Министерство образования Республики Беларусь
Учреждение образования Белорусский государственный университет информатики и радиоэлектроники

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Объектно ориентированное программирование (ООП).

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту на тему

«Многопользовательская IO игра»

Выполнил: студент

Кременевский В.С

Студент группы 953501

Руководитель: ассистент кафедры информатики Рогов М.Г.

Минск 2021

Содержание.

Введение

С развитием цифровых технологий компьютеры все больше вливаются в жизнь человека. Если раньше ЭВМ использовались исключительно для сложных математических вычислений, то сегодня сфера их применения существенно расширилась. Компьютерные игры - одно из наиболее массовых применений электронных вычислительных машин.

Развитие игровой индустрии шло стремительным темпом, и особенно пользовалось популярностью у подростков. Первые игры отличались простотой интерфейса и логики, но со временем они становились все сложнее и сложнее, над их созданием работал уже не один человек, а целая команда разработчиков.

Современные игры требуют достаточно большой производительности от компьютера, и не каждая офисная машина в силах воспроизводить их. Однако для отдыха от монотонной работы зачастую достаточно простой, не требовательной к технике, игры.

Множество компьютерных игр разработано сейчас. Открыто множество компаний, где люди трудятся над созданием игр различного геймплея. Одна часть трудится над графикой, вторая над звуком, третья над производительностью, четвертая над созданием деталей. Каждый занят своим делом. И в итоге через долгие месяцы совместной работы выходит какой-нибудь шедевр компьютерной индустрии.

Какие же типы компьютерных игр существуют на данный момент? Сейчас известно 7 типов компьютерных игр:

1) Квесты

Осуществление одного или нескольких персонажей к конечной цели, осуществляя преодоление препятствий.

2) Экшн (action)

Это игры от первого лица. Говоря простым языком: "бродишь и стреляешь".

3) Ролевые игры (рпг)

В этом жанре игрок исполняет роль отдельного персонажа в игре, проигрывает определенный промежуток его жизни. Выполняет поставленные перед персонажем задачи. Обычно в игре такого жанра очень много истории персонажа, мира и его деталей. Такие игры занимают много свободного времени.

Эти три жанра существуют очень популярны. Большинство игр создается именно в этих трех жанрах. По последним подсчетам, эти виды компьютерных игр наиболее популярны в человеческой аудитории. Большой минус таких типов игры, что она вызывает стойкой привыкание. Игрок вживается в своего персонажа, переживает его проблемы как свои, радуется его победе, путает виртуальный мир с реальностью. Разумное привыкание ведет к тому, что игрок начинает читать книги, связанные с этой игрой, изучать историю, развивает фантазию. Также такой жанр игры потрясающе разряжает человека, позволяет ему расслабиться в полной мере. Игры онлайн предпочитают многие пользователи, ведь для того чтобы в полной мере насладиться игрой, нет необходимости её скачивать, достаточно просто зайти на нужный сайт. Особенной популярностью пользуются сайты, на которых не нужна регистрация и где игры доступны совершенно бесплатно.

4) Стратегии

В игре такого жанра отсутствует отдельный персонаж. Там человек управляет определенными внутриигровыми процессами. Сначала используя логику, игрок выполняет свою логическую цепочку решения, а после наблюдает последствия своих действий.

5) Симуляторы.

Достаточно популярный жанр. Суть его в том, что он имитирует управление на автомобиле, самолете, корабле, космическом аппарате. В таких играх нет сюжета, они не развивают какого то привыкания.

6) Логические игры

Идет направленность на мозг человека. Такой вид игры развивает

мыслительную деятельность, подводит человека к собственному логическому размышлению, заставляет креативно мыслить, и строить логические цепочки.

7) Азартные игры

Есть связь с логическими играми. Азартные тоже влияет на мозговую активность, но в отличие от логических, эти игры вызывают страсть или сильное увлечение игровым процессом. Обычно это игры, направленные на победу или проигрыш в определенном турнире. В таких играх все зависит от везения и умения игрока правильно осмысливать и оценивать ситуацию.

Вот 7 основных типов игр. Выбор между видами зависит от игровых предпочтений человека. Каждый жанр игры имеет свою аудиторию. Поэтому лучше выбрать игру по интересу.

Почти все игры также делятся на однопользовательские, с участием одного игрока, как правило в оффлайне, и многопользовательские, в которых одновременно взаимодействуют несколько игроков.

Последние и представляют наибольший интерес сегодня. Почему? Скорее всего - соревновательная система, у игроков появляется еще больше желания развиваться в этих играх, чтобы обойти своих виртуальных соперников и в целом, проходить игру, взаимодействовать с другими игроками гораздо интересней, чем в одиночку.

Последний жанр стал активно развиваться, когда начало появляться стабильное интернет соединение, потому что все взаимодействие и общение игроков происходит чаще всего по интернету.

Именно такой разработке посвящен данный курсовой проект - мйогопольвательская экшен игра.

1. Постановка задачи

1.1 Описание предметной области

По технической реализации многопользовательские игры делятся на

- Многопользовательские на одном компьютере:

1. Игроки участвуют в игре одновременно. В спортивных симуляторах это естественный режим. В других играх, например, гонках, экран делится пополам.
2. Игроки участвуют в игре поочередно. Используется в некоторых пошаговых стратегиях, например шахматы.
3. Один компьютер - несколько терминалов. На данный момент применяется в основном в игровых автоматах.

- Игры по сети. Несколько компьютеров соединены в вычислительную сеть.

1. Через последовательные или параллельные порты.
2. Через модем.
3. Через локальную сеть или интернет, по протоколам IPX или TCP/ IP

4. Онлайн игры

По организации связи делятся на:

1. «Равный с равным». В этом режиме связи нет чётко выделенного главного компьютера; от каждого игрока информация передаётся всем остальным компьютерам. Каждый из компьютеров имеет достоверную информацию об игровом мире. Один из компьютеров обычно является ведущим, его роль ограничивается заданием темпа игры и управлением игрой (смена уровня, изменение настроек игры). Если ведущий выходит, роль ведущего может взять на себя любой другой компьютер. В этом режиме обычно работают стратегии в реальном времени.

- Достоинства: простота реализации; при архитектуре, когда передаются команды управления, игровой мир может быть сколь угодно сложным, но трафик зависит лишь от количества игроков; минимальная нагрузка на ведущего (равняющаяся нагрузке на ведомых); минимальные задержки передачи; при выходе ведущего любой компьютер может взять на себя обязанности ведущего.
- Недостатки: взломав игру, можно следить за остальными игроками; большой трафик при большом количестве игроков; практически невозможно реализовать противодействие некачественной связи; проблемы с входом в начатую игру; требуется установленный канал связи каждого с каждым; высокая нагрузка на ведомые машины.
- Применение: первые сетевые игры, игры для небольшого количества игроков, игры на слабых машинах

2. Звездообразная связь. Архитектура напоминает «равный с равным», однако вся связь ведётся через один центральный компьютер. Является переходным между «равный с равным» и «клиент-сервер».

- Достоинства: простота реализации; малая нагрузка на сеть; можно реализовать противодействие задержкам, передачу ведущего на другой компьютер и вход в начатую игру.
- Недостатки: взломав игру, всё ещё можно следить за остальными игроками; всё ещё высокая вычислительная нагрузка на ведомые машины; более высокие задержки передачи.
- Применение: аналогично одноранговым сетям.

