**Министерство образования и науки Российской Федерации**

Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

---

Институт Кибернетики

Направление подготовки (специальность) Информационные системы и технологии

Кафедра Вычислительной техники

**ОТЧЕТ**

**по учебной практике**

**Игра «Точки»**

(Название работы)

Выполнил студент гр. 8И32 _____ Ветров А.А.
(подпись)

_____ _____ 20 ___ г.
(дата сдачи отчета)

Отчет принят:

Старший преподаватель _____ Лепустин А.В.
(подпись)

_____ _____ _____ 20 ___ г.
(оценка) (дата проверки отчета)

Учебную практику студент Ветров А.А. выполнил и защитил

с оценкой _____

Члены комиссии: _____

_____

_____

_____ _____ 20 ___ г.
(дата защиты)

Томск 2014 г.

## Оглавление

Стр.

## Введение

Игра «точки» появилась в 80-х годах из-за неправильной трактовки правил игры в «го» — широко известной и довольно сложной игры. Для появившейся игры четких правил создано не было, поэтому на данный момент существует большой количество вариаций «точек». В основу данной работы был взяты наиболее популярный вариант, правила которого описаны на большинстве тематических ресурсов.

Данная игра предназначена для игры двух человек на клетчатой бумаге, но, с появлением интернета почти в каждом доме, наиболее предпочтительным вариантом является игра по сети в интернет-браузере. На данный момент не существует нормальной кроссплатформенной релизаций игры «точки» в браузере без использования Flash. Это вызвано высокой сложностью алгоритмов, используемых в процессе обработки игры, поэтому разработка программного продукта, удовлетворяющая большинство пользователей актуальна в наши дни.

## 1. Цель работы

Целью данной работы является освоение современных технологий web-разработки, таких как язык разметки HTML5, язык описания стилей CSS3, языка программирования JavaScript, описаного в стандарте ECMAScript 5, реализация которого встроена в браузер, а также в веб-сервер Node.js.

## 2. Задачи

1. Изучение функциональных возможностей Javascript
2. Изучение Javascript-фреймворка AngularJS
3. Изучение веб-сервера Node.js
4. Проектирование схемы взаимодействия между клиентом и сервером
5. Проектирование и разработка серверной части приложения
6. Проектирование и разработка клиентской части приложения
7. Тестирование готового программного обеспечения

## 3. Задание

### 3.1 Описание игры

Данная игра проводится между двумя игроками на клетчатом поле с шириной 39 и высотой 32 клетки. На каждом шаге игрок должен поставить точку своего цвета на одно из свободных пересечений линий на поле (пункт). Пункт считается свободным в том случае, если он не занят другой точкой или не находится в зоне окружения. Программа должна образовывать зону окружения при возможности соединения точек одного цвета непрерывной замкнутой линией и наличии точек другого цвета в замыкаемой области, причем зона начинает считаться окружающей только в ход игрока, владеющего этой зоной. Точки возможно соединить линией, если они соседствуют на одной клетке (находятся в разных ее углах). Точки соперника, попавшие в зону окружения далее не участвуют в образовании новых зон окружения. На каждом ходе

программа должна подсчитывать в каждой замкнутой зоне одного игрока количество точек другого игрока и вывести суммарное количество в качестве очков этого игрока. Зоны, окруженные другой зоной в подсчете не участвуют. Игра заканчивается либо при отсутствии свободных пунктов, либо при взаимном согласии двух игроков. Побеждает игрок с наибольшим количеством очков.

### 3.2 Архитекура

Программа должна состоять из клиентской части, написанной на языке программирования JavaScript с использованием языка разметки HTML5, загружающейся в браузере пользователя, и серверной части, написанной на языке программирования JavaScript, запускаемой на сервере, и к которой присоединяется клиентская часть.

### 3.3 Клиентская часть

Клиентская часть должна быть представлена в виде web-приложения. При первом заходе в игру программа должна предоставить пользователю возможность зарегистрироваться на сайте, введя имя пользователя и пароль. После регистрации пользователю назначается начальный рейтинг 1600. Рейтинг характеризует опытность пользователя (чем больше – тем лучше) и пересчитывается по итогам каждой игры с участием этого пользователя. Далее программа должна позволять пользователю войти под своими данными в систему путем ввода в окне авторизации имени пользователя и пароля, указанного при регистрации.

После входа игрок попадает в режим ожидания соперника. В этом случае программа должна отобразить список игроков вместе с их рейтингом, также ожидающих игру. Игроки отсортированы по рейтингу в возрастающем порядке. Также программа должна позволять игроку применить фильтр к данному списку путем нажатия кнопки «фильтр» сверху списка пользователей. После нажатия кнопки программа должна отобразить панель, в котором можно выбрать способ фильтрования. Фильтровать позволяется либо по рейтингу

(вводится диапазон значений с помощью двух ползунков на одной линии, задающих минимальный и максимальный рейтинг), либо по имени. Чтобы применить фильтр, пользователь должен нажать на кнопку «Применить», чтобы сбросить фильтр, пользователь должен нажать на кнопку «Сброс». После того, как список удовлетворяющих соперников был сформирован, игрок должен иметь возможность выбрать понравившегося игрока, нажав на него в списке. После нажатия кнопки программа должна отобразить панель, содержащую надпись «ожидание ответа», кнопку, позволяющую отменить выбор и таймер обратного отсчета, имеющий начальное значение 15 секунд. При отказе соперника программа должна отобразить сообщение о том, что соперник отказался от игры. По окончании времени программа должна отобразить сообщение о том, что время ожидания истекло. У выбранного соперника программа должна вывести подтверждение, содержащее информацию об игроке, выразившем желание сразиться с ним, кнопку, с помощью которой он может начать игру, кнопку, с помощью которой он может отказаться от игры и аналогичный таймер обратного отсчета, как и у инициатора игры.

После начала игры программа должна обоим игрокам вывести одинаковое пустое поле, имя пользователя и счет первого игрока слева от поля, имя пользователя и счет второго игрока справа от поля, и состояние хода («ваш ход» или «ход соперника») сверху. Первым считается игрок, инициализировавший партию. Чтобы совершить ход, игрок должен нажать левой кнопкой мыши на пункт, который он хочет занять. После каждого хода программа должна отправлять ход на сервер и запросить у него новый счет и поле, на котором красным цветом выделяются точки и замкнутые линии первого игрока, а синим цветом точки и замкнутые линии второго игрока. В случае некорректности ход должен игнорироваться, а пользователю должна предоставиться возможность сделать ход заново. Также программа должна предоставить возможность либо предложить сопернику ничью, либо завершить игру с текущим счетом путем нажатия на специальные кнопки, расположенные под полем. После нажатия кнопки программа должна отобразить панель, содержащую надпись «ожидание ответа», кнопку, позволяющую отменить предложение и таймер обратного отсчета, имеющий начальное значение 15 секунд. При отказе соперника программа должна отобразить сообщение о том, что соперник отказался от предложения. По окончании времени программа должна отобра-

зить сообщение о том, что время ожидания истекло. У выбранного соперника программа должна вывести подтверждение, содержащее кнопку, с помощью которой он может принять предложение, кнопку, с помощью которой он может отказаться от предложения и аналогичный таймер обратного отсчета, как и у предложившего игрока. Также должна быть возможность выйти из игры без подтверждения, закрыв окно браузера либо нажав на специальную кнопку, тем самым признав поражение. После окончания игры пользователь возвращается на экран ожидания.

### 3.4 Серверная часть

Серверная часть должна обрабатывать регистрации пользователей и хранить в базе данных информацию о них (имя пользователя, пароль, рейтинг). Также она должна принимать ходы игроков, проверять их корректность, находить и обводить окружающие зоны, производить пересчет очков игроков, синхронизировать поля между игроками. Синхронизация должна выполняться с помощью pub/sub каналов: для каждой игры должен организовываться собственный канал, на который подписываются клиентские части обоих игроков, на данном канале серверная часть должна публиковать события, такие как ход игрока, предложения ничьи и завершение игры. Клиентская часть в реальном времени обрабатывает эти события и отображает на экране.

После окончания игры программа должна подсчитывать новый рейтинг игроков в зависимости от исхода игры по системе Эло. Сначала программа дожна вычислить вероятность выигрыша игрока $A$ против игрока $B$. Эта вероятность одновременно равна наиболее вероятному количеству очков, которое наберёт игрок $A$ в партии с $B$:

$$E_A = \frac{1}{1 + 10^{\frac{R_B - R_A}{400}}}$$

где:
$E_A$ — ожидаемое количество очков, которое наберёт игрок $A$ в партии с $B$,
$R_A$ — рейтинг игрока $A$,
$R_B$ — рейтинг игрока $B$.

Новый рейтинг игрока $A$ рассчитывается по формуле:

$$R_A = R_A + K * (S_A - E_A)$$

где:
$K$ — коэффициент, значение которого равно $10$ для сильнейших игроков (рейтинг $2400$ и выше), $15$ — для игроков с рейтингом меньшим чем $2400$ и $25$ — для игроков с рейтингом меньше чем $1700$,
$S_A$ — фактически набранное игроком $A$ количество очков ($1$ очко за победу, $0.5$ — за ничью и $0$ — за поражение),
$R_A$ – новый рейтинг игрока $A$.

# 4. Реализация программы

## 4.1 Серверная часть

Серверная часть является связующим элементом между клиентами. Он обрабатывает большое количество событий, происходящие до, во время и после игры, поэтому в качестве веб-сервера была выбрана программная платформа Node.js. В его основе лежит событийно-ориентированный и асинхронный подход к программированию с неблокирующим вводом/выводом, что позволяет одновременно обрабатывать множество запросов, не боясь того, что долгие вычисления одного запроса будут задерживать остальные.

### 4.1.1 Хранение данных

Используя асинхронный подход, программист не может хранить данные в глобальных переменных, т.к. в случае одновременного обращения к ней из двух точек программы могут возникнуть проблемы. Поэтому для хранения данных было использовано хранилище данных Redis, которое поддерживает атомарные неконкурирующие операции. Также оно обладает высокой скоростью доступа, т.к. в процессе работы все данные хранятся в оперативной памяти и периодически копируются на жесткий диск.

Redis является хранилищем вида «ключ-значение», где в качестве значения может быть строка, словарь вида «ключ-значение», множество неповторяющихся строк и множество строк, отсортированное по определенному значению. Ключи и их значения представлены в таблице А.1.

### 4.1.2 Клиент-серверное взаимодействие

Для обрабоки клиентских запросов было выбрано расширение для веб-сервера Express, позволяющее использовать методы протокола HTTP, такие как GET и POST, а также заголовки предаваемых запросов, такие как адрес запрашиваемой страницы, для описания запроса. На каждое описание Express позволяет назначать свой обработчик. Таким образом работа приложения разграничивается по функционалу. Все типы клиентских запросов и их обработики указаны в таблице А.4, данные, которые должен передать клиент в ходе запроса, указаны в таблице А.5, а данные и коды ошибок, которые возвращает сервер, указаны в таблице А.6.

Взаимодействуя с клиентом, серверу необходимо не только отвечать на поступающие запросы, но и отправлять некоторые данные клиенту. Для этого был выбрана модель «издатель-подписчик» (в дальнейшем — pub/sub).

В этой модели существуют «каналы», на которые клиент или сервер имеют право публиковать сообщения. Сервер также может фильтровать и оставлять только те сообщения, содержимое которых соответствует API клиент-серверного взаимодействия. Также клиенты могут изъявить желание получать все сообщения, публикуемые на определенном канале, для этого они подписываются на них. Данная модель обеспечивается расширением для веб-сервера Faye, скрывающее внутри себя особенности взаимодействия между подписчиками и издателями и предоставляющее для этого удобный интерфейс. За обработку публикуемых сообщений на сервере отвечает функция `incomingMessage`, вызываемая при каждом сообщении. Клиент должен к каждому сообщению присоединять объект `auth`, который позволяет идентифицировать пользователя. Структура данного объекта описана в таблице А.8. Сервер в свою очередь проверяет правильность авторизационных данных и удаляет данный объект из сообщения для того, чтобы другие пользователи не могли его увидеть. Если данный объект отсутствует или авторизационные данные не верны, сервер возвращает пользователю код ошибки «401::Authentication required» и не публикует его сообщение. В случае успешной идентификации к сообщению прикрепляется поле `id`, содержащее идентификационный номер пользователя. В дальнейшем сервер проверяет канал, на который было опубликовано сообщение. Если канал служебный и начинается с префикса «/meta», то сообщение пропускается без обработки, за исключением канала «/meta/disconnect». Сообщения на данный канал публикуются пользователем только при отключении от сервера, поэтому другим игрокам сообщается о выходе игрока с помощью сообщения типа «quit». Далее если сообщение публикуется на канал «/game/queue», его обработка передается функции `queueChannel`, а если канал подходит под регулярное выражение `/game/([a-f0-9]+)$`, то оно было отправлено на канал идущей в данный момент игры и его обработка передается функции `gameChannel`.

### 4.1.3 Канал «/game/queue»

На данном канале находятся игроки, находящиеся в режиме ожидания соперника. Подписавшись на данный канал, игроки получают актуальную информацию обо всех событиях, происходящих в очереди ожидания, таких как появление нового игрока, выход другого игрока, предложения о создании игры и само создание новой игры. Каждое событие сопровождается публикуемым

сообщением на данном канале. Подробная информация о каждом типе сообщений указана в таблице А.7. Обработчик `queueChannel` обрабатывает все эти сообщения и проверяет их корректность. Также он поддерживает правильность списка ожидающих игроков в хранилище данных, в частности, при сообщении типа «heartbeat» он записывает или обновляет POSIX время последнего появления игрока, отправившего это сообщение, в множество «queue» в хранилище данных. При сообщении типа «quit» он удаляет данного пользователя из множества «queue». При сообщении типа «request» он добавляет имя игрока, отправившего запрос, и его рейтинг, а также вызывает функцию `addRequest`, которая добавляет информацию о запросе в словарь «request:[id]», а также POSIX время запроса в множество «requests» хранилища данных для того, чтобы сборщик мусора удалял запросы с истекшим сроком годности. Также запросы удаляются при сообщении типа «decline» с помощью функции `cancelRequest`. При сообщении типа «accept» обработчик передает запрос в функцию `createGame`, которая проверяет, актуален ли запрос и может ли данный пользователь подтвердить данный запрос. Наконец, если запрос подтвержден успешно, сервер создает новую игру и приглашает в нее игроков с помощью сообщения типа «newgame».

