

Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное

учреждение высшего образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра автоматизированных систем управления (АСУ)

## ГРАФИЧЕСКИЙ УЧЕБНЫЙ WEB ИСПОЛНИТЕЛЬ (WEB ГРИСП)

Отчет по результатам учебной практики:

получение первичных навыков научно-исследовательской работы (рассред.)

(тип практики)

Обучающийся гр. 439-3  
(группа)

А. С. Мазовец  
(подпись) (И. О. Фамилия)

« \_\_\_\_\_ » \_\_\_\_\_ 2020 г.  
(дата)

Руководитель практики  
от Университета:

ассистент каф. АСУ  
(должность, ученая степень, звание)

А. Е. Косова  
(подпись) (И. О. Фамилия)

\_\_\_\_\_ « \_\_\_\_\_ » \_\_\_\_\_ 2020 г.  
(оценка) (дата)

Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное  
учреждение высшего образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра автоматизированных систем управления (АСУ)

Утверждаю

и. о. зав. кафедрой АСУ

канд. тех. наук, доцент

(должность, ученая степень, звание)

В. В. Романенко

(подпись)

(И. О. Фамилия)

« \_\_\_\_ » \_\_\_\_\_ 2020 г.  
(дата)

## ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ

на учебную практику:

получение первичных навыков научно-исследовательской работы (рассред.)

(тип практики)

Студенту гр. 439-3 факультета систем управления

(группа)

(наименование направления подготовки, специальность)

Мазовцу Александру Сергеевичу

(Ф. И. О. студента полностью)

- 1) **Тема практики:** графический учебный Web исполнитель (Web ГРИСП).
- 2) **Цель практики:** разработать контроллер Web ГРИСП и реализовать динамическое отображение работы исполнителя в браузере.
- 3) **Задачи практики:**
  - а) Изучить и проанализировать предметную область;
  - б) Спроектировать контроллер Web ГРИСП;
  - в) Разработать контроллер Web ГРИСП;
  - г) Спроектировать Web-интерфейс;

- д) Разработать Web-интерфейс;
- е) Протестировать программный продукт;
- ж) Вести дневник;
- з) Написать отчёт;
- и) Подготовить презентацию с докладом.

#### 4) Сроки практики:

С « 1 » сентября 2020 г. По « 31 » декабря 2020 г.  
(дата) (дата)

### Рабочий график (план) проведения практики

№ п/п	Перечень заданий	Сроки выполнения
1	Провести анализ предметной области	28.09.2020
2	Подобрать и изучить технологии	17.10.2020
3	Разработать структуру программы	12.11.2020
4	Реализовать Web ГРИСП	25.11.2020
5	Построить бизнес-модель проекта	9.12.2020

Дата выдачи задания:

« 1 » сентября 2020 г.  
(дата)

Руководитель практики от Университета:

ассистент каф. АСУ  
(должность, ученая степень, звание)

\_\_\_\_\_ А. Е. Косова  
(подпись) (И. О. Фамилия)

Задание принял к исполнению:

Студент гр. 439-3  
(група)

\_\_\_\_\_ А. С. Мазовец  
(подпись) (И. О. Фамилия)

## Содержание

1 Введение.....	7
1.1 Актуальность.....	7
1.2 Цель работы.....	7
1.3 Задачи.....	7
2 Обзор предметной области.....	8
2.1 Описание предметной области.....	8
2.2 Пользователи системы.....	9
2.3 Обзор аналогов.....	9
3 Обзор программного обеспечения.....	12
3.1 Сервер Web ГРИСП.....	12
3.2 Web-интерфейс Web ГРИСП.....	16
4 Обзор нормативной документации.....	19
4.1 Единая система программной документации.....	19
4.2 ГОСТ 19.XXX.....	19
5 Обзор графических нотаций.....	23
5.1 Семейство IDEF.....	23
5.2 ARIS и eEPC.....	24
5.3 Диаграмма Ганта.....	24
5.4 Блок-схемы.....	24
5.5 Язык моделирования UML.....	25
6 Описание структуры разрабатываемой программы.....	27
6.1 Постановка задачи.....	27
6.2 Ограничения и допущения.....	27
6.3 Структура сервера Web ГРИСП.....	28
6.4 Описание протокола взаимодействия с сервером Web ГРИСП.....	30
6.5 Графический интерфейс Web ГРИСП.....	34
6.6 Структура клиента отрисовщика Web ГРИСП.....	35
7 Реализация.....	36
7.1 Запуск сервера.....	36

7.2 Сайт.....	36
8 Маркетинговый анализ и построение бизнес-модели проекта.....	39
8.1 Конкурентный анализ.....	39
9 Заключение.....	42
9.1 Перспективы развития web-приложения.....	43
10 Список использованных источников.....	44

# **1 Введение**

## **1.1 Актуальность**

Тема является актуальной, так как на данный момент большинство исполнителей для учебных целей ориентированны на развитие исключительно алгоритмических знаний и навыков, тогда как разработанный мной исполнитель я позиционирую, как удобное средство для обучения Web взаимодействия с удаленным исполнителем с помощью современных и актуальных протоколов сети Интернет. Так же, в современной разработке систем обучения программированию для детей разработка учебных исполнителей до сих пор популярна и востребована.

## **1.2 Цель работы**

Разработка контроллера Web ГРИСП и реализация динамического отображения работы исполнителя в браузере.

## **1.3 Задачи**

- 1) Изучить и проанализировать предметную область;
- 2) Спроектировать контроллер Web ГРИСП;
- 3) Разработать контроллер Web ГРИСП;
- 4) Спроектировать Web-интерфейс;
- 5) Разработать Web-интерфейс.

При разработке использовались следующие графические нотации: UML [7], ДРАКОН [8].

## 2 Обзор предметной области

В рамках обзора предметной области необходимо выполнить следующие подзадачи:

- 1) Определить основные понятия, связанные с темой проекта;
- 2) Определить пользователей Web ГРИСП;
- 3) Провести анализ аналогичных решений;
- 4) Выбрать подходящую модель взаимодействия с Web ГРИСП.

### 2.1 Описание предметной области

Графический учебный исполнитель, как следует из названия, это система, состоящая из *исполнителя* и метода графического изображения его работы в динамике. ГРИСП разрабатываются в первую очередь для образовательных целей и используется для практики навыков алгоритмизации, так как является удобным и наглядным способом показать работу алгоритма, что способствует лучшему пониманию базовых принципов программирования и алгоритмизации обучающимися. Существуют учебные программы, успешно интегрирующие ГРИСП в образовательный процесс начального программирования [1].

В школьной учебной литературе дается следующее определение исполнителя [2]:

Исполнитель алгоритма — человек или техническое устройство, которые понимают команды алгоритма и умеют правильно их выполнять.

Понятие *исполнителя* изначально вытекает из определения алгоритма. Первая попытка дать точное математическое определение алгоритму Аланом Тьюрингом привело его к идее машины Тьюринга (МТ), представлявшего из себя [3] «математическую модель идеализированной цифровой вычислительной машины», являющегося по своей сути примером исполнителя. МТ состоит из четырех основных частей:

- ленты — теоретически бесконечной, разбитой на равные части ленты, содержащей символы внешнего алфавита;



- считывающей головки — устройства, позволяющего считывать символы внешнего алфавита с ленты МТ;
- внутренней памяти — набора внутренних состояний МТ, позволяющего МТ выполнять различные действия в зависимости от считанного символа внешнего алфавита;
- устройства управления — устройства, позволяющего исполнять переход от одного внутреннего состояния МТ, к другому в зависимости от текущего состояния и считанного символа внешнего алфавита и осуществлять переход по ленте к следующему символу внешнего алфавита.

В связи с тем, что исполнитель будет реализовываться на существующем языке программирования, роль устройства управления и ленты будет обеспечено средствами языка. Таким образом, для реализации исполнителя необходимо разработать набор команд и внутренних состояний. Особенностью является учет того, что исполнитель в составе ГРИСП должен учитывать условия виртуального окружения, в котором находится.

## 2.2 Пользователи системы

Разрабатываемую Web ГРИСП планируется использовать в образовательных целях в рамках предмета информатики и связанных с ней дисциплин в качестве дополнительного образования для учеников старших классов.

Пользователями системы могут быть:

- 1) Ученики старших классов для изучения и практики удаленно, из дома или в классе работы с Web ГРИСП;
- 2) Преподаватели, в качестве средства проведения занятий с учениками;

## 2.3 Обзор аналогов

Автором были найдены следующие примеры ГРИСП:

- 1) <https://docs.python.org/3/library/turtle.html> — классический исполнитель «Черепашка», реализованный на языке Python;

- 2) <http://pascalabc.net/downloads/pabcnethelp/topics/ForEducation/Executors/robotstart.html> — реализация исполнителя «Робот» в среде PascalABC.NET;
- 3) <http://pascalabc.net/downloads/pabcnethelp/topics/ForEducation/Executors/dmstart.html> — реализация исполнителя «Чертежник» в среде PascalABC.NET;
- 4) Реализация исполнителей «Робот» и «Чертежник» в среде языка «Ку-Мир».
- 5) <https://scratch.mit.edu> — образовательная площадка Scratch, пример современной ГРИСП, суть которой заключается в первую очередь в обучении программированию в простой и наглядной среде с использованием некоторого улучшенного аналога исполнителя «Черепашка» на собственном разработанном визуальном языке программирования.

Из обзора следует, что в качестве среды исполнителя выбирают в первую очередь языки программирования, легкие в обучении или специально созданные, чтобы максимально сфокусировать внимание обучающегося на изучении принципов алгоритмизации и не отвлекаться на трудности изучения самого языка.

Меня заинтересовал продукт **Scratch** своим подходом к созданию среды, в которой могут одновременно участвовать несколько исполнителей. Это максимально соответствует моей задумке реализовать среду для исполнителя, в которой можно было бы совершать взаимодействие нескольких исполнителей одновременно.

2.3.1 Все возможности для разработки в **Scratch** реализованы непосредственно в окне браузера и не требуют установки какого-либо дополнительного программного обеспечения.

2.3.2 Этот продукт является совершенно бесплатным и доступен без регистрации, не ограничивает и публикации работ для неавторизованных пользователей.

2.3.3 Реализация визуального языка простая и удобная, позволяет создавать собственные функции на основе базовых команд, являющихся стандартными для всех исполнителей типа «Черепашка».

2.3.4 Из минусов я обнаружил, что в среде **Scratch** нет возможности воспользоваться иными языками программирования, кроме предоставленного визуального языка.

2.3.5 Так же, по умолчанию не предусмотрена возможность управления несколькими исполнителями с различных устройств посредством сети Интернет.

Таким образом, в работе решено использовать язык Python для создания демонстрационного клиента для взаимодействия с Web ГРИСП, как достаточно легкого в освоении школьниками языка программирования.

Взаимодействие с исполнителем должно представлять из себя модель клиент-сервера, как позволяющую создавать одновременное управление исполнителями с различных устройств. В протоколе взаимодействия должен быть интерфейс как создания новой учебной комнаты, так и подключения к существующей комнате.

Основным критерием при разработке протокола взаимодействия с Web ГРИСП будет являться максимальная простота и интуитивность взаимодействия с исполнителем посредством сети Интернет чтобы ученик мог самостоятельно разрабатывать для себя собственные клиенты взаимодействия с исполнителем.

По примеру **Scratch** отображение работы исполнителя должно быть исполнено посредством браузера, чтобы минимизировать требуемое для установки и настройки программное обеспечение.

## 3 Обзор программного обеспечения

### 3.1 Сервер Web ГРИСП

Задачи сервера Web ГРИСП:

1) При запросе *клиента браузера*:

а) Возвращать множеству клиентов необходимый, исполняемый непосредственно в браузере код графического окружения;

б) Возвращать множеству клиентов обертку для кода в виде html файла страницы и css файла стилей;

2) При запросе *клиента контроллера*:

а) Устанавливать постоянное Web соединение с клиентом;

б) Принимать запросы с множества подключенных клиентов на обновление состояния исполнителя;

в) Возвращать актуальную информацию об исполнителе.

Для реализации сервера можно реализовать обе эти задачи внутри единого сервера, занимающегося и отдачей статической информации об исполнителе (код и скрипты страницы) и динамической обработкой команд клиента контроллера, либо разделить их между двумя серверами таким образом, чтобы один сервер обрабатывал все статические запросы, а второй сервер занимался только динамической обработкой команд контроллера.

У последнего варианта существуют преимущества:

— Разделяя различные подзадачи на несколько программ это позволяет использовать внутри проекта готовые проверенные решения, создавая большую стабильность работы системы;

— Серверы, предназначенные для работы со статическим контентом часто предоставляют возможность проксирования запросов сразу по нескольким различным протоколам и на различные порты хоста, благодаря чему можно реализовывать шифрование передачи данных без риска повлиять на работу кода, обрабатывающего команды клиента контроллера.