3. Клиент-сервер. Один из компьютеров (сервер) содержит полную и достоверную информацию об игровом мире. Остальным компьютерам

(клиентам) передаётся лишь та доля информации, которая позволяет вести игру и адекватно отображать игровой мир.

- Достоинства: минимальная нагрузка на клиенты; можно установить выделенный сервер (в остальных моделях это бесполезно); наибольшая стойкость к читерству; естественный выбор, когда игровой мир очень велик, но каждый отдельный пользователь имеет дело с небольшой его частью; часто при изменениях в серверной части пользователям не нужно обновлять клиенты.
- Недостатки: сложная реализация; большая нагрузка на сеть, когда игровой мир насыщен активными объектами, миграция сервера на другой компьютер практически невозможна; высокая задержка между нажатием клавиши и действием.
- Применение: большинство современных игр (за исключением стратегий, на которых выгодно применять одноранговую или звездообразную связь).

Остановимся подробнее на сетевой клиент серверной онлайн игре поддерживающей одновременное взаимодействие большого количества игроков (от 2 до 100). Такие игры получили название IO игры.

1.2 IO игры

Так называемые .io-игры — это браузерные massively multiplayer action-игры, в которых множество людей борются с излишками свободного времени. Massively multiplayer — это значит, что игра представляет собой многопользовательскую массовку из большого количества (сотен+) игроков, играющих в общей локации. Существует мнение, что все началось с игры Agar.io (клетки в чашке Петри). Другими успешными примерами можно назвать Slither.io (змейки) и Diep.io (танчики). Если верить статистике, то каждый день в эти игры играют миллионы игроков. Сегодня существуют

десятки различных .io-игр, большинство из которых можно найти, загуглив по запросу «io game list».

В 2015 году на просторах интернета внезапно появились ИО игры. Это многопользовательские браузерные игры. Давай разжую. Помнишь, как долго ты искал, где скачать игру. Потом качал и устанавливал её. Вот уже твой обеденный час кончился, а ты так и не поиграл. Тут всё по другому, ставить не чего не надо качать тоже просто запустил специализированный сайт на котором есть игры ио и отправился захватывать галактики, отстреливаться от зомби или играть в змейку с другими игроками в режиме онлайн. Это ли не чудо.

Первой игрой этого жанра была Agar.io она то как раз и вышла в 2015 году. А дальше рынок заполнили разработчики ИО игр, и теперь ты можешь найти тут всё что душе угодно. Многопользовательский режим добавил особую вишенку на торт. Ведь теперь не надо гонять ботов по карте. Гораздо интереснее играть с живыми людьми.

В 2019 году сложно уже удивить графикой или спецэффектами тут ставка на геймплей. Отличие таких игр заключается в небогатой графике и соревновательном духе. Пройти ИО игру невозможно, зато есть возможность побеждать в каждом раунде и возглавить рейтинг лучших игроков. Соревновательный дух во всей красе. Благодаря маленьким требованиям ты можешь запускать эти игры как на рабочем телефоне пока босс не видит, так и на мамкином планшете.

Продуманная игровая механика ИО игр впечатляет. Ещё бы, когда тебе не надо тратить время на графику ты занимаешься реально нужными вещами. Баллистика, гравитация, повреждения всё это продумано до мелочей. Приятно играть.

1.3 Цели и задачи проектирования

Подчеркнем, что целью данного проекта ставится создание как раз похожей IO игры.

Игра будет браузерная, с возможностью подключения большого количества пользователей.

Так как игроков будет много велик шанс взлома игры и получения доступа к серверному компьютеру. Поэтому будет небольшой упор на безопасность.

Также необходимо предусмотреть, чтобы никакие игроки не могла вмешиваться во внутреннюю механику игры, менять какие-то серверные данные, к которым у них не должно быть доступа, так как игры будет браузерная, а все браузеры имеют встроенную консоль, через которую можно немного переопределять поведение игры в лучшую для себя сторону. Это так называемое читерство, которые мы тоже постараемся избежать обрабатывая полностью игровой процесс на так называемом сервере.

Поэтому нужно будет создать сильный мощный сервер, в котором будет вся логика игры, обработка всей механики.

И легкого клиента, чтобы даже пользователи с плохим интернет соединением могли без проблем и задержек играть в нашу игру. Для этого нужно задуматься о количестве передаваемой информации с сервера на клиент, и как можно сильнее минимизировать ее. Ведь чем меньше данных необходимо подгружать пользователю каждую секунду, тем более слабое соединение мы сможем поддерживать.

1.4 Требования к разрабатываемому программному обеспечению

Веб-игра принципиально отличается и от обычной компьютерной игры и от обычного сайта в браузере. Обычной игре все игровые ресурсы доступны

оффлайн, игровому движку известны сведения о процессоре, памяти и видеокарте компьютера. Обычному сайту требуется немного ресурсов компьютера, а в случае неполадок, можно просто перезагрузить страницу.

Предположения об особенностях браузерной — значительные ограничения на доступный и используемый размер оперативной памяти, баланс между качеством игровых ресурсов (изображения, текстуры) и скоростью их скачивания.

К этому добавились требования от клиента — игра должна запускаться и работать во всех заявленных десктопных браузерах, на минимально возможных аппаратных характеристиках.

Готовый программный продукт должен удовлетворять следующим требованиям:

1. Иметь простой и понятный веб-интерфейс
2. Не должно возникать больших задержек при получении данных с сервера
3. Клиентская часть должна быть предельно простая, чтобы любой игрок попавший на сайт, смог без проблем установить связь с сервером и начать собственно игру.
4. Сервер должен использовать эффективные алгоритмы работы с данными, ведь наша игра в перспективе может поддерживать вплоть до 100 пользователей, поэтому необходимо быстрая обработка данных, генерация новых и отсылка по всем пользователям
5. Игра должна быть интересной, но в тоже время простой, своего рода убивалкой времени, не требующей большого порога входа.
6. Систему прокачки игрока, чтобы ему было интересней играть и затягивала его.
7. Продуманный баланс. Под балансом понимается то, что на какую ветку прокачки не сделал бы упор игрок, он примерно будет такой же по силе как и другие, чтобы не было сильного доминирования одних игроков над другими, а примерно равные шансы на выживание.

8. Интересную механику.

9. Обработку всех исключительных ситуации. Таких как резкий выход из строя сервера или потерю соединения на стороне клиента

10. Возможность создания аккаунта, для дальнейшего сохранения игровых результатов иди в перспективе, кастомизации своего персонажа и сохранения всех изменений.

11. Возможную систему ролей, чтобы определенные пользователи, например админы, могли в ран тайме, следить за состоянием игры, и вносить какие-нибудь изменения в игровой процесс.

Таким образом, задачей данного курсового проекта сводится к разработке браузерной онлайн игры на клиент-серверной архитектуре, для поддержания и игры большого количество пользователей (IO игры). Готовое приложение должно иметь понятный интерфейс и быть удобным в использовании.