### 4.1.4 Канал «/game/[channel]

На данном канале находятся игроки, играющие в игру с буквенно-цифровым идентификатором «[channel]»(канал игры). Канал игры формируется случайным образом при ее создании и состоит из 40 битного случайного числа, представленного в шестнадцетеричном формате. На данном канале публикуются все события, происходящие в данной игре,  На каждом сообщении, публикуемом на канал, обработчик указывает номер игрока в данной игре: «1» — если сообщение отправил первый игрок, «2» — если сообщение отправил второй игрок, «0» — если игрок, отправивший сообщение, не участвует в игре. Технически, если номер игрока равен «0», то он не может повлиять на игру, но может ее смотреть, но режим просмотра чужой игры отсутствует и не тестировался на реализованном клиенте. Также обработчик запоминает POSIX время игроков в полях «timestamp_1» и «timestamp_2» при получении сообщения типа «heartbeat». При сообщении типа «move» обработчик вызывает функцию `doMove`, которая обрабатывает ход и по итогам хода добавляет недостающие данные («score», «zones», «captured») в запрос. При

сообщении типа «request» он вызывает функцию `addRequest`, которая добавляет информацию о запросе в словарь «request:[id]», а также POSIX время запроса в множество «requests» хранилища данных для того, чтобы сборщик мусора удалял запросы с истекшим сроком годности. Также запросы удаляются при сообщении типа «decline» с помощью функции `cancelRequest`. При сообщении типа «accept» обработчик проверяет, актуален ли запрос и может ли данный пользователь подтвердить данный запрос, и в случае успеха вызывает функцию gameOver, которая останавливает игру с выбранным результатом (ничья или по текущему счету), рассчитывает новый рейтинг Эло игроков и отправляет сообщение типа «gameover».

### 4.1.5 Обработка хода

Обработка хода — самая трудоемкая задача сервера. Во первых, обработчик проверяет, является ли пункт, на который совершатся ход, захваченным или занятым другой точкой. Если пункт свободен, он занимается точкой цвета того игрока, который выполняет ход. Далее массив, содержащий все зоны окружения, очищается и начинается его заполнение заново. Зоны окружения создаются с помощью функции `findZones`. Для каждого цвета данная функция вызывается отдельно, сначала для цвета игрока, который выполнил ход, затем для цвета его противника. В начале функция `findZones` выполняет «закраску» поля из всех граничных точек. Для этого она вызывает функцию `fillArea` для каждой точки на границе поля. `fillArea` в свою очередь выполняет поиск в глубину из заданной точки, заходя в верхнего, нижнего, левого и правого соседа точки (не выходя за границы поля). Зайти в точку функция `fillArea` может в двух случаях:

1. Цвет точки (массив `field.color`) не равен цвету, по которому в данный момент строятся зоны окружения.
2. Состояние захвата (массив `field.captured`) такое, что точка захвачена противником.

Получившаяся в итоге закрашенная область будет так называемой «открытой» областью, на которой нет зон выбранного цвета. Если пройти по массиву и заменить закрашенную точку незакрашенной и наоборот (инвертировать область), то все зоны выбранного цвета станут закрашенными. Далее вызывается функция `findAreas` для нахождения отдельных зон в этой области. Она выполняет поиск точек сочленения с помощью функции `findCutpoints`, а

затем разделяет область с помощью найденых точек сочленения на зоны с помощью функции `splitAreas`. Дальше убираются те зоны, которые не содержат точек соперника. Для оставшихся функция `borderLine` строит замкнутые линии, ограничивающие данные области, а затем проверяет, чтобы данные линии не проходили через захваченные соперником точки. Для оставшихся после данных операций областей вызывается функция `updateCaptured`, которая «захватывает» каждые точки этих областей выбранным цветом. В конце концов оставшиеся зоны помещаются в массив `field.zones`. После того, как все зоны каждого цвета были найдены, функция `updateScores` обновляет текущий счет игры.

## 4.2 Клиентская часть

Клиентская часть представляет собой одностраничный HTML-документ. Он выполняется в браузере пользователя и является полностью статичным. Динамичным приложение делают скрипты, написанные на языке программирования JavaScript, который подключаются к этому HTML-документу, взаимодействует с сервером и изменяют содержимое страницы. На этапе проектирования был выбран шаблон проектирования MVC, аббревиатура которого расшифровывается как «модель-представление-контролер». Данный шаблон представляет собой цикл, в котором пользователь управляет контролером, контролер изменяет модель, модель влияет на представление, которое в конце концов видит пользователь. В качестве модели выступает сервер, хранящий и перерабатывающий данные, в качестве контролера выступает скрипт «controller.js» который взаимодействует с сервером, синхронизируя локальные переменные в соответствии с моделью, и с пользователем, а в качестве представления выступают шаблоны на языке HTML5, которые с помощью специальных атрибутов и HTML тегов (директив) самостоятельно обновляются в соответствии с локальными переменными. Каркасом для приложения, работающего по описанной архитектуре, является AngularJS — фреймворк, написаный на языке программирования JavaScript.

### 4.2.1 Корневой документ

Главным документом данного приложения является файл «index.html». В нем загружаются AngularJS и все остальные скрипты. Связующим элемен-

том для них является файл «app.js», в котором описывается структура приложения, а именно подключаемые модули. К данному приложению подключаются следующие модули:

- ngRoute — встроенный в AngularJS модуль, который позволяет использовать URL адрес для определения выводимого шаблона. Данный модуль позволяет эмулировать многостраничный сайт, используя всего один HTML документ.
- ngStorage — подключаемый сторонний модуль, который позволяет взаимодействовать с локальным хранилищем браузера для хранения своих данных. В данном приложении оно используется вместо cookies для хранения авторизационных данных. Эти данные описаны в таблице А.8.
- dots.controllers — модуль, в котором хранятся собственные контролеры, описанные в таблице А.11. Данный модуль хранится в файле «controllers.js».
- dots.directives — модуль, в котором хранятся собственные директивы. Данный модуль храниться в файле «directives.js»

Также в файле описываются правила маршрутизации. Эти правила привязывают к определенным URL адресам вызываемые контролеры и шаблоны, они также описаны в таблице А.11. Примечательно то, что, при переходе к другому адресу, пользователь не переходит на другой документ, а остается в корневом документе, так как URL адрес приписывается к «index.html» через символ «#». Для этого в «index.html» добавлен тег `<div ng-view></div>`, в который AngularJS загружает с помощью AJAX шаблон в соответствии с URL адресом.

### 4.2.2 Кореневой контролер

Каждый контролер имеет собственную область видимости и присоединяется к какому-либо HTML тегу. Но также возможно присоединить контролер к тегу, охватывающему все остальные, например к тегу `<body>`. Такой контролер называется корневым, а все остальные, подключенные к тегам внутри `<body>`, являются его потомками, которые наследуют область видимости корневого контролера. Таким образом в корневом контролере можно хранить данные, общие для всех страниц, например авторизационные данные.

В данном приложении корневым контролером является `appCtrl`. При запуске приложения данный контролер запрашивает данные из локального

хранилища. Структура этих данных описана в таблице А.13. Затем корневой контролер запускает клиент для сервера публикации/подписок Faye и настраивает его так, чтобы к каждому сообщению прикреплялись авторизационные данные. Также он настраивает обработчик ошибок при AJAX запросах: в случае ошибки «401::Unauthorized» пользователь должен переместиться на страницу входа.

### 4.2.3 Страница входа

При открытии приложения пользователь отправляется на страницу входа, на которой находится HTML форма, в которую вводятся имя пользователя и пароль уже зарегистрированного пользователя. Так как имя пользователя и пароль не должны быть пустыми, к полям ввода данных был применена встроенная в AngularJS директива `required`, означающая обязательность ввода. В случае отсутствия данных, пользователю отображается подсказка о необходимости ввести данные. После того, как пользователь нажал на кнопку «Войти», вызывается функция `$scope.login`, которая занимается обработкой формы. Она проверяет наличие введенных данных и отправляет эти данные на сервер. Если сервер успешно авторизировал пользователя, то он отправляет авторизационные данные, описанные в таблице А.8, которые контролер записывает в локальное хранилище. Затем пользователь отправляется в комнату ожидания соперника. Если сервер вернул код ошибки, контролер сообщает клиенту о неправильно введенных данных. Если пользователь еще не зарегистрирован, то он может нажать на ссылку «Зарегистрироваться» и перейти на страницу регистрации.

### 4.2.4 Страница регистрации

Если пользователь заходит в приложение в первый раз, ему необходимо пройти процесс регистрации. На странице регистрации также находится HTML форма, в которую вводятся желаемые имя пользователя и пароль регистрируемого пользователя. К обоим полям применена директива `required` , которая означает обязательность ввода данных в эти поля. Также к полю ввода пароля была применена директива `ng-minlength="6"`, которая означает минимальную длину вводимого пароля. При нажатии на кнопку «Зарегистрироваться» вызывается функция `$scope.register`, которая занимается обработкой формы. Она проверяет соответствие введенных данных требованиям и отправляет эти данные на сервер. Если данные не соответствуют требова-

ниям, пользователю отображается соответствующая подсказка. Если сервер успешно произвел регистрацию, пользователь уведомляется об этом и появляется ссылка, ведущая на страницу входа. Если сервер не смог зарегистрировать пользователя, значит пользователь с таким именем уже существует, о чем и уведомляется пользователь.

### 4.2.5 Комната ожидания соперника

При входе в комнату ожидания соперника, контролер проверяет наличие авторизационных данных в локальном хранилище и в случае неудачи отправляет пользователя на страницу входа. Далее контролер запрашивает у сервера информацию о других пользователях, и сохраняет ее в переменной `$scope.queue`, в соответствии с которой обновляется представление в виде HTML шаблона. Далее контролер подписывается на канал «/game/queue», на котором он следит за событиями, описаными в таблице А.9. Также запускается счетчик, который каждые 5 секунд публикует сообщения типа «heartbeat». При входе/выходе игроков соответственно обновляется очередь ожидающих игроков. Если пользователю предлагается начать новую игру, то отображается счетчик обратного отсчета на 15 секунд и кнопки, с помощью которых пользователь может начать или отменить новую игру. По истечении времени или в случае отмены предложения соперником игрок теряет возможность начать предложенную игру, о чем он уведомляется на одну секунду.

Если пользователь самостоятельно выбирает понравившегося пользователя, то ему отображается счетчик обратного отсчета на 15 секунд и кнопка, с помощью которой пользователь может отменить предложение. По истечении времени или в случае отказа противника игрок уведомляется об этом на одну секунду.

Если оба пользователя согласны с тем, чтобы начать новую игру, контролер отписывается от канала «/game/queue» и отправляется на страницу игры.

### 4.2.6 Счетчик обратного отсчета

Все счетчики обратного отсчета в игре ализованы с помощью созданой для этой цели директивы `<countdown>`, описаной в файле «directives.js». Данная директива также имеет свой шаблон, описаный в файле `countdown.html`. Данной директиве можно передать время, на которое будет включаться счетчик, переменные, за которыми она будет следить и в соответствии с которыми

она будет изменять свое состояние и представление. Также в данную директиву передаются функции, вызываемые при определенных событиях и сообщения, отображаемые вместе с этими событиями. Все параметры и их описания указаны в таблице А.12.

### 4.2.7 Игровой процесс

При входе на страницу игры, контролер проверяет наличие авторизационных данных в локальном хранилище и в случае неудачи отправляет пользователя на страницу входа. Далее контролер запрашивает у сервера информацию о текущем состоянии игры, а именно поле, которое сохраняет в объект `$scope.field`, структура которого описана в таблице А.2, идентификационные номера игроков, участвующих в игре, и текущий счет. В зависимости от идентификационных номеров игроков, участвующих в игре контролер определяет номер игрока, которым является пользователь и сохраняет этот номер в переменной `$scope.player`. Далее он выполняет функцию отрисовки поля `$scope.redrawField` и функцию отрисовки зон окружения `$scope.redrawZones`.

Игровое поле представляет из себя HTML тег `<canvas>`. Данный тег позволяет рисовать внутри себя геометрические примитивы, а также, как и любой другой тег, позволяет обрабатывать нажатия мышкой внутри поля. Но примитивы нельзя удалять по отдельности, поэтому поле отрисовывается каждый раз при изменении состояния игры заново. Также нельзя определить примитив, на которой пользователь нажал мышкой, известны только координаты относительно левого верхнего угла, поэтому определением пункта, на который было совершено нажатие, занимается функция `$scope.canvasClick`. Контролеру известно расстояние между линиями, на которых размещаются пункты (оно является постоянным и находится в переменных `$scope.ceilWidth` и `$scope.ceilHeight`), а также расстояние от начала координат до первой линии (оно является постоянным и находится в переменных `$scope.offsetX` и `$scope.offsetY`). Для того, чтобы определить координаты пункта, из координат нажатия мышки необходимо вычесть расстояние от начала координат до первой линии, а затем разделить на расстояние между линиями и округлить до ближайшего целого.

После того, как игровое поле было отрисовано, контролер подписывается на канал игры «/game/[channel]» и начинает следить за событиями, которые там происходят и которые описаны в таблице А.10. Затем запускается счетчик, который каждые 5 секунд публикует сообщения типа «heartbeat».

Контролер поддерживает номер игрока, который в данный момент должен совершать ход, который хранится в переменной `$scope.current_player` и в соответствии с которым обновляется надпись над игровым полем. Если номер игрока соответствует тому, кто должен совершать ход, пользователь может выбрать пункт, на который он будет ставить точку. При нажатии на пункт публикуется сообщение типа «move». Такое же сообщение может опубликовать и соперник, если он совершает ход. Контролер следит за этими сообщениями и обновляет состояние игры в соответствии с ними. Также контролер следит за предложениями о ничье или завершении игры и отображает счетчики обратного отсчета. Наконец, если на канале будет опубликовано сообщение типа «gameover», контролер перемещает игрока в комнату ожидания соперника.

## 5. Руководство пользователя

Чтобы открыть игру, вам необходимо открыть сайт «http://acm.tpu.ru/dots/». После этого вы будете перемещены на страницу входа.

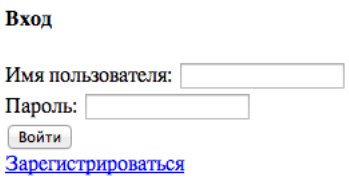


Рисунок 5.1 — Страница входа

Чтобы войти в игру, вы должны ввести имя пользователя и пароль. После этого нажмите кнопку «Войти». Если какое-либо поле не было заполнено, будет отображена ошибка.

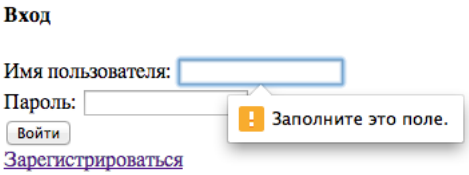


Рисунок 5.2 — Пустое поле

Если введенные имя пользователя и пароль оказались неверными, будет отображена ошибка.

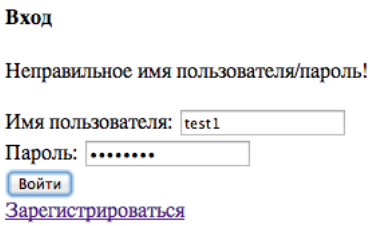


Рисунок 5.3 — Неправильное имя пользователя/пароль

Если вы заходите в игру в первый раз, вам необходимо пройти процедуру регистрации. Для этого нажмите кнопку «Зарегистрироваться» на странице входа. После этого вы будете перемещены на страницу регистрации.