Опираясь на перечисленные плюсы второго подхода решено использовать именно такую модель. В качестве сервера, задачи которого заключаются в обработке статических запросов и проксирования запросов для команд клиента контроллера, был выбран Web сервер **nginx**. Он позволяет реализовать все вышеперечисленные требования. Также он распространяется по открытой лицензии и не накладывает никаких ограничений на использование его в проектах.

Настройка **nginx** довольно простая и происходит в конфигурационных файлах в форме, похожей на формат *json* по определенным правилам [4].

Рассмотрим пример базовой настройки сервера, принимающего запросы по порту 8080 с доменного имени *mazovec.ru* и перенаправляющего запросы по URI */service* на порт 4000 и по умолчанию отдавая файл *index.html*, расположенный по пути */home/asmazovec/html*:

```

1 | events {
2 | }
3 |
4 | http {
5 |     include /etc/nginx/mime.types;
6 |
7 |     map $http_upgrade $connection_upgrade {
8 |         default upgrade;
9 |         ''       close;
10 |    }
11 |
12 |    server {
13 |        listen      8080;
14 |        server_name  mazovec.ru www.mazovec.ru;
15 |
16 |        location / {
17 |            root /home/asmazovec/html;
18 |            index index.html;
19 |        }
20 |
21 |        location /server {
22 |            proxy_pass http://localhost:4000;
23 |            proxy_http_version 1.1;
24 |            proxy_set_header Upgrade $http_upgrade;
25 |            proxy_set_header Connection $connection_upgrade;
26 |        }
27 |    }
28 | }
```

3.1.1 Различные параметры для настройки разбиты по специальным блокам;

3.1.2 Блоки и параметры могут иметь различное значение в зависимости от контекста, в котором они используются.

3.1.3 Основные блоки, в которых происходит настройка — *location* и *server*:

3.1.4 Блок *server* необходим, чтобы на одном хосте и на одном IP адресе использовать различные доменные имена и порты соединения;

3.1.5 С помощью блока *location* указываются конкретные URI адреса, с которыми соотносится физическая директория на хосте или IP адрес, порт и, при необходимости, дополнительные заголовки, по которым необходимо совершать переадресацию, что позволяет использовать сервер *nginx* для проксирования запросов.

**Nginx** предпочтительнее использовать в проектной работе, чем существующий аналог — **apache**, так как он довольно минималистичен, меньше нагружает систему и быстрее обрабатывает запросы.

Для простоты разработки сервера Web ГРИСП, обрабатывающего команды клиента контроллера удобно использовать средства создания сервера, непосредственно встроенные в язык программирования. Для такой задачи подошел язык **Python 3**, являющийся универсальным и позволяющий использовать различные дополнительные библиотеки для создания Web серверов и установления соединения по Web сокет-протоколу. Для создания сервера на языке Python использовалась библиотека **aiohttp**, реализующая функции асинхронного Web приложения по протоколу Web сокетов. Для работы с библиотекой *aiohttp* написана исчерпывающая документация.

Рассмотрим базовый пример создания сервера, работающего по 4000 порту, принимающего запросы по протоколу Web сокетов по URI */service* и обработчика этих запросов:

```

1 | from aiohttp import web
2 |
3 | class WSService(web.View):
4 |     def __init__(self, request):
5 |         self._request = request
6 |

```

```

7 |     async def get(self):
8 |         ws = web.WebSocketResponse()
9 |         await ws.prepare(self.request)
10 |
11 |         dispatcher = {
12 |             # заголовки запросов соотносятся с методами класса
13 |         }
14 |
15 |         async for msg in ws:
16 |             if msg.type == web.WSMsgType.text:
17 |                 if msg.data == 'close':
18 |                     await ws.close()
19 |                 else:
20 |                     event = json.loads(msg.data)
21 |                     if ws.closed:
22 |                         await ws.close()
23 |                     else:
24 |                         await ws.send_json( \
25 |                             await dispatcher \
26 |                             [event['e']] (ws=ws, data=event))
27 |             elif msg.type == web.WSMsgType.error:
28 |                 print(f'error {ws.exception()}')
29 |
30 |         print(f'ws connection closed')
31 |         return ws
32 |
33 | def main():
34 |     server = web.Application()
35 |     server.router.add_get('/service', WSService)
36 |
37 |     web.run_app(server, '127.0.0.1', port=4000)
38 |
39 | if __name__ == '__main__':
40 |     main()

```

3.1.6 Создание сервера начинается с инициализации объекта класса `web.Application`;

3.1.7 Сервер оперирует множеством сервисов, запрос к которым может происходить посредством URI адреса, а за перенаправление к определенным обработчикам запроса отвечает атрибут класса `web.Application` — `router`. Метод `router.add_get` связывает URI и обработчик запросов — объект класса `web.View`;

3.1.8 Запуск асинхронного сервера производится с помощью метода `web.run_app`.

Каждый новый запрос по определенному URI вызывает инициализацию объекта класса `web.View`, являющегося наследником абстрактного типа данных

Python — словаря (dictionary). Объект класса *web.View* хранит очередь запросов к данному обработчику запросов. Новое соединение по протоколу Web сокетов требует особенного порядка установления соединения, связанного с запросом обновления протокола:

3.1.9 Методы *web.WebSocketResponse* и *prepare*, которые отправляют фрейм клиенту, содержащий информацию, необходимую для установления постоянного соединения с заголовком *update*;

3.1.10 После успешного установления соединения все запросы данного клиента будет принимать связанный с ним обработчик запросов.

Все запросы в цикле асинхронно обрабатываются обработчиком запросов:

3.1.11 Все запросы для удобства можно расшифровывать из текстового формата *.json* в структуру словаря. Такой подход позволяет эффективно обрабатывать запросы используя функцию — диспетчер (*dispatcher*).

## 3.2 Web-интерфейс Web ГРИСП

Интерфейсы ГРИСП отличаются тем, что должны наглядно и оригинально отображать работу исполнителя. Так как решено использовать браузер в качестве среды отрисовки, была выбрана технология **WebGL** и основанная на ней библиотека **JavaScript — Three.js** [6]. Она позволяет легко создавать примитивы 3D объектов, размещать источники света на сцене и обрабатывать тени от 3D объектов.

Рассмотрим пример скрипта на JavaScript, выводящего на экран примитив шара, плоскость и источник света с динамическим расчетом теней используя библиотеку *Three.js*:

```
1 | import * as THREE from
   | 'https://threejsfundamentals.org/threejs/resources/threejs/r11
   | 9/build/three.module.js';
2 |
3 | const canvas = document.querySelector('#c');
4 |
5 |
6 | let scene = new THREE.Scene();
7 |
8 | let renderer = new THREE.WebGLRenderer({canvas});
```



```

9 | renderer.shadowMapEnabled = true;
10 |
11 | const fov      = 60;
12 | const aspect  = 2;
13 | const near     = 1;
14 | const far      = 300;
15 | let camera = new THREE.PerspectiveCamera(fov, aspect, near,
    far);
16 | camera.position.set(0, 50, 0);
17 | camera.lookAt(0, 0, 0);
18 |
19 | let spotLight = new THREE.SpotLight(0xffffffff);
20 | spotLight.position.set(-40, 60, -10);
21 | spotLight.castShadow = true;
22 | scene.add(spotLight);
23 |
24 | let planeGeometry = new THREE.PlaneGeometry(100,100);
25 | let planeMaterial = new THREE.MeshLambertMaterial({color:
    0xffffffff});
26 | let plane = new THREE.Mesh(planeGeometry,planeMaterial);
27 | plane.position.x = 0;
28 | plane.position.y = 0;
29 | plane.position.z = 0;
30 | plane.receiveShadow = true;
31 | scene.add(plane);
32 |
33 | let sphereGeometry = new THREE.SphereGeometry(4,8,6);
34 | let sphereMaterial = new THREE.MeshLambertMaterial({color:
    BBBBBF});
35 | let sphere = new THREE.Mesh(sphereGeometry,sphereMaterial);
36 | scene.add(sphere);
37 |
38 | renderer.render(scene, camera);

```

3.2.1 Файлы библиотеки в JavaScript могут быть подключены по ссылке на исходные файлы требуемой библиотеки;

3.2.2 Все объекты размещаются в сцене. Создание сцены осуществляется вызовом конструктора *Scene*;

3.2.3 За отрисовку всех объектов в сцене отвечает класс *WebGLRenderer*. В качестве аргумента ему передается ссылка на существующий на html странице тег *canvas*. *ShadowMapEnabled* — поле, указывающее, что тень от требуемых объектов необходимо просчитывать каждый кадр;

- Объект камеры с перспективой определен как класс *PerspectiveCamera*;
- Класс точечного источника света — *SpotLight*;
- Класс плоскости, полигональная сетка — *PlaneGeometry*;

— Класс примитива сферы, полигональная сетка — `SphereGeometry`.

3.2.4 В библиотеке `Three.js`, помимо геометрического примитива (`Geometry`), каждый объект имеет материал (`Material`), который участвует в отрисовке объекта. Для объектов, реагирующих на источники освещения и отбрасывающих тень определен класс `MeshLambertMaterial`.

3.2.5 Геометрический примитив и материал объекта объединяются в меше (`Mesh`). Отрисовщик будет отрисовывать тень для всех мешей на сцене с атрибутом `receiveShadow`.

3.2.6 После создания меша, его необходимо добавить на сцену `scene.add`.

## **4 Обзор нормативной документации**

### **4.1 Единая система программной документации**

Единая система программной документации - комплекс государственных стандартов, устанавливающих взаимосвязанные правила разработки, оформления и обращения программ и программной документации (ГОСТ 19.001-77).

В стандартах ЕСПД устанавливают требования, регламентирующие разработку, сопровождение, изготовление и эксплуатацию программ, что обеспечивает возможность:

- Унификации программных изделий для взаимного обмена программами и применения ранее разработанных программ в новых разработках;
- Снижения трудоемкости и повышения эффективности разработки, сопровождения, изготовления и эксплуатации программных изделий;
- Автоматизации изготовления и хранения программной документации.
- Сопровождение программы включает анализ функционирования, развитие и совершенствование программы, а также внесение изменений в нее с целью устранения ошибок.

### **4.2 ГОСТ 19.XXX**

ГОСТ 19.XXX - комплекс стандартов единой программной документации (ЕСПД). Часто применяется наряду с ГОСТ 34 при создании программ и автоматизированных систем, особенно, когда в качестве заказчиков выступают государственные или крупные коммерческие организации.

#### **4.2.1 Общие положения:**

- ГОСТ 19.001-77. ЕСПД. Общие положения.
- ГОСТ 19.002-80. ЕСПД. Схемы алгоритмов и программ. Правила выполнения. — Заменен на ГОСТ 19.701-90
- ГОСТ 19.003-80. ЕСПД. Схемы алгоритмов и программ. Обозначения условные графические. — Заменен на ГОСТ 19.701-90

- ГОСТ 19.004-80. ЕСПД. Термины и определения. — Заменен на ГОСТ 19.781-90

- ГОСТ 19.005-85. ЕСПД. Р-схемы алгоритмов и программ. Обозначения условные графические и правила выполнения.

#### 4.2.2 Основопологающие стандарты:

- ГОСТ 19.101-77. ЕСПД. Виды программ и программных документов.

- ГОСТ 19.102-77. ЕСПД. Стадии разработки.

- ГОСТ 19.103-77. ЕСПД. Обозначение программ и программных документов.

- ГОСТ 19.104-78. ЕСПД. Основные надписи.

- ГОСТ 19.105-78. ЕСПД. Общие требования к программным документам.

- ГОСТ 19.106-78. ЕСПД. Требования к программным документам, выполненным печатным способом.

#### 4.2.3 Правила выполнения документации разработки:

- ГОСТ 19.201-78. ЕСПД. Техническое задание. Требования к содержанию и оформлению.

- ГОСТ 19.202-78. ЕСПД. Спецификация. Требования к содержанию и оформлению.

#### 4.2.4 Правила выполнения документации изготовления:

- ГОСТ 19.301-79. ЕСПД. Программа и методика испытаний. Требования к содержанию и оформлению.

- Правила выполнения документации сопровождения

- ГОСТ 19.401-78. ЕСПД. Текст программы. Требования к содержанию и оформлению.

- ГОСТ 19.402-78. ЕСПД. Описание программы.

- ГОСТ 19.403-79. ЕСПД. Ведомость держателей подлинников.

- ГОСТ 19.404-79. ЕСПД. Пояснительная записка. Требования к содержанию и оформлению.

#### 4.2.5 Правила выполнения эксплуатационной документации:

- ГОСТ 19.501-78. ЕСПД. Формуляр. Требования к содержанию и оформлению.

- ГОСТ 19.502-78. ЕСПД. Описание применения. Требования к содержанию и оформлению.

- ГОСТ 19.503-79. ЕСПД. Руководство системного программиста. Требования к содержанию и оформлению.

- ГОСТ 19.504-79. ЕСПД. Руководство программиста. Требования к содержанию и оформлению.