2. Проектирование игры

2.1 Игровой процесс

Для начала нужно было решить, о чем будет игра. Обдумав игровую механику двух наиболее успешных представителей — Агарио и Диепио — было решено, что игра должна обладать следующими свойствами:

1. Управляемый персонаж должен набираться сил постепенно в течение длительного времени, и этот набор сил должен отражаться визуально.
2. Нужно чтобы был смысл играть очень долго — несколько часов подряд. Игрок, проигравший 2 часа, должен иметь определенное преимущество перед тем, кто проиграл 5 минут.
3. Игровое преимущество не должно зависеть исключительно от времени, проведенного в игре. Должен быть разумный баланс между отыгранным временем и игровым мастерством. Человек, отыгравший несколько минут, должен иметь возможность победить того, кто играет уже несколько часов. Пусть это будет очень сложно, пусть здесь понадобится фактор удачи, но такая возможность должна быть.
4. Бонусы, полученные в результате победы на другим игроком, не должны безоговорочно доставаться победителю — у других игроков должна быть возможность отхватить их часть. (Пока в разработке)
5. Каждый игрок мог бы сам выбирать уникальную прокачку и постепенно улучшать свой “Пузырь”. Поэтому при получения нового уровня, игрок может выбрать, чтобы он хотел улучшить в своем герое. На выбор предоставляется:
    1. Уровень Здоровья
    2. Скорость регенерации здоровья
    3. Скорость игрока
    4. Дальность стрельбы
    5. Скорость стрельбы
    6. Наносимый урон

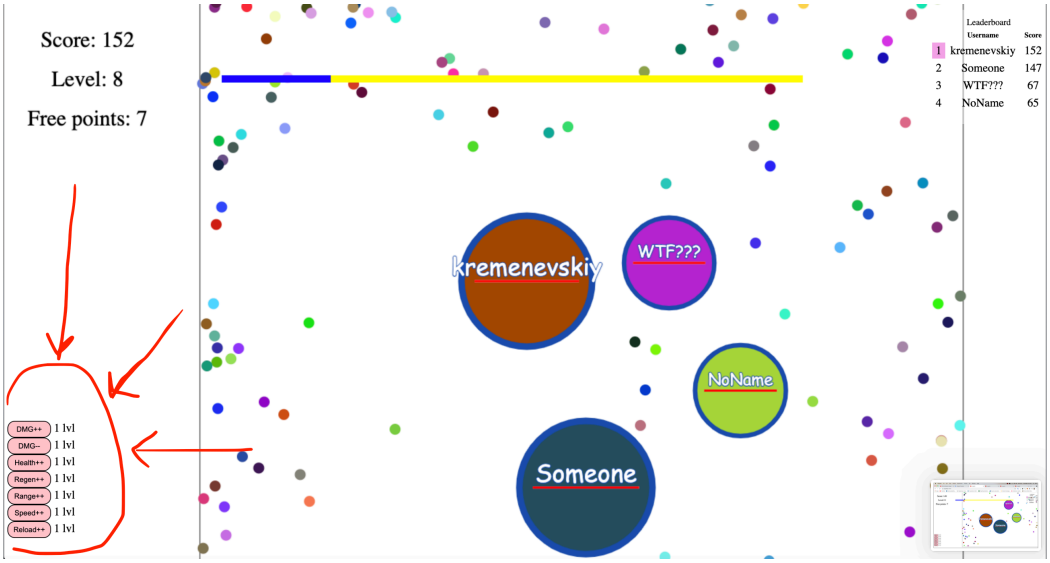

Рис. 1 Выбор системы прокачки в игре

Суть игры заключается в том, чтобы уничтожить как можно больше других игроков и забрать все их очки, в то же время оставшись самому в живых.

Прокачивать своего персонажа можно съедая маленькую еду, которая рандомно генерится на карте или уничтожая других игроков.

Само взаимодействие игроков осуществляется путем создания постоянной угрозы своим оппонентам. Убивать других игроков, можно просто съедая их, если масса нашего шара на 20% больше массы противника, непосредственно делать это можно, предварительно прокачав ветку скорости и здоровья или стреляю по ним, но важно помнить, что стреляя, вы отдаете свою массу, поэтому на каждом этапе игры у игрока есть выбор попробовать убить противника патронами или съесть или вообще уйти и не подставлять себя под угрозу.

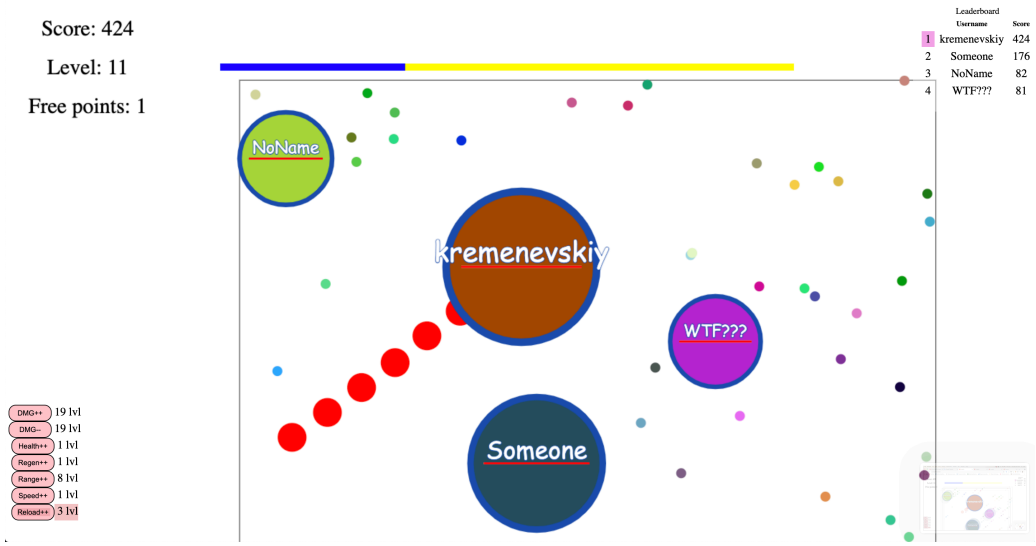

Рис. 2 Анимация стрельбы

2.1 Проектирование архитектуры приложения

На рисунке представлена спроектированная схема серверной и клиентской части приложения на языке uml. На ней можно видеть сущности и типы связей

