

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»

Профиль подготовки: «Программное обеспечение средств вычислительной
техники и автоматизированных систем»

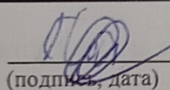
БАКАЛАВРСКАЯ РАБОТА

на тему

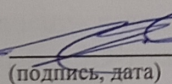
ИНТЕРАКТИВНЫЙ 3D-ТРЕНАЖЁР

«ЭВАКУАЦИЯ ИЗ ЗДАНИЯ»

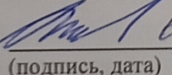
Студент

 03.06.24 Пантюшов Егор Игоревич
(подпись, дата)

Руководитель

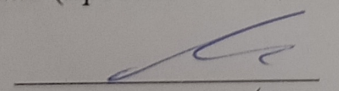
 С.А. Зинкин
(подпись, дата) 03.06.24

Нормоконтролёр

 03.06.24 А.С. Бычков
(подпись, дата)

Работа допущена к защите (протокол заседания кафедры от 27.05.24 № 11)

Заведующий кафедрой


(подпись)

М.А. Митрохин
(фамилия, инициалы)

Работа защищена с отметкой Отлично (протокол заседания ГЭК от

17.06.2024г. № 27)

Секретарь ГЭК


(подпись)

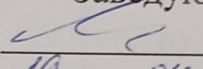
Гурин Е.И.
(фамилия, инициалы)

Пенза 2024

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
 М.А. Митрохин
« 19 » 04 20 24 г.

**ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ**

1. Студент Пантюшов Егор Игоревич гр.20ВВП2 факультета вычислительной техники

2. Тема работы: Интерактивный 3D-тренажёр «Эвакуация из здания»

Тема утверждена приказом ПГУ № 405/о от 19 апреля 2024 г.

3. Руководитель бакалаврской работы д.т.н., профессор ВТ ПГУ Зинкин С. А

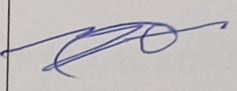
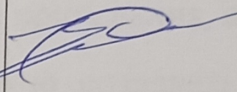
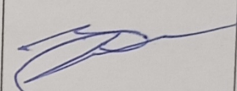
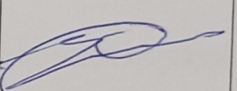
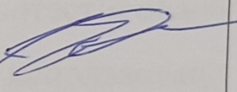
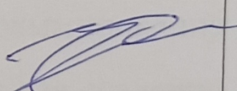
4. Задание на работу (назначение разработки, исходные данные и т.п.)

Разработать интерактивный 3D-тренажёр «Эвакуация из здания», позволяющий пользователю интерактивно пройти симуляцию эвакуации из здания при пожаре. Приложение должно позволять пользователю перемещаться по зданию от первого лица, совершать прыжки и приседания. Приложение должно создавать здания формы многоугольника, заданного пользователем с учётом проёмов, этажей и окон. Также приложение должно поддерживать запуск в браузере.

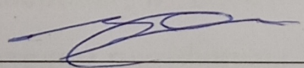
5. Перечень подлежащих разработке вопросов

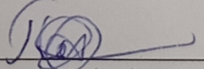
Требуется произвести анализ предметной области: изучить существующие аналоги, изучить способы проектирования и разработки 3D-приложения. Также необходимо изучить технологий для разработки приложения, такие как *Bevy*, *Trunk* и т.д.

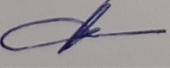
6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Изучение предметной области	50	26.04.2024-30.04.2024	
2.	Анализ требований	10	01.05.2024-02.05.2024	
3.	Выбор технологий, применяемых при проектировании и разработке	50	03.05.2024-07.05.2024	
4.	Описание алгоритмов	40	08.05.2024-11.05.2024	
5.	Разработка интерактивного 3D-тренажёра «Эвакуация из здания»	150	12.05.2024-29.05.2024	
6.	Оформление пояснительной записки	24	30.05.2024-31.05.2024	

Дата выдачи задания 19 апреля 2024 г.

Руководитель бакалаврской работы  Зинкин С.А.
(подпись, дата)

Задание к исполнению принял студент  Пантюшов Е.И.
(подпись, дата)

Работу к защите допустить
Декан факультета  Фионова Л.Р.
13.06.2024

Реферат

Бакалаврская работа содержит 103 листов, 25 рисунков, 6 источников и 4 приложения

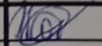
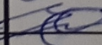
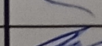
3D-ПРИЛОЖЕНИЕ, RUST, BEVY, TRUNK, ECS, RUSTROVER, WEB-ASSEMBLY

Цель работы – разработать интерактивный 3D-тренажёр, позволяющий пользователю пройти симуляцию эвакуации из здания при пожаре.

Разработанное приложение должно позволять пользователям передвигаться по зданию от первого лица, иметь возможность совершать прыжки и приседания, а также поддерживать запуск в браузере. Также приложение должно создавать здания формы многоугольника, заданной пользователем, с учётом этажей, окон и проёмов.

Результатом проделанной работы является проект, который удовлетворяет поставленной задаче. А именно, разработано, реализовано и протестировано 3D-приложение по эвакуации из здания. При разработке было использовано большое количество технологий, к которым относятся: язык программирования Rust, игровой движок Bevy, среда разработки RustRover, инструмент для создания и упаковки веб-приложений Trunk и низкоуровневый язык WebAssembly.

Результаты данной работы применены в работе ООО «СЕТАП» для организации, а также доступны пользователям платформы WebSchool.au.