- ГОСТ 19.505-79. ЕСПД. Руководство оператора. Требования к содержанию и оформлению.

- ГОСТ 19.506-79. ЕСПД. Описание языка. Требования к содержанию и оформлению.

- ГОСТ 19.507-79. ЕСПД. Ведомость эксплуатационных документов.

- ГОСТ 19.508-79. ЕСПД. Руководство по техническому обслуживанию. Требования к содержанию и оформлению.

#### 4.2.6 Правила обращения программной документации:

- ГОСТ 19.601-78. ЕСПД. Общие правила дублирования, учета и хранения.

- ГОСТ 19.602-78. ЕСПД. Правила дублирования, учета и хранения программных документов, выполненных печатным способом.

- ГОСТ 19.603-78. ЕСПД. Общие правила внесения изменений.

- ГОСТ 19.604-78. ЕСПД. Правила внесения изменений в программные документы, выполненные печатным способом.

#### 4.2.7 Резервные группы:

- ГОСТ 19.701-90 (ИСО 5807-85). ЕСПД. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения.

– ГОСТ 19.781-90. Обеспечение систем обработки информации программное. Термины и определения.

## 5 Обзор графических нотаций

Для разработки проекта и его описания требуется использовать *нотации*. Нотация - система условных обозначений, принятая в какой-либо области знаний или деятельности. Включает множество символов, используемых для представления понятий и их взаимоотношений, составляющее алфавит нотации, а также правила их применения [10].

Существует множество принятых графических нотаций, каждая из которых решает определенную задачу, имеет ограничения, плюсы и минусы. Примеры графических нотаций:

- Семейство IDEF (Integrated DEFinition);
- ARIS eEPC (Extended Event Driven Process Chain);
- Диаграмма Ганта;
- блок-схемы;
- Язык моделирования UML.

### 5.1 Семейство IDEF

Семейство IDEF представлено тремя графическими нотациями:

- 1) IDEF0;
- 2) IDEF1.X;
- 3) IDEF3.

IDEF0 — нотация функционального моделирования. Ее назначение — описание бизнес процессов, отражает логического отношения между процессами, но *не отражает* последовательности выполнения процессов. Основным плюсом этой нотации является то, что она позволяет представить все процессы во взаимодействии в общем виде, не углубляясь в подробности, однако, в связи с тем, что использование этой нотации требует подробной подготовки, данная нотация выходит из употребления. Так же эта графическая нотация не имеет средств описания.

IDEF3 — продолжение разработки графической нотации IDEF0. Она предназначена для декомпозиции структурных элементов IDEF0. В ней, в отличие от IDEF0 отображается *очередность* выполнения работ, имеет сходства с блок-схемами. Как наследник IDEF0, имеет те же достоинства и недостатки.

## 5.2 ARIS и eEPC

Отображает цепочку исполнения бизнес процесса. Она используется для описания низших слоев бизнес-модели, отображает ход выполнения бизнес-процесса. Она позволяет детально описать выполнения бизнес-процесса всеми исполнителями и графически изобразить все эти процессы. Недостатком является большой, строго определенный набор графических элементов, что затрудняет понимание.

## 5.3 Диаграмма Ганта

Эта нотация создана для описания работ над проектом, позволяет оценивать время на выполнение конкретных работ. Она наглядна, и позволяет легко планировать независимые друг от друга работы.

## 5.4 Блок-схемы

Графическая модель, описывающая *алгоритм* или *процесс*. В качестве примера графической нотации блок схем рассмотрим ДРАКОН (Дружелюбный Русский Алгоритмический Язык, Который Обеспечивает Наглядность). Основными видами ДРАКОН схем являются примитив и силуэт. Примитив отображает алгоритм в «двумерной» форме, тогда как силуэт объединяет множество примитивов и образует «двумерное» представление алгоритма, за счет чего достигается большая наглядность сложных алгоритмов. Структурными элементами ДРАКОН схемы являются строго определенные *иконы*. Каждая ДРАКОН схема начинается иконой *начало* и заканчивается иконой *конец*. Ветвление изображается с помощью иконы *условие*. Выполнение определенного шага алгоритма описывается в иконе *действие*. За счет этих икон можно производить проектирование алгоритма по модели «черного ящика», когда задача разбивается на



подзадачи, решение которых является *черным ящиком* и требует описания на следующих этапах декомпозиции задачи. Для каждого черного ящика описываются входные и выходные данные.

### 5.5 Язык моделирования UML

UML это графическая нотация, позволяющая графически описывать модели в парадигме объектного моделирования, но так же является достаточно гибким для моделирования бизнес-процессов — UML является графической нотацией широкого профиля. UML предлагает несколько видов диаграмм:

- Структурные диаграммы (диаграммы классов, компонентов, объектов и т.д.);
- Диаграммы поведения (диаграммы деятельности, состояний, вариантов использования);
- Диаграммы взаимодействия.

UML позволяет описать систему со всех возможных сторон, однако обладает неточной семантикой, что приводит к тому, что его сложно изучать и использовать, но есть наиболее устойчивые диаграммы. Для описания структуры проекта диаграмму классов UML определяет класс и виды отношений между классами. Любой класс обладает названием, аргументами и методами. Виды отношений между классами:

- Наследование;
- Композиция;
- Агрегация;
- Ассоциация;
- Зависимость.

Наследование моделирует тип отношения «является» между двумя объектами или отношение «родитель - наследник». Конструктор нового объекта сначала создает родительский элемент с его атрибутами и методами, а затем непосредственно объект наследник, расширяя или заменяя атрибуты и методы роди-

тельского класса. В диаграммах классов изображается в виде линии между двумя классами, на класс — родитель указывает пустая треугольная стрелка.

Композиция моделирует тип отношения «имеет». Этот вид отношения означает, что объект содержит в себе один или множество элементов второго класса, но существование каждого объекта, содержащегося в классе имеет смысл и за пределами этого класса. В диаграмме классов изображается в виде линии между двумя объектами, на класс — контейнер указывает закрашенный ромб.

Агрегация моделирует тип отношения «имеет», но в этом виде отношения существование «дочернего» объекта не имеет смысла вне «родительского» объекта. Изображается линией между двумя объектами, на класс - «родитель» указывает незакрашенный ромб.

Ассоциация моделирует тип отношения «использует» и не является, в отличие от вышеперечисленных отношений вида «часть — целое». Изображается просто линией между объектами.

Зависимость используется только в тех случаях, если необходимо указать посредника в построении отношений между другими объектами. Изображается как пунктирная стрелка между объектом и линией отношения двух других объектов.

## 6 Описание структуры разрабатываемой программы

### 6.1 Постановка задачи

#### 6.1.1 Функции Web ГРИСП:

##### 1) Функции для работы с клиентом отрисовщика:

- а) Получение данных о сцене, необходимых для отрисовки;
- б) Обработка запроса на обновление сцены.

##### 2) Функции для работы с клиентом контроллера:

- а) Исполнение команд клиента контроллера;
- б) Передача актуальных данных об исполнителе клиенту контроллера.

#### 6.1.2 Входные данные:

- Управляющие команды клиента контроллера;
- Запрос на обновление от клиента отрисовщика.

#### 6.1.3 Выходные данные:

- Графическое отображение работы исполнителя.

В общем виде, графически можно изобразить решение задачи в виде диаграммы потоков данных (см. Рисунок 4.1.1).

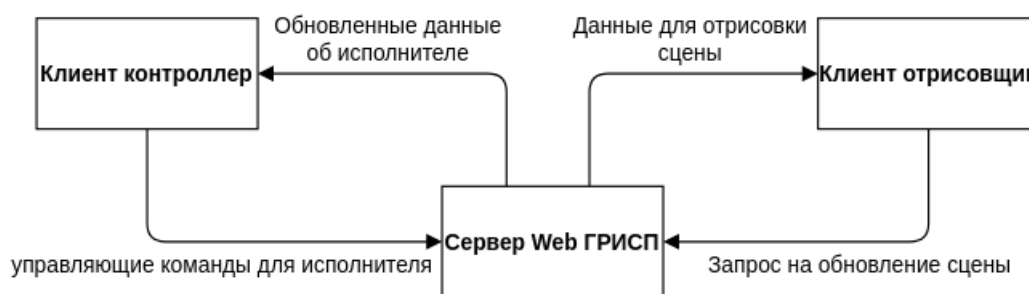


Рисунок 4.1.1 — диаграмма потока данных.

### 6.2 Ограничения и допущения

- К серверу Web ГРИСП может быть подключено большое множество клиентов отрисовщиков и клиентов контроллеров;

- В один момент времени на клиенте отрисовщика должна отображаться работа только ограниченного установленного числа исполнителей;
- В один момент времени одним исполнителем на сервере должен управлять только один клиент контроллер;
- Исполнители должны взаимодействовать с определенными исполнителями на сервере Web ГРИСП.

### 6.3 Структура сервера Web ГРИСП

Введем для сервера понятие *комнаты*. Каждая комната содержит в себе множество объектов и исполнителей. Каждый исполнитель ассоциируется с отдельным клиентом контроллера используя доступ по паролю.

6.3.1 Комната является контейнерным классом для объектов сцены — он хранит объекты в структуре словаря, ключом выступает идентификационный номер каждого объекта. Это позволяет быстро производить поиск нужных объектов в комнате, не прибегая к перебору.

6.3.2 Все объекты сцены наследуются от класса `Object`. Это позволяет минимизировать количество кода и гарантирует предсказуемость поведения объектов новых создаваемых классов-наследников;

6.3.3 Класс `Player` является исполнителем и для него определены методы, с которыми будут ассоциироваться команды контроллера. Методы специально вынесены в класс родителя — `Movable`, что позволяет в будущем добавлять в комнату объекты, управляемые непосредственно сервером, а не клиентом контроллера;

6.3.4 Каждый объект класса клиента ассоциируется с комнатой. При создании комнаты ей присваивается идентификационный номер, по которому могут подключаться клиенты. Для клиента контроллера, чтобы защитить другие исполнители от перехвата управления, требуется пароль для ассоциации с объектом исполнителя `Player` (см. Рисунок 4.3.1);

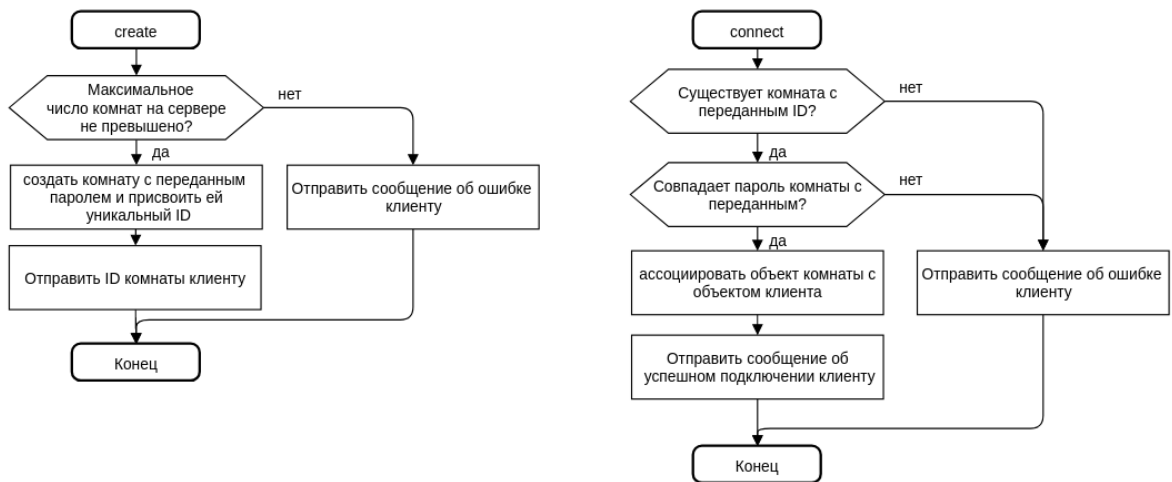


Рисунок 4.3.1 — ДРАКОН схема алгоритма ассоциации клиента с комнатой.

6.3.5 Структуру сервера можно представить для наглядности в виде uml диаграммы классов (см. Рисунок 4.3.2).