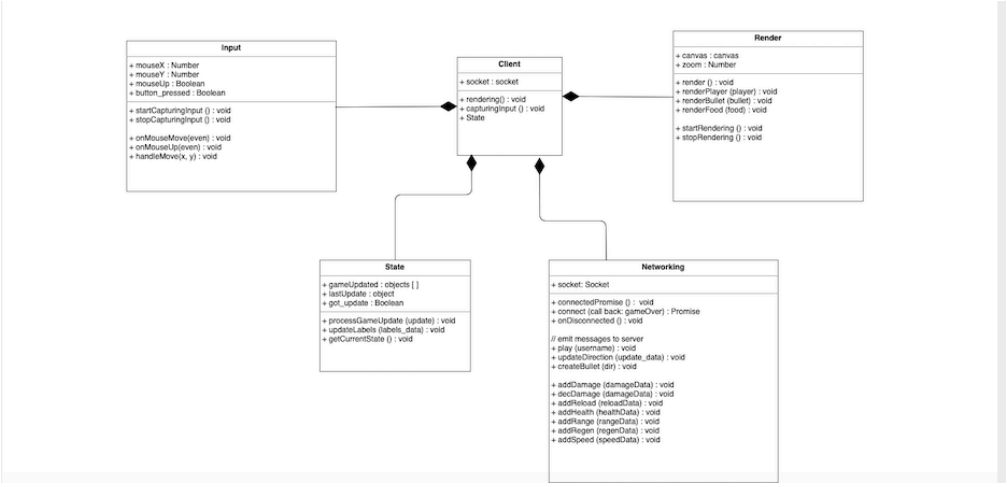

Рис. 3 Uml схема архитектуры клиента

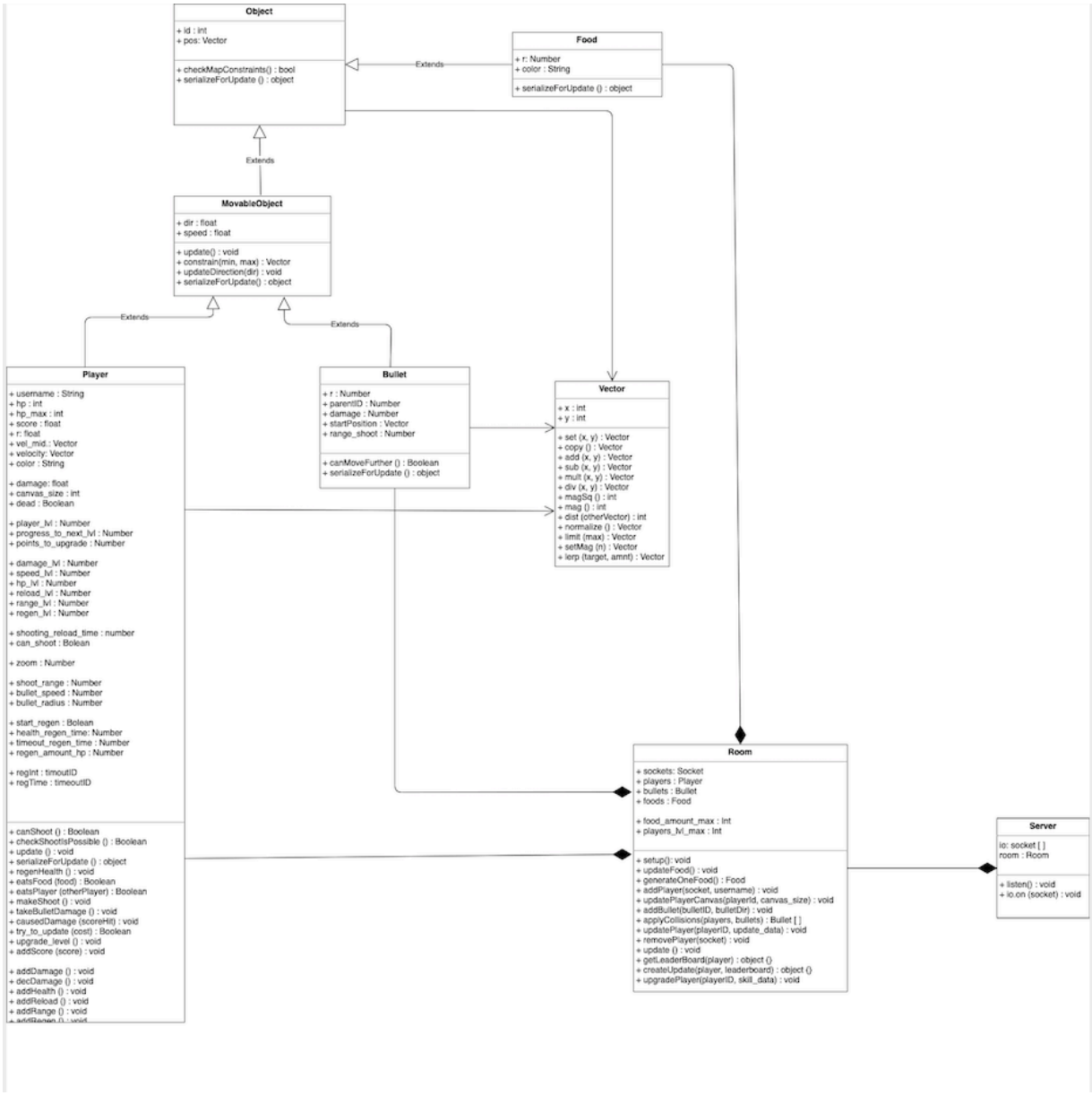

Рис. 3 Uml схема архитектуры сервера

2.2 Стек технологий

Приложение было разработано на JavaScript с использованием Run-time среды выполнения Node JS. В качестве используемой базы данных была выбрана MongoDb. В качестве среды разработке был выбран WebStorm.

1. Node JS - весь бэкэнд.
2. HTML + CSS - верстка сайта

3. WebSockets - создание сети и взаимодействия клиента и сервера. JavaScript + NodeJs - мощнейшая связка для создания веб приложений любой сложности. Достигается за счет возможности подключения внешних библиотек непосредственно с помощью Node JS, которые позволяют найти и подключить к приложению почти любой необходимый функционал.

3. Программная реализация

В данном разделе будет рассмотрена архитектура приложения, а также будут приведены примеры некоторых пользовательских функций и хранимых процедур.

3.1 Физическая структура приложения

Как говорилось ранее, при разработке приложение была использована клиент-серверная архитектура.

Это значит, что у нас имеется один мощный сервер, который запускает приложение, устанавливает соединение с базой данных, и в дальнейшем ожидает подключения новых пользователей. Сервер запускается на специально заданным порту и айпи адресу.

Production / development mode.

Сервер позволяет развернуть приложение в двух режимах:

1. production mode.
2. Development mode.

Установка режима development обеспечивает минимальное логирование системы, обрабатывая только критические ошибки, а также для повышения этапа сборки приложения.

Например, библиотека шаблонов, используемая Express, компилируется в режиме отладки, если **NODE_ENV** не установлен как **production**. Шаблоны Express компилируются в каждом запросе в режиме разработки, в то время как в production версии они кэшируются.

Также файлы с расширением .js, .html, .cs сильно сжимаются без потери информации и обрабатываются в браузере гораздо быстрее в режиме production, но не содержат абсолютно никакой информации для отладки программы.

Сервер — это приложение, внутри которого и происходит вся игра. Сервер состоит из игровых комнат. Игровая комната — это экземпляр игровой локации. Комната имеет адекватные для геймплея геометрические размеры и лимит на количество игроков в ней.

3.2 Физическая структура клиента

Игровой клиент — это браузер с открытой в нем html-страницей. Технологии те же, что и у большинства других .io-игр, а именно: язык программирования — JavaScript, графика — HTML5 Canvas + JS, взаимодействие с игровым сервером — через WebSockets. Сложилось, что .io-игры не принято озвучивать (хотя все технические возможности для этого имеются), поэтому было решено, что и наша игра будет без звука.

Упрощенно говоря, вся движуха происходит на сервере, а клиент является лишь монитором-визуализатором этой движухи. Все что умеет делать клиент помимо пассивного визуализирования — это отправлять на сервер действия игрока (в нашем случае — координаты курсора мыши и состояния ее кнопок). Больше ничего клиент не умеет, и не должен уметь — никакая игровая логика не должна выполняться на клиенте, иначе это моментально станет возможностью для читерства. Таким образом, с программной точки зрения, клиент — это объект, который, среди прочего

1. Умеет отправлять на сервер действия пользователя.
2. Умеет принимать с сервера сообщения и диспетчеризировать их.
3. Хранит у себя копию игрового мира, попадающую в его поле зрения.
4. Визуализирует хранимую копию игрового мира.

При загрузке сайта с игрой, пользователи попадает на html следующую html страницу, на которой он может либо создать аккаунт либо начать игру.