					ПГУ 1.090301.34.001 ПЗ		
Изм.	Лист	№ докум.	Подпись	Дата			
Разраб.		Пантюшов Е.И.		23.06.24	Интерактивный 3D-тренажёр "Эвакуация из здания"	Лит.	Лист
Провер.		Зинкин С.А.		23.06.24			2
Н. Контр.		Бычков А. С.		23.06.24		ФВТ, каф. ВТ гр. 20ВВП2	
Утверд.		Митрохин М.А.		23.06.24			

Содержание

Перечень сокращений и обозначений.....	5
Введение.....	6
1 Интерактивный 3D-тренажёр «Эвакуация из здания»	8
1.1 Анализ актуальности темы.....	8
1.2 Анализ аналогов интерактивных 3D-тренажёров.....	9
1.2.1 VR-приложение «Fire Safety Lab VR»	10
1.2.2 Программа «Pathfinder»	11
1.2.3 3D-приложение «Emergency Evacuation Simulator»	12
1.3 Постановка задачи.....	13
1.4 Выбор игрового движка.....	14
1.4.1 Игровой движок «Unity»	14
1.4.2 Игровой движок «Unreal Engine»	15
1.4.3 Игровой движок «Bevy»	16
1.5 Выбор языка программирования	17
1.6 Выводы	18
1.7 Выбор инструментов разработки	19
1.8 Архитектура приложения	20
1.9 Разработка приложения	23
1.9.1 Плагин AssetsLoadPlugin	23
1.9.2 Плагин InstanseSpawnerPlugin.....	25
1.9.3 Плагин FloorPlugin	27

1.9.4 Плагин FloorDefaultPlugin	29
1.9.5 Плагин FullViewportPlugin	31
1.9.6 Плагин CrosshairPlugin	33
1.9.7 Внешние плагины.....	34
1.9.8 Запуск приложения	35
1.10 Тестирование работы приложения	37
2 Безопасность жизнедеятельности.....	45
2.1 Вред здоровью при работе с ПК.....	45
2.2 Снижение вреда для здоровья при работе с ПК.....	45
2.3 Техника безопасности при работе с компьютером.....	46
Заключение	47
Список использованных источников	48
Приложение А	49
Приложение А.1 – Диаграмма компонентов	50
Приложение А.2 – Диаграмма последовательностей плагина AssetLoadPlugin	51
Приложение А.3 – Диаграмма последовательностей плагина InstanseSpawnerPlugin.....	52
Приложение А.4 – Диаграмма последовательностей плагина FloorPlugin	53
Приложение Б	54
Приложение В.....	98

Перечень сокращений и обозначений

В настоящей работе применяют следующие сокращения и обозначения

БД – база данных

ПК – персональный компьютер

API (application programming interface) – программный интерфейс

AR (augmented reality) – среда, в которой физические и виртуальные объекты объединяются в реальном времени

ECS (англ. entity component system) – наиболее часто используемый архитектурный шаблон в программном обеспечении для разработки 3D-приложений

GLB (GL Transmission Forma) – формат файлов для хранения 3D-сцен и моделей

IDE (integrated development environment) – набор программных средств, используемый программистами для разработки программного обеспечения

PNG (portable network graphics) – растровый формат хранения графической информации

URL (Uniform Resource Locator) – адрес ресурса в интернете

VR (virtual reality) – технология для погружения в виртуальный мир

WASM (Web Assembly) – язык программирования низкого уровня

					ПГУ 1.090301.34.001 ПЗ	Лист
						5
Изм.	Лист	№ докум.	Подпись	Дата		

Введение

В современном мире, когда безопасность людей является важным приоритетом, особенно важно уметь правильно действовать в чрезвычайных ситуациях. Эвакуация из здания во время пожара или другого опасного происшествия – это навык, который может спасти жизнь и уберечь от негативных последствий.

Традиционные методы обучения эвакуации, такие как лекции, теоретические занятия и письменные инструкции, не всегда обеспечивают достаточную подготовку. Люди могут забывать информацию, полученную на таких занятиях, или не уметь правильно ее применить в стрессовой ситуации.

В последнее время, всё чаще разрабатываются интерактивные 3D-тренажеры, которые предлагают более эффективный подход к навыку эвакуации. Они помогают научиться правильно и безопасно вести себя во время чрезвычайных ситуаций. 3D-моделирование позволяет создать максимально приближенную к реальности обстановку, что повышает эффективность обучения. Благодаря этому тренажеры могут быть адаптированы к различным типам зданий и сценариям эвакуации.

Интерактивные 3D-тренажеры применяются в различных областях. Они могут быть использованы для обучения сотрудников различных организаций правилам эвакуации. Это может помочь повысить уровень безопасности на рабочих местах. Также тренажеры могут быть использованы для подготовки студентов и школьников к действиям в случае различных чрезвычайных ситуациях, что поможет им развить необходимые навыки и знания для безопасной эвакуации из школы или университета. Тренажеры также используются для инструктажа населения по вопросам эвакуации. Они повышают осведомленность людей о правилах эвакуации и делают их более подготовленными к чрезвычайным ситуациям.

В последнее время появляются тренажеры, адаптированные под VR-приложения. Так, пользователи надевают очки виртуальной реальности и

					ПГУ 1.090301.34.001 ПЗ	Лист
						6
Изм.	Лист	№ докум.	Подпись	Дата		

попадают в виртуальную среду здания, из которого им необходимо эвакуироваться. Следуя инструкциям виртуального инструктора, пользователи должны пройти по безопасному маршруту к выходу из здания, избегая при этом различных опасностей. В процессе эвакуации система оценки отслеживает действия пользователей и оценивает их навыки. По завершении эвакуации пользователи получают отчет о своих результатах, который может включать в себя рекомендации по улучшению навыков эвакуации.

Таким образом, интерактивные 3D-тренажеры позволяют создать реалистичную и интерактивную среду для обучения, что повышает его эффективность. Тренажеры могут быть использованы для обучения людей разных возрастов и профессий, что делает их универсальным инструментом для повышения уровня безопасности [1].

					ПГУ 1.090301.34.001 ПЗ	Лист
						7
Изм.	Лист	№ докум.	Подпись	Дата		

1 Интерактивный 3D-тренажёр «Эвакуация из здания»

Начало разработки программного продукта всегда связано с анализом предметной области. Необходимо определить кто будет пользоваться продуктом, выбрать актуальные технологии, которые позволят в будущем поддерживать и развивать продукт. В данном разделе рассматриваются общие теоретические вопросы, связанные с проектированием и реализацией интерактивного 3D-тренажёра. Изучение данных вопросов поможет определить основные проблемы и задачи для разработки программного продукта.