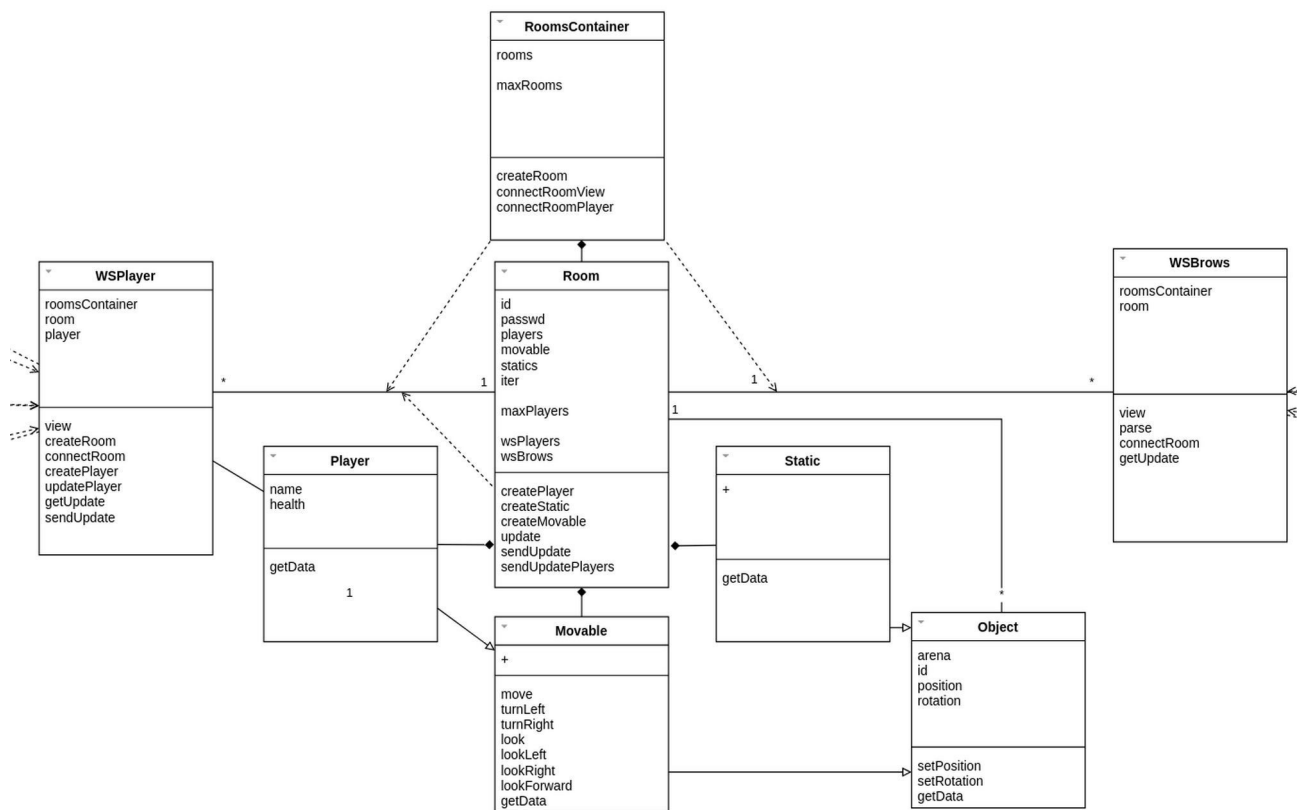


Рисунок 4.3.2 — Диаграмма классов сервера Web ГРИСП.

Комната инкапсулирует методы создания объектов сцены и обновления сцены, что позволяет гарантировать ограниченность количества подключенных к комнате исполнителей. Максимальное количество исполнителей определяется атрибутом maxPlayers.

С клиентов отрисовщика может приходиться множество запросов на обновление комнаты, чтобы гарантировать синхронизированность запросов в классе Room итерация каждого запрашиваемого обновления сравнивается с атрибутом iteration (см. Рисунок 4.3.3).



Рисунок 4.3.3 — ДРАКОН-схема алгоритма обработки запроса обновления комнаты.

## 6.4 Описание протокола взаимодействия с сервером Web ГРИСП

Каждый запрос определяет порядок дальнейшего взаимодействия с сервером. Все запросы принимаются в текстовом формате в структуре json. Основным и общим для всех запросов ключом структуры является поле **e** (от event — событие).

### 1) Запросы от клиента отрисовщика:

а) connect — запрос на подключение к комнате:

```

1 | e::=      <connect>
2 | roomId::= <идентификационный номер комнаты>
  
```

б) getUpdate — запрос на обновление комнаты или синхронизацию:

```

1 | e::=      <getUpdate>
2 | iter::=   <запрашиваемая итерация комнаты>
  
```

### 2) Запросы от клиента контроллера:

а) create — запрос на создание новой комнаты на сервере:

```

1 | e::=      <create>
2 | passwd::= <пароль для подключения контроллеров к комнате>
  
```

```

3 | minPl ::= <минимальное число возможных исполнителей>
4 | maxPl ::= <максимальное число возможных исполнителей>

```

б) connect — запрос на подключение к комнате на сервере:

```

1 | e ::= <connect>
2 | roomID ::= <идентификационный номер комнаты>
3 | passwd ::= <пароль для подключения к комнате>

```

в) createPl — запрос на создание исполнителя для управления контроллером, выполняется строго после подключения к комнате:

```

1 | e ::= <createPl>
2 | name ::= <имя исполнителя>

```

г) getUpdate — запрос на получение актуальный данных о состоянии контролируемого исполнителя:

```

1 | e ::= <getUpdate>

```

д) UpdatePl — заголовок для управляющих команд:

```

1 | e ::= <UpdatePl>
2 | do ::= <skip> || <move> || <turnLeft> || <turnRight>

```

Запрос getUpdate на обновление клиента принимается классом WSRoom.

Поле **e** ответа на запрос содержит заголовок, соответствующий заголовку запроса. За ним следует поле **m** (от message — сообщение), в которой содержится информация об успешности или неудаче обработки запроса.

1) Ответы клиенту отрисовщику:

а) Успешный connect — сообщение об успешном подключении к комнате:

```

1 | e ::= <connect>
2 | m ::= <ok>

```

б) Ошибка connect:

```

1 | e ::= <create>
2 | m ::= <error>

```

в) Успешный getUpdate — обновленные данные всех объектов:

```

1 | e ::= <getUpdate>
2 | m ::= <ok>
3 | id ::= <идентификационный номер i-го объекта>

```

```

4 | type ::= <тип i-го объекта>
5 | pos ::= x ::= <позиция объекта по x>
6 |       y ::= <позиция объекта по y>
7 | rot ::= x ::= <косинус угла поворота>
8 |       y ::= <синус угла поворота>

```

г) Ошибка getUpdate:

```

1 | e ::= <getUpdate>
2 | m ::= <error>
3 | iter ::= <актуальная итерация симуляции>

```

2) Ответы клиенту контроллеру:

а) Успешный create — сообщение об успешном создании комнаты:

```

1 | e ::= <create>
2 | m ::= <roomID > + <идентификационный номер комнаты>

```

б) Ошибка create:

```

1 | e ::= <create>
2 | m ::= <error>

```

в) Успешный connect — сообщение об успешном подключении к комнате:

```

1 | e ::= <connect>
2 | m ::= <ok>

```

г) Ошибка connect:

```

1 | e ::= <connect>
2 | m ::= <error>

```

д) Успешный createPl — сообщение об успешном создании исполнителя в комнате:

```

1 | e ::= <createPl>
2 | m ::= <ok>

```

е) Ошибка createPl:

```

1 | e ::= <createPl>
2 | m ::= <ok>

```

ж) Успешный getUpdate — все данные, которые может получить исполнитель, просканировав окружение вокруг себя и впереди:

```

1 | e ::= <getUpdate>
2 | m ::= <ok>

```



```

3 | id ::=      <идентификационный номер исполнителя>
4 | type ::=   <тип i-го объекта>
5 | pos ::=    x ::= <позиция объекта по x>
6 |           y ::= <позиция объекта по y>
7 | rot ::=    x ::= <косинус угла поворота>
8 |           y ::= <синус угла поворота>
9 | [update] ::= [look] ::= type ::= <тип обнаруженного объекта>
10 |              pos ::= x ::= <x этого объекта>
11 |              y ::= <y этого объекта>
12 |              [lookF] ::= type ::= <тип обнаруженного объекта>
13 |              pos ::= x ::= <x этого объекта>
14 |              y ::= <y этого объекта>
15 |              [lookL] ::= type ::= <тип обнаруженного объекта>
16 |              pos ::= x ::= <x этого объекта>
17 |              y ::= <y этого объекта>
18 |              [lookR] ::= type ::= <тип обнаруженного объекта>
19 |              pos ::= x ::= <x этого объекта>
20 |              y ::= <y этого объекта>
21 |

```

### з) Ошибка getUpdate:

```

1 | e ::=      <getUpdate>
2 | m ::=      <error>

```

и) Успешный updatePl — успешное исполнение переданной клиентом контроллера команды:

```

1 | e ::=      <updatePl>
2 | m ::=      <ok>

```

### к) Ошибка updatePl:

```

1 | e ::=      <updatePl>
2 | m ::=      <error>

```

Схема взаимодействия с сервером Web ГРИСП представлена в виде схемы (см. Рисунок 4.4.1).

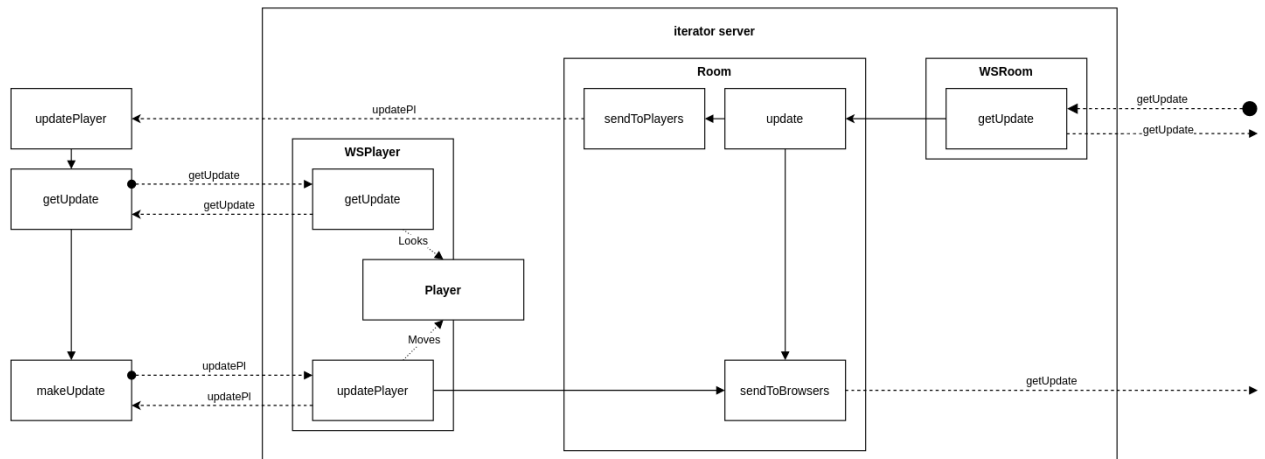


Рисунок 4.4.1 — Схема протокола взаимодействия с сервером Web ГРИСП.

## 6.5 Графический интерфейс Web ГРИСП

Структура html файла содержит минимально необходимые теги для работы библиотеки Three.js — подключена таблица стилей css, скрипты клиента отрисовщика, canvas.

```

1 | <!DOCTYPE html>
2 | <html>
3 |     <head>
4 |         <title>canvas</title>
5 |         <meta charset="utf-8">
6 |         <link rel="stylesheet" type="text/css"
7 | href="css/style.css" />
8 |     </head>
9 |
10 |     <body>
11 |         <canvas id="c"></canvas>
12 |     </body>
13 |
14 | <script type="module" src="js/classes.js"></script>
15 | <script type="module" src="js/websocket.js"></script>
16 | <script type="module" src="js/main.js"></script>
17 | </html>

```

Таблица стилей css:

```

1 | html, body {
2 |     margin: 0;
3 |     height: 100%;
4 | }
5 |
6 | #c {
7 |     width: 100%;
8 |     height: 100%;

```

```

9 |      display: block;
10 |  }
```

## 6.6 Структура клиента отрисовщика Web ГРИСП

Основным элементом клиента отрисовщика является функция асинхронного обработчика запросов Web сокетов — `wsStart`. Она отвечает за обработку всех сообщений от сервера Web ГРИСП. Так же через каждый фиксированный интервал из этой функции на сервер передается запрос на обновление комнаты в соответствии с протоколом взаимодействия с сервером Web ГРИСП.

Все данные, передаваемые с сервера и принимаемые клиентом отрисовщика, проходят обработку, и на основании этих данных создаются объекты класса `Room` и `GObject`. При создании `GObject` каждому новому объекту присваивается соответственный идентификационный номер, соответствующий идентификационному номеру этого объекта на сервере. Если же приходит запрос на обновление состояния объекта сцены `Three.js`, вызывается метод класса `Gobject` — `move`, совершающий плавное перемещение объекта по сцене. В зависимости от типа объекта, для него определяется свой уникальный примитив и цвет.

В бесконечном цикле происходит перерисовка сцены.

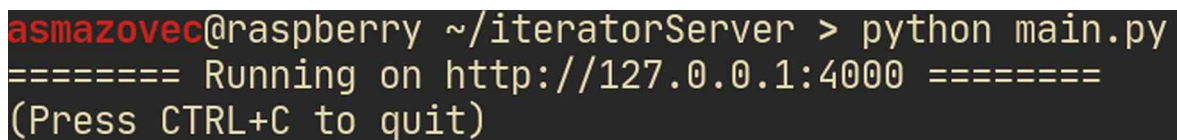
Элементы управления реализованы вспомогательной для `Three.js` библиотеки `dat.gui.module.js`. На каждое поле GUI в структуре назначены вышеперечисленные функции, на основании них в html коде страницы автоматически создаются элементы управления Web ГРИСП.