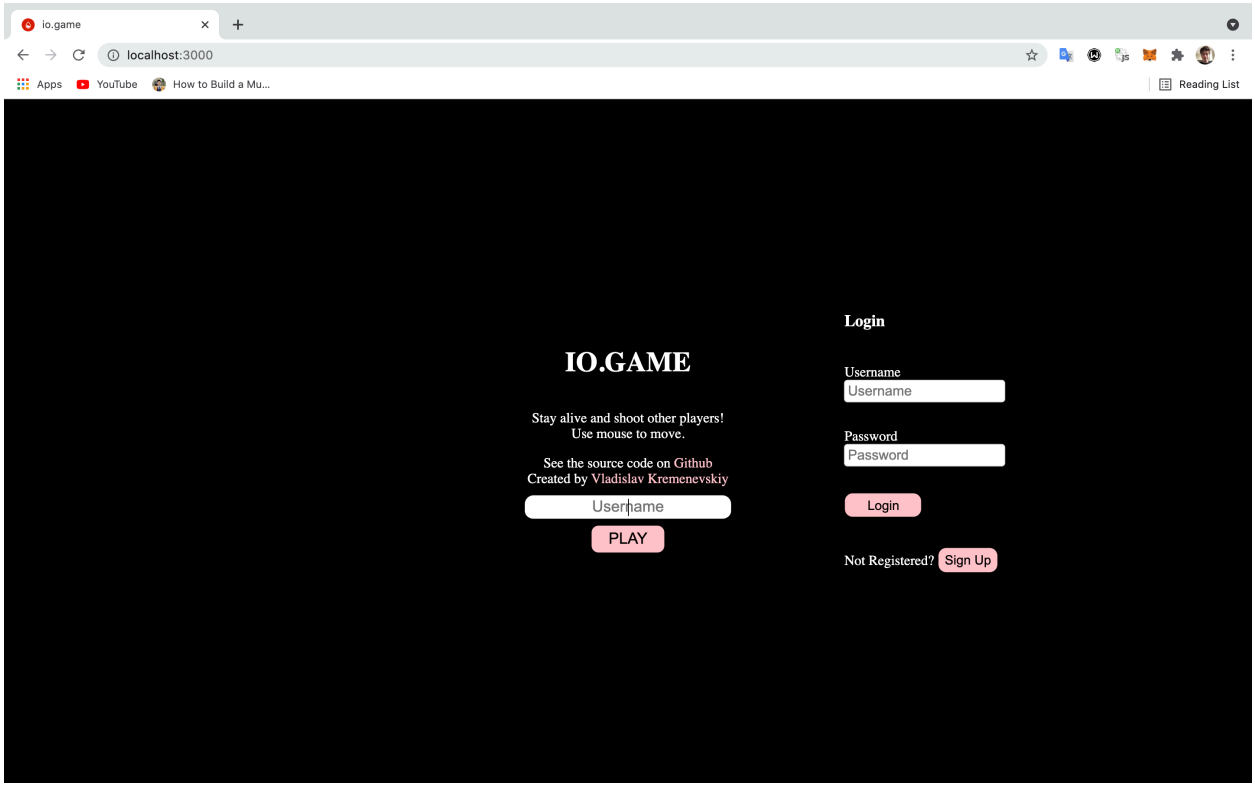

Рис 5. Стартовая html страница

Клиенту не нужно заботиться о том, как все происходит под капотом, что существует какой-то сервер, который все обрабатывает, все что от него необходимо - нажать кнопку PLAY и начать играть. При нажатии PLAY пользователю становится доступен канвас с возможностью передвигаться по игровой карте.

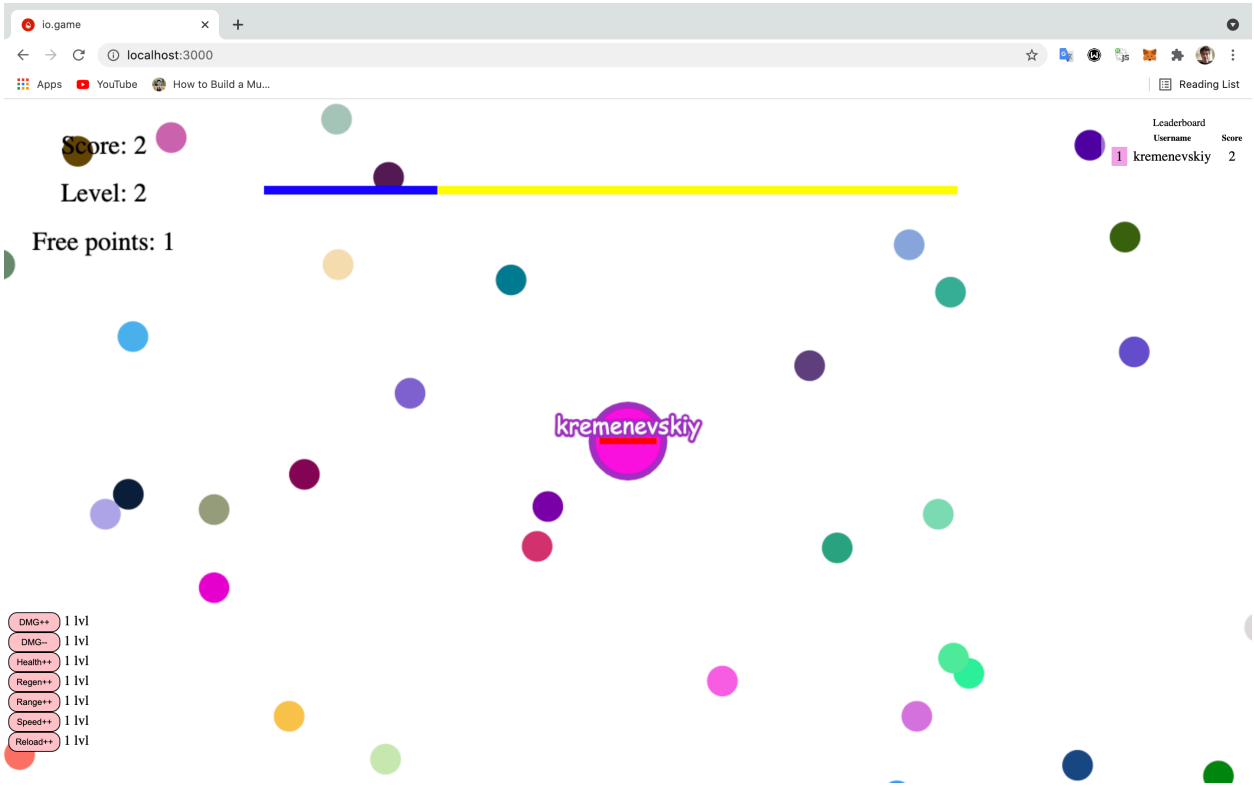

Рис 6. Начало игры

3.3 Физическая структура сервера

При запуску сервера, на нем запускается express сервер. Который позволяет устанавливать соединение с другими игроками, заходящими на сайт.

Далее, при подключении клиента, устанавливается соединение между сервером и клиентом через WebSocket, который позволяет общаться и взаимодействовать серверу и клиенту между собой путем отправки различных данных по вебсокету и правильной расшифровке их.

Рис. 7 WebSocket

Сокет непосредственно создается на стороне клиента, и еще одно часть, с которой он общается на стороне сервера, далее на сервере заносится в коллекцию, чтобы в любой момент иметь возможность отослать необходимую информацию по обновлению мира или дисконект.