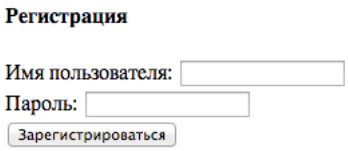


Рисунок 5.4 — Страница входа

На странице регистрации вам необходимо ввести желаемые имя пользователя и пароль и нажать на кнопку «Зарегистрироваться». Если какое-либо поле не было заполнено, будет отображена ошибка рядом с незаполненным полем. Также пароль должен содержать не менее 6 символов. Если введенный пароль менее 6 символов, будет отображена ошибка.

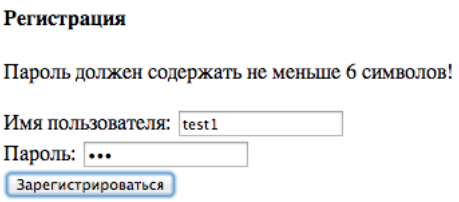


Рисунок 5.5 — Пароль содержит менее 6 символов

Если введенное имя пользователя уже существует, будет отображена ошибка.

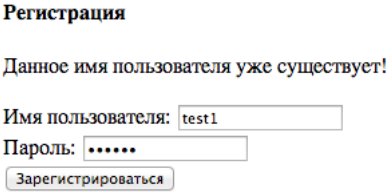


Рисунок 5.6 — Страница входа

Если регистрация пройдет успешно, вам будет отображено сообщение.

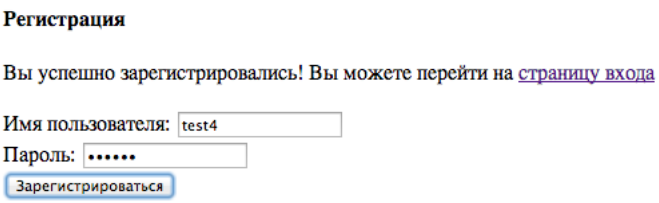


Рисунок 5.7 — Успешная регистрация

После этого вы можете перейти по ссылке на страницу входа и ввести свое имя пользователя и пароль. Если вход в систему будет произведен успешно, вы переместитесь в комнату игроков, ожидающих соперника.

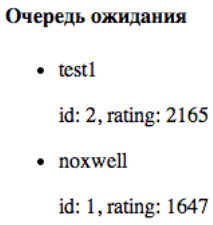


Рисунок 5.8 — Комната ожидания

В этой комнате вы можете выбрать из списка игроков соперника, с которым вы хотите начать новую игру. После того, как вы определились с выбором, вы можете нажать на имя пользователя соперника. После этого вы увидите счетчик обратного отсчета.

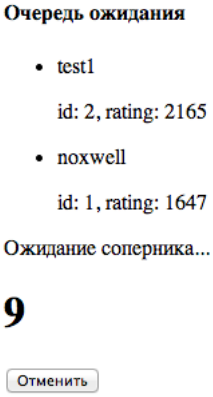


Рисунок 5.9 — Ожидание соперника

В это время у соперника также появится счетчик обратного отсчета.

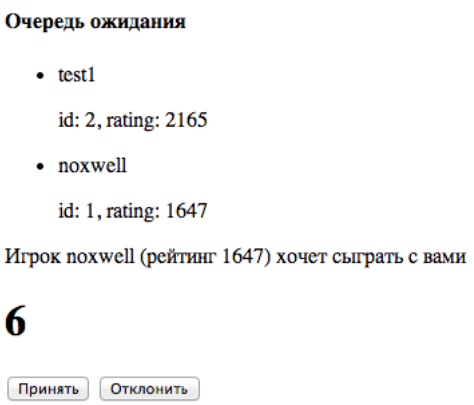


Рисунок 5.10 — Игрок хочет сыграть с вами

В течение 15 секунд вы можете отменить свое решение начать новую игру с этим соперником. Для этого вы должны нажать на кнопку «Отменить». В этом случае сопернику будет отображено сообщение.

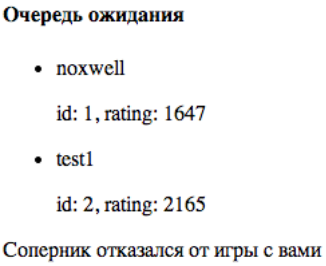


Рисунок 5.11 — Соперник отказался от игры с вами

Также и соперник может отказаться от игры с вами, нажав кнопку «Отклонить». В этом случае вам будет отображено такое же сообщение, как на рисунке 5.11. Если ни вы, ни соперник не примете решение в течение 15 секунд, вам обоим отобразится сообщение.

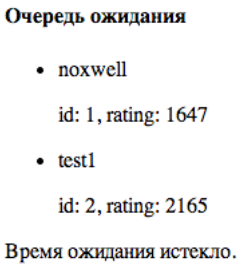


Рисунок 5.12 — Время ожидания истекло

Если соперник согласится начать с вами новую игру нажав кнопку «Принять», то вы оба перейдете на страницу игры.

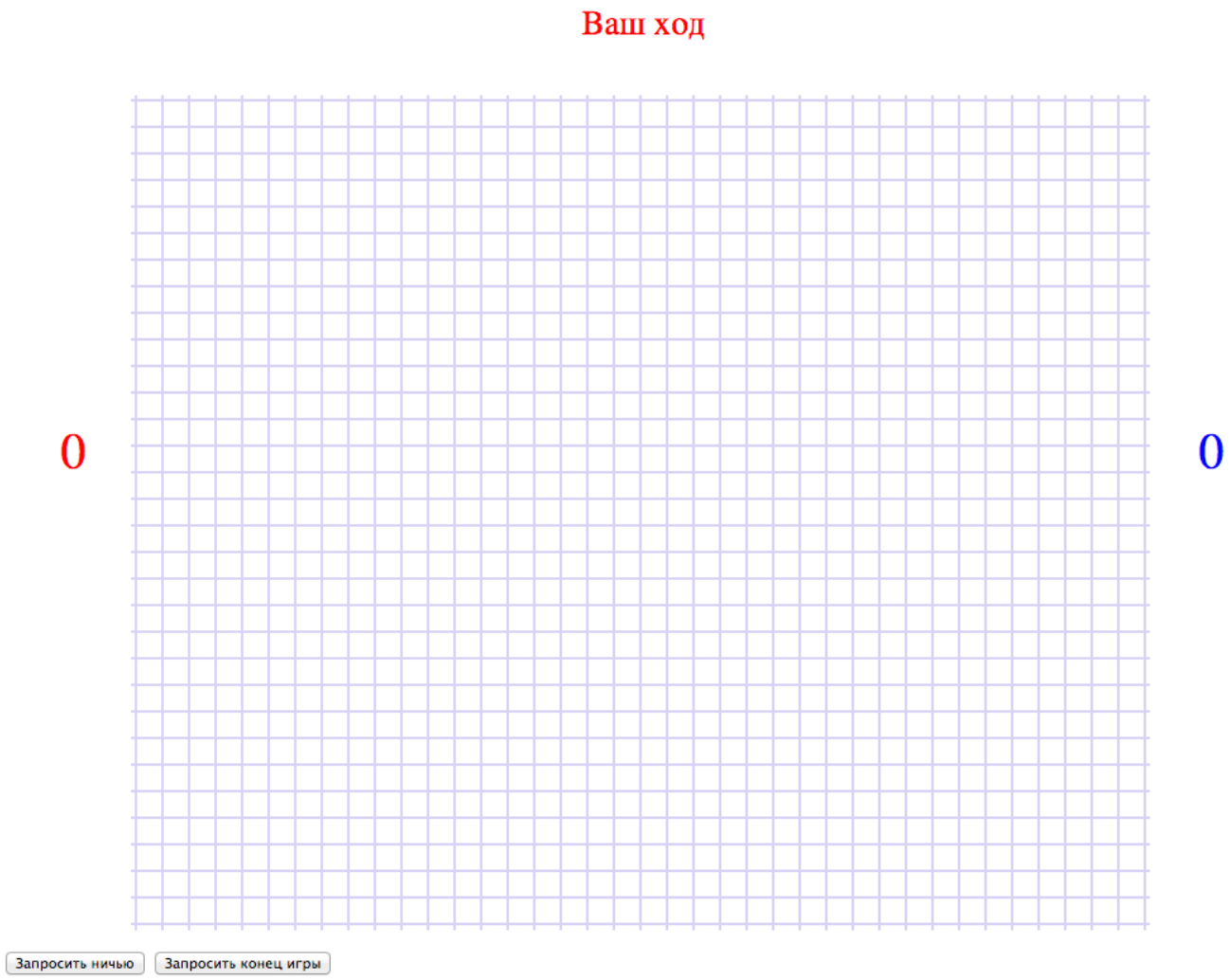


Рисунок 5.13 — Игровой экран

Посередине экрана располагается игровое поле. Слева располагаются набранные очки первого игрока, справа — очки второго игрока. Игрок, предложивший начать игру ходит первым. Если в данный момент вы должны ходить, над игровым полем будет отображено сообщение «Ваш ход», как на рисунке 5.13. Если в данный момент ходить должен соперник, вы не можете ходить и вам будте отображено сообщение «Ход соперника».


Рисунок 5.14 — Ход соперника

Ход осуществляется путем нажатия левой кнопкой мыши на пересечение вертикальных и горизонтальных линий на игровом поле (пункт). После того, как ход был совершен, на поле появляется точка определенного цвета, для точек первого игрока — красный, для точек второго — синий. Если после совершения хода точки вашего цвета можно соединить непрерывной замкнутой линией, а также в замыкаемой области находятся точки соперника, данная область окружается, а вам присваиваются очки за каждую точку соперника в этой области.

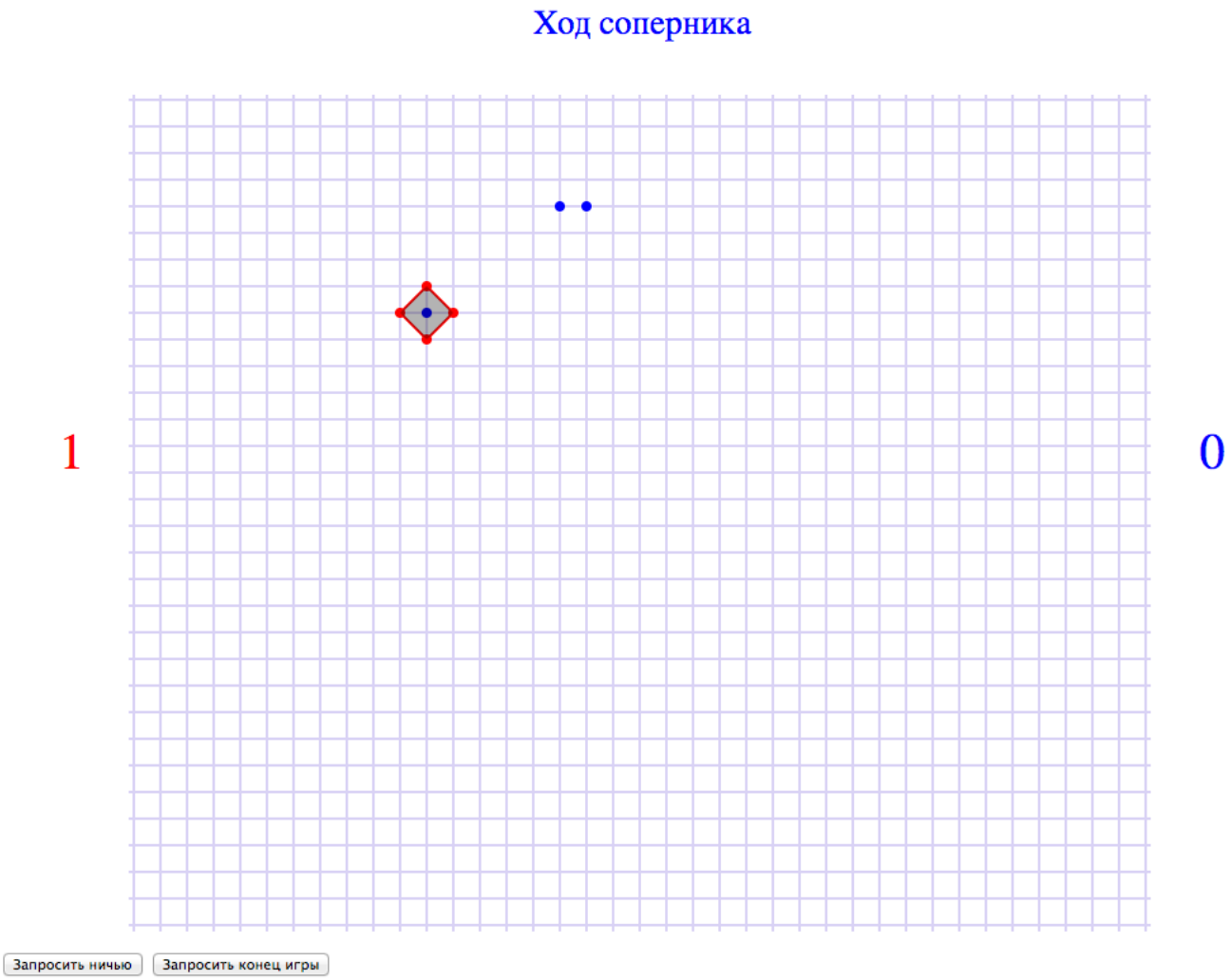


Рисунок 5.15 — Зона окружения

Вы можете попросить соперника завершить игру вничью, нажав кнопку «Запросить ничью», или с текущим счетом, нажав на кнопку «Запросить конец игры». После этого у вас запустится счетчик обратного отсчета.