## 7 Реализация

### 7.1 Запуск сервера

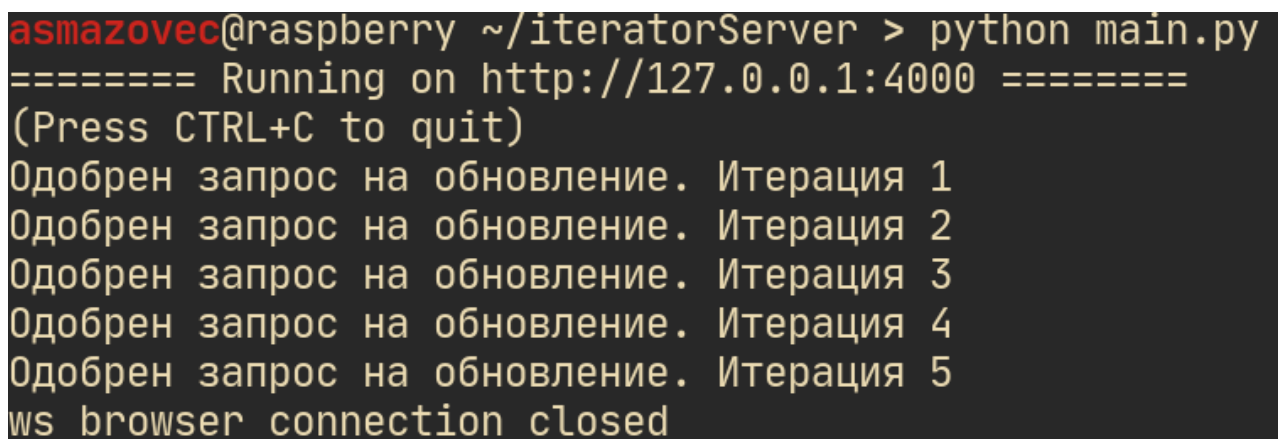
Запуск сервера может выполняться с помощью исполняемого файла либо из консоли, путем ввода команды `python main.py` из директории сервера. Для запуска обязательно требуется наличие предустановленной библиотеки `aiohttp`, установка которой осуществляется командой `pip install aiohttp`.

После запуска сервера, в окно консоли начнет выводиться отладочная информация (см. Рисунок 5.1, Рисунок 5.2).



```
asmazovec@raspberrypi ~/iteratorServer > python main.py
===== Running on http://127.0.0.1:4000 =====
(Press CTRL+C to quit)
```

Рисунок 5.1 — Скриншот консольного вывода работы сервера.

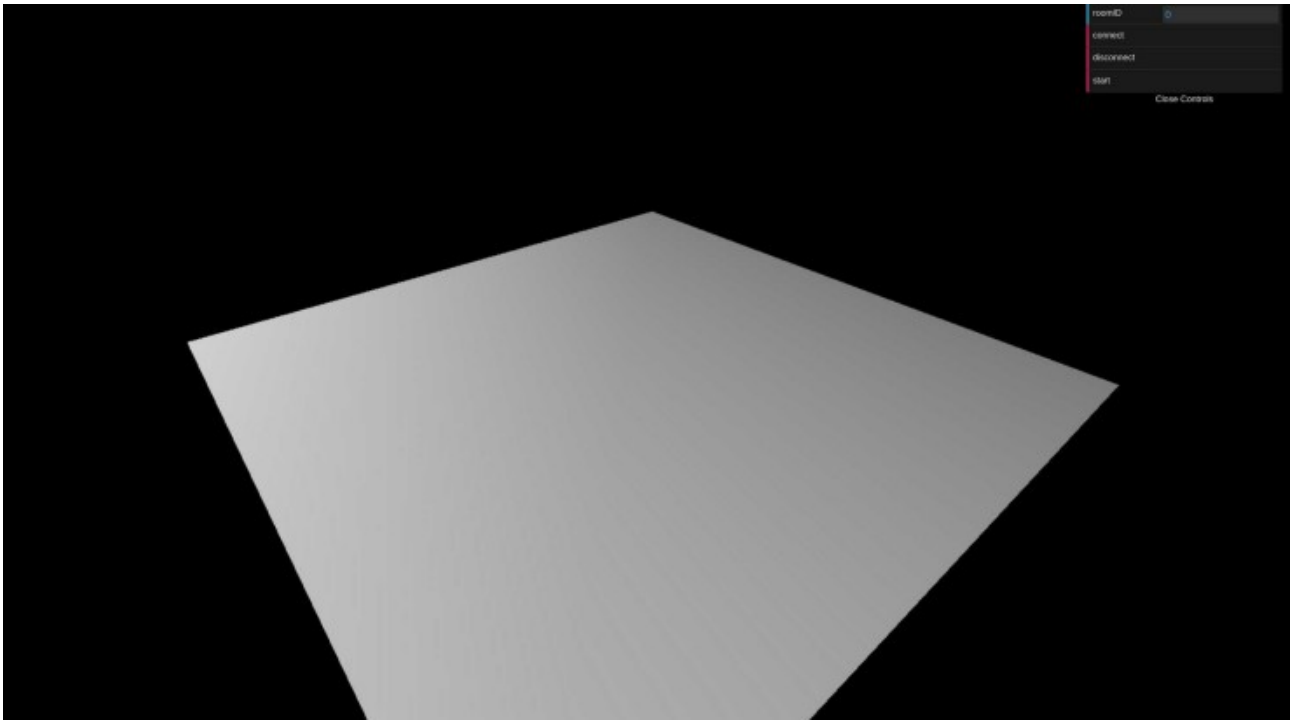


```
asmazovec@raspberrypi ~/iteratorServer > python main.py
===== Running on http://127.0.0.1:4000 =====
(Press CTRL+C to quit)
Одобен запрос на обновление. Итерация 1
Одобен запрос на обновление. Итерация 2
Одобен запрос на обновление. Итерация 3
Одобен запрос на обновление. Итерация 4
Одобен запрос на обновление. Итерация 5
ws browser connection closed
```

Рисунок 5.2 — Скриншот вывод отладочной информации.

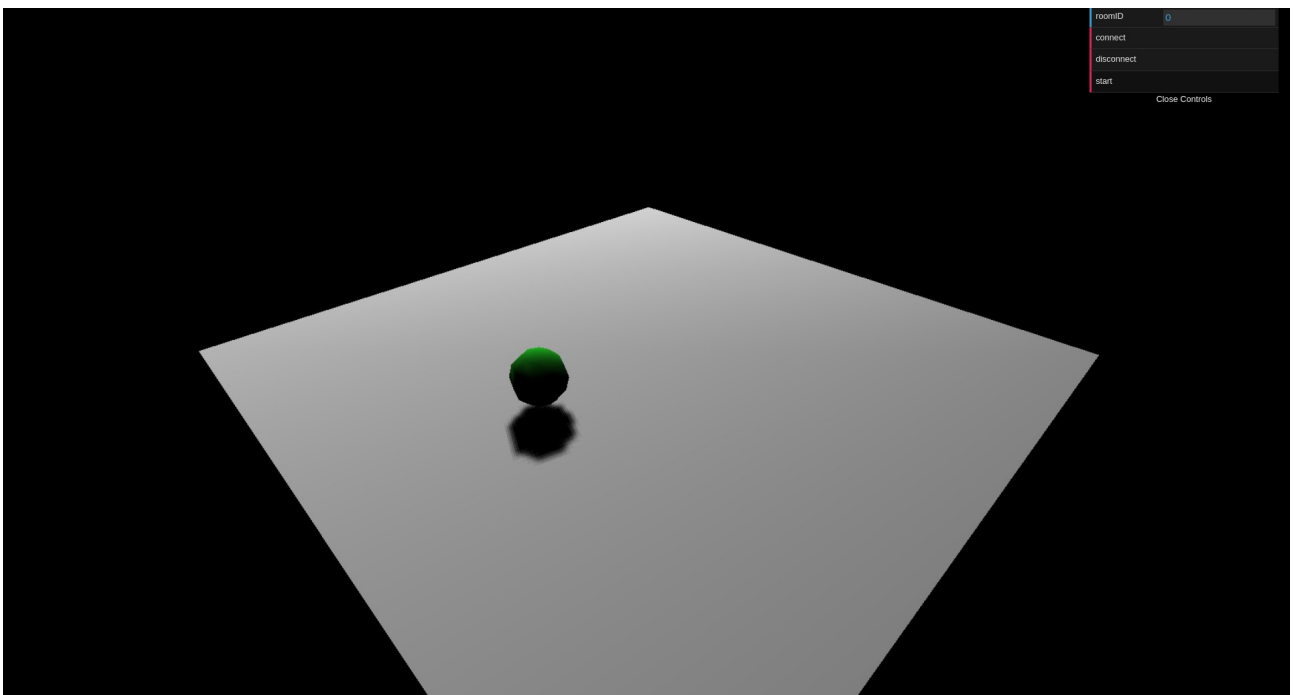
### 7.2 Сайт

Сайт размещен по доменному имени `mazovec.ru`. На рисунке 5.3 представлен внешний вид стартовой страницы. Кнопки управления расположены таким образом, что заполняя поля в порядке от верхнего к нижнему можно запустить среду исполнителя и прервать ее работу.



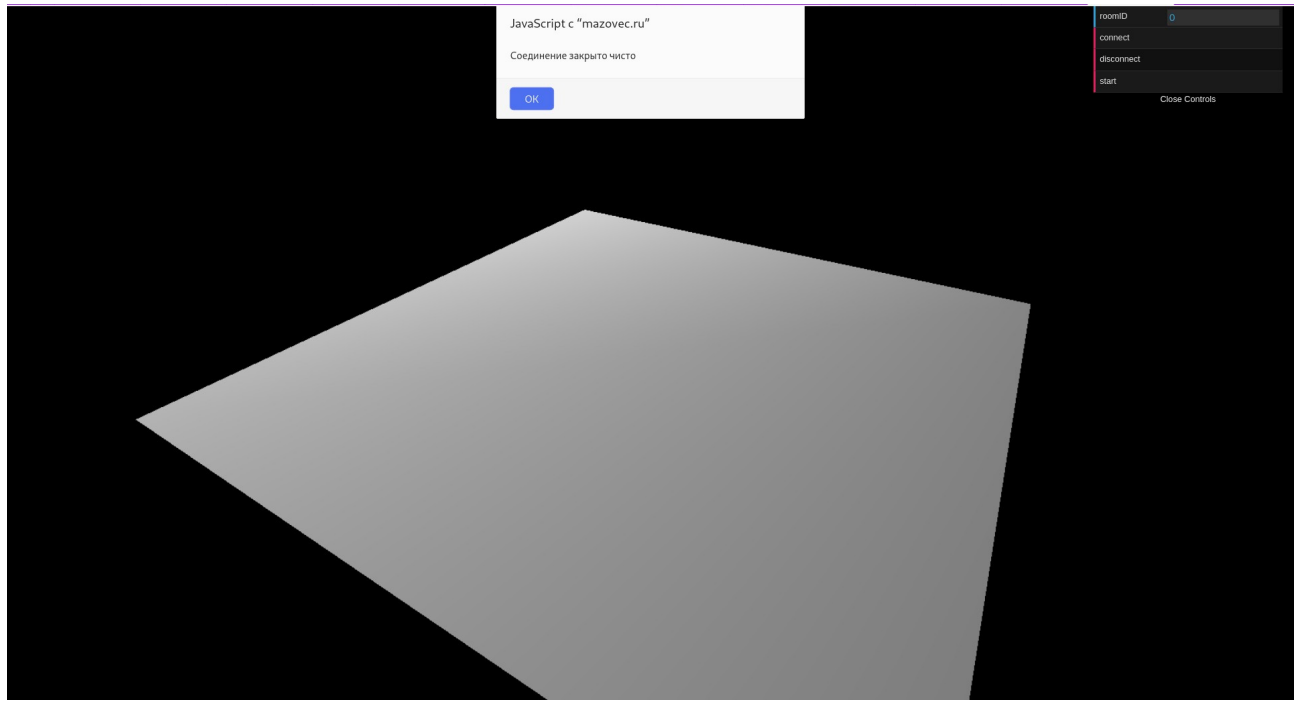
*Рисунок 5.3 - Стартовая страница*

В поле RoomID заносится идентификационный номер созданной клиентом контроллера комнаты, далее требуется произвести подключение к серверу нажатием кнопки connect (сервер Web ГРИСП должен быть запущен и на нем должна существовать комната с указанным ID). После подключения и нажатия кнопки start на поле должен появиться исполнитель (см. Рисунок 5.4).