3.4 Архитектура серверной части

При запуске сервера, одновременно с ним создается объект класса Room, который пожалуй, один из важнейших классов в нашей игре, в котором хранится информация о всех пользователях, соединенных с ними сокетов, всех объектах на карте, всей еды, всех пуль.

```
io.on('connection', newConnection);
function newConnection(socket){
    console.log("New client: " + socket.id);
    socket.on(Constants.MSG_TYPES.JOIN_GAME, joinGame);
    socket.on(Constants.MSG_TYPES.UPDATE_INPUT, updatePlayer);
    socket.on(Constants.MSG_TYPES.NEW_BULLET, addBullet);
    socket.on(Constants.MSG_TYPES.CANVAS_GET, setCanvasSize);

    // update players
    socket.on(Constants.MSG_TYPES.DAMAGE_ADD, damage_add);
    socket.on(Constants.MSG_TYPES.DAMAGE_DEC, damage_dec);
    socket.on(Constants.MSG_TYPES.HEALTH_ADD, health_add);
    socket.on(Constants.MSG_TYPES.REGEN_ADD, regen_add);
    socket.on(Constants.MSG_TYPES.SPEED_ADD, speed_add);
    socket.on(Constants.MSG_TYPES.RELOAD_ADD, reload_add);
    socket.on(Constants.MSG_TYPES.RANGE_ADD, range_add);

    socket.on('disconnect', onDisconnect);
}

const room = new Room();
room.setup();
```

Рис 8. Запуск сервера

```
class Room {
    constructor() {
        this.sockets = {}; // словарь подлюченных сокетов
        this.players = {}; // словарь подлюченных игроков
        this.bullets = []; // все живые пули
        this.food_amount_max = Constants.MAP_SIZE / 3;
        this.foods = []; // вся еда на карте
        setInterval(this.update.bind(this), timeout: 1000/60); // устанавливаемм частоту посылок данных на клиента
        setInterval(this.updateFood.bind(this), timeout: 1000/5); // частота добавления игры на карту
        this.players_lvl_max = 20; // максимальный уровень игрока
    }
```

Рис. 9 Создание объекта Room для игры

Игровая комната

Здесь происходит вся игровая логика. Комната — это объект, который, среди прочего:

1. Хранит контейнеры игровых объектов: существ, еды, пуль.
2. Умеет принимать от клиентов сообщения и диспетчеризировать их.
3. Умеет уведомлять клиентов о событиях, наступивших в комнате. Некоторые из уведомлений отправляются только если данные события наступили в поле зрения клиента.
4. Реализует функцию перехода комнаты из состояния An в состояние An+1 (апдейт игрового мира).
5. Периодически выполняет апдейт игрового мира с течением времени (в нашем случае — 60 раз в секунду).

Апдейт игрового мира

Это функция, которая реализует изменения в игровом мире, произошедшие за фиксированный отрезок времени dt (в нашем случае — 16 миллисекунд), будем называть эту функцию update. В общих чертах update игровой комнаты — это вызов update для игровых объектов, то есть вызов update далее вниз по лестнице агрегации, от общего к частному. Простой пример: пусть у нас есть игрок, находящаяся в координатах position, и двигающийся с вектором скорости velocity. Тогда ее функция update — это вычисление ее позиции через dt времени:

```
update(){
    var vel = new Vector(this.vel_mid.x, this.vel_mid.y)
    vel.div( x: 10);
    vel.limit( max: 4 + this.speed * 0.2);
    this.velocity.lerp(vel,  amnt: 0.1);
    this.pos.add(this.velocity);
    this.constrain(-Constants.MAP_SIZE + this.r, Constants.MAP_SIZE - this.r);
}
```

Рис. 10 Функция обновление положения игрока на карте

Также для пуль:

```
update() {
    this.pos.x += Math.cos(this.dir) * this.speed;
    this.pos.y += Math.sin(this.dir) * this.speed;
}
```

Рис. 11 Функция обновления координат поли

Далее каждые 60 кадров в секунду, сервер прогоняет функцию **update(),** цели которой:

1. Посчитать новые координаты всех игроков
2. Новые координаты всех пуль
3. Проверить не вышли ли какие-нибудь пули за пределы карты
4. Проверить были ли столкновения каких-нибудь пуль с игроками, чтобы обновить очки и здоровья игрокам.

```
applyCollisions(players, bullets){
    const destroyedBullets = [];
    for(let i = 0; i < bullets.length; ++i) {
        for(let j = 0; j < players.length; ++j) {
            const bullet = bullets[i];
            const player = players[j];

            if (bullet.parentID !== player.id &&
                player.pos.dist(bullet.pos) <= player.r + bullet.r){
                destroyedBullets.push(bullet);
                player.takeBulletDamage(bullet.damage);
                // console.log('player got damage ' + player.id + " from: " + this.players[bullet.id].id)
                break;
            }
        }
    }
    return destroyedBullets;
}
```

Рис 12. функция applyCollisions()

5. Проверить нет ли умерших игроков и если есть удалить их.

6. Проверить ни ест ли какой-нибудь игрок других игроков.

7. Разослать новую информацию о всех объектах на карте всем живым игрокам

И тут возникает проблема. Дело в том что если на сервере большое количество игроков, большой размер карты, активная игра, тогда количество объектов на карте становится предельно большим. И если каждому игроку высылать полный объем данных могут возникнуть следующие проблемы:

1. Клиентам понадобится очень хорошее соединение, чтобы успевать принимать так много новых данных.
2. Также если сервер находится достаточно далеко от клиента, то вероятность того, что какой-то кусочек не дойдет до клиента гораздо больше, и придется повторять ве повторно, а это потеря времени, нам важна каждая миллисекунда

Решением этой проблемы стало отсылание каждому пользователю информации об объектах, находятся в пределах видимости данного игрока, то есть конкретно на экране данного игрока. Остальные объекты не будут

отсылать, ведь они попросту не нужны. Зачем игроку получать информацию о том, что происходит на другом конце карты, если это никак его не касается?