1.1 Анализ актуальности темы

Число пожаров, стихийных бедствий и других чрезвычайных ситуаций в мире постоянно увеличивается. Это приводит к росту числа людей, находящихся в опасности и нуждающихся в эвакуации. Несмотря на усилия по повышению осведомленности общества о правилах эвакуации, многие люди не знают, как правильно действовать в случае чрезвычайной ситуации. Это может привести к панике, травмам и даже смерти людей.

Именно поэтому интерактивные 3D-тренажёры является незаменимым продуктом в современном мире. Традиционные методы обучения эвакуационным процедурам, такие как лекции и практические занятия, не всегда позволяют в полной мере подготовить людей к реальным угрозам. Лекции могут быть скучными и неэффективными, а практические занятия могут быть опасными или нереалистичными. Интерактивные 3D-тренажеры предлагают инновационный подход к обучению, который позволяет создать реалистичную и интерактивную среду для обучения. Это позволяет людям развивать навыки эвакуации в безопасной среде, учиться на своих ошибках без риска для жизни и здоровья и повышать свою уверенность и готовность к действиям в случае чрезвычайной ситуации.

					ПГУ 1.090301.34.001 ПЗ	Лист
						8
Изм.	Лист	№ докум.	Подпись	Дата		

Таким образом, интерактивные 3D-тренажеры являются актуальным и эффективным инструментом для обучения людей эвакуационным процедурам. Они позволяют создать реалистичную и интерактивную среду для обучения, что повышает его эффективность.

Тренажеры могут быть использованы для обучения людей разных возрастов и профессий, что делает их универсальным инструментом для повышения уровня безопасности.

Внедрение интерактивных 3D-тренажеров "Эвакуация из здания" может помочь повысить уровень подготовки населения к действиям в случае чрезвычайной ситуации и спасти жизни.

1.2 Анализ аналогов интерактивных 3D-тренажеров

В современном мире каждый год на рынок выходят новые сервисы и приложения, расширяющие возможности пользователей и предоставляющие инновационные решения для их повседневных задач. Одним из направлений, которое активно развивается, является область обучения и безопасности, особенно в контексте эвакуации из зданий в чрезвычайных ситуациях. В таких ситуациях важно обучить людей действиям в случае пожара или другой чрезвычайной ситуации.

Для анализа подобных приложений мы рассмотрим несколько популярных аналогов, таких как:

- Emergency Evacuation Simulator;
- Fire Safety Lab VR;
- Pathfinder.

Каждый из них представляет собой инструменты для тренировки и симуляции эвакуации из зданий с использованием различных технологий, включая виртуальную реальность и компьютерные моделирования, а проведение анализов аналогов поможет выявить особенности и возможности этих

приложений, их преимущества и недостатки, а также определить требования к разработке интерактивного 3D-тренажера "Эвакуация из здания".

1.2.1 VR-приложение «Fire Safety Lab VR»

Особенностями данного аналога является наличие четырёх видов сценариев которые позволяют узнать последовательность действий при пожаре. Например, вызов пожарных, включение пожарной тревоги, выключение электроснабжения приборов и использование доступных противопожарных средств. Интерфейс VR-приложения «FireSafetyLabVR» представлен на рисунке 1.



Рисунок 1 – FireSafetyLabVR

Приложение «Fire Safety Lab VR» предоставляет своим пользователям широкий спектр функционала. Ниже перечислены основные функции и возможности, которые оно предоставляет:

- перемещения и взаимодействия с предметами в VR;
- нелинейный сценарий пожара, в котором вы должны не только вспомнить последовательность действий, но и принимать логические решения на основе событий;

– теоретическую информацию о классах пожара, основных факторах опасности при пожаре, а также о различных видах противопожарного оборудования и средств индивидуальной защиты;

– выбор режима, который может выбрать пользователь и пройти обучение, поучаствовав в различных сценариях и пройти экзамен [2].

1.2.2 Программа «Pathfinder»

Pathfinder – программа для моделирования эвакуации в чрезвычайных ситуациях с учетом возможности спасения людей, включает в себя пользовательской графический интерфейс для создания модели и модуль для просмотра анимированных трехмерных результатов.

Интерфейс программы «Pathfinder» представлен на рисунке 2.