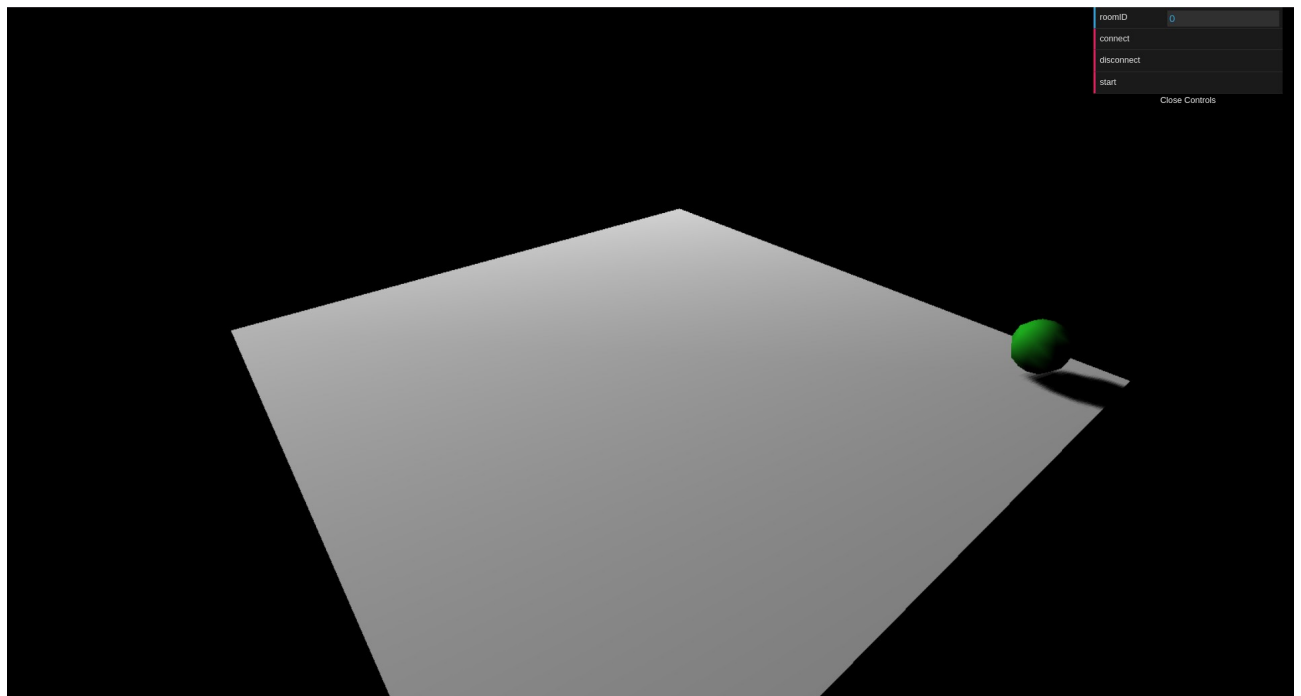


*Рисунок 5.4 - Первая итерация работы исполнителя.*

Если для исполнителя был описан какой-либо алгоритм на клиенте контроллера, то он немедленно начнет исполнять его, при соблюдении всех условий, заданных при создании комнаты (минимальное число пользователей и т. д.). Работу исполнителя можно прерывать и возобновлять (см. Рисунок 5.5 — 5.6).



*Рисунок 5.5 — Приостановленная работа исполнителя*



*Рисунок 5.6 — возобновление работы исполнителя*

## 8 Маркетинговый анализ и построение бизнес-модели проекта

Структура разрабатываемой бизнес-модели, опираясь на работу Александра Остервальдера и Ива Пинье, кратко описывает девять элементов шаблона бизнес-модели [9]:

- Потребительские сегменты (ПС);
- Ценностные предложения (ЦП);
- Каналы сбыта (КС);
- Взаимоотношения с клиентами (ВК);
- Потоки поступления доходов (ПД);
- Ключевые ресурсы (КР);
- Ключевые виды деятельности (КД);
- Ключевые партнеры (КП);
- Структура издержек (СИ);

### 8.1 Конкурентный анализ

Как правило, ГРИСП разрабатывается в образовательных целях и является инструментом привлечения образовательных учреждений к использованию сред программирования, как пример, среда разработки PascalABC.NET. Такие проекты являются проектами с открытым исходным кодом и поддерживаются OpenSource сообществом и образовательными учреждениями.

Другой пример — исполнители, нацеленные в первую очередь развлекать потребителя, обучая его в процессе игры. Такие проекты распространяются по платной лицензии через игровые торговые площадки. Например, проект Human Resource Machine от Gog, являющийся обучающей игрой, использующей механику исполнителя и в процессе игры обучающий навыкам программирования на низкоуровневых языках программирования, таких как ассемблер. Цена на игровых площадках начинается от 300 рублей.

8.1.1 Проект рассчитан на нишевый рынок, так как тематика проекта ориентирована на особый потребительский сегмент — школьные образовательные

учреждения, учреждения дополнительного школьного образования, кванториумы.

8.1.2 Из первого пункта следует, что ключевыми партнерами в бизнес-модели могут выступать учебные сайты в которые можно будет встраивать функционал разработанного проекта. Это будет выгодно и для образовательных сайтов, как метод привлечения клиентов для прохождения обучения. Они же могут выступать в роли основного канала сбыта разрабатываемого Web сервиса.

8.1.3 Главное ценностное предложение, это предоставление услуги обучения школьников, интересующихся Web разработкой, предоставление удобного средства обучения специальным знаниям, выходящим за рамки школьной программы. Такое ценностное предложение может поспособствовать увеличению конкурентноспособности образовательного учреждения и привлечет мотивированных на дополнительное образование школьников. Так же, для образовательных учреждений, участвующих в разработке или предоставляющих собственное оборудование для размещения сервера разрабатываемый сервис будет распространяться бесплатно.

8.1.4 Основным потоком получения доходов могут выступать образовательные учреждения, не имеющие возможности участвовать в разработке или предоставлять оборудование. Для таких учреждений будет предоставлен бесплатный доступ к исходному коду, но без поддержки, ориентированный исключительно на самообслуживание, либо использование по лицензии. Использование сервиса для предоставления частных образовательных услуг (например, репетиторства) может регулироваться условиями лицензии, но, так как сервис рассчитан на предоставление услуг дополнительного школьного образования, этот вид дохода не будет являться ключевым.

8.1.5 Ключевым видом деятельности будет в дальнейшем будет продолжение разработки и улучшения продукта, а так же разработка учебных задач.

Представленную выше структуру удобно изображать в виде шаблона бизнес-модели (см. Рисунок 7.1).



КП  Учебные сайты (поставка ресурсов и совместная деятельность).	КД  Разработка ПО, Разработка учебных задач.	ЦП  Простое средство обучения специальной теме,  бесплатный доступ при участии в разработке.	ВК  Самообслуживание, прямая связь с разработчиком	ПС  Нишевый рынок — школы, кванториумы.
	КР  Оборудование.		КС  Образовательные учреждения: школы, лицеи, кванториумы	
СИ  Предоставление услуг хостинга.			ПД  Лицензированное использование.	

*Рисунок 7.1 — Шаблон бизнес-модели.*

## 9 Заключение

В результате работы был разработан контроллер Web ГРИСП и реализовано динамическое отображение работы исполнителя в браузере.

В ходе работы над проектом были выполнены следующие задачи:

- 1) Определены пользователи разработанного программного продукта;
- 2) Проанализированы аналогичные решения;
- 3) Определены базовые функции Web ГРИСП;
- 4) Изучены и выбраны для разработки технологии создания Web сервера;
- 5) Изучены и выбраны для разработки технологии взаимодействия с Web-сервисами;
- 6) Составлена модель взаимодействия с контроллером - сервером;
- 7) Составлена UML-диаграмма классов контроллера;
- 8) Разработан контроллер - сервер Web ГРИСП;
- 9) Изучены и выбраны для разработки технологии отрисовки графики в окне браузера;
- 10) Разработан Web-интерфейс Web ГРИСП;
- 11) Выбран хостинг и домен;
- 12) Выполнена публикация работы в сети Интернет.

Цель работы достигнута.

- Исходный код Web сервера см. Приложение А;
- Исходный код Web интерфейса см. Приложение Б.

Были получены знания и навыки разработки Web приложений, создания и настройки Web сервера, использования нотации UML для отображения диаграммы классов, построения бизнес-моделей, программирования на языке JavaScript.

Были закреплены знания и навыки объектно-ориентированного программирования, построения ДРАКОН схем, программирования на языке Python.

### **9.1 Перспективы развития web-приложения**

Интерфейс пользователя требует дальнейшей разработки и оптимизации для маленьких экранов, но несмотря на это, основные заложенные функции выполняются. В дальнейшем можно добавить редактор сцен, чтобы разрабатывать задачи под ГРИСП через удобный интерфейс.

## 10 Список использованных источников

- 1 А. Г. Кушниренко, Г. В. Лебедева и Я. Н. Зайдельмана «Информатика 7–9 классы», М., 2001. — 224 с.
- 2 Математическая логика и теория алгоритмов: учебное пособие / Т. О. Перемитина – Томск : ФДО, ТУСУР, 2016. — 132 с.
- 3 Понятие исполнителя. Режим доступа: <https://www.sites.google.com/a/gkl-kemerovo.ru/informatics/algorithms/ponatie-ispolnitela>.
- 4 Документация по настройке Web сервера nginx. Режим доступа: <https://nginx.org/ru/docs/>.
- 5 Документация по использованию Web сервера Python aiohttp. Режим доступа: <https://docs.aiohttp.org/en/stable/web.html>.
- 6 Документация по использованию Three.js. Режим доступа: <https://threejs.org/docs/index.html#manual/en/introduction/Creating-a-scene>
- 7 Леоненков А.В. Самоучитель UML. – СПб.: БХВ-Петербург, 2002. – 304 с
- 8 Владимир Паронджанов язык ДРАКОН краткое описание. Режим доступа: [https://drakon.su/\\_media/biblioteka/drakondescription.pdf](https://drakon.su/_media/biblioteka/drakondescription.pdf).
- 9 Построение бизнес-моделей: Настольная книга стратега и новатора / Александр Остервальдер, Ив Пинье; Пер. с англ. - М.: Альпина Паблишер, 2011. — 288 с.
- 10 Системная инженерия. Режим доступа: <http://sewiki.ru/OCL>.

## Приложение А

### Исходный код Web сервера

#### Файл main.py

```

1 | from aiohttp import web
2 | from wsplayer import WSPlayer
3 | from wsbrows import WSBrows
4 |
5 |
6 | def main():
7 |     app = web.Application()
8 |     app.router.add_get('/ws', WSBrows) # browser api
9 |     app.router.add_get('/api', WSPlayer) # player api
10 |
11 |     web.run_app(app, host='127.0.0.1', port=4000)
12 |
13 |
14 | if __name__ == '__main__':
15 |     main()

```

#### Файл classes.py

```

1 | from room import *
2 | import json
3 |
4 |
5 | class Object(object):
6 |     globalID: int = 0
7 |
8 |     def __init__(self,
9 |                 position: list = [0, 0],
10 |                 rotation: complex = complex(1, 0)):
11 |         self.id: int = Object.globalID;
12 |         self.room: 'room.Room'
13 |         self.position: list = position
14 |         self.rotation: list = rotation
15 |         #
16 |         Object.globalID += 1
17 |         #
18 |         self.data: dict = {
19 |             "id": self.id,
20 |             "type": "object",
21 |             "pos": {
22 |                 "x": self.position[0],
23 |                 "y": self.position[1]
24 |             },
25 |             "rot": {
26 |                 "x": int(self.rotation.real),
27 |                 "y": int(self.rotation.imag)
28 |             }
29 |         }

```

```

30 |
31 |     def setPosition(self,
32 |         x: int,
33 |         y: int):
34 |         self.position[0] = x
35 |         self.position[1] = y
36 |         self.data["pos"]["x"] = self.position[0]
37 |         self.data["pos"]["y"] = self.position[1]
38 |         return self
39 |
40 |     def setRotation(self,
41 |         x: int,
42 |         y: int):
43 |         self.rotation = complex(x, y)
44 |         self.data["rot"]["x"] = int(self.rotation.real)
45 |         self.data["rot"]["y"] = int(self.rotation.imag)
46 |         return self
47 |
48 |     def getData(self):
49 |         a = {}
50 |         a.update(self.data)
51 |         return a
52 |
53 |
54 | class Static(Object):
55 |     def __init__(self,
56 |         position: list = [0, 0],
57 |         rotation: complex = complex(1, 0)):
58 |         super().__init__(position, rotation)
59 |         self.data["type"] = "static"
60 |
61 |
62 | class Movable(Object):
63 |     def __init__(self,
64 |         position: list = [0, 0],
65 |         rotation: complex = complex(1, 0)):
66 |         super().__init__(position, rotation)
67 |         self.data["type"] = "movable"
68 |         # Заголовок для временных данных
69 |         self.data.update({"update": {}})
70 |
71 |     def lookAtXY(self):
72 |         """ Возвращает координату, на которую в данных момент
    смотрит объект """
73 |         look = [self.position[0] + int(self.rotation.real),
74 |                 self.position[1] + int(self.rotation.imag)]
75 |         return look
76 |
77 |
78 |     def look(self):
79 |         """ Возвращает объект, который стоит непосредственно
    перед объектом,
80 |         если такой есть