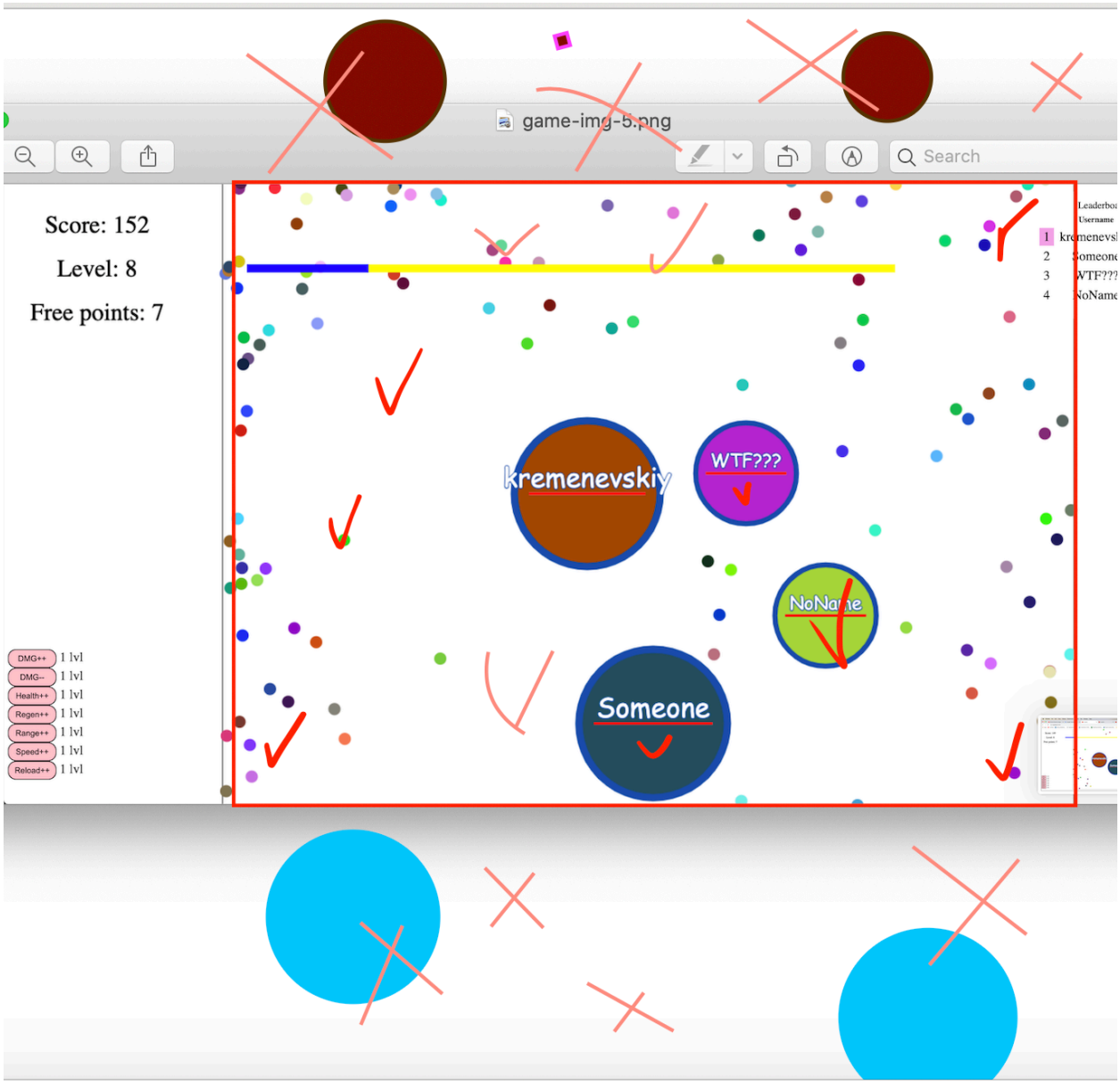

Рис. 13 Отправка обновления kremenevskiy только красного прямоугольника

```
createUpdate(player, leaderboard){
    // console.log('sending update');
    var data = {
        me: player.serializeForUpdate(),
        others: Object.values(this.players).map(p => p.serializeForUpdate()),
        bullets: this.bullets.map(b => b.serializeForUpdate())
    };


    let newZoom = 30 / player.r;
    player.zoom = lerp(player.zoom, newZoom, t: 0.1);


    // const visible_dist = Constants.MAP_SIZE * 4;
    const visible_dist = player.canvas_size / 2 * Math.sqrt( x: 2) / (player.zoom + 0.5);
    const nearbyPlayers = Object.values(this.players)
        .filter(p => p!==player && p.pos.dist(player.pos) <= visible_dist);
    const nearbyBullets = this.bullets.filter(b => b.pos.dist(player.pos) <= visible_dist);
    const nearbyFood = this.foods.filter(f => f.pos.dist(player.pos) <= visible_dist);

    return {
        me: player.serializeForUpdate(),
        others: nearbyPlayers.map(p => p.serializeForUpdate()),
        bullets: nearbyBullets.map(b => b.serializeForUpdate()),
        food: nearbyFood.map(f => f.serializeForUpdate()),
        leaderboard: leaderboard
    }
}
```

Рис. 14 Функция генерации данных обновления конкретному пользователю

Однако за это нам приходится платить более высокой нагрузкой на сервер, ведь перед каждой отсылкой обновления необходимо перебрать все имеющиеся объекты и посчитать расстояние до текущего, к которому идет новая информация. Но предполагается, что наш серверный компьютер очень мощный.

3.5 Архитектура клиентской части

Далее все обновления отсылаются на клиента с частотой отправки указанной на сервере.

Все эти обновления сохраняются в массиве обновлений на клиенте и при необходимости достается нужное последнее. Если уже имеются новые, а

старые все ровно еще не успели обработаться, то они удаляются, потому что нам не нужно хранить старые обновления, если у нас уже есть более новое.

```
const gameUpdates = [];
var lastGameUpdate = null;
var got_update = false;

export function processGameUpdate(update) {
    got_update = true;
    lastGameUpdate = update;
    updateLeaderboard(update.leaderboard)
    if (update.me.update_points < 1) {
        upgradeMenu.classList.add('disabled');
    }
    else {
        upgradeMenu.classList.remove( tokens: 'disabled');
    }
    // console.log('got new update from server')
    // console.log(update)
}


export function updateLabels(labels_data){
    update_lvl_labels(labels_data);
}


export function getCurrentState() {
    if (!got_update) {
        return false
    }
    return lastGameUpdate;
}
```

Рис. 15 Обработка пришедших обновлений

И после этого с указанной на клиенте скоростью перерисовки (кадрах в секунду), которые могут отличаться от тех, что установлены на сервере, клиент берет последние пришедшие данные и выполняют функцию отрисовки всех необходимых объектов. За это отвечают функции render(), renderBullet(bullet), renderFood(food), renderPlayer(player).

```
function renderPlayer(player) {
    // draw player
    c.beginPath();
    c.arc(player.position.x, player.position.y, player.r,  startAngle: 0,  endAngle: Math.PI * 2,  anticlockwise: false);
    c.fillStyle = player.color;
    c.fill();
    if (player.r < 30){
        c.lineWidth = 4;
    } else{
        c.lineWidth = player.r / 10;
    }
    c.strokeStyle = strokeColor;
    c.stroke();

    // draw health bar
    c.beginPath();
    c.fillStyle = 'white';
    c.fillRect( x: player.position.x - player.r * 0.8,  y: player.position.y - 2,  w: player.r * 2 * 0.8,  h: 4);
    c.fillStyle = 'red';
    c.fillRect( x: player.position.x - player.r * 0.8,  y: player.position.y - 2,  w: player.r * 2 * 0.8 *
        ((1 - ((player.hp_max - player.hp) / player.hp_max))),  h: 4);
    c.fillStyle = player.color;
    c.textAlign = 'center';

    // draw nickname
    c.beginPath();
    let font_size = Math.floor((player.r / 2.7)).toString() + 'px';
    let font = " Comic Sans MS";

    c.font = player.r > 35 ? font_size + font : "15px Comic Sans MS";
    let text_offset = player.r < 30 ? 5 : 10;

    c.strokeStyle = strokeColor;
    c.lineWidth = 4;
    c.strokeText(player.nickname, player.position.x,  y: player.position.y - text_offset);
    c.fillStyle = 'white';
    c.fillText(player.nickname, player.position.x,  y: player.position.y - text_offset);
}
```

Рис. 16 Функция отрисовки игрока

```
function renderBullet(bullet) {
    c.beginPath();
    c.arc(bullet.position.x, bullet.position.y, bullet.r,  startAngle: 0,  endAngle: Math.PI * 2,  anticlockwise: false);
    c.fillStyle = 'red';
    c.fill();
}
```

Рис. 17 Функция отрисовки снаряда

```
function renderFood(food){
    c.beginPath();
    c.arc(food.position.x, food.position.y, food.r,  startAngle: 0,  endAngle: Math.PI * 2,  anticlockwise: false);
    c.fillStyle = food.color;
    c.fill();
}
```