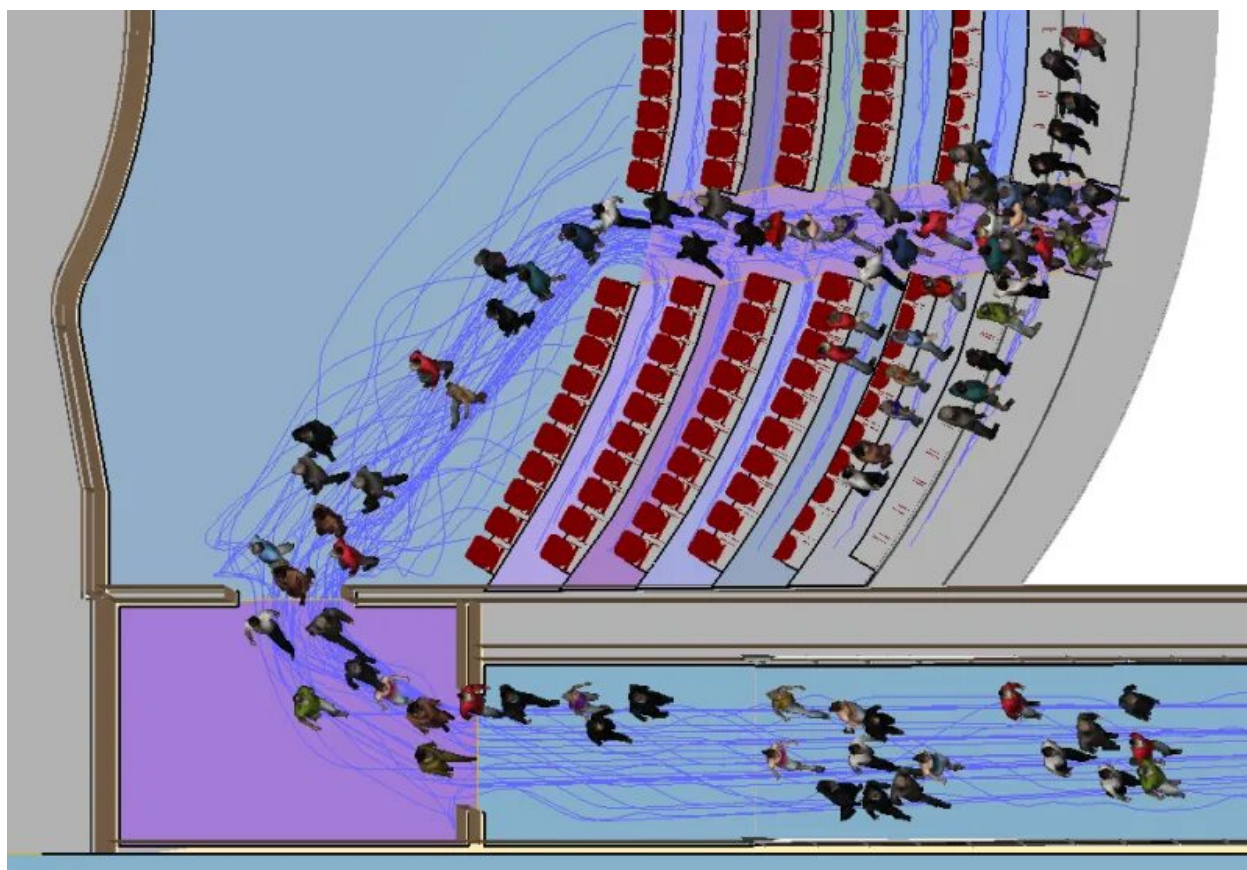


Рисунок 2–Интерфейс программы «Pathfinder»

Основные особенности Pathfinder:

– моделирование движения. Pathfinder использует алгоритмы

моделирования движения для анализа потока людей в здании в различных сценариях эвакуации;

– реалистичная симуляция. Программа предоставляет возможность создавать реалистичные модели зданий и окружающей среды для точного моделирования ситуации эвакуации;

– визуализация и анализ. Pathfinder позволяет визуализировать результаты моделирования в виде графиков, диаграмм и анимаций, что помогает анализировать эффективность эвакуационных планов и выявлять узкие места.

1.2.3 3D-приложение «Emergency Evacuation Simulator»

Основная цель этого приложения – провести обучение эвакуации сотрудников, работающих в больших зданиях со сложной планировкой этажей и маршрутами эвакуации.

Интерфейс приложения «EmergencyEvacuationSimulator» представлен на рисунке 3.



Рисунок 3–Интерфейс программы «Emergency Evacuation Simulator»

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.34.001 ПЗ

Лист

12

Основные особенности Emergency Evacuation Simulator:

- реалистичное моделирование различных сценариев чрезвычайных ситуаций: пожар, утечка газа, дым, землетрясение, торнадо, наводнение;
- возможность ставить перед участниками обучения неограниченное количество задач, в том числе специальные (например, вызов пожарно-спасательной команды), за выполнение которых начисляются дополнительные баллы;
- возможность организации сотрудничества и выполнения групповых действий во время тренинга с неограниченным количеством участников;
- возможность добавлять столько случайных людей (ботов), сколько потребуется для поддержания реалистичных условий обучения;
- активные индикаторы состояния здоровья.

1.3 Постановка задачи

Интерактивный 3D-тренажёр «Эвакуация из здания» должен иметь следующий функционал:

- создание трёхмерной модели здания с учетом параметров его конструкции, этажности и расположения выходов, окон и проёмов, заданных пользователем;
- возможность управления персонажем в трехмерном пространстве;
- возможность осуществлять различные действия, например прыжки, приседания;
- реализовывать эффект возгорания в некоторых участках здания;
- поддержка запуска из браузера.

Минимальные системные требования:

- процессор: двухъядерный 2,5 ГГц или быстрее возможность управления персонажем в трехмерном пространстве;
- оперативная память: 4 ГБ;

- графическая карта: поддерживающая OpenGL 3.3 или более новую версию;
- операционная система: Windows 7/8/10, macOS 10.12+, Linux;
- браузер: Google Chrome, Mozilla Firefox, Safari, Microsoft Edge.

1.4 Выбор игрового движка

В настоящее время существует широкий спектр инструментов для создания 3D-приложений от простых библиотек до масштабных платформ с обилием функционала. Для разработки высококачественных 3D-приложений критически важно выбирать качественные инструменты. Использование готовых движков значительно упрощает этот процесс, предоставляя разработчикам возможность создавать приложения без излишних сложностей и затрат времени. Это ключевой аспект при создании увлекательных и инновационных 3D-приложений, которые будут востребованы среди пользователей.

1.4.1 Игровой движок «Unity»

Unity – это популярный игровой движок, используемый для создания игр на различных платформах, включая ПК, мобильные устройства, игровые консоли и виртуальную реальность. Основные особенности движка Unity:

- широкий выбор языков программирования. Unity позволяет использовать различные языки программирования, включая C#, JavaScript(UnityScript) и Boo. Это дает разработчикам возможность выбрать наиболее подходящий язык для их задач;
- поддержка VR и AR. Unity имеет широкую поддержку виртуальной и дополненной реальности, что позволяет разработчикам создавать игры и приложения для этих платформ;
- интеграция с другими инструментами.

Однако Unity может быть не идеальным выбором для некоторых 3D-приложений из-за сложности использования, потенциальных проблем с производительностью и размером сборок, а также из-за зависимости от компании

и стоимости лицензирования. Для некоторых проектов могут быть предпочтительны более легковесные и настраиваемые инструменты разработки [3].