```

```

81 |         """
82 |         for obj in self.room.getObjects():
83 |             if obj.position == self.lookAtXY():
84 |                 self.data["update"].update({"look": {}})
85 |                 self.data["update"]["look"].update({
86 |                     "type": obj.data["type"],
87 |                     "pos": obj.data["pos"]
88 |                 })
89 |             return obj
90 |         return None
91 |
92 |     def lookForward(self):
93 |         """ Возвращает объект, на который смотрит объект, если
такой есть """
94 |         front = []
95 |         for i in range(1, 20):
96 |             front = [self.position[0] +
int(self.rotation.real)*i,
97 |                     self.position[1] +
int(self.rotation.imag)*i]
98 |             for obj in self.room.getObjects():
99 |                 if front == obj.position:
100 |                     self.data["update"].update({"lookF": {}})
101 |                     self.data["update"]["lookF"].update({
102 |                         "type": obj.data["type"],
103 |                         "pos": obj.data["pos"]
104 |                     })
105 |                 return obj
106 |         return None
107 |
108 |     def move(self):
109 |         if not self.look():
110 |             lookAtXY = self.lookAtXY()
111 |             self.setPosition(lookAtXY[0], lookAtXY[1])
112 |         return self
113 |
114 |     def turnLeft(self):
115 |         self.rotation *= complex(0, 1)
116 |         self.data["rot"]["x"] = int(self.rotation.real)
117 |         self.data["rot"]["y"] = int(self.rotation.imag)
118 |         return self
119 |
120 |     def turnRight(self):
121 |         self.rotation *= complex(0, -1)
122 |         self.data["rot"]["x"] = int(self.rotation.real)
123 |         self.data["rot"]["y"] = int(self.rotation.imag)
124 |         return self
125 |
126 |     def lookLeft(self):
127 |         """ Возвращает объект слева от объекта, если такой
есть """
128 |         obj = self.turnLeft().look()
129 |         self.turnRight()

```

```

130 |         if obj:
131 |             self.data.update({"lookL": {}})
132 |             self.data["update"]["lookL"].update({ "type":
obj.data["type"], "pos": obj.data["pos"]
133 |             })
134 |             return obj
135 |
136 |     def lookRight(self):
137 |         """ Возвращает объект справа от объекта, если такой
есть """
138 |         obj = self.turnRight().look()
139 |         self.turnLeft()
140 |         if obj:
141 |             self.data.update({"lookR": {}})
142 |             self.data["update"]["lookR"].update({
143 |                 "type": obj.data["type"],
144 |                 "pos": obj.data["pos"]
145 |             })
146 |             return obj
147 |
148 |     def getData(self):
149 |         """ Возвращет словарь данных объекта, очищает поле
временных данных """
150 |         a = {}
151 |         a.update(self.data)
152 |         self.data["update"] = {}
153 |         return a
154 |
155 |
156 | class Player(Movable):
157 |     def __init__(self,
158 |                 name: str,
159 |                 health: int = 100,
160 |                 position: list = [0, 0],
161 |                 rotation: complex = complex(1, 0)):
162 |         super().__init__(position, rotation)
163 |         self.data["type"] = "player"
164 |         self.name = name
165 |         self.health = health
166 |         self.data.update({"name": self.name, "health":
self.health})

```

Файл room.py

```

1 | from classes import Static, Player, Movable
2 | from aiohttp import web
3 | from random import *
4 |
5 |
6 | class Room(object):
7 |     globalID: int = 0
8 |
9 |     def __init__(self,

```



```

10 |         passwd: str,
11 |         minPlayers: int = 2,
12 |         maxPlayers: int = 2):
13 |     if minPlayers < 2:
14 |         minPlayers = 2
15 |     if maxPlayers < minPlayers:
16 |         maxPlayers = minPlayers
17 |     self.id: int = Room.globalID
18 |     self.passwd: str = passwd
19 |     self.players: dict = {}
20 |     self.movable: dict = {}
21 |     self.statics: dict = {}
22 |     # Ограничения на количество игроков в комнате
23 |     self.maxPlayers: int = maxPlayers
24 |     self.minPlayers: int = minPlayers
25 |     # Подключённые в данный момент сессии
26 |     self.wsBrows: list = []
27 |     self.wsPlayers: list = []
28 |     # Итерация комнаты и глобальный ID
29 |     self.iter: int = 0
30 |     Room.globalID += 1
31 |
32 |     def getObjects(self):
33 |         a = list(self.movable.values()) + \
34 |             list(self.players.values()) + \
35 |             list(self.statics.values())
36 |         return a
37 |
38 |     def createPlayer(self,
39 |                     name: str):
40 |         if len(self.players) < self.maxPlayers:
41 |             player = Player(name, position=[randint(-3, 3),
42 |             randint(-3, 3)])
43 |             player.room = self
44 |             self.players[player.id] = player
45 |             return player
46 |         return None
47 |
48 |     def createStatic(self,
49 |                     position: list,
50 |                     rotation: complex = complex(1, 0)):
51 |         static = Static(position, rotation)
52 |         static.room = self
53 |         self.statics[static.id] = static
54 |
55 |     def createMovable(self,
56 |                     position: list,
57 |                     rotation: complex = complex(1, 0)):
58 |         movable = Movable(position, rotation)
59 |         movable.room = self
60 |         self.movable[movable.id] = movable
61 |
62 |     async def sendToPlayers(self, data: dict):

```

```

62 |         for pl in self.wsPlayers:
63 |             if pl.closed:
64 |                 self.wsPlayers.remove(pl)
65 |                 print(f'Отключился игрок {pl}')
66 |             else:
67 |                 await pl.send_json(data)
68 |
69 |     async def sendToBrowsers(self, data: dict):
70 |         for b in self.wsBrows:
71 |             if b.closed:
72 |                 self.wsBrows.remove(b)
73 |                 print(f'Отключился клиент {b}')
74 |             else:
75 |                 await b.send_json(data)
76 |
77 |     async def update(self,
78 |                       iteration: int):
79 |         if iteration > self.iter:
80 |             print(f'Одобен запрос на обновление. Итерация
81 | {iteration}')
82 |             self.iter = iteration
83 |             for obj in list(self.movable.values()) + \
84 |                 list(self.statics.values()):
85 |                 mvblData = mvbl.getData()
86 |                 await self.sendToBrowsers({
87 |                     'e': 'getUpdate',
88 |                     'id': mvblData['id'],
89 |                     'type': mvblData['type'],
90 |                     'pos': mvblData['pos'],
91 |                     'rot': mvblData['rot']
92 |                 })
93 |                 await self.sendToPlayers({'e': 'updatePl'})
94 |                 return {'e': 'getUpdate', 'm': 'ok'}
95 |             return {'e': 'getUpdate', 'm': 'error', 'iter':
96 | self.iter }
97 |
98 | class RoomsContainer(object):
99 |     def __init__(self,
100 |                  maxRooms: int = 10):
101 |         self.rooms = {}
102 |         self.maxRooms = maxRooms
103 |
104 |     def createRoom(self,
105 |                   passwd: str,
106 |                   minPlayers: int = 2,
107 |                   maxPlayers: int = 2):
108 |         if len(self.rooms) < self.maxRooms:
109 |             room = Room(passwd, minPlayers, maxPlayers)
110 |             self.rooms[room.id] = room
111 |             return room
112 |         return None

```

```

113 |     def connectRoomView(self,
114 |         roomID: int):
115 |         if roomID in self.rooms:
116 |             return self.rooms[roomID]
117 |         return None
118 |
119 |     def connectRoomPlayer(self,
120 |         roomID: int,
121 |         passwd: str):
122 |         if roomID in self.rooms:
123 |             if self.rooms[roomID].passwd == passwd:
124 |                 return self.room[roomID]
125 |             return None
126 |
127 | roomsContainer = RoomsContainer()

```

Файл wsBrows.py

```

1 | from room import roomsContainer
2 | from aiohttp import web
3 | import asyncio
4 | import aiohttp
5 | import room
6 | import classes
7 | import json
8 |
9 |
10 | class WSBrows(web.View):
11 |     def __init__(self, request):
12 |         self.roomsContainer = roomsContainer
13 |         self.room: 'classes.Room' = None
14 |         self._request = request
15 |
16 |     async def connectRoom(self,
17 |         ws,
18 |         data: dict):
19 |         await asyncio.sleep(0)
20 |         room =
self.roomsContainer.connectRoomView(data['roomID'])
21 |         self.room = None
22 |         if room:
23 |             self.room = room
24 |             self.room.wsBrows.append(ws)
25 |             return {'e': 'connect', 'm': 'ok'}
26 |         return {'e': 'connect', 'm': 'error'}
27 |
28 |     async def getUpdate(self,
29 |         ws,
30 |         data: dict):
31 |         await asyncio.sleep(0)
32 |         if self.room:
33 |             return await self.room.update(data['iter'])
34 |         return {'e': 'getUpdate', 'm': 'error'}

```

```

35 |
36 |     async def get(self):
37 |         ws = web.WebSocketResponse()
38 |         await ws.prepare(self.request)
39 |         dispatcher = {
40 |             'connect': self.connectRoom,
41 |             'getUpdate': self.getUpdate
42 |         }
43 |         #
44 |         async for msg in ws:
45 |             if msg.type == web.WSMsgType.text:
46 |                 if msg.data == 'close':
47 |                     await ws.close()
48 |                 else:
49 |                     event = json.loads(msg.data)
50 |                     if ws.closed:
51 |                         print(f'ws browser connection closed')
52 |                         self.room.wsBrows.remove(ws)
53 |                         await ws.close()
54 |                     else:
55 |                         await ws.send_json(await
dispatcher[event['e']](ws=ws, data=event))
56 |                 elif msg.type == web.WSMsgType.error:
57 |                     print(f'error {ws.exception()}')
58 |             #
59 |             self.room.wsBrows.remove(ws)
60 |             print(f'ws browser connection closed')
61 |             return ws

```

Файл wsplayer.py

```

1 | from room import roomsContainer
2 | from aiohttp import web
3 | import asyncio
4 | import aiohttp
5 | import room
6 | import classes
7 | import json
8 |
9 |
10 | class WSPlayer(web.View):
11 |     def __init__(self, request):
12 |         self.roomsContainer = roomsContainer
13 |         self.room: 'classes.Room' = None
14 |         self.player: 'classes.Player' = None
15 |         self._request = request
16 |         ws = None
17 |
18 |     async def createRoom(self,
19 |                          ws,
20 |                          data: dict):
21 |         await asyncio.sleep(0)

```

```

22 |         room = self.roomsContainer.createRoom(data['passwd'],
data['minPl'], data['maxPl'])
23 |         self.room = None
24 |         self.player = None
25 |         if room:
26 |             self.room = room
27 |             self.room.wsPlayers.append(ws)
28 |             return {'e': 'create', 'm': 'roomID ' +
str(self.room.id)}
29 |         return {'e': 'create', 'm': 'error'}
30 |
31 |     async def connectRoom(self,
32 |         ws,
33 |         data: dict):
34 |         await asyncio.sleep(0)
35 |         room =
self.roomsContainer.connectRoomPlayer(data['roomID'],
data['passwd'])
36 |         self.room = None
37 |         self.player = None
38 |         if room:
39 |             self.room = room
40 |             self.room.wsPlayers.append(ws)
41 |             return {'e': 'connect', 'm': 'ok'}
42 |         return {'e': 'connect', 'm': 'error'}
43 |
44 |     async def createPlayer(self,
45 |         ws,
46 |         data: dict):
47 |         await asyncio.sleep(0)
48 |         if self.room:
49 |             player = self.room.createPlayer(data['name'])
50 |             self.player = None
51 |             if player:
52 |                 self.player = player
53 |                 return {'e': 'createPl', 'm': 'ok'}
54 |             return {'e': 'createPl', 'm': 'error'}
55 |
56 |     async def getUpdate(self,
57 |         ws,
58 |         data: dict):
59 |         await asyncio.sleep(0)
60 |         if self.player:
61 |             self.player.look()
62 |             self.player.lookForward()
63 |             self.player.lookLeft()
64 |             self.player.lookRight()
65 |             a = {'e': 'getUpdate'}
66 |             a.update(self.player.getData())
67 |             return a
68 |         return {'e': 'getUpdate', 'm': 'error'}
69 |
70 |     async def updatePlayer(self,