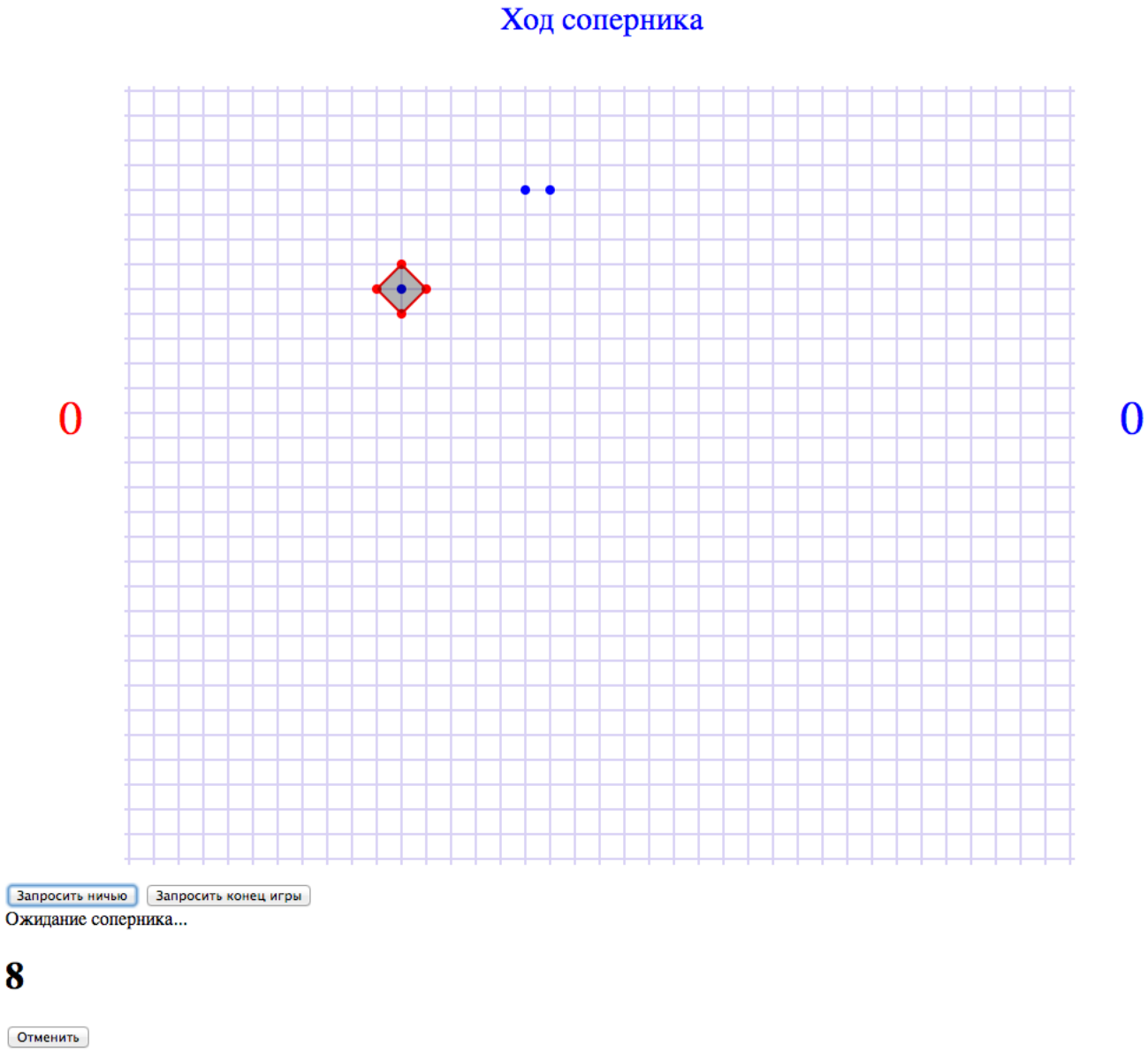


Рисунок 5.16 — Ожидание соперника

Аналогично у соперника появится счетчик обратного отсчета.

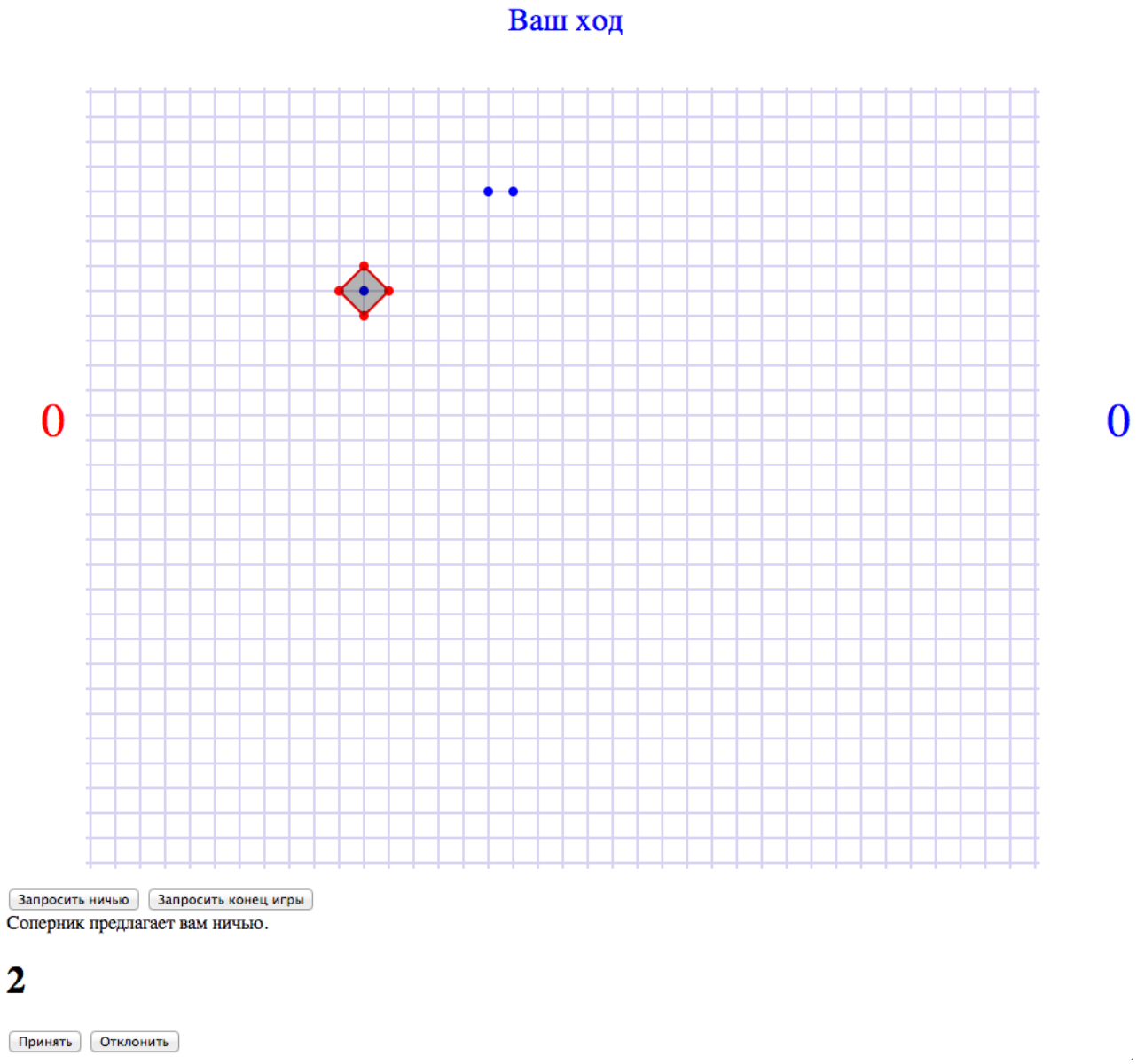


Рисунок 5.17 — Соперник предлагает завершить игру

В течение 15 секунд вы или соперник должны сделать выбор. Счетчик работает аналогично тому, который был в комнате ожидания, с помощью кнопок под счетчиком вы можете отменить запрос, аналогично соперник может отклонить его, о чем вам будет сообщено. Если соперник согласиться завершить игру, игра завершается, победитель награждается повышением рейтинга, у проигравшего рейтинг понижается. В случае ничьи рейтинг увеличивается у того, у кого изначально рейтинг был меньше. Рейтинг изменяется по особой формуле. После завершения игры вы возвращаетесь на страницу ожидания соперника.

## Заключение

В ходе проделанной работы были освоены современные технологии web-разработки, такие как язык разметки HTML5, язык описания стилей CSS3, язык программирования JavaScript, а также в веб-сервер Node.js.

Было изучено асинхронное программирование, а также шаблон проектирования «модель-представление-контролер» и применен на реальной задаче.

Была изучена событийно-ориентированная парадигма программирования, являющаяся основной в языке JavaScript. Данный подход позволяет строить высоконагруженные асинхронные приложения, абстрагируясь от тех действий, которые вызывают эти события. Также разработанное приложения является отличным примером легко поддерживаемой расширяемой модульной архитектуры, так как модули работают практически независимо друг от друга и изменение одного не влияет на другие.

Данные технологии являются актуальными и широко используются в настоящее время, постепенно оттесняя другие, бывшие когда-то стандартом технологии, например язык программирования PHP.

## Приложение А

Таблица А.1 — Структура хранилища данных

| Ключ | Тип | Значение |
|---|---|---|
| user:[id] | Словарь | Содержит данные об игроках. Вместо [id] необходимо указывать идентицикационный номер пользователя. |
| user:[id]->name | Строка | Имя пользователя. |
| user:[id]->password | Строка | Пароль в открытом виде. |
| user:[id]->token | Строка | Авторизационный токен, который клиент должен передавать в каждом запросе для подтверждения авторизации. Выдается клиенту при вводе логина и пароля в окне авторизации. |
| user:[id]->rating | Строка | Рейтинг пользователя. |
| last_userid | Строка | Идентицикационный номер пользователя, использованный при регистрации последнего пользователя. Используется для обеспечения уникальности идентификационных номеров. Начальное значение — «0» |
| users | Словарь | Имена пользователей, сопоставленные с их идентификационными номерами. |

Продолжение таблицы А.1

| Ключ | Тип | Значение |
|---|---|---|
| queue | Упорядоченное множество | Идентификационные номера пользователей, находящихся в режиме ожидания соперника. Номера упорядочены по времени последнего появления, представленного в виде количества секунд с 1 января 1970 года (POSIX время). |
| requests | Упорядоченное множество | Идентификационные номера пользователей, требующих некоторого действия от другого пользователя (запрос), упорядоченные по времени создания запроса. |
| request:[id] | Словарь | Содержит данные о запросе. Вместо [id] необходимо указывать идентицикационный номер пользователя. |
| request:[id]->type | Строка | Тип запрашиваемого действия. Может принимать значения: «newgame» — создание новой игры, «draw» — закончить игру в ничью, «surrender» — закончить игру с текущим счетом. |

Продолжение таблицы А.1

| Ключ | Тип | Значение |
|---|---|---|
| request:[id]->target | Строка | Идентификационный номер пользователя, от которого требуется принять решение — выполнить предлагаемый запрос или нет. |
| request:[id]->gameid | Строка | Идентификационный номер игры, в рамках которой должен выполнится запрос. Если тип запроса равен «newgame», то значением данной строки будет являться «0». |
| games | Словарь | Буквенно-цифровой идентификатор игры (канал), сопоставленный с идентификационным номером игры. |
| last_gameid | Строка | Идентификационный номер последней созданной игры. Используется для обеспечения уникальности идентификационных номеров. Начальное значение — «0». |
| game:[id] | Словарь | Содержит данные об игре. Вместо [id] необходимо указывать идентификационный номер игры. |
| game:[id]->channel | Строка | Канал игры |

Продолжение таблицы А.1

| Ключ | Тип | Значение |
|---|---|---|
| game:[id]->field | Строка | Состояние поля в текущий момент. Храниться в виде объекта, представленного в JSON-нотации. Подробное описание полей этого объекта можно узнать в таблице А.2 |
| game:[id]->player_1 | Строка | Идентицикационный номер первого игрока. |
| game:[id]->score_1 | Строка | Количество очков, набранное первым игроком на текущий момент. |
| game:[id]->timestamp_1 | Строка | Время последнего появления первого игрока в формате POSIX. |
| game:[id]->player_2 | Строка | Идентицикационный номер второго игрока. |
| game:[id]->score_2 | Строка | Количество очков, набранное вторым игроком на текущий момент. |
| game:[id]->timestamp_2 | Строка | Время последнего появления второго игрока в формате POSIX. |
| game:[id]->current_player | Строка | Номер игрока, который совершает ход в данный момент времени. «1» — если в данный момент времени ходит первый игрок, «2» — если в данный момент времени ходит второй игрок. |

Таблица А.2 — Структура объекта «game[id]:field»

| Поле | Значение |
|---|---|
| width | Ширина поля. Значение по умолчанию — 39 |
| height | Высота поля. Значение по умолчанию — 32 |
| color | Двумерный массив, содержащий цвет каждого пункта на поле. Цвет равен «0» если данный пункт не содержит точки, «1» — если точка на пункте принадлежит первому игроку, «2» — если точка на пункте принадлежит второму игроку. |
| captured | Двумерный массив, содержащий состояние захвата каждого пункта на поле. Состояние равно «0» если пункт не находится в зоне окружения, «1» — если пункт находится в зоне окружения (включая границы) перового игрока, «2» — если пункт находится в зоне окружения (включая границы) второго игрока. |
| zones | Массив, содержащий все зоны окружения существующие на поле в данный момент времени. Каждый элемент массива также является массивом, содержащим пункты, находящие на границе выбранной зоны окружения. Пункт является объектом, поля которого описаны в таблице А.3 |

Таблица А.3 — Структура объекта, содержащего координаты пункта

| Поле | Значение |
|---|---|
| x | Номер горизонтальной линии, на которой располагается пункт. Линии нумеруются сверху-вниз, начиная с нуля. |
| y | Номер вертикальной линии, на которой располагается пункт. Линии нумеруются слева-направо, начиная с нуля. |

Таблица А.4 — HTTP запросы

| URL | Метод | Обработчик | Описание |
|---|---|---|---|
| /auth | POST | onAuth | Авторизация в системе. В ответ клиент получает уникальные авторизационные данные, которые ему необходимо передавать в каждых последующих запросах для подтверждение личности. |
| /register | POST | onRegister | Регистрации нового пользователя в системе. |
| /queue | GET | getQueue | Получение списка игроков, находящихся в режиме ожидания соперника. |
| /gamedata | GET | getGameData | Получение текущего состояния игры. |

Таблица А.5 — Передаваемые данные HTTP запросов

| URL | Метод | Передаваемые параметры | Значение |
|---|---|---|---|
| /auth | POST | name | Имя авторизуемого пользователя |
| | | password | Пароль |
| /register | POST | name | Имя регистрируемого пользователя |
| | | password | Пароль |
| /queue | GET | id | Идентификационный номер пользователя |
| | | token | Авторизационный токен |
| /gamedata | GET | id | Идентификационный номер пользователя |
| | | token | Авторизационный токен |
| | | channel | Канал запрашиваемой игры |

Таблица А.6 — Получаемые данные HTTP запросов

| URL | Получаемые данные | Значение |
|---|---|---|
| /auth | id | Идентификационный номер авторизированного пользователя |
| | token | Авторизационный токен |
| | HTTP статус | 200 в случае успешной авторизации, 401 в случае неправильных авторизационных данных. |
| /register | HTTP статус | 200 в случае успешной регистрации, 406 если уже существует пользователь с регистрируемым именем пользователя. |
| /queue | queue | Массив в формате JSON, содержащий очередь игроков, ожидающих соперника. Каждый элемент массива является объектом, структура которого описана в таблице А.7 |
| | HTTP статус | 200 в случае успешного получения данных, 401 в случае неавторизированного доступа. |
| /gamedata | game | JSON объект, содержащий текущее состояние игры. Данный объект является словарем «game:[id]», структура которого описана в таблице А.1. |
| | HTTP статус | 200 в случае успешного получения данных, 401 в случае неавторизированного доступа. |