1.4.2 Игровой движок «Unreal Engine»

Unreal Engine - популярный игровой движок, разработанный Epic Games, известный своей высококачественной графикой, надежным набором инструментов и универсальностью.

Unreal Engine обладает графикой довольно высокого качества. Этот движок известен своими расширенными графическими возможностями, включая реалистичное освещение, детализированные текстуры и передовые технологии затенения. Он часто используется для создания визуально потрясающих игр и симуляторов.

Ещё одним преимуществом Unreal Engine является большое сообщество и широкий набор инструментов. Огромное количество разработчиков которые создают инструменты, плагины и дополнения для улучшения движка. Это позволяет программистам быстро и легко находить решения для любой задачи;

Также движок обладает системой визуального программирования. Blueprints от Unreal позволяет разработчикам создавать логику 3D-приложений без написания кода, доступную художникам и дизайнерам. Этот визуальный скрипт может ускорить процесс разработки и обеспечить быстрое создание прототипов;

Unreal Engine содержит полный набор инструментов для анимации, физики, искусственного интеллекта и многого другого. Он также включает в себя качественный редактор для создания уровней, материалов и пользовательских интерфейсов.

Несмотря на систему blueprints, Unreal Engine отличается высокой скоростью обучения, особенно для начинающих. Количество функций и инструментов весьма огромно, и освоение их требует времени и усилий;

					ПГУ 1.090301.34.001 ПЗ	Лист
						15
Изм.	Лист	№ докум.	Подпись	Дата		

Также движок Unreal Engine может быть ресурсоемким и вызывать проблемы с производительностью, особенно на недорогих устройствах. Иногда бывает сложно оптимизировать приложение таким образом, чтобы оно безотказно работало на всех целевых платформах;

Для небольших игр или простых проектов Unreal Engine может оказаться излишним. В приложениях, которые не требуют улучшенной графики или дополнительных функций, сложность и мощность движка могут оказаться ненужными. Тем более, для успешного применения всех технологий требуется покупка лицензии за большие деньги;

Таким образом, Unreal Engine является неплохим выбором при разработке 3D-приложений, однако для обычного обучающего тренажёра стоимость, возможности и ресурсы движка неэкономичны.[4]

1.4.3 Игровой движок «Bevy»

Bevy – это относительно новый и быстро развивающийся игровой движок, созданный на Rust, который отличает простота, гибкость и новизна. Основные особенности движка Bevy:

- модульная архитектура. Bevy использует модульную архитектуру, которая позволяет разработчикам легко настраивать и расширять функциональность движка, добавляя новые компоненты и системы. Это делает его гибким и масштабируемым инструментом для создания различных типов приложений;

- параллельная обработка данных. Bevy предлагает параллельную обработку данных, что позволяет эффективно обрабатывать большие объемы данных, таких как 3D-сцены, анимации и физику. Это обеспечивает высокую производительность и позволяет создавать сложные и быстрые приложения.;

- простота в использовании. Bevy разработан с учетом простоты использования и ясности синтаксиса. Он предоставляет удобный API и инструменты для создания игр и приложений без излишних сложностей и затрат времени;

					ПГУ 1.090301.34.001 ПЗ	Лист
						16
Изм.	Лист	№ докум.	Подпись	Дата		

– язык программирования Rust. Bevy написан на языке программирования Rust, который известен своей безопасностью, производительностью и эффективным управлением памятью. Это обеспечивает высокую производительность и надежность при создании игр и приложений;

– активное сообщество. Несмотря на то, что Bevy-относительно молодой проект, здесь уже существует активное сообщество разработчиков, которые активно участвуют в разработке, создают ресурсы и обмениваются опытом [5].

1.5 Выбор языка программирования

Bevy использует язык программирования Rust, который обладает рядом преимуществ, делающих его идеальным выбором для разработки игр. Rust обеспечивает высокую безопасность памяти благодаря системе защиты и контроля. Уникальная концепция borrowchecker и строгая система типов предотвращают целый класс ошибок, таких как недопустимый доступ к памяти и конфликты данных.

Также Rust предоставляет прямой доступ к системным ресурсам, позволяя программистам полностью контролировать управление ресурсами. Это делает его очень эффективным и подходящим для создания высокопроизводительных приложений. Rust имеет встроенную поддержку параллельного и асинхронного программирования, что позволяет создавать эффективные и масштабируемые приложения, способные эффективно использовать многоядерные процессоры.

Язык обладает богатой библиотекой и хорошо развитой экосистемой инструментов, которые облегчают разработку различных типов приложений, включая веб-сервисы, мобильные приложения и системное программирование. Кроме того, синтаксис языка современный и понятный, что делает его доступным для широкого круга программистов.

Используя Rust, можно создавать менее подверженные ошибкам и уязвимостям безопасности программы за счет мощных функций управления памятью. Кроме того, Rust – проект с открытым исходным кодом, и активное

сообщество разработчиков поддерживает его и продолжает добавлять новую функциональность и удобные инструменты. Например, компилятор языка генерирует очень точные и информативные сообщения об ошибках, выделяет проблемный код и рекомендует способы его исправления.

Хотя Rust не так доступен, как другие языки программирования, в целом на нем легче программировать, потому что легче предотвратить нежелательное поведение. Поэтому Rust – отличный язык для продуктивного написания кода, не беспокоясь о проблемах с памятью или незамеченных ошибках.

Таким образом, в сочетании с хорошей поддержкой сообщества и исчерпывающей документацией Rust становится превосходным выбором для создания надежных, высокопроизводительных 3D-приложений [6].

1.6 Выводы

Bevy является важным и современным инструментом для разработки 3D-приложений и выделяется своей модульной архитектурой, параллельной обработкой данных и использованием языка программирования Rust. Эти функции являются привлекательным вариантом для разработчиков, стремящихся создавать высокопроизводительные и надежные приложения.