Рис. 18 Функция отрисовки еды

```
function render() {
    if (!getCurrentState()){
        return;
    }
    const {me, others, bullets, food} = getCurrentState();

    c.clearRect( x: 0,  y: 0, canvas.width, canvas.height);
    c.save();
    c.translate( x: canvas.width/2,  y: canvas.height/2);
    let newZoom = 30  / me.r;
    zoom = lerp(zoom, newZoom,  t: 0.1);

    c.scale( x: zoom + 0.5,  y: zoom + 0.5);
    c.translate(-me.position.x, -me.position.y);

    // draw boundaries
    c.fillStyle = 'black';
    c.lineWidth = 1;
    c.strokeRect(-Constants.MAP_SIZE, -Constants.MAP_SIZE,  w: Constants.MAP_SIZE*2,  h: Constants.MAP_SIZE*2);

    food.forEach(foody => renderFood(foody));
    bullets.forEach(bullet => renderBullet(bullet));

    others.forEach(player => renderPlayer(player));
    renderPlayer(me);

    c.restore();

    // draw score_to_next lvl

    c.beginPath();
    c.fillStyle = 'yellow';
    c.fillRect( x: 300,  y: 100,  w: 800,  h: 10);
    c.fillStyle = 'blue';
    c.fillRect( x: 300,  y: 100, w: 800 *  (1 - ((Math.pow(me.lvl,  y: 2) - me.score_to_next_lvl) /
        Math.pow(me.lvl,  y: 2))),  h: 10);

    // Draw score
    c.beginPath();
    c.fillStyle = "black";
    c.textAlign = "center";
    c.font = '30px serif';
    let score_msg = "Score: " + me.score.toString();
    // console.log("mess: " + score_msg);
    c.fillText(score_msg,  x: canvasWidth * 0.08,  y: canvasHeight * 0.08);
    c.fill();

    // draw level
    c.beginPath();
    c.fillStyle = "black";
    c.textAlign = "center";
    c.font = '30px serif';
    let level_msg = "Level: " + me.lvl.toString();
    c.fillText(level_msg,  x: canvasWidth * 0.08,  y: canvasHeight * 0.15);

    // draw free point to update
    c.beginPath();
    c.fillStyle = "black";
    c.textAlign = "center";
    c.font = '30px serif';
    let points_msg = "Free points: " + me.update_points;
    c.fillText(points_msg,  x: canvasWidth * 0.08,  y: canvasHeight * 0.22);
}
```

Рис. 19 Функция рендеринга игрового мира

3.6 Проблема отсылки данных от пользователя

Также в начале написания приложения взаимодействие клиента и сервера было организовано таким образом, что новое местоположение клиента, подсчитывалось непосредственно на стороне клиента, а это как мы уже поняли очень плохо. Потому что любой игрок имеющей доступ к браузерной консоли сможет отослать на сервер неверную информацию.

Почему это плохо? Предположим, что игрок находится в точке (20, 20), вы ожидаем, что на сервер придет через пару миллисекунд придет новая координата, например (18, 22). Но вдруг нам приходит (500, 500). И данное положение транслируется по всем игрокам и в том числе текущем, получается что игрок просто телепортировался. Это так называемое читерство в играх.

Однако было придумано решение данной проблемы.

Вместо того, чтобы отсылать с клиента новые координаты игрока на сервер, мы будем хранить их на стороне сервере, и каждый заданный интервал передавать только данные об изменении положения курсора на карте. И с учетом нового положения курсора на стороне сервера посчитаем новые координаты игрока и отошлем ему же и всем остальным.

Это решение является безопасным, потому что от положения мыши на экране меняется только арктангенс направления игрока, а игрок и так и так может двигаться в любом направлении.

```
function handleMove(x, y) {
    // console.log('hangle move');
    // console.log('size: ' + window.innerWidth/2 + " " + window.innerWidth/2)
    const angle = Math.atan2( y: y - canvasHeight / 2,  x: x - canvasWidth / 2);
    var update_data = {
        dir: angle,
        vel_mid: {
            x: mouseX - window.innerWidth / 2,
            y: mouseY - window.innerHeight / 2
        }
    }
    updateDirection(update_data);
}
```

Рис. 20 Обработка движения мыши

Далее эти безопасные данные отправляются на сервер и там производится перерасчет новых координат.

4. Развертывание игры

Сначала необходимо установить все необходимые инструменты для запуска сервера игры.

Весь код приложения можно скачать по ссылке: https://github.com/kremenevskiy/io.game .

Далее необходимо скачать Node Js на компьютер и выполнить команду

>>> npm install

Это необходимо для того, чтобы скачать и установить все необходимые библиотеки, используемые в приложении.

Запуск выполняется в 2 этапа.

1. Выбираем режим запуска сервера, production или development. Далее в зависимости от этого пишем в терминале

Запуск в режиме разработки

>>> npm run all

Запуск в режиме продакшена

>>> npm run prod

2. По указанному в функции server.js айпи и порту переходим на соответствующий IP::PORT в браузере (PORT: 3000 - по умолчанию).

Например, запустив предварительно сервер по адресу localhost:3000 или 127.0.0.1:3000 в строке браузера просто пишем:

>>> localhost:3000

Ну или другой выбранный айпишник до этого. Далее можно звать друзей и начинать покорять игровое пространство.

Заключение

В рамках работы над курсовым проектом была разработана многопользовательская ИО игра. Разработан мощный сервер, способный обрабатывать всю логику игры и привлекательная клиентская версия для браузера.

За время написания игры был изучен язык JavaScript, а его рантайм среда NodeJS. Изучена архитектура построения и развертывания современных браузерных игр.

Также был изучен большой объем информации по проектированию и администрированию баз данных.

Были проанализированы различные подходы и технологии, из которых были выбраны те, которые являются наиболее устойчивыми и безопасными.

Разработанное программное средство представляет собой законченный программный продукт, готовый к использованию. Но при желании приложение можно доработать: расширить функциональность приложения, изменить дизайн, добавлять новые уровни игры, дорабатывать логику, добавить магазин скинов и вывести приложение для получения дохода.

Хорошим дополнением было бы добавление ботов. Причем боты должны заходить на сервер по сети, по тому же протоколу, что и клиенты, и «видеть» ту же информацию, что и реальные игроки. Это существенно облегчит отладку, а также позволит оценить потребляемые ресурсы в условиях, приближенных к боевым. Сколько всего игроков онлайн потянет сервер? Когда мы упремся в CPU или ширину сетевого канала? На эти и некоторые другие вопросы вам помогут ответить боты, тем самым избавив нас от неприятных сюрпризов после релиза. Кроме того, боты позволят сделать игру не слишком скучной, пока в ней мало живых игроков.

Также было бы неплохо отслеживать сетевой трафик. Обычно у хостинг-провайдеров трафик либо платный по счетчику, либо предоплаченный пакет с возможностью докупки, либо безлимитный, но со звездочкой-сноской, по которой находится оговорка, что на самом деле не такой уж он и

безлимитный — при исчерпании определенного объема вас будут ждать карательные меры в виде, например, урезания ширины канала, или же придется таки платить сверху. Поэтому необходимо продумать протокол и экономить каждый байт! У нас может быть до 10000 живых игроков, отыгравших за сутки, которые могут за эти сутки генерировать порядка 200 гигабайт игрового трафика. Наверно это может показаться не так уж много, но нужно помнить, что объем потребляемого трафика будет расти вместе с аудиторией. Несколько сэкономленных байт в отдельно взятом пакете могут сэкономить сотни мегабайт или даже гигабайты трафика в сутки.

Список используемых источников

1. https://nodejs.dev/learn
2. https://mongoosejs.com/docs/