Таблица А.7 — Структура объекта, хранящего данные о пользователе

| Поле | Значение |
|---|---|
| id | Идентификационный номер пользователя. |
| name | Имя пользователя. |
| rating | Рейтинг пользователя. |

Таблица А.8 — Структура объекта, хранящего авторизационные данные

| Поле | Значение |
|---|---|
| id | Идентификационный номер пользователя. |
| token | Авторизационный токен, выданный клиенту при авторизации. |

Таблица А.9 — Сообщения, публикуемые на канале «/game/queue»

| Тип | Параметры | Описание |
|---|---|---|
| heartbeat | type | Данный тип сообщений предназначен для поддержания актуальности списка игроков, ожидающих соперника. Игрок исключается из этого списка, если в течении 10 секунд он не публиковал ни одного сообщения данного типа. |
| | id | Идентификационный номер пользователя, отправившего сообщение. Данное поле добавляется сервером. |
| | name | Имя пользователя, отправившего сообщение. Данное поле добавляется сервером. |
| | rating | Рейтинг пользователя, отправившего сообщение. Данное поле добавляется сервером. |
| quit | type | Данный тип сообщений публикуется игроком в том случае, если он добровольно исключает себя из списка игроков, ожидающих соперника, либо сервером, если игрок принудительно исключается из этого списка. Игрок может уже отсутствовать в списке, т.к. игрок может отправить это сообщение одновременно с сервером. |
| | id | Идентификационный номер выходящего пользователя. Данное поле добавляется сервером. |

Продолжение таблицы А.9

| Тип | Параметры | Описание |
|---|---|---|
| request | type | Данный тип сообщений отправляется игроком в том случае, если он предлагает другому игроку начать новую игру. |
| | id | Идентификационный номер пользователя, отправившего предложение. Данное поле добавляется сервером. |
| | user.name | Имя пользователя, отправившего предложение. Данное поле добавляется сервером. |
| | user.rating | Рейтинг пользователя, отправившего предложение. Данное поле добавляется сервером. |
| | target | Идентификационный номер пользователя, которому предлагается начать новую игру. |
| accept | type | Данный тип сообщения отправляется игроком при согласии с предложением начать новую игру. |
| | id | Идентификационный номер согласившегося пользователя. Данное поле добавляется сервером. |
| | target | Идентификационный номер пользователя, предложение которого одобряет согласившийся игрок. |
| decline | type | Данный тип сообщения отправляется игроком для отмены предложения. Отменить предложение может либо игрок, отправивший его, либо игрок, принимающий его. |
| | id | Идентификационный номер игрока, отменяющего предложение. Должен совпадать либо с полем «init», либо с полем «target». Данное поле добавляется сервером. |
| | init | Идентификационный номер игрока, изначально отправившего предложение. |
| | target | Идентификационный номер игрока, который должен принять предложение. |

Продолжение таблицы А.9

| Тип | Параметры | Описание |
|---|---|---|
| newgame | type | Данный тип сообщения обозначает начало новой игры. Его имеет право отправить только сервер. |
| | id | Идентификационный номер пользователя, отправившего сообщение. В данном случае он может принимать единственное значение «0», обозначающее сервер игры. |
| | player_1 | Первый игрок, участвующий в новой игре. |
| | player_2 | Второй игрок, участвующий в новой игре. |
| | channel | Канал игры, на который приглашаются игроки. |

Таблица А.10 — Сообщения, публикуемые на канале «/game/[channel]»

| Тип | Параметры | Описание |
|---|---|---|
| heartbeat | type | Данный тип сообщений предназначен для проверки наличия игроков в игре. Если игрок не отправлял сообщения данного типа в течение 10 секунд, игрок считается выбывшим. |
| | id | Идентификационный номер пользователя, отправившего сообщение. Данное поле добавляется сервером. |
| | player | Номер игрока, отправившего сообщение, «1» — если первый игрок, «2» — если второй игрок, «0» — если игрок не участвует в игре. Данное поле добавляется сервером. |

Продолжение таблицы А.10

| Тип | Параметры | Описание |
|---|---|---|
| move | type | Данный тип сообщения отправляется игроком в том случае, если он совершает ход. |
| | id | Идентификационный номер пользователя, совершившего ход. Данное поле добавляется сервером. |
| | player | Номер игрока, совершившего ход. Данный номер обязательно должен совпадать с игроком, который в данный момент должен ходить. Данное поле добавляется сервером. |
| | point | Объект, содержащий пункт, на который сходил игрок. Структура объекта описана в таблице А.3. |
| | score | Массив, содержащий количество очков у каждого игрока. Элемент с индексом [0] всегда равен «0», с индексом [1] равен количеству очков первого игрока, с индексом [2] — количество очком второго игрока. Данное поле добавляется сервером. |
| | zones | Массив, содержащий все зоны окружения существующие на поле в данный момент времени. Является полной копией поля «zones», содержащегося в хранилище данных объекта «game[id]:field», описанного в таблице А.2. Данное поле добавляется сервером. |
| | captured | Двумерный массив, содержащий состояние захвата каждого пункта на поле. Является полной копией поля «captured», содержащегося в хранилище данных объекта «game[id]:field», описанного в таблице А.2. Данное поле добавляется сервером. |

Продолжение таблицы А.10

| Тип | Параметры | Описание |
|---|---|---|
| request | type | Данный тип сообщений отправляется игроком в том случае, если он предлагает другому игроку закончить игру. |
| | id | Идентификационный номер пользователя, отправившего предложение. Данное поле добавляется сервером. |
| | player | Номер игрока, отправившего предложение. Не может принимать значение «0», т.к. закончить игру может только участник этой игры. Данное поле добавляется сервером. |
| | requestType | Способ, по которому в случае окончания игры будет выбираться победитель. «draw» — если игру предлагается закончить в ничью, «surrender» — победитель будет выбран согласно текущему счету. |
| accept | type | Данный тип сообщения отправляется игроком при согласии с предложением закончить игру. |
| | id | Идентификационный номер согласившегося игрока. Данное поле добавляется сервером. |
| | player | Номер согласившегося игрока. Не может принимать значение «0», т.к. закончить игру может только участник этой игры. Данное поле добавляется сервером. |
| decline | type | Данный тип сообщения отправляется игроком для отмены предложения. |
| | id | Идентификатор игрока, отменяющего предложение. Данное поле добавляется сервером. |
| | player | Номер игрока, отменяющего предложение. Не может принимать значение «0», т.к. закончить игру может только участник этой игры. Данное поле добавляется сервером. |

Продолжение таблицы А.10

| Тип | Параметры | Описание |
|---|---|---|
| gameover | type | Данный тип сообщения обозначает конец игры. Его имеет право отправить только сервер. |
| | id | Идентификационный номер пользователя, отправившего сообщение. В данном случае он может принимать единственное значение «0», обозначающее сервер игры. |
| | score_1 | Количество очков, набранное первым игроком. |
| | score_2 | Количество очков, набранное вторым игроком. |
| | winner | Результат игры. «1» — если победил первый игрок, «2» — если победил второй игрок, «0» — если ничья. |

Таблица А.11 — Контролеры

| Название | Сопоставляемый URL | Шаблон | Описание |
|---|---|---|---|
| appCtrl | / | index.html | Корневой контроллер. |
| loginFormCtrl | /login | login.html | Контроллер формы входа. |
| registrationFormCtrl | /register | register.html | Контроллер формы регистрации. |
| waitingRoomCtrl | /queue | queue.html | Контроллер комнаты ожидания соперника. |
| gameScreenCtrl | /game/[channel] | game.html | Контроллер игрового процесса. |

Таблица А.12 — Параметры к директиве обратного отсчета

| Поле | Описание |
|---|---|
| ng-show | Внешняя логическая переменная, которая «истинна», когда нужно включить и показать счетчик обратного отсчета и «ложна», когда его нужно выключить и скрыть. Если счетчик останавливается с помощью данной переменной, то никаких других событий не вызывается. |
| time | Параметр, характеризующий время, на которое включается счетчик. |
| on-accept | Функция, вызываемая при нажатии кнопки «Принять». Необязательный параметр, если присутствует параметр «on-cancel». |
| on-decline | Функция, вызываемая при нажатии кнопки «Отклонить». Необязательный параметр, если присутствует параметр «on-cancel». |
| on-cancel | Функция, вызываемая при нажатии кнопки «Отменить». Необязательный параметр, если присутствует параметр «on-accept» и «on-decline». |
| is-failure | Внешняя логическая переменная, которая в состоянии «истинно» выключает счетчик и показывает сообщение «failure-message». |
| failure-message | Сообщение, показываемое при выключении счетчика с помощью переменной «is-failure». |
| timeout | Сообщение, показываемое при выключении счетчика по окончании времени. |

Таблица А.13 — Структура локального хранилища

| Поле | Описание |
|---|---|
| loggedIn | Логическая переменная, показывающее состояние авторизации. Если переменная «истинна», то пользователь авторизирован в системе и имеет право заходить на любые страницы. Если переменна «ложна», то пользователь допускается только к страницам входа и регистрации. |

Продолжение таблицы А.13

| Поле | Описание |
|---|---|
| user | Объект, хранящий данные об авторизированном пользователе. Структура этого объекта описана в таблице А.7. |
| auth | Объект, хранящий авторизационные данные. Структура этого объекта описана в таблице А.8. |

## Приложение В

### Исходный код В.1 — index.html

```
<!DOCTYPE html>
<html ng-app="dots">
<head>
  <meta charset="utf-8">
  <title>Точки</title>
  <link href="css/style.css" rel="stylesheet">
  <script type="text/javascript" src="lib/jquery.min.js"></script>
  <script type="text/javascript" src="lib/faye-browser-min.js"></script>
</head>
<body ng-controller="appCtrl">
  <div ng-view></div>
  <script type="text/javascript" src="lib/angular.js"></script>
  <script type="text/javascript" src="lib/angular-route.js"></script>
  <script type="text/javascript" src="lib/ngStorage.min.js"></script>
  <script type="text/javascript" src="js/app.js"></script>
  <script type="text/javascript" src="js/controllers.js"></script>
  <script type="text/javascript" src="js/directives.js"></script>
</body>
</html>
```

### Исходный код В.2 — partials/login.html

```
<h4>Вход</h4>
<form name="loginForm">
  <span ng-show="failure" id="login_failure">
    Неправильное имя пользователя/пароль!
    <br>
    <br>
  </span>
  Имя пользователя: <input type="text" ng-model="name" name="name" required>
  <br>
  Пароль: <input type="password" ng-model="password" name="password" required>
  <br>
  <button ng-click="login()">Войти</button>
  <br>
  <a href="#/register">Зарегистрироваться</a>
</form>
```

### Исходный код В.3 — partials/register.html

```
<h4>Регистрация</h4>
<form name="registrationForm">
  <span ng-show="success" id="registration_success">
    Вы успешно зарегистрировались! Вы можете перейти на <a href="#/login">
        страницу входа</a>
```

```
5     <br><br>
6   </span>
7   <span ng-show="failure" id="registration_failure">
8     <span ng-show="alreadyExists">Данное имя пользователя уже существует!<br></
          span>
9     <span ng-show="registrationForm.password.$error.minlength">Пароль должен
          содержать не меньше 6 символов!<br></span>
10    <br>
11  </span>
12  Имя пользователя: <input type="text" name="name" ng-model="name" required>
13  <br>
14  Пароль: <input type="password" name="password" ng-minlength="6" ng-model="
        password" required>
15  <br>
16  <button ng-click="register()">Зарегистрироваться</button>
17 </form>
```

Исходный код B.4 — paritals/queue.html

```
1 <h4>Очередь ожидания</h4>
2 <span id="no_queue" ng-show="queue.length == 0">Никого нет онлайн</span>
3 <ul class="queue">
4   <li ng-repeat="user in queue" ng-click="requestGame(user.id)">
5     {{user.name}}
6     <p>id: {{user.id}}, rating: {{user.rating}}</p>
7   </li>
8 </ul>
9 <countdown ng-show="acceptingRequest" time="15" on-accept="acceptRequest()" on-
      decline="declineRequest()" is-failure="requestDeclined" failure-message="
      Соперник отказался от игры с вами." timeout="Время ожидания соперника истекло
      .">Игрок {{requestDetails.name}} (рейтинг {{requestDetails.rating}}) хочет
      сыграть с вами</countdown>
10 <countdown ng-show="waitingAccept" time="15" on-cancel="cancelRequest()" is-
      failure="requestDeclined" failure-message="Соперник отказался от игры с вами"
       timeout="Время ожидания истекло.">Ожидание соперника...</countdown>
```

Исходный код B.5 — partials/countdown.html

```
1 <div class="countdown" ng-show="timeLeft > 0">
2   <div class="countdown_message" ng-transclude></div>
3   <h1>{{timeLeft}}</h1>
4     <button ng-show="canAccept" ng-click="accept()">Принять</button>
5     <button ng-show="canAccept" ng-click="decline()">Отклонить</button>
6     <button ng-hide="canAccept" ng-click="cancel()">Отменить</button>
7 </div>
8 <div class="error_message" ng-hide="timeLeft > 0">
9   {{error}}
10 </div>
```

Исходный код B.6 — partials/game.html

```
<div id="current_player" ng-attr-class="{{'color_' + current_player}}">
  <span ng-show="current_player == player">Ваш ход</span>
  <span ng-show="current_player != player">Ход соперника</span>
</div>
<div id="scores">
  <div id="score_1">{{score[1]}}</div>
  <div id="score_2">{{score[2]}}</div>
</div>
<div id="game_field">
  <canvas id='field_canvas' ng-click="canvasClick($event)" width="806" height="
      660"></canvas>
</div>
<div id="control_panel">
  <button ng-click="requestDraw()">Запросить ничью</button>
  <button ng-click="requestSurrender()">Запросить конец игры</button>
  <div id="draw_request" ng-show="requestedDraw">
    <countdown ng-show="acceptingRequest" time="15" on-accept="acceptRequest()"
       on-decline="declineRequest()" is-failure="requestDeclined" failure-
       message="Соперник отказался от ничьи." timeout="Время ожидания соперника
       истекло.">Соперник предлагает вам ничью.</countdown>
    <countdown ng-show="waitingAccept" time="15" on-cancel="declineRequest()" is
       -failure="requestDeclined" failure-message="Соперник отказался от ничьи."
        timeout="Время ожидания истекло.">Ожидание соперника...</countdown>
  </div>
  <div id="surrender_request" ng-show="requestedSurrender">
    <countdown ng-show="acceptingRequest" time="15" on-accept="acceptRequest()"
       on-decline="declineRequest()" is-failure="requestDeclined" failure-
       message="Соперник отказался от вышего предложения." timeout="Время
       ожидания соперника истекло.">Соперник предлагает вам завершить игру с
       текущим счетом.</countdown>
    <countdown ng-show="waitingAccept" time="15" on-cancel="declineRequest()" is
       -failure="requestDeclined" failure-message="Соперник отказался от ничьи."
        timeout="Время ожидания истекло.">Ожидание соперника...</countdown>
  </div>
</div>
```