Модульная архитектура позволяет разработчикам легко настраивать и расширять функциональность движка, обеспечивая гибкость и масштабируемость для различных типов проектов. Параллельная обработка данных обеспечивает эффективную работу с большими объемами данных. Это особенно важно для 3D-приложений со сложной графикой и физикой.

Использование языка программирования Rust обеспечивает высокую производительность, безопасность и эффективное управление памятью, что делает Bevy идеальным выбором для разработки современных и инновационных приложений.

Кроме того, активное сообщество разработчиков и открытый исходный код способствуют росту и развитию Bevy, обеспечивая поддержку и содействие разработчикам на всех этапах создания проекта.

Таким образом, Bevy - мощное, гибкое и современное средство 3D-разработки, которое может стать лучшим выбором для тех, кто хочет создавать высококачественные и производительные приложения.

1.7 Выбор инструментов разработки

В качестве среды разработки было решено выбрать RustRover – уникальная интегрированная среда разработки для разработки на Rust. Она разрабатывается компанией JetBrains на замену бесплатного плагина поддержки Rust для IntelliJ IDEA и CLion.

RustRover позволяет вам быстрее писать свой код, благодаря автозавершению названия всего кода. Среда IDE предлагает заменить отсутствующие поля, методы импорта и атрибутов, а также помогает создать типичный дизайн с использованием шаблонов.

RustRover использует встроенную проверку для анализа кода на лету и предоставления быстрых исправлений для автоматического устранения проблем. Среда IDE предоставляет множество возможностей для рефакторинга, которые применяются ко всей базе кода.

WebAssembly (Wasm) – формат двоичных команд для виртуальных машин на основе стека. Он предназначен для запуска кода практически с обычной скоростью в веб-браузере. WebAssembly позволяет разработчикам писать код на различных языках программирования (например, Rust, C, C++) и запускать его в веб-браузере.

Wasm код выполняется значительно быстрее, чем JavaScript, что позволяет создавать более производительные веб-приложения, игры и другие интерактивные элементы. Также он открывает доступ к новым возможностям,

недоступным в JavaScript, таким как работа с низкоуровневым аппаратным обеспечением и оптимизация производительности на уровне машинного кода.

Wasm код может быть написан на различных языках программирования, что позволяет разработчикам использовать свои любимые языки и инструменты для создания веб-приложений. Модули wasm можно легко портировать между различными веб-браузерами и платформами, что упрощает разработку кроссплатформенных приложений.

Trunk – инструмент для создания и упаковки веб-приложений на Rust с популярным инструментом сборки- WebAssembly (Wasm). Trunk используется для разработки приложений Rust в браузере и для выполнения таких задач, как создание проектов, управление зависимостями, копирование ресурсов и запуск серверов разработки.

Trunk компилирует Rust код в высокопроизводительный Wasm модуль, который может быть интерпретирован веб-браузером. Затем, генерируется HTML-страница, которая служит основой веб-приложения. Созданный Java-Script код, загружает Wasm модуль и обеспечивает взаимодействие между ним и HTML-страницей.

Trunk идеально подходит для создания высокопроизводительных веб-приложений с 3D-графикой и физикой. Он имеет простой и понятный интерфейс командной строки, что делает его доступным для разработчиков с любым уровнем опыта.

1.8 Архитектура приложения

В этом разделе будет рассмотрена архитектура программного модуля, для интерактивного 3D-тренажёра «Эвакуация из здания». Ключевой особенностью архитектуры приложений с использованием Bevy является использование паттерна ECS (Entity Component System). Эта парадигма проектирования разделяет приложение на 3 части: Entities, Components и Systems. Entities –

					ПГУ 1.090301.34.001 ПЗ	Лист
						20
Изм.	Лист	№ докум.	Подпись	Дата		

уникальные «вещи», которым назначены группы Components, которые затем обрабатываются с помощью Systems.

Шаблон ECS поощряет чистые, несвязанные проекты, заставляя разбивать данные и логику приложения на его основные компоненты. Это также помогает ускорить ваш код за счет оптимизации шаблонов доступа к памяти и упрощения параллелизма. Структуру паттерна можно увидеть на рисунке 4.

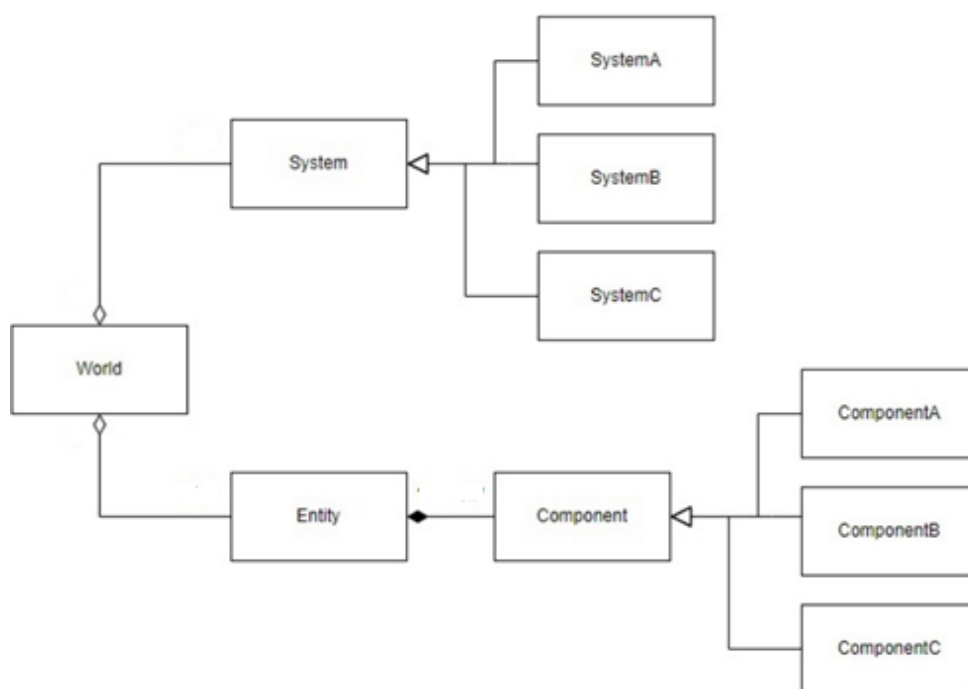


Рисунок 4-Структура паттерна ECS

Согласно приведенной структуре, ECS-мир содержит в себе коллекции систем и сущностей. Сущности содержат изменяемый набор компонентов. Доступ из систем к набору сущностей, содержащих определенный набор компонентов, предоставляется публичными методами ECS-мира. Специфика вызова этих методов зависит от реализации в конкретном ECS-фреймворке, но в большинстве из них получение доступа к определенному набору сущностей схоже с формированием запроса к базе данных.

