50%, -50%) rotate(45deg);

}

.cell.X:after {

  transform: translate(-50%, -50%) rotate(-45deg);

}

.cell.O:before,

.cell.O:after {

  content: "";

  position: absolute;

  top: 50%;

  left: 50%;

  transform: translate(-50%, -50%);

  width: 10vmin;

  height: 10vmin;

  border-radius: 50%;

  border: 2vmin solid #f2ebd3;

}

.message {

  margin-bottom: 2vmin;

  font-family: Arial, Helvetica, sans-serif;

  font-size: 4vmin;

  font-weight: bold;

  text-transform: uppercase;

  color: #f2ebd3;

}

.cell.O-img:before {

  background-image: var(--bg-img-before);

  background-size: cover; /\* Растягиваем изображение, чтобы оно занимало всю площадь \*/

}

.cell.O-img:after {

  background-image: var(--bg-img-after);

  background-size: cover; /\* Растягиваем изображение, чтобы оно занимало всю площадь \*/

}

.cell.O-img:before,

.cell.O-img:after {

  content: "";

  position: absolute;

  top: 50%;

  left: 50%;

  transform: translate(-50%, -50%);

  width: 17vmin;

  height: 17vmin;

}

#changeImg {

  margin-top: 20px;

  display: flex;

  flex-direction: column;

  align-items: center;

}

#changeImageBtn {

  margin-top: 10px;

  padding: 8px 16px;

  font-size: 16px;

  cursor: pointer;

}

#imageDropdown {

  margin-top: 10px;

  padding: 8px 16px;

  font-size: 16px;

}

#readyButton {

  margin-top: 10px;

  padding: 8px 16px;

  font-size: 16px;

  cursor: pointer;

}

Код сервера

// server.js

const express = require("express");

const path = require("path");

const http = require("http");

const WebSocket = require("ws");

const Player = require("./player");

const game = require("./game");

// Класс сервера "крестики-нолики"

class TicTacToeServer {

  // Конструктор класса TicTacToeServer

  constructor() {

    // Инициализация объектов сервера, WebSocket и соответствующих списков

    this.app = express(); // Экспресс-приложение

    this.httpServer = http.createServer(); // HTTP-сервер

    this.wss = new WebSocket.Server({ server: this.httpServer }); // Сервер WebSocket

    // Список активных соединений клиентов WebSocket

    this.clientConnections = {};

    this.games = [];

    // Словарь для отслеживания соответствия оппонентов по их clientId

    this.opponents = {};

    // Список clientId ожидающих матча

    this.clientIdsWaitingMatch = [];

    // Вызов методов настройки для Express и WebSocket

    this.setupExpress();

    this.setupWebSocket();

  }

  setupExpress() {

    this.app.use(express.static(path.join(\_\_dirname, "..", "client")));

    this.app.listen(3000);

    this.httpServer.listen(8080);

  }

  setupWebSocket() {

    this.wss.on("connection", this.handleConnection.bind(this));

  }

  // Метод обработки нового подключения к серверу WebSocket

  handleConnection(connection) {

    // Создание нового объекта игрока, передавая ему соединение и ссылку на текущий сервер

    const player = new Player(connection, this);

    // Добавление игрока в список клиентов с использованием clientId в качестве ключа

    this.clientConnections[player.clientId] = player;

    // Попытка сопоставить игроков для начала матча

    this.matchClients(player.clientId);

  }

  findGameIndex(clientId) {

    // Ищем игру, в которой участвует игрок с указанным clientId

    for (let i = 0; i < this.games.length; i++) {

      const game = this.games[i];

      if (game.currentPlayers.some((player) => player.clientId === clientId)) {

        return i; // Возвращаем индекс игры

      }

    }

    return -1; // Если игра не найдена

  }

  matchClients(clientId) {

    this.clientIdsWaitingMatch.push(clientId);

    if (this.clientIdsWaitingMatch.length < 2) {

      //console.log ("[SERVER]  Waiting match length: ", this.clientIdsWaitingMatch.length);

      //console.log("[SERVER] Choosing a host: ", this.clientConnections[clientId]);

      // Если данный ИГРОК подключился к игре первее, он будет хостом. Ставим флаг хоста true

      this.clientConnections[clientId].isHost = true;

      // Отсылаем интерфейсу информацию о хосте

      this.clientConnections[clientId].sendHostMessage();

      // Создаем игру

      this.games.push(new game(0));

      // Добавим игрока в список игроков в новой игре

      this.games[this.games.length - 1].currentPlayers.push(

        this.clientConnections[clientId]

      );

      // Назначим первому игроку номер игры

      this.clientConnections[clientId].gameId =

        this.games[this.games.length - 1].gameId;

      return;

    }

    // Извлекаем первого и второго клиентов из очереди ожидания

    const firstClientId = this.clientIdsWaitingMatch.shift();

    const secondClientId = this.clientIdsWaitingMatch.shift();

    // присваиваем номер игры

    this.clientConnections[secondClientId].gameId =

      this.clientConnections[firstClientId].gameId;

    // Добавим игрока в список игроков в игровой комнате

    this.games[this.games.length - 1].currentPlayers.push(

      this.clientConnections[secondClientId]

    );

    // Устанавливаем соответствие между первым и вторым клиентами

    this.opponents[firstClientId] = secondClientId;

    this.opponents[secondClientId] = firstClientId;

    // Отправляем сообщение о присоединении первому и второму клиентам

    this.clientConnections[firstClientId].sendJoinMessage("X");

    this.clientConnections[secondClientId].sendJoinMessage("O");

  }

  moveHandler(result, clientId) {

    // Получаем ID противника текущего игрока.

    const opponentClientId = this.opponents[clientId];

    // Находим индекс игры, в которой участвует текущий игрок.

    const gameIndex = this.findGameIndex(clientId);

    // Проверяем, выиграл ли текущий игрок.

    if (this.games[gameIndex].checkWin(result.field)) {

      // Если выиграл, отправляем сообщение о победе обоим игрокам.

      [clientId, opponentClientId].forEach((cid) => {

        this.clientConnections[cid].sendResultMessage(

          `${result.symbol} win`,

          result.field,

          result.size

        );

      });

    }

    // Проверяем, закончилась ли игра вничью.

    if (this.games[gameIndex].checkDraw(result.field)) {

      // Если ничья, отправляем сообщение о ничьей обоим игрокам.

      [clientId, opponentClientId].forEach((cid) => {

        this.clientConnections[cid].sendResultMessage("Draw", result.field);

      });

    }

    // Отправляем обновление игрового поля обоим игрокам.

    [clientId, opponentClientId].forEach((cid) => {

      this.clientConnections[cid].sendUpdateMessage(

        result.symbol === "X" ? "O" : "X", // Определяем следующий символ (ход другого игрока).

        result.field // Обновленное игровое поле.

      );

    });

  }

  closeClient(player) {

    // Если игрок ожидал начала матча, удаляем его из списка ожидающих.

    if (player.isWaitingMatch) {

      this.clientIdsWaitingMatch = this.clientIdsWaitingMatch.filter(

        (id) => id !== player.clientId

      );

    } else {

      // Если игрок был в игре, уведомляем его противника о выходе.

      const opponentClientId = this.opponents[player.clientId];

      if (this.clientConnections[opponentClientId]) {

        this.clientConnections[opponentClientId].sendExitMessage();

      }

    }

    // Закрываем соединение текущего игрока.

    player.connection.close();

    // Удаляем данные соединения текущего игрока из списка активных соединений.

    delete this.clientConnections[player.clientId];

  }

  handleReadyMessage(clientId, category) {

    const opponentClientId = this.opponents[clientId];

    // Отправляем сообщение оппоненту

    if (opponentClientId && this.clientConnections[opponentClientId]) {

      this.clientConnections[opponentClientId].sendHostReadyMessage(category);

    }

  }

  handleResizeMessage(clientId) {

    const opponentClientId = this.opponents[clientId];

    // Отправляем сообщение оппоненту

    if (opponentClientId && this.clientConnections[opponentClientId]) {

      this.clientConnections[opponentClientId].sendResizeMessage();

    }

  }

}

const ticTacToeServer = new TicTacToeServer();

Код самой игры сервера

// Класс игры "крестики-нолики"

class TicTacToeGame {

  static gameIdCounter = 0;

  constructor(size) {

    this.gameId = ++TicTacToeGame.gameIdCounter;

    this.size = size;

    this.winningCombos = this.generateWinningCombos(size);

    this.currentPlayers = [];

  }

  generateWinningCombos() {

    const winningCombos = [];

    const size = this.size;

    // Rows (Горизонтали)

    for (let i = 0; i < size; i++) {

      winningCombos.push(Array.from({ length: size }, (\_, j) => i \* size + j));

    }

    // Columns (Вертикали)

    for (let i = 0; i < size; i++) {

      winningCombos.push(Array.from({ length: size }, (\_, j) => i + j \* size));

    }

    // Diagonals (Диагонали)

    winningCombos.push(Array.from({ length: size }, (\_, i) => i \* (size + 1)));  // Главная диагональ

    winningCombos.push(Array.from({ length: size }, (\_, i) => (size - 1) \* (i + 1)));  // Побочная диагональ

    return winningCombos;

  }

  renewSize(size) {

    //console.log("[GAME] Got new size: ", size)

    this.size = size;

    this.winningCombos = this.generateWinningCombos();

  }

  checkWin(field) {

    const winningCombos = this.winningCombos;

    // Проверка по горизонтали, вертикали и диагонали

    const checkCombo = (combo) => {

      const symbols = combo.map(index => field[index]);

      return symbols.every(symbol => symbol !== "" && symbol === symbols[0]);

    };

    return winningCombos.some(combo => checkCombo(combo));

  }

  checkDraw(field) {

    // Проверяем, что все ячейки заполнены

    const isFieldFilled = field.every(symbol => symbol === "X" || symbol === "O");

    // Если поле заполнено и при этом нет победителя, то это ничья

    return isFieldFilled && !this.checkWin(field);

  }

}

module.exports = TicTacToeGame;

Код клиента

const WebSocket = require("ws");

// Класс игрока

class Player {

  static clientIdCounter = 0;

  constructor(connection, server) {

    this.connection = connection; // ссылка на экземпляр подключения WS

    this.server = server; // ссылка на экземпляр сервера

    this.clientId = ++Player.clientIdCounter; // айди игрока

    this.isWaitingMatch = true; // изначально игрок ждет старта матча

    this.isHost = false; // Станет true в особом случае на сервере

    this.gameId = null; // изначально у игрока нет gameId, он появится сразу после создания объекта игры

    this.connection.on("message", this.handleMessage.bind(this)); // при получении сообщения по WS, обрабатываем это

    this.connection.on("close", this.handleClose.bind(this)); // при получении close так же обрабатываем (WS)

  }

  // Обработчик получения сообщений

  handleMessage(message) {

    const result = JSON.parse(message);

    //console.log("[PLAYER] Got new message: ", result);

    const gameIndex = this.server.findGameIndex(this.clientId);

    // ход игрока

    if (result.method === "move") {

      this.server.moveHandler(result, this.clientId);

    }

    // изменение размера игрового поля

    if (result.method === "resize") {

      this.server.games[gameIndex].renewSize(result.size);

      this.server.handleResizeMessage(this.clientId, result.size);

    }

    // старт игры, хост готов

    if (result.method === "hostReady") {

      this.server.handleReadyMessage(this.clientId, result.category);

    }

  }

  // обработка закрытия, завершаем работу клиента (закрываем соединения)

  handleClose() {

    this.server.closeClient(this);

  }

  // отправка Join-сообщения, в котором мы сообщим о размере поля, и о том какой у нас символ

  sendJoinMessage(symbol) {

    const gameIndex = this.server.findGameIndex(this.clientId);

    this.connection.send(

      JSON.stringify({

        method: "join",

        symbol: symbol,

        turn: "X",

        size: this.server.games[gameIndex].size,

      })

    );

    this.isWaitingMatch = false;

  }

  // отправка сообщения с результатом

  sendResultMessage(message, field, size) {

    this.connection.send(

      JSON.stringify({

        method: "result",

        message: message,

        field: field,

        size: size,

      })

    );

  }

  // отправка сообщения о выходе

  sendExitMessage() {

    if (this.connection.readyState === WebSocket.OPEN) {

      this.connection.send(

        JSON.stringify({

          method: "left",

          message: "opponent left",

        })

      );

    }

  }

  // отправка сообщения о ходе

  sendUpdateMessage(turn, field) {

    this.connection.send(

      JSON.stringify({

        method: "update",

        turn: turn,

        field: field,

      })

    );

  }

  // отправка сообщения о том что текущий игрок - хост

  sendHostMessage() {

    if (this.isHost) {

      this.connection.send(

        JSON.stringify({

          method: "isHost",

        })

      );

    }

  }

  // отправка сообщения о готовности хоста

  sendHostReadyMessage(category) {

    this.connection.send(

      JSON.stringify({

        method: "hostReady",

        selectedCategory: category,

      })

    );

  }

  // отправка сообщения об изменении размера игрового поля

  sendResizeMessage() {

    const gameIndex = this.server.findGameIndex(this.clientId);

    this.connection.send(

      JSON.stringify({

        method: "resize",

        size: this.server.games[gameIndex].size,

      })

    );

  }

}

module.exports = Player;