Исходный код B.7 — css/style.css

```
.color_0
{
  color: rgba(0, 0, 0, 0);
}

.color_1
{
  color: rgba(255, 0, 0, 1);
}
```

```
10
11 .color_2
12 {
13   color: rgba(0, 0, 255, 1);
14 }
15
16 .point
17 {
18   margin-right: 12px;
19 }
20
21 .point > span:before
22 {
23   content: '\2022';
24 }
25
26 .row
27 {
28   margin-top: -10px;
29   margin-bottom: 0px;
30   cursor: default;
31 }
32
33 #current_player
34 {
35   width: 300px;
36   margin-left: auto;
37   margin-right: auto;
38   clear: both;
39   font-size: 20pt;
40   text-align: center;
41 }
42
43 #scores
44 {
45   width: 980px;
46   margin-top: 300px;
47   margin-left: auto;
48   margin-right: auto;
49   font-size: 30pt;
50   text-align: center;
51 }
52
53 #score_1
54 {
55   float: left;
56   width: 80px;
57   color: red;
```

```
58 }
59
60 #score_2
61 {
62   float: right;
63   width: 80px;
64   color: blue;
65 }
66
67 #game_field
68 {
69   /*background: url('../img/bg.png') repeat;
70   background-position-x: 4px;
71   background-position-y: 4px;*/
72   position: absolute;
73   top: 50px;
74   left: 50%;
75   width: 810px;
76   height: 663px;
77   margin-top: 30px;
78   margin-left: -405px;
79   font-size: 20pt;
80 }
81
82 #field_canvas
83 {
84   /*position: relative;
85   top: -672px;
86   left: -7px;*/
87   z-index: 10;
88 }
89
90 #control_panel
91 {
92   position: relative;
93   top: 370px;
94   clear: both;
95   float: left;
96   display: block;
97   z-index: 20;
98 }
```

## Исходный код B.8 — js/app.js

```
1 'use strict';
2
3
4 // Declare app level module which depends on filters, and services
5 var app = angular.module('dots', [
```

```
  'ngRoute',
  'ngStorage',
  'dots.controllers',
  'dots.directives'
]);

app.config(function($routeProvider) {
  $routeProvider.
  when("/login", {templateUrl:'partials/login.html', controller:'loginFormCtrl'}).
  when("/register", {templateUrl:'partials/register.html', controller:'registrationFormCtrl'}).
  when("/queue", {templateUrl:'partials/queue.html', controller:'waitingRoomCtrl'}).
  when("/game/:channel", {templateUrl:'partials/game.html', controller:'gameScreenCtrl'}).
  otherwise({redirectTo: '/login'});
});
```

Исходный код B.9 — js/controllers.js

```
'use strict';

/* Controllers */

var controllers = angular.module('dots.controllers', []);
var server = 'http://localhost:8888';

controllers.controller('appCtrl', ['$scope', '$localStorage', '$location',
    function($scope, $localStorage, $location) {
  $scope.$storage = $localStorage.$default({
      loggedIn: false,
      user: {name:'', rating: ''},
      auth: {id:'', token:''}
  });

  $(document).ajaxError(function(event, xhr) {
    if(xhr.status == 401)
    {
      $scope.$apply(function(){
        $scope.$storage.loggedIn = false;
        $location.path('/login');
      });
    }
  });

  $scope.pubsub = new Faye.Client(server + '/game');
  $scope.leaveChannel = function(channel){
    $scope.pubsub.publish(channel, {type: 'quit'});
```

```
28   };
29   $scope.pubsub.addExtension({
30     outgoing: function(message, callback) {
31         if (message.channel.substring(0, 5) == '/meta' && message.channel != '/
               meta/subscribe' && message.channel != '/meta/disconnect')
32           return callback(message);
33         message.auth = $scope.$storage.auth;
34         callback(message);
35     }
36   });
37   $scope.$on('$destroy', function(){
38     $scope.pubsub.disconnect();
39   });
40 }]);
41
42 controllers.controller('loginFormCtrl', ['$scope', '$location', function($scope,
       $location) {
43   $scope.failure = false;
44   $scope.login = function() {
45     if($scope.loginForm.name.$valid && $scope.loginForm.password.$valid)
46     {
47       $.post(server + '/auth', {name: $scope.name, password: $scope.password},
             function(data) {
48         $scope.$apply(function(){
49           $scope.failure = false;
50           $scope.$parent.$storage.user.name = $scope.name;
51           $scope.$parent.$storage.auth = {id: data.id, token: data.token};
52           $scope.$parent.$storage.loggedIn = true;
53           $location.path('/queue');
54         }, 'JSON');
55       }).fail(function(){
56         $scope.failure = true;
57       });
58     }
59   };
60 }]);
61
62 controllers.controller('registrationFormCtrl', ['$scope', '$location', function(
       $scope, $location) {
63   $scope.success = false;
64   $scope.failure = false;
65   $scope.alreadyExists = false;
66   $scope.name = '';
67   $scope.password = '';
68   $scope.register = function(){
69     if($scope.registrationForm.name.$valid && $scope.registrationForm.password.
          $valid)
70     {
```

```
71        $.post(server + '/register', {name: $scope.name, password: $scope.password
              }, function(data) {
72          $scope.$apply(function(){
73            $scope.failure = false;
74            $scope.alreadyExists = false;
75            $scope.success = true;
76          }, 'JSON');
77        }).fail(function(){
78          $scope.$apply(function(){
79            $scope.failure = true;
80            $scope.alreadyExists = true;
81          });
82        });
83      }
84      else
85      {
86        $scope.failure = true;
87      }
88    };
89 }]);
90
91 controllers.controller('waitingRoomCtrl', ['$scope', '$location', '$interval',
       function($scope, $location, $interval) {
92    if($scope.$storage.loggedIn == false)
93      $location.path('/login');
94    $scope.queue = [];
95    $.get(server + '/queue', $scope.$storage.auth, function(data) { //get current
          queue
96      $scope.$apply(function(){
97        $scope.queue = data.queue;
98      });
99    }, 'JSON');
100
101   $scope.updateUser = function(user)
102   {
103     if(user.id == $scope.$storage.auth.id)
104       return;
105     $scope.$apply(function(){
106       for(var i = 0; i < $scope.queue.length; i++)
107       {
108         if($scope.queue[i].id == user.id)
109         {
110           $scope.queue[i] = user;
111           return;
112         }
113       }
114       $scope.queue.push(user); //add user if not exists
115     });
```

```
116   };
117   $scope.removeUser = function(id)
118   {
119     if(id == $scope.$storage.auth.id)
120       return;
121
122     $scope.$apply(function(){
123       for(var i = 0; i < $scope.queue.length; i++)
124       {
125         if($scope.queue[i].id == id)
126         {
127           $scope.queue.splice(i, 1);
128           return;
129         }
130       }
131     });
132   };
133
134   $scope.waitingAccept = false;
135   $scope.acceptingRequest = false;
136   $scope.requestDetails = {name: '', rating: ''};
137   $scope.requestDeclined = false;
138
139   $scope.requestGame = function(id)
140   {
141     $scope.pubsub.publish('/game/queue', {type: 'request', target: id});
142   };
143
144   $scope.acceptRequest = function()
145   {
146     $scope.pubsub.publish('/game/queue', {type: 'accept', target: $scope.
          acceptingRequest});
147   };
148
149   $scope.declineRequest = function()
150   {
151     var init = $scope.acceptingRequest;
152     var target = $scope.$storage.auth.id;
153     $scope.pubsub.publish('/game/queue', {type: 'decline', init: init, target:
          target});
154   }
155
156   $scope.cancelRequest = function()
157   {
158     var init = $scope.$storage.auth.id;
159     var target = $scope.waitingAccept;
160     $scope.pubsub.publish('/game/queue', {type: 'decline', init: init, target:
          target});
```

```
161 };
162
163 $scope.onQueueMessage = function(message)
164 {
165   switch(message.type)
166   {
167     case 'heartbeat':
168       $scope.updateUser({id: message.id, name: message.name, rating: message.
              rating});
169       break;
170     case 'quit':
171       $scope.removeUser(message.id);
172       break;
173     case 'request':
174       if($scope.waitingAccept || $scope.acceptingRequest)
175         break;
176       if(message.id == $scope.$storage.auth.id)
177       {
178         console.log('I wanna play game with', message.target);
179         $scope.$apply(function(){
180           $scope.waitingAccept = message.target;
181         });
182       }
183       else if(message.target == $scope.$storage.auth.id)
184       {
185         console.log(message.id, 'wants play game with me');
186         $scope.$apply(function(){
187           $scope.requestDetails = message.user;
188           $scope.acceptingRequest = message.id;
189         });
190       }
191       break;
192     case 'accept':
193       if(message.id == $scope.$storage.auth.id)
194       {
195         console.log('I accepted game with', message.target);
196         $scope.$apply(function(){
197           $scope.acceptingRequest = false;
198         });
199       }
200       else if(message.target == $scope.$storage.auth.id)
201       {
202         console.log(message.id, ' accepted my game');
203         $scope.$apply(function(){
204           $scope.waitingAccept = false;
205         });
206       }
207       break;
```

```
208       case 'decline':
209         if(message.init == $scope.$storage.auth.id && message.target == $scope.
                waitingAccept ||
210           message.init == $scope.acceptingRequest && message.target == $scope.
                $storage.auth.id )
211         {
212           $scope.$apply(function(){
213             $scope.requestDeclined = true;
214           });
215         }
216         break;
217       case 'newgame':
218         if(message.player_1 == $scope.$storage.auth.id || message.player_2 ==
                $scope.$storage.auth.id)
219         {
220           $scope.$apply(function(){
221             $location.path('/game/' + message.channel);
222           });
223         }
224         break;
225       default:
226         console.log('Unknown message type:' + message.type);
227     };
228   };
229
230   $scope.subscription = $scope.pubsub.subscribe('/game/queue', $scope.
        onQueueMessage);
231
232   $scope.iamalive = function(){
233     $scope.pubsub.publish('/game/queue', {type: 'heartbeat'});
234   };
235   $scope.heartbeat = $interval(function(){
236     $scope.iamalive();
237   }, 5000);
238   $scope.iamalive(); //because interval executes only after some time
239
240   $scope.$on('$destroy', function(){
241     $scope.leaveChannel('/game/queue');
242     $scope.subscription.cancel();
243     $interval.cancel($scope.heartbeat);
244   });
245 }]);
246
247 controllers.controller('gameScreenCtrl', ['$scope', '$location', '$routeParams',
        '$interval', function($scope, $location, $routeParams, $interval) {
248   if($scope.$storage.loggedIn == false)
249     $location.path('/login');
250   $scope.channel = $routeParams.channel;
```

```
$scope.player = 0; //0 - spectator
$scope.current_player = 1;
$scope.score = [0, 0, 0];
$scope.field = {};

$scope.canvas = $('#field_canvas')[0].getContext('2d');
$scope.canvasWidth = $('#field_canvas')[0].width;
$scope.canvasHeight = $('#field_canvas')[0].height;
$scope.offsetX = 4;
$scope.offsetY = 4;
$scope.ceilWidth = 21;
$scope.ceilHeight = 21;

$scope.requestedDraw = false;
$scope.requestedSurrender = false;
$scope.acceptingRequest = false;
$scope.waitingAccept = false;
$scope.requestDeclined = false;

$.get(server + '/gamedata', {id: $scope.$storage.auth.id, token: $scope.
    $storage.auth.token, channel: $scope.channel} , function(data) { //get
    current game
  $scope.$apply(function(){
    $scope.field = JSON.parse(data.field);
    $scope.redrawField();
    $scope.redrawZones();
    if(data.player_1 == $scope.$storage.auth.id)
      $scope.player = 1;
    else if(data.player_2 == $scope.$storage.auth.id)
      $scope.player = 2;
    $scope.current_player = data.current_player;
    $scope.score[1] = data.score_1;
    $scope.score[2] = data.score_2;
  });
}, 'JSON');

$scope.redrawZones = function()
{
  for(var i = 0; i < $scope.field.zones.length; i++)
  {
    $scope.drawZone($scope.field.zones[i]);
  }
}

$scope.redrawField = function()
{
  $scope.canvas.clearRect(0, 0, $scope.canvasWidth, $scope.canvasHeight);
  for(var i = 0; i < $scope.field.height; i++)
```

```
297     {
298       $scope.canvas.beginPath();
299       $scope.canvas.moveTo(0, $scope.offsetY + i * $scope.ceilHeight);
300           $scope.canvas.lineTo($scope.canvasWidth, $scope.offsetY + i * $scope.
                  ceilHeight);
301       $scope.canvas.closePath();
302       $scope.canvas.lineWidth = 2;
303       $scope.canvas.strokeStyle = 'rgb(217, 210, 245)';
304           $scope.canvas.stroke();
305     }
306     for(var i = 0; i < $scope.field.width; i++)
307     {
308       $scope.canvas.beginPath();
309       $scope.canvas.moveTo($scope.offsetX + i * $scope.ceilWidth, 0);
310           $scope.canvas.lineTo($scope.offsetX + i * $scope.ceilWidth, $scope.
                  canvasHeight);
311       $scope.canvas.closePath();
312       $scope.canvas.lineWidth = 2;
313       $scope.canvas.strokeStyle = 'rgb(217, 210, 245)';
314           $scope.canvas.stroke();
315     }
316     for(var i = 0; i < $scope.field.height; i++)
317     {
318       for(var j = 0; j < $scope.field.width; j++)
319       {
320         if($scope.field.color[i][j] != 0)
321         {
322           $scope.canvas.beginPath();
323           $scope.canvas.arc($scope.offsetX + j * $scope.ceilWidth, $scope.
                  offsetY + i * $scope.ceilHeight, 4, 0, 2 * Math.PI, false);
324           $scope.canvas.fillStyle = ($scope.field.color[i][j] == 1) ? 'red' : '
                  blue';
325           $scope.canvas.fill();
326         }
327       }
328     }
329   }
330
331   $scope.drawZone = function(zone)
332   {
333     $scope.canvas.beginPath();
334     $scope.canvas.moveTo($scope.offsetX + zone[0].y * $scope.ceilWidth, $scope.
          offsetY + zone[0].x * $scope.ceilHeight);
335     for(var i = 1; i < zone.length; i++)
336     {
337       $scope.canvas.lineTo($scope.offsetX + zone[i].y * $scope.ceilWidth, $scope
             .offsetY + zone[i].x * $scope.ceilHeight); //remember, that x points at
              row and y points at column!!!
```

```
338     }
339     $scope.canvas.closePath();
340     $scope.canvas.lineWidth = 2;
341         $scope.canvas.strokeStyle = ($scope.field.captured[zone[0].x][zone[0].y]
                == 1) ? 'red' : 'blue';
342         $scope.canvas.stroke();
343         $scope.canvas.fillStyle = 'rgba(0, 0, 0, 0.3)';
344         $scope.canvas.fill();
345   };
346
347   $scope.requestDraw = function(){
348     console.log('request_d');
349     $scope.pubsub.publish('/game/' + $scope.channel, {type: 'request',
          requestType: 'draw'});
350   };
351
352   $scope.requestSurrender = function(){
353     $scope.pubsub.publish('/game/' + $scope.channel, {type: 'request',
          requestType: 'surrender'});
354   };
355
356   $scope.acceptRequest = function(){
357     $scope.pubsub.publish('/game/' + $scope.channel, {type: 'accept',
          requestType: ($scope.requestedDraw ? 'draw' : 'surrender')});
358   };
359
360   $scope.declineRequest = function(){
361     $scope.pubsub.publish('/game/' + $scope.channel, {type: 'decline', init: (3
          - $scope.player), target: $scope.player});
362   };
363
364   $scope.cancelRequest = function(){
365     $scope.pubsub.publish('/game/' + $scope.channel, {type: 'decline', init:
          $scope.player, target: (3 - $scope.player)});
366   };
367
368   $scope.quitGame = function(){
369
370   };
371
372   $scope.onGameMessage = function(message){
373     switch(message.type)
374     {
375       case 'heartbeat':
376         break;
377       case 'request':
378         $scope.$apply(function(){
379           $scope.requestedDraw = (message.requestType == 'draw');
```

```
380           $scope.requestedSurrender = (message.requestType == 'surrender');
381           $scope.waitingAccept = (message.player == $scope.player);
382           $scope.acceptingRequest = (message.player == (3 - $scope.player));
383         });
384         break;
385       case 'accept':
386         $scope.$apply(function(){
387           $scope.requestedDraw = false;
388           $scope.requestedSurrender = false;
389           $scope.waitingAccept = false;
390           $scope.acceptingRequest = false;
391         });
392         break;
393       case 'decline':
394         if(message.player != $scope.player)
395         {
396           $scope.$apply(function(){
397             $scope.requestDeclined = true;
398           });
399         }
400         break;
401       case 'move':
402         console.log('mov');
403         $scope.$apply(function(){
404           $scope.field.color[message.point.x][message.point.y] = message.player;
405           $scope.current_player = ($scope.current_player == 1) ? 2 : 1;
406           $scope.score = message.score;
407           $scope.field.zones = message.zones;
408           $scope.field.captured = message.captured;
409           $scope.redrawField();
410           $scope.redrawZones();
411         });
412         break;
413       case 'gameover':
414         $scope.$apply(function(){
415           $location.path('/queue');
416         });
417         break;
418       default:
419         console.log('Unknown message type:' + message.type);
420     };
421   };
422   $scope.subscription = $scope.pubsub.subscribe('/game/' + $scope.channel,
        $scope.onGameMessage);
423
424   $scope.iamalive = function(){
425     $scope.pubsub.publish('/game/' + $scope.channel, {type: 'heartbeat'});
426   };
```

```
427   $scope.heartbeat = $interval(function(){
428     $scope.iamalive();
429   }, 5000);
430   $scope.iamalive(); //because interval executes only after some time
431
432   $scope.canvasClick = function(event)
433   {
434     //console.log(event.offsetX, event.offsetY);
435     $scope.doMove(Math.abs(Math.round((event.offsetY - $scope.offsetY) / $scope.
          ceilHeight)), Math.abs(Math.round((event.offsetX - $scope.offsetX) /
          $scope.ceilWidth)));
436   }
437   $scope.doMove = function(x, y){
438     console.log(x, y);
439     if($scope.current_player == $scope.player && !$scope.field.color[x][y] && !
          $scope.field.captured[x][y])
440       $scope.pubsub.publish('/game/' + $scope.channel, {type: 'move', point: {x:
            x, y: y}});
441   }
442
443   $scope.$on('$destroy', function(){
444     $scope.leaveChannel('/game/' + $scope.channel);
445     $scope.subscription.cancel();
446     $interval.cancel($scope.heartbeat);
447   });
448 }]);
```