В Entity-Component-System данные строго отделены от логики. Поведение объекта определяется не интерфейсами, как в классическом объектно-

ориентированном программировании, а присвоенными объекту свойствами с данными и существующей отдельно логикой обработки этих данных. Главное свойство ECS, которое выделяет его на фоне других подходов к разработке – всё есть данные. Данными выражены и свойства объекта, и его характеристики, и события. Такой подход поможет соблюдать принцип Composition Over Inheritance.

Логика реализуется конвейерной обработкой данных. В каждой итерации игрового цикла системы последовательно получают доступ к определенным компонентам сущностей и изменяют данные этих компонентов. Таким образом, порядок выполнения систем играет большую роль. На рисунке 5 представлена конвейерная схема работы систем.

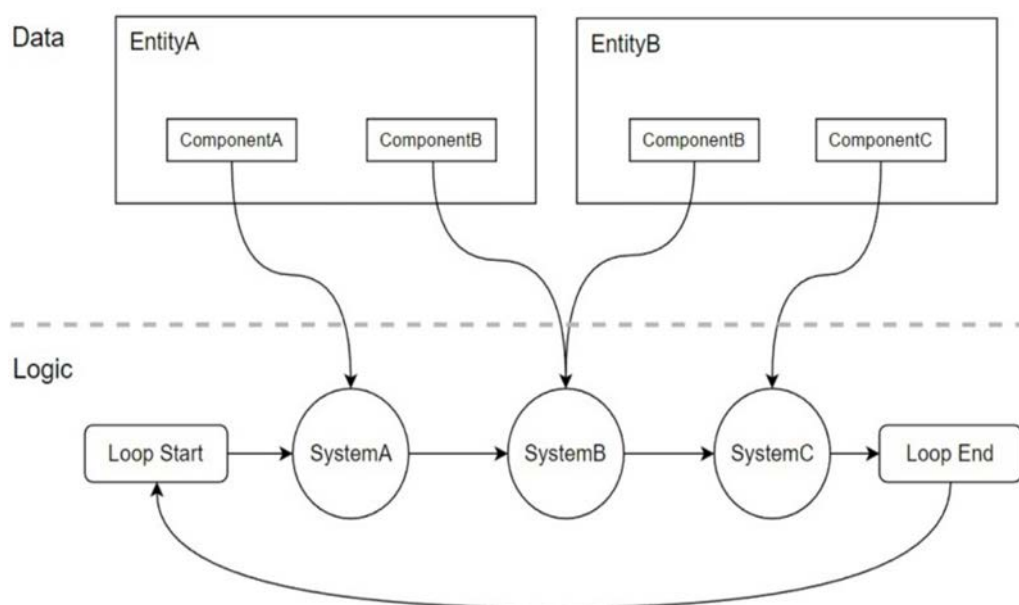


Рисунок 5- Конвейерная схема работы систем

1.9 Разработка приложения

Одним из основных принципов Bevu является модульность. Все функции движка Bevu реализованы в виде плагинов — коллекций кода, которые изменяют файл App-контейнер логики и данных приложения. Схема проекта представлена на рисунке 6.

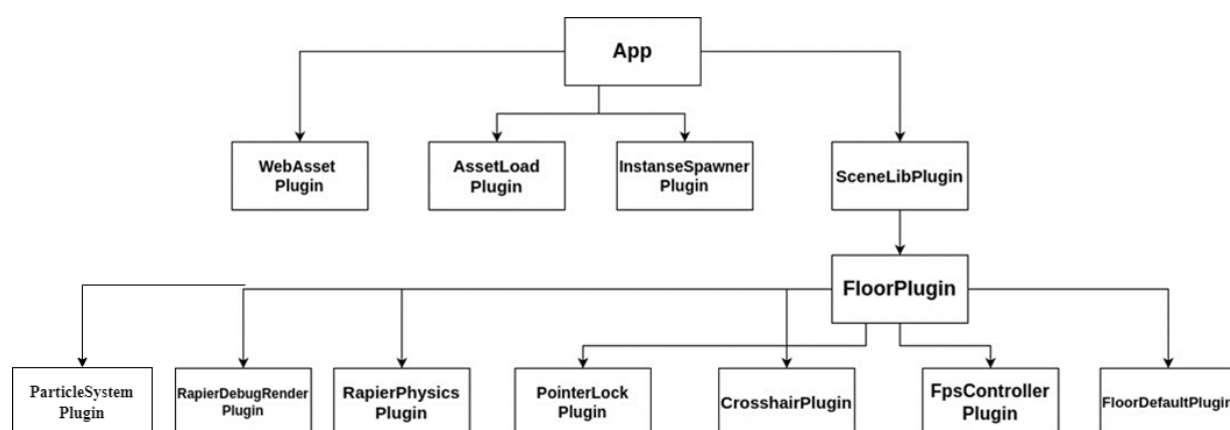


Рисунок 6-Схема плагинов

1.9.1 Плагин AssetsLoadPlugin

AssetsLoadPlugin нужен для загрузки ресурсов-данных, с которыми работает игровой движок: изображения, 3D-модели, звуки, сцены и многое другое. В данном приложении необходимо использовать GLB модели окон и дверей, а также текстуры в формате PNG для тех же самых моделей и пола. Плагин был реализован с возможностью выбора типа загрузки: параллельной или последовательной, в зависимости от ваших целей приложения. В данном плагине определены следующие структуры данных:

- AssetType определяет типы ресурсов (Mesh или TextureMap);
- AssetsLoadPlugin является и структурой, и плагином. Он реализует трейт Plugin, что позволяет его использовать как плагин в Bevu-приложении. В то же время, он также является структурой, содержащей поле asset_load_algorithm, которое определяет алгоритм загрузки активов. Поле представляет собой тип данных Enum, в зависимости от значения которого и происходит выбор

дальнейшей загрузки;

- AssetInQueue содержит информацию о ресурсе, который нужно загрузить;
- AssetDescriptor хранит информацию о загруженном ресурсе;
- AssetsLoadedRes ресурс, содержащий список загруженных ресурсов;
- AssetsLoadQueue ресурс, содержащий список ресурсов, ожидающих загрузки;

– AssetsLoadedEvent: событие, срабатывающее при завершении загрузки всех ресурсов. Оно нужно, чтобы InstanceSpawnerPlugin не начинал свою работу до загрузки ресурсов.

В зависимости от типа загрузки будут выполняться определённые системы.

При последовательном варианте:

- setup_sequential(): подготовка очереди для загрузки;
- sequential_load(): загрузка ресурсом последовательно.

При параллельном варианте:

– setup_parallel (): подготовка и загрузка данных параллельно. В данной системе вызывается функция load_map(), которая загружает текстурные карты для материалов;

– check_parallel_assetss_ready(): проверка состояния загруженности всех ресурсов в случае параллельной загрузки. В данной системе вызывается функция is_all_ready(), которая проверяется все ли ресурсы загружены и в случае чего, отправляет событие AssetssLoadedEvent, также функция выводит состояния загружаемых ресурсов.

Схема плагина представлена на рисунке 7.

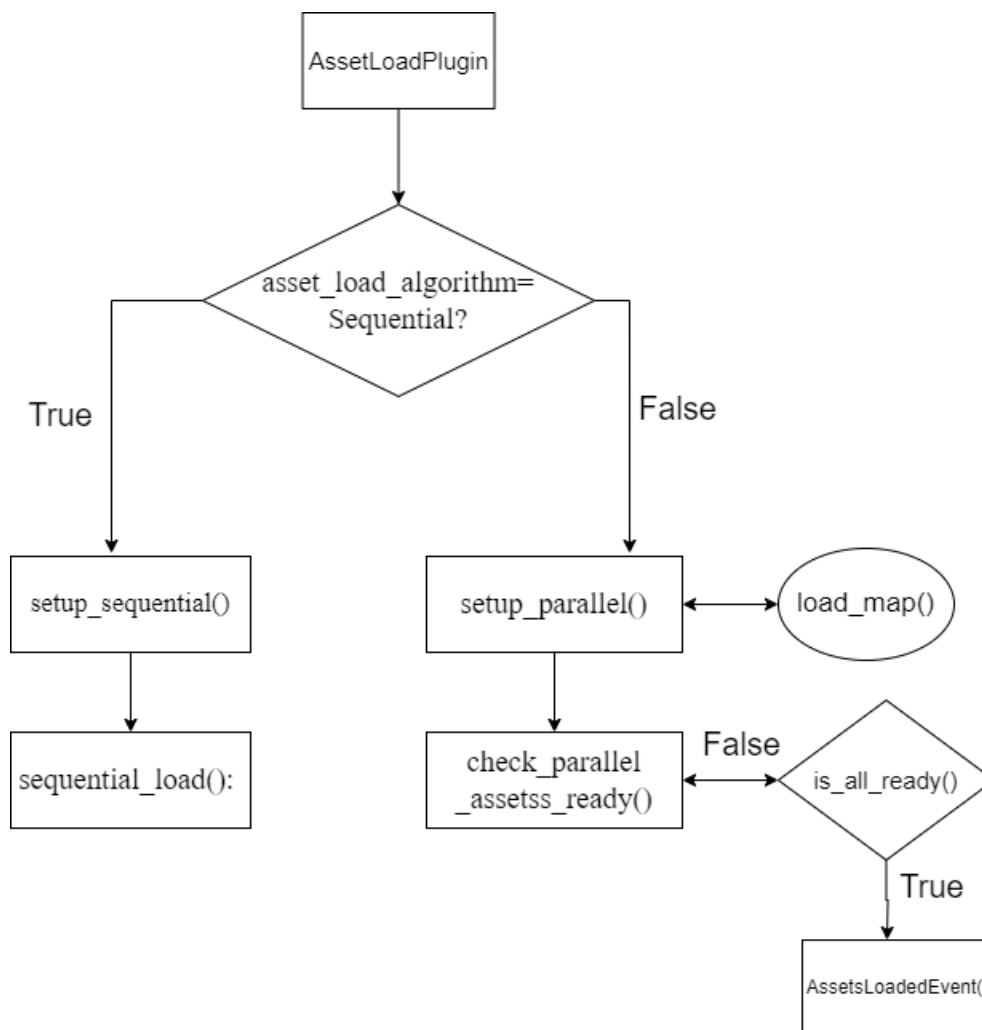


Рисунок 7-Схема AssetLoadPlugin

1.9.2 Плагин InstanceSpawnerPlugin

Данный плагин инициализирует сцены, создаёт и располагает объекты, а также обрабатывает события для них. Он содержит ресурс IsSceneStarted, который принимает параметр типа bool и если он равен true, то ресурс предотвращает повторную инициализацию сцены, если она уже была запущена. Кроме того, система стартует по приходу события AssetsLoadEvent() из плагина AssetsLoadedPlugin. Оно уведомляет о том, что необходимые ресурсы загружены. В плагине выполняются две системы: on_assets_loaded() и spawn_window(). В системе on_assets_loaded