```

```

71 |         ws,
72 |         data: dict):
73 |     if self.player:
74 |         dispatcher = {
75 |             'skip': self.player.look,
76 |             'move': self.player.move,
77 |             'turnLeft': self.player.turnLeft,
78 |             'turnRight': self.player.turnRight
79 |         }
80 |         dispatcher[data['do']]()
81 |         plData = self.player.getData()
82 |         await self.room.sendToBrowsers({
83 |             'e': 'getUpdate',
84 |             'id': plData['id'],
85 |             'type': plData['type'],
86 |             'pos': plData['pos'],
87 |             'rot': plData['rot']
88 |         })
89 |         return {'e': 'updatePl', 'm': 'ok'}
90 |     return {'e': 'updatePl', 'm': 'error'}
91 |
92 |     async def get(self):
93 |         ws = web.WebSocketResponse()
94 |         await ws.prepare(self.request)
95 |         dispatcher = {
96 |             'create': self.createRoom,
97 |             'connect': self.connectRoom,
98 |             'createPl': self.createPlayer,
99 |             'getUpdate': self.getUpdate,
100 |             'updatePl': self.updatePlayer
101 |         }
102 |         #
103 |         async for msg in ws:
104 |             if msg.type == web.WSMsgType.text:
105 |                 if msg.data == 'close':
106 |                     await ws.close()
107 |                 else:
108 |                     event = json.loads(msg.data)
109 |                     if ws.closed:
110 |                         print(f'ws player connection closed')
111 |                         self.room.wsPlayers.remove(ws)
112 |                         await ws.close()
113 |                     else:
114 |                         await ws.send_json(await
dispatcher[event['e']](ws=ws, data=event))
115 |                         elif msg.type == web.WSMsgType.error:
116 |                             print(f'error {ws.exception()}')
117 |                         #
118 |                         self.room.wsPlayers.remove(ws)
119 |                         print(f'ws player connection closed')
120 |                         return ws

```

## Приложение Б

### Исходный код Web-интерфейса

#### Файл index.html

```

1 | <!DOCTYPE html>
2 | <html>
3 |     <head>
4 |         <title>canvas</title>
5 |         <meta charset="utf-8">
6 |         <link rel="stylesheet" type="text/css"
  href="css/style.css" />
7 |     </head>
8 |
9 |     <body>
10 |         <canvas id="c"></canvas>
11 |     </body>
12 |
13 | <script type="module" src="js/classes.js"></script>
14 | <script type="module" src="js/websocket.js"></script>
15 | <script type="module" src="js/main.js"></script>
16 | </html>

```

#### Файл style.css

```

1 | html, body {
2 |     margin: 0;
3 |     height: 100%;
4 | }
5 |
6 | #c {
7 |     width: 100%;
8 |     height: 100%;
9 |     display: block;
10 | }

```

#### Файл main.js

```

1 | import * as THREE from
  'https://threejsfundamentals.org/threejs/resources/threejs/r11
  9/build/three.module.js';
2 | import { GUI } from
  'https://threejsfundamentals.org/threejs/.../3rdparty/dat.gui.m
  odule.js';
3 | import { room, roomController } from './websocket.js'
4 | import { scene } from './classes.js'
5 |
6 |
7 | const canvas = document.querySelector('#c');
8 |
9 | const fov = 60;
10 | const aspect = 2;

```

```

11 | const near = 1;
12 | const far = 300;
13 |
14 |
15 | let renderer = new THREE.WebGLRenderer({canvas});
16 | renderer.shadowMapEnabled = true;
17 |
18 |
19 | let camera = new THREE.PerspectiveCamera(fov, aspect, near,
    far);
20 | camera.position.set(0, 50, 0);
21 | camera.lookAt(0, 0, 0);
22 | let clock = new THREE.Clock();
23 | let angle = 0; // текущий угол камеры
24 | let angularSpeed = THREE.Math.degToRad(5); // угловая скорость
25 | let delta = 0;
26 | let radius = 80;
27 |
28 |
29 | let gui = new GUI();
30 | gui.add(roomController, 'roomID');
31 | gui.add(roomController, 'connect');
32 | gui.add(roomController, 'disconnect');
33 | gui.add(roomController, 'start');
34 |
35 |
36 | let spotLight = new THREE.SpotLight(0xffffffff );
37 | spotLight.position.set( -40, 60, -10 );
38 | spotLight.castShadow = true;
39 | scene.add(spotLight );
40 |
41 |
42 | let planeGeometry = new THREE.PlaneGeometry(100,100);
43 | let planeMaterial = new THREE.MeshLambertMaterial({color:
    0xffffffff});
44 | let plane = new THREE.Mesh(planeGeometry,planeMaterial);
45 | plane.rotation.x = -0.5*Math.PI;
46 | plane.position.x = 0;
47 | plane.position.y = 0;
48 | plane.position.z = 0;
49 | plane.receiveShadow = true;
50 | scene.add(plane);
51 |
52 |
53 | function resizeRendererToDisplaySize(renderer) {
54 |     const canvas = renderer.domElement;
55 |     const width  = canvas.clientWidth;
56 |     const height = canvas.clientHeight;
57 |     const needResize = canvas.width !== width || canvas.height
        !== height;
58 |     if (needResize) {
59 |         renderer.setSize(width, height, false);
60 |     }

```



```

61 |     return needResize;
62 | }
63 |
64 | function renderScene() {
65 |     camera.position.x = Math.cos(angle) * radius;
66 |     camera.position.z = Math.sin(angle) * radius;
67 |     angle += angularSpeed * delta;
68 |
69 |     camera.lookAt(0, 0, 0);
70 |
71 |     if(resizeRendererToDisplaySize(renderer)) {
72 |         const canvas = renderer.domElement;
73 |         camera.aspect = canvas.clientWidth /
canvas.clientHeight;
74 |         camera.updateProjectionMatrix();
75 |     }
76 |     for(let obj in room.objects) {
77 |         room.objects[obj].move();
78 |     }
79 |
80 |     renderer.render(scene, camera);
81 |     delta = clock.getDelta();
82 |     requestAnimationFrame(renderScene);
83 | }
84 |
85 | renderScene();

```

Файл classes.js

```

1 | import * as THREE from
  | 'https://threejsfundamentals.org/threejs/resources/threejs/r11
  | 9/build/three.module.js';
2 |
3 |
4 | let shift = 10;
5 | let scene = new THREE.Scene();
6 |
7 | class GObject {
8 |     constructor(id, type, x, y, rot, geometry, material) {
9 |         this.id = id;
10 |         this.type = type;
11 |         this.x = x;
12 |         this.y = y;
13 |         this.rot = rot;
14 |
15 |         this.mesh = new THREE.Mesh(geometry, material);
16 |         this.mesh.position.x = shift * this.x;
17 |         this.mesh.position.z = shift * this.y;
18 |         this.mesh.position.y = 5;
19 |         this.mesh.castShadow = true;
20 |
21 |         scene.add(this.mesh);
22 |     }

```

```

23 |
24 |     setPos(x, y) {
25 |         this.x = x;
26 |         this.y = y;
27 |         return this;
28 |     }
29 |
30 |     setRot(rot) {
31 |         this.rot.x = rot.x;
32 |         this.rot.y = rot.y;
33 |         return this;
34 |     }
35 |
36 |     move() {
37 |         let dx = (shift * this.x - this.mesh.position.x);
38 |         let dy = (shift * this.y - this.mesh.position.z);
39 |         if(Math.abs(dx) >= 0.01) {
40 |             this.mesh.position.x += Math.tanh(0.4 * dx);
41 |         }
42 |         if(Math.abs(dy) >= 0.01) {
43 |             this.mesh.position.z += Math.tanh(0.4 * dy);
44 |         }
45 |     }
46 |
47 |     remove() {
48 |         scene.remove(this.mesh);
49 |     }
50 | }
51 |
52 | class Room {
53 |     constructor(roomID) {
54 |         this.id = roomID
55 |         this.objects = {};
56 |         this.connected = false;
57 |         this.iter = 0;
58 |     }
59 |
60 |     connectRoom() {
61 |         return {
62 |             e: 'connect',
63 |             roomID: this.id,
64 |         };
65 |     }
66 |
67 |     disconnectRoom() {
68 |         for(let obj in this.objects) {
69 |             this.objects[obj].remove();
70 |             delete this.objects[obj];
71 |         }
72 |         this.id = 0;
73 |         this.objects = {};
74 |         this.connected = false;
75 |         this.iter = 0;

```

```

76 |     }
77 |
78 |     getUpdate() {
79 |         this.iter += 1;
80 |         return {
81 |             e: 'getUpdate',
82 |             iter: this.iter,
83 |         };
84 |     }
85 |
86 |     updateObject(objID, type, x, y, rot) {
87 |         if(objID.toString() in this.objects) {
88 |             this.objects[objID.toString()].setPos(x,
89 | y).setRot(rot);
90 |         } else {
91 |             let geometry;
92 |             let material;
93 |             if(type == 'player') {
94 |                 geometry = new THREE.SphereGeometry(4, 8, 6);
95 |                 material = new
96 | THREE.MeshLambertMaterial({color: 0x22BB22});
97 |             } else if(type == 'movable') {
98 |                 geometry = new THREE.CylinderGeometry(3, 4, 9,
99 | 8, 1);
100 |                 material = new
101 | THREE.MeshLambertMaterial({color: 0xBB2222});
102 |             } else if(type == 'static') {
103 |                 geometry = new THREE.BoxGeometry( 10, 6, 10 );
104 |                 material = new
105 | THREE.MeshLambertMaterial({color: 0BBBBBFF});
106 |             } else {
107 |                 geometry = new THREE.SphereGeometry(4, 8, 6);
108 |                 material = new
109 | THREE.MeshLambertMaterial({color: 0BBBBBFF});
110 |             }
111 |             this.objects[objID.toString()] = new
112 | GObject(objID, type, x, y, rot, geometry, material);
113 |         }
114 |     }
115 | }
116 |
117 | export { Room, GObject, scene };

```

Файл websocket.js

```

1 | import * as THREE from
  | 'https://threejsfundamentals.org/threejs/resources/threejs/r11
  | 9/build/three.module.js';
2 | import { Room, GObject } from './classes.js';
3 |
4 | let ws;
5 | let connectID;
6 | let connectDelay = 2000;

```

```

7 | let connectTimes = 15;
8 |
9 | let timerID;
10 | let gameDelay = 1000;
11 |
12 | function wsStart(host='ws://localhost/ws') {
13 |     ws = new WebSocket(host);
14 |
15 |     if(connectTimes == 0) {
16 |         alert("Проблемы с подключением к серверу")
17 |         window.location.href = "/error/onServerError.html";
18 |     }
19 |
20 |     ws.onopen = function(event) {
21 |         connectTimes = 15;
22 |         clearInterval(connectID);
23 |         if(room.connected) {
24 |             alert("Соединение установлено");
25 |             clearInterval(timerID);
26 |             timerID = setInterval(function() {
27 |                 ws.send(JSON.stringify(room.getUpdate()));
28 |             }, gameDelay);
29 |         } else {
30 |             ws.send(JSON.stringify(room.connectRoom()));
31 |         }
32 |     }
33 |
34 |     ws.onclose = function(event) {
35 |         if(event.wasClean) {
36 |             alert("Соединение закрыто чисто");
37 |             ws = false;
38 |         } else {
39 |             ws = null;
40 |             clearInterval(timerID);
41 |             connectTimes -= 1;
42 |             connectID = setTimeout(wsStart, connectDelay,
43 | host);
44 |         }
45 |     }
46 |
47 |     ws.onmessage = function(event) {
48 |         let resp = JSON.parse(event.data)
49 |
50 |         if(resp.e == 'connect' && resp.m == 'ok') {
51 |             room.connected = true;
52 |         }
53 |         if('m' in resp) {
54 |             if(resp.m == 'error' && 'iter' in resp) {
55 |                 room.iter = resp.iter;
56 |             }
57 |         } else {
58 |             console.log(event.data);

```

```

58 |         room.updateObject(resp.id, resp.type, resp.pos.x,
    | resp.pos.y, resp.rot);
59 |     }
60 | }
61 |
62 |     ws.onerror = function(event) {
63 |     }
64 | }
65 | //wsStart('ws://mazovec.ru:8000/ws');
66 |
67 |
68 | let room = new Room(0);
69 |
70 | let roomController = {
71 |     roomID: 0,
72 |     connect: function() {
73 |         if(!room.connected) {
74 |             room.disconnectRoom();
75 |             room.id = this.roomID;
76 |             if(!ws) { wsStart('ws://46.161.155.209/ws'); }
77 |         }
78 |     },
79 |     disconnect: function() {
80 |         if(room.connected) {
81 |             room.disconnectRoom();
82 |             clearInterval(timerID);
83 |             clearTimeout(connectID);
84 |             if(ws) { ws.close(); }
85 |         }
86 |     },
87 |     start: function() {
88 |         if(room.connected) {
89 |             clearInterval(timerID);
90 |             timerID = setInterval(function() {
91 |                 if(ws)
    | { ws.send(JSON.stringify(room.getUpdate())); }
92 |                 }, gameDelay);
93 |         }
94 |     }
95 | }
96 |
97 | export { room, roomController };

```