Исходный код B.10 — js/directives.js

```
1  'use strict';
2
3  /* Directives */
4
5  angular.module('dots.directives', []).
6    directive('countdown', ['$timeout', '$interval', function($timeout, $interval)
         {
7      return {
8          restrict: 'E',
9        templateUrl: 'partials/countdown.html',
10       scope: {
11         ngShow: '=',
12         time: '@',
13         onAccept: '&',
14         onDecline: '&',
15         onCancel: '&',
16         isFailure: '=',
17         failureMessage: '@',
18         timeout: '@'
19       },
```

```
20        transclude: true,
21        link: function link($scope, element, attrs) {
22          $scope.canAccept = (typeof attrs.onAccept !== 'undefined');
23          $scope.timeLeft = 0;
24          $scope.error = '';
25          $scope.showError = function(message)
26          {
27            $scope.stopCountdown();
28            $scope.error = message;
29            $timeout(function(){
30              $scope.isFailure = false;
31              $scope.ngShow = false;
32            }, 1000, true);
33          };
34          $scope.accept = function() {
35            $scope.onAccept();
36            $scope.ngShow = false;
37          };
38          $scope.decline = function() {
39            $scope.onDecline();
40            $scope.ngShow = false;
41          };
42          $scope.cancel = function() {
43            $scope.onCancel();
44            $scope.ngShow = false;
45          };
46          $scope.onFailure = function() {
47            if($scope.ngShow != false && $scope.isFailure != false)
48              $scope.showError($scope.failureMessage);
49          };
50          $scope.onTimeout = function() {
51            $scope.showError($scope.timeout);
52          };
53          $scope.countdown = false;
54
55          $scope.startCountdown = function(){
56            $scope.timeLeft = $scope.time;
57            $scope.countdown = $interval(function(){
58              $scope.timeLeft--;
59              if($scope.timeLeft == 0)
60                $scope.onTimeout();
61            }, 1000, $scope.time, true);
62          };
63          $scope.stopCountdown = function(){
64            if($scope.timeLeft == 0)
65              return;
66            $scope.timeLeft = 0;
67            $interval.cancel($scope.countdown);
```

```
68          };
69          $scope.$watch('ngShow', function(){
70            if($scope.ngShow)
71              $scope.startCountdown();
72            else
73              $scope.stopCountdown();
74          });
75          $scope.$watch('isFailure', $scope.onFailure);
76          $scope.$on('$destroy', function(){
77            if($scope.countdown)
78              $interval.cancel($scope.countdown);
79          });
80        }
81      }
82    }]);
```

