1) Описание проекта

**Проект**: игра Морской бой, чат

**Платформа**: веб (<http://ea.wtf:4242>)

**Студенты:**

* Косякина Светлана
* Конова Ксения
* Сафронов Денис
* Ершов Андрей
* Егоров Евгений

Ключевой идеей было написание веб-приложения, которое могло бы обрабатывать запросы пользователей в режиме реального времени и работать со современным стандартом WebSocket.

Стек технологий:

**Front-end**:

* HTML5
* CSS3
* JavaScript (JQuery, Socket.IO)

**Back-end:**

* Node.js
* express (веб-сервер, аналог Django, Nginx, Apache)
* Socket.IO (библиотека для работы с WebSocket с клиента)

**Структура приложения:**

public/ - **файлы для клиента**

css/ - **файлы стилей**

img/ - **изображения**

js/ - **файлы JavaScript**

index.html – **основной HTML документ**

node\_modules/ - **библиотеки Node.js**

app/ - **логика сервера**

settings.js – **количество строк/столбцов и кораблей**

ship.js – **класс корабля**

player.js – **класс игрока**

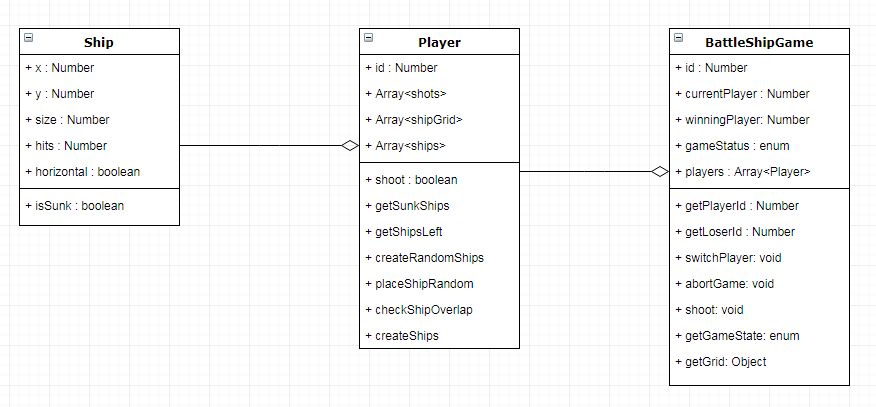
gameStatus.js – **enumeration статуса игры**

game.js – **класс игры**

server.js – **запуск сервера**

2) UML диаграммы

Т.к. проекты на JavaScript не всегда отличаются изобилием большого количества классов (в отличие от C++ или уж тем более Java), то код был написан больше в функциональном стиле с использованием событийно-ориентированной парадигмой.

****

Логика бэкэнда примерно такова.

Да, здесь везде плюсики (в JS нет “private”, “protected”, это лишь условности, но по факту private’a нет.)

3) Тесты для проверки

Тесты клиентской части, интерфейс

1) Клик во время хода соперника ничего не делает

2) Клик по полю, куда уже был произведён выстрел – ничего не делает

3) При промахе ход переходит сопернику

4) При попадании в корабль врага ход остаётся за игроком

5) При отправке сообщения оно появляется в списке

6) Сообщения имеют верного автора

Тесты серверной части, логика

7) При выстреле в координаты, куда уже был выстрел, действие не обрабатываются

8) При выстреле за пределы координат, действие не обрабатываются

9) При попытке выстрела в ход соперника, действие не будет обработано

10) Когда один игрок покидает игру, второму автоматически присваивается победа.

Для тестирования можно открыть две вкладки с приложением.

4) Алгоритм установки и запуска

Front-end часть работает на HTML/CSS, JavaScript для запуска которых потребуется любой браузер, но желательно использовать Google Chrome последней версии.

Для запуска сервера необходима платформа Node.js с пакетным менеджером npm. Скачать их можно по ссылке отсюда:

<https://nodejs.org/en/download/> далее настроить PATH и прочее, чтобы Node был доступен везде.

Теперь необходимо распаковать исходный код в нужную директорию. Далее необходимо установить необходимые пакеты (зависимости проекта), они уже прописаны в файле package.json

Сделать это можно командой: *npm i*

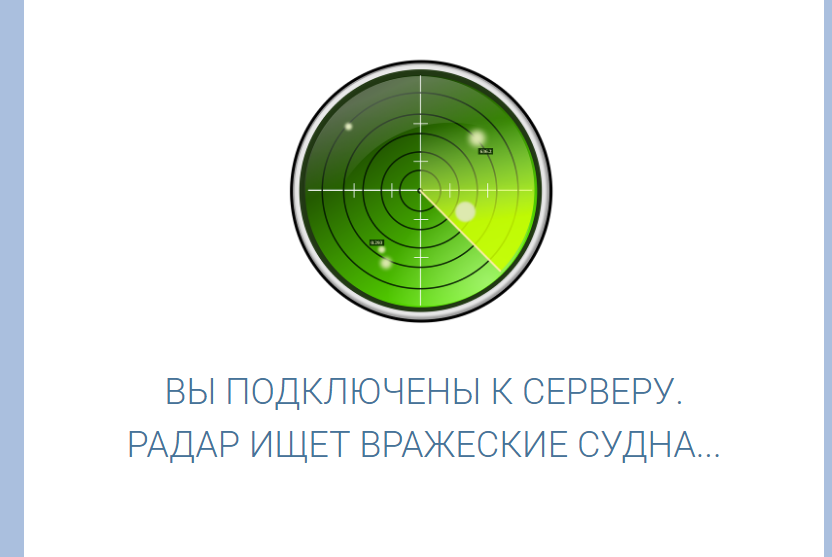
После того, как пакеты будут установлены можно запускать сервер командой: *node server.js*

В случае успеха вы увидите сообщение:

**Game started at localhost: {порт из настроек}**

Веб-сервер успешно запущен и теперь можно пройти в браузере по ссылке: [http://localhost:{порт}](http://localhost:%7bпорт%7d) и попасть в игру.

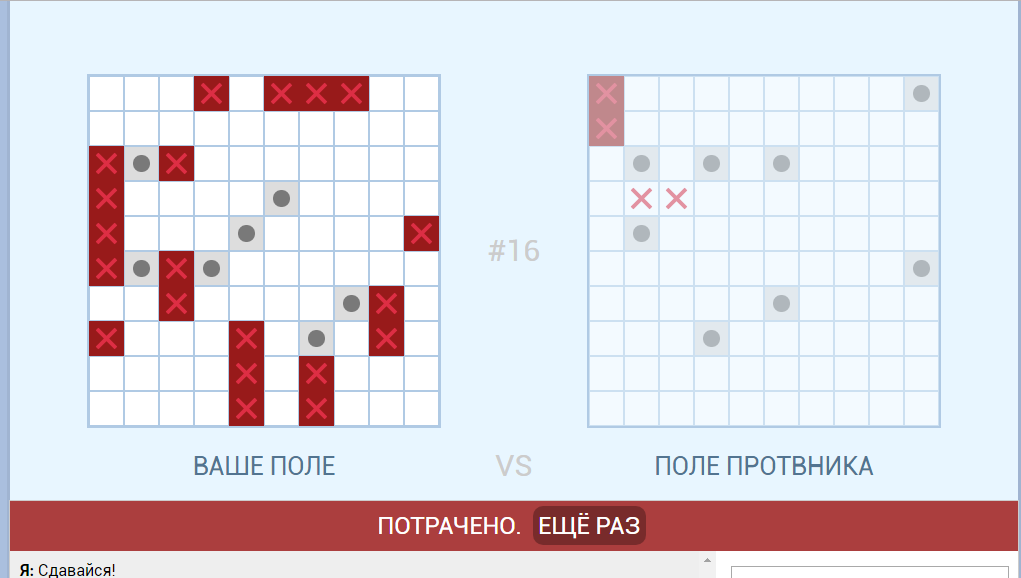
**Комната ожидания:**



**Победа:**



**Поражение:**



5) Планы на будущее

Из минусов – отсутствие возможности вручную выбирать позиции для кораблей для выбора стратегии игры, корабли могут располагаться вплотную друг к другу, чего в оригинальных правилах быть не должно.

Также можно добавить регистрацию и таблицу рекордов победителей.

Создавать сессии для пользователей, чтобы при перезагрузке страницы игрок не проигрывал, а продолжал играть.