Исходный код B.11 — server.js

```
1  var http = require('http');
2  var express = require('express');
3  var bodyParser = require('body-parser');
4  var faye = require('faye');
5  var redis = require('redis');
6  var url = require('url');
7  var crypto = require('crypto')
8
9  var app = express();
10 var server = http.createServer(app);
11 var db = redis.createClient(6379, 'acm.tpu.ru');
12 var bayeux = new faye.NodeAdapter({mount: '/game', timeout: 45});
13 var ajax = 'file://';
14
15 app.use(bodyParser.urlencoded({ extended: false }));
16
17 function onAuth(request, response)
18 {
19   var name = request.param('name');
20   var password = request.param('password');
21   response.setHeader('Access-Control-Allow-Origin', ajax);
22   if(name == null || name == "" || password == null || password == "")
23     response.status(400).send('Bad request');
24   db.hget('users', name, function(err, id){
25     if(err) throw err;
26     if(id == null)
27     {
28       response.status(401).send('Invalid username/password');
29     }
30     else
31     {
```

```
32        db.hget('user:' + id, 'password', function(err, passwd){
33          if(err) throw err;
34          if(password != passwd)
35          {
36            response.status(401).send('Invalid username/password');
37          }
38          else
39          {
40            crypto.randomBytes(10, function(err, buf){
41              var token = buf.toString('hex');
42              db.hset('user:' + id, 'token', token);
43              response.json({id: id, token: token});
44            });
45          }
46        });
47      }
48    });
49  }
50
51  function checkAuth(auth, succ, fail)
52  {
53    db.hgetall('user:' + auth.id, function(err, reply){
54      if(err) throw err;
55      if(reply != null && reply.token != '' && auth.token == reply.token)
56        succ(reply);
57      else
58        fail();
59    });
60  }
61
62  function onRegister(request, response)
63  {
64    var name = request.param('name');
65    var password = request.param('password');
66    response.setHeader('Access-Control-Allow-Origin', ajax);
67    if(name == null || name == "" || password == null || password == "")
68      response.status(400).send('Bad request');
69    db.hget('users', name, function(err, id){
70      if(err) throw err;
71      if(id != null)
72      {
73        response.status(406).send('Already exists');
74      }
75      else
76      {
77        db.incr('last_userid', function(err, user_id){
78          db.hset('users', name, user_id);
79          db.hmset('user:' + user_id, 'name', name, 'password', password, 'token',
```

```
            '', 'rating', '1600');
80        response.send('OK!');
81      });
82    }
83  });
84 }
85
86 function getQueue(request, response)
87 {
88   response.setHeader('Access-Control-Allow-Origin', ajax);
89   var auth = {id: request.param('id'), token: request.param('token')}
90   if(auth.id == null || auth.id == "" || auth.token == null || auth.token == "")
91     response.status(400).send('Bad request');
92   checkAuth(auth, function(){ //authorized
93     db.sort('queue', 'by', 'user:*->rating', 'get', '#', 'get', 'user:*->name',
         'get', 'user:*->rating', function(err, reply){
94       var queue = [];
95       if(reply != null)
96       {
97         for(i = 0; i < reply.length / 3; i++)
98         {
99           var ix = i * 3;
100          if(reply[ix] != auth.id)
101            queue.push({id: reply[ix], name: reply[ix + 1], rating: reply[ix +
                 2]});
102        }
103      }
104      response.json({queue: queue});
105    });
106  }, function(){
107    response.status(401).send('Invalid token!');
108  });
109 }
110
111 function getGameData(request, response)
112 {
113   response.setHeader('Access-Control-Allow-Origin', ajax);
114   var auth = {id: request.param('id'), token: request.param('token')};
115   var channel = request.param('channel');
116   if(auth.id == null || auth.id == "" || auth.token == null || auth.token == ""
        || channel == null || channel == "")
117     response.status(400).send('Bad request');
118   checkAuth(auth, function(){
119     db.hget('games', channel, function(err, game_id){
120       if(game_id == null)
121       {
122         return response.status(404).send('Game not found');
123       }
```

```
124       db.hgetall('game:' + game_id, function(err, game){
125         return response.json(game);
126       });
127     });
128   }, function(){
129     return response.status(401).send('Invalid token!');
130   });
131 }
132
133 app.post('/auth', onAuth);
134 app.post('/register', onRegister);
135 app.get('/queue', getQueue);
136 app.get('/gamedata', getGameData);
137
138 function timestamp()
139 {
140   return Math.round(Date.now() / 1000);
141 }
142
143 function kickPlayer(id)
144 {
145   bayeux.getClient().publish('/game/queue', {type: 'quit', id: id});
146 }
147
148 function addRequest(id, type, target, message, callback)
149 {
150   db.zscore('requests', id, function(err, reply){
151     if(reply != null && reply > timestamp() - 15) //user already requested
          something
152     {
153       message.error = '406::Already exists';
154     }
155     else
156     {
157       db.zadd('requests', timestamp(), id);
158       db.hmset('request:' + id, 'type', type, 'target', target);
159     }
160     callback(message);
161   });
162 }
163
164 function cancelRequest(id, target, message, callback)
165 {
166   db.hget('request:' + id, 'target', function(err, reply){
167     if(reply == null || reply != target)
168     {
169       message.error = '403::Authentication required';
170     }
```

```
171     else
172     {
173       db.zrem('requests', id);
174       db.del('request:' + id);
175     }
176     callback(message);
177   });
178 }
179
180 function emptyField()
181 {
182   field = {width: 39, height: 32, color: [], captured: [], zones: []};
183   for(var i = 0; i < field.height; i++)
184   {
185     field.color[i] = [];
186     field.captured[i] = [];
187     for(var j = 0; j < field.width; j++)
188     {
189       field.color[i][j] = 0;
190       field.captured[i][j] = 0;
191     }
192   }
193   return JSON.stringify(field);
194 }
195
196 function createGame(player_1, player_2, message, callback)
197 {
198   db.hget('request:' + player_1, 'target', function(err, reply){
199     if(reply == null || reply != player_2)
200     {
201       message.error = '406::Not allowed';
202     }
203     else
204     {
205       crypto.randomBytes(5, function(err, buf){
206         var channel = buf.toString('hex');
207         db.incr('last_gameid', function(err, game_id){
208           db.hset('games', channel, game_id);
209           db.hmset('game:' + game_id, 'channel', channel, 'field', emptyField(),
                'player_1', player_1, 'score_1', 0, 'player_2', player_2, 'score_2
               ', 0, 'current_player', 1);
210           bayeux.getClient().publish('/game/queue', {type: 'newgame', id: 0,
                channel: channel, player_1: player_1, player_2: player_2});
211         });
212       });
213     }
214     callback(message);
215   });
```

```
216 }
217
218 var neighbours = [{x: 0, y: -1}, {x: -1, y: -1}, {x: -1, y: 0}, {x: -1, y: 1}, {
      x: 0, y: 1}, {x: 1, y: 1}, {x: 1, y: 0}, {x: 1, y: -1}];
219
220 function emptyArea(height, width)
221 {
222   var area = [];
223   for(var i = 0; i < height; i++)
224   {
225     area[i] = [];
226     for(var j = 0; j < width; j++)
227       area[i][j] = 0;
228   }
229   return area;
230 }
231
232 function fillArea(area, field, point, color)
233 {
234   if(point.x < 0 || point.x >= field.height || point.y < 0 || point.y >= field.
        width || area[point.x][point.y] || (field.color[point.x][point.y] == color
        && field.captured[point.x][point.y] != (3 - color)))
235     return;
236   area[point.x][point.y] = 1;
237   for(var i = 0; i < neighbours.length; i += 2) //we don't need corners
238   {
239     var x = point.x + neighbours[i].x;
240     var y = point.y + neighbours[i].y;
241     fillArea(area, field, {x: x, y: y}, color);
242   }
243 }
244
245 function findCutpoints(area, timer, tin, fup, open_area, field, point, parent)
246 {
247   area[point.x][point.y] = 1;
248   tin[point.x][point.y] = fup[point.x][point.y] = ++timer.time;
249   var children = 0;
250   for(var i = 0; i < neighbours.length; i++)
251   {
252     var x = point.x + neighbours[i].x;
253     var y = point.y + neighbours[i].y;
254     if(x < 0 || x >= field.height || y < 0 || y >= field.width || open_area[x][y
          ] || (x == parent.x && y == parent.y))
255       continue;
256     if(area[x][y])
257     {
258       fup[point.x][point.y] = Math.min(fup[point.x][point.y], tin[x][y]);
259     }
```

```
260     else
261     {
262       findCutpoints(area, timer, tin, fup, open_area, field, {x: x, y: y}, point
            );
263       fup[point.x][point.y] = Math.min(fup[point.x][point.y], fup[x][y]);
264       if (fup[x][y] >= tin[point.x][point.y] && parent.x != -1)
265       {
266         area[point.x][point.y] = 2;
267       }
268       children++;
269     }
270   }
271   if(parent.x == -1 && children > 1)
272     area[point.x][point.y] = 2;
273 }
274
275 function clearOpenArea(area, open_area, field)
276 {
277   for(var i = 0; i < field.height; i++)
278     for(var j = 0; j < field.width; j++)
279       if(area[i][j])
280         open_area[i][j] = 1;
281 }
282
283 function splitArea(used, area, areas, current, tin, fup, field, point, parent)
284 {
285   used[point.x][point.y] = 1;
286   areas[current][point.x][point.y] = 1;
287   var children = 0;
288   for(var i = 0; i < neighbours.length; i++)
289   {
290     var x = point.x + neighbours[i].x;
291     var y = point.y + neighbours[i].y;
292     if(x < 0 || x >= field.height || y < 0 || y >= field.width || !area[x][y] ||
          used[x][y] || (x == parent.x && y == parent.y))
293       continue;
294     children++;
295     if((fup[x][y] >= tin[point.x][point.y] && parent.x != -1) || (parent.x == -1
          && children > 1))
296     {
297       var new_current = areas.length;
298       areas[new_current] = emptyArea(field.height, field.width);
299       areas[new_current][point.x][point.y] = 1;
300       splitArea(used, area, areas, new_current, tin, fup, field, {x: x, y: y},
            point);
301     }
302     else
303     {
```

```
304       splitArea(used, area, areas, current, tin, fup, field, {x: x, y: y}, point
              );
305     }
306   }
307 }
308
309 function findAreas(open_area, field)
310 {
311   var areas = [];
312   var current = 0;
313   for(var x = 0; x < field.height; x++)
314   {
315     for(var y = 0; y < field.width; y++)
316     {
317       if(!open_area[x][y])
318       {
319         var area = emptyArea(field.height, field.width);
320         var tin = emptyArea(field.height, field.width);
321         var fup = emptyArea(field.height, field.width);
322         findCutpoints(area, {time: 0}, tin, fup, open_area, field, {x: x, y: y},
                {x: -1, y: -1});
323         clearOpenArea(area, open_area, field);
324         areas[current] = emptyArea(field.height, field.width);
325         splitArea(emptyArea(field.height, field.width), area, areas, current,
              tin, fup, field, {x: x, y: y}, {x: -1, y: -1});
326         current = areas.length;
327       }
328     }
329   }
330   return areas;
331 }
332
333 function borderLine(area, field)
334 {
335   var start = null;
336   for(var i = 0; i < field.height; i++)
337   {
338     for(var j = 0; j < field.width; j++)
339     {
340       if(area[i][j])
341       {
342         start = {x: i, y: j};
343         break;
344       }
345     }
346     if(start != null)
347       break;
348   }
```

```
349   var point = start;
350   var parent = {x: -1, y: -1}
351   console.log(start);
352   var line = [];
353   do
354   {
355     var i = 0;
356     if(parent.x != -1)
357     {
358       while(true)
359       {
360         var x = point.x + neighbours[i].x;
361         var y = point.y + neighbours[i].y;
362         if(parent.x == x && parent.y == y)
363           break;
364         i++;
365       }
366     }
367     while(true)
368     {
369       var j = (i + 1) % neighbours.length;
370       var xi = point.x + neighbours[i].x, xj = point.x + neighbours[j].x;
371       var yi = point.y + neighbours[i].y, yj = point.y + neighbours[j].y;
372       if((xi < 0 || xi >= field.height || yi < 0 || yi >= field.width || area[xi
              ][yi] == 0) && (xj >= 0 && xj < field.height && yj >= 0 && yj < field.
              width && area[xj][yj] != 0))
373       {
374         parent = point;
375         point = {x: xj, y: yj};
376         break;
377       }
378       i = j;
379     }
380     console.log(point);
381     line.push(point);
382   } while(point.x != start.x || point.y != start.y);
383   return line;
384 }
385
386 function checkArea(area, field, color)
387 {
388   var anticolor = 3 - color;
389   for(var i = 0; i < field.height; i++)
390   {
391     for(var j = 0; j < field.width; j++)
392     {
393       if(area[i][j] && (field.color[i][j] == anticolor))
394         return true;
```

```
395     }
396   }
397   return false;
398 }
399
400 function checkLine(line, field, color)
401 {
402   var anticolor = 3 - color;
403   for(var i = 0; i < line.length; i++)
404   {
405     if(field.captured[line[i].x][line[i].y] == anticolor)
406       return false;
407   }
408   return true;
409 }
410
411 function updateCaptured(area, field, color)
412 {
413   for(var i = 0; i < field.height; i++)
414     for(var j = 0; j < field.width; j++)
415       if(area[i][j])
416         field.captured[i][j] = color;
417 }
418
419 function findZones(message, field, color)
420 {
421   var open_area = emptyArea(field.height, field.width);
422   for(var i = 0; i < field.width; i++)
423   {
424     fillArea(open_area, field, {x: 0, y: i}, color);
425     fillArea(open_area, field, {x: field.height - 1, y: i}, color);
426   }
427   for(var i = 0; i < field.height; i++)
428   {
429     fillArea(open_area, field, {x: i, y: 0}, color);
430     fillArea(open_area, field, {x: i, y: field.width - 1}, color);
431   }
432   areas = findAreas(open_area, field);
433   for(var i = 0; i < areas.length; i++)
434     if(checkArea(areas[i], field, color))
435     {
436       var line = borderLine(areas[i], field);
437       if(checkLine(line, field, color))
438       {
439         field.zones.push(borderLine(areas[i], field));
440         updateCaptured(areas[i], field, color);
441       }
442     }
```

```
443 }
444
445 function updateScores(field)
446 {
447   score = [0, 0, 0];
448   for(var i = 0; i < field.height; i++)
449   {
450     for(var j = 0; j < field.width; j++)
451     {
452       if(field.color[i][j] != 0 && field.captured[i][j] != 0 && field.captured[i
              ][j] != field.color[i][j])
453       {
454         score[field.captured[i][j]]++;
455       }
456     }
457   }
458   field.score_1 = score[1];
459   field.score_2 = score[2];
460 }
461
462 function doMove(message, callback, game_id, field, point)
463 {
464   if(typeof point === 'undefined' || typeof point.x === 'undefined' || typeof
         point.y === 'undefined')
465   {
466     message.error = '400::Bad request';
467     return callback(message);
468   }
469   if(field.color[point.x][point.y] != 0 || field.captured[point.x][point.y] !=
         0)
470   {
471     message.error = '406::Point is busy';
472     return callback(message);
473   }
474   field.color[point.x][point.y] = message.data.player;
475   console.log(field.color[point.x][point.y]);
476   field.zones = [];
477   findZones(message, field, message.data.player);
478   findZones(message, field, 3 - message.data.player);
479   message.data.zones = field.zones;
480   message.data.captured = field.captured;
481   updateScores(field);
482   message.data.score = [0, field.score_1, field.score_2];
483   db.hmset('game:' + game_id, 'current_player', ((message.data.player == 1) ? 2
         : 1), 'field', JSON.stringify(field));
484   callback(message);
485 }
486
```

```
487 function eloCoefficient(rating)
488 {
489   var res = 10;
490   if(rating < 2400)
491     res = 15;
492   if(rating < 1700)
493     res = 25;
494   return res;
495 }
496
497 function updateRatings(game, winner)
498 {
499   db.hget('user:' + game.player_1, 'rating', function(err, rating_1){
500     db.hget('user:' + game.player_2, 'rating', function(err, rating_2){
501       rating_1 = Number(rating_1);
502       rating_2 = Number(rating_2);
503       var e1 = 1.0 / (1.0 + Math.pow(10, (rating_2 - rating_1) / 400.0));
504       var e2 = 1.0 / (1.0 + Math.pow(10, (rating_1 - rating_2) / 400.0));
505       var s1 = 0.5;
506       var s2 = 0.5
507       if(winner == 1)
508       {
509         var s1 = 1;
510         var s2 = 0;
511       }
512       else if(winner == 2)
513       {
514         var s1 = 0;
515         var s2 = 1;
516       }
517       var new_rating_1 = Math.round(rating_1 + eloCoefficient(rating_1) * (s1 -
              e1));
518       var new_rating_2 = Math.round(rating_2 + eloCoefficient(rating_2) * (s2 -
              e2));
519       db.hset('user:' + game.player_1, 'rating', new_rating_1);
520       db.hset('user:' + game.player_2, 'rating', new_rating_2);
521     });
522   });
523 }
524
525 function gameOver(game_id, game, draw, message, callback)
526 {
527   var winner;
528   if(game.score_1 == game.score_2 || draw)
529     winner = 0;
530   else if(game.score_1 > game.score_2)
531     winner = 1;
532   else
```

```
533     winner = 2;
534   var rating = [0, 0];
535   updateRatings(game, winner);
536   bayeux.getClient().publish('/game/' + game.channel, {type: 'gameover', id: 0,
          score_1: game.score_1, score_2: game.score_2, winner: winner});
537   db.hdel('games', game.channel);
538   db.del('game:' + game_id);
539   callback(message);
540 }
541
542 function unauthorized_message(message, callback){
543   return function(){
544     console.log('unauth, ', message.channel);
545     message.error = '403::Authentication required';
546     callback(message);
547   };
548 }
549
550 function queueChannel(user, message, callback)
551 {
552   if(message.type == 'newgame' && message.data.id != 0) //only server can send
          newgame messages
553       return unauthorized_message(message, callback)();
554   if(message.data.type == 'heartbeat')
555   {
556     message.data.name = user.name;
557     message.data.rating = user.rating;
558     db.zadd('queue', timestamp(), message.data.id);
559   }
560
561   if(message.data.type == 'quit')
562   {
563     console.log('Player', message.data.id, 'quit');
564     db.zrem('queue', message.data.id);
565   }
566
567   if(message.data.type == 'request')
568   {
569     message.data.user = {name: user.name, rating: user.rating};
570     return addRequest(message.data.id, 'newgame', message.data.target, message,
          callback);
571   }
572
573   if(message.data.type == 'accept')
574   {
575     return createGame(message.data.target, message.data.id, message, callback);
576   }
577
```

```
578   if(message.data.type == 'decline' && (message.data.id == message.data.init ||
          message.data.id == message.data.target))
579   {
580     return cancelRequest(message.data.init, message.data.target, message,
            callback);
581   }
582   return callback(message);
583 }
584
585 function gameChannel(channel, id, message, callback)
586 {
587   db.hget('games', channel, function(err, game_id){
588     if(game_id == null)
589     {
590       message.error = '404::Game not found';
591       callback(message);
592     }
593     db.hgetall('game:' + game_id, function(err, game){
594       if(game == null)
595       {
596         message.error = '404::Game not found';
597         return callback(message);
598       }
599       if(game.player_1 == id)
600         message.data.player = 1;
601       else if (game.player_2 == id)
602         message.data.player = 2;
603       else
604         message.data.player = 0;
605
606       var other_player = 3 - message.data.player;
607
608       game.field = JSON.parse(game.field);
609
610       if(message.data.type == 'heartbeat' && message.data.player != 0)
611       {
612         db.hset('game', 'timestamp_' + message.data.player, timestamp());
613       }
614
615       if(message.data.type == 'move')
616       {
617         if(message.data.player != game.current_player)
618           message.error = '406::Not allowed';
619         else
620           return doMove(message, callback, game_id, game.field, message.data.
                point);
621       }
622
```

```
623       if(message.data.type == 'request')
624       {
625         if(message.data.player != 0 && (message.data.requestType == 'draw' ||
              message.data.requestType == 'surrender'))
626         {
627           return addRequest(game['player_' + message.data.player], message.data.
                requestType, game['player_' + other_player], message, callback);
628         }
629       }
630
631       if(message.data.type == 'accept')
632       {
633         if(message.data.player != 0)
634         {
635           return db.hget('request:' + game['player_' + other_player], 'target',
                function(err, reply){
636             if(reply == null || reply != game['player_' + message.data.player])
637             {
638               message.error = '406::Not allowed';
639               return callback(message);
640             }
641             gameOver(id, game, (message.data.requestType == 'draw'), message,
                  callback);
642           });
643         }
644       }
645
646       if(message.data.type == 'decline')
647       {
648         if(message.data.player != 0)
649         {
650           return cancelRequest(game['player_' + message.data.init], game['
                player_' + message.data.target], message, callback);
651         }
652       }
653
654       callback(message);
655     });
656   });
657 }
658
659 function incomingMessage(message, callback){
660   if (message.channel.substring(0, 5) == '/meta' && message.channel != '/meta/
        subscribe' && message.channel != '/meta/disconnect')
661       return callback(message);
662
663   var auth = message.auth;
664   if(auth == null)
```

```
665   {
666     return unauthorized_message(message, callback)();
667   }
668   delete message.auth;
669
670   if(auth.token == server_token) //message sent by server
671   {
672     return callback(message);
673   }
674
675   checkAuth(auth, function(user){
676     if(message.data != null)
677       message.data.id = auth.id;
678
679     if(message.channel == '/meta/disconnect')
680     {
681       kickPlayer(auth.id);
682     }
683
684     if(message.channel == '/game/queue')
685       return queueChannel(user, message, callback);
686
687     var game_channel = /\/game\/([a-f0-9]+)$/.exec(message.channel);
688     if(game_channel != null && typeof game_channel[1] !== 'undefined')
689       return gameChannel(game_channel[1], auth.id, message, callback);
690
691     callback(message);
692   }, unauthorized_message(message, callback));
693 }
694
695 //pub-sub engine
696 var server_token = 'suppasecretservertoken';
697
698 bayeux.getClient().addExtension({
699   outgoing: function(message, callback) {
700       if (message.channel.substring(0, 5) == '/meta' && message.channel != '/
          meta/subscribe')
701         return callback(message);
702       message.auth = {id: 0, token: server_token};
703       callback(message);
704   }
705 });
706
707 bayeux.addExtension({incoming: incomingMessage});
708
709 function garbageCollector()
710 {
711   //find offline users
```

```
712   db.zrangebyscore('queue', '-inf', timestamp() - 10, function(err, result){
713     if(result == null)
714       return;
715     for(i = 0; i < result.length; i++)
716     {
717       kickPlayer(result[i]);
718     }
719   });
720   db.zremrangebyscore('queue', '-inf', timestamp() - 10);
721
722   //find outdated requests
723   db.zrangebyscore('requests', '-inf', timestamp() - 15, function(err, result){
724     if(result == null)
725       return;
726     for(i = 0; i < result.length; i++)
727     {
728       result[i] = 'request:' + result[i];
729     }
730     db.del(result);
731   });
732   db.zremrangebyscore('requests', '-inf', timestamp() - 15);
733 }
734
735 setInterval(garbageCollector, 10000);
736
737 bayeux.attach(server);
738 server.listen(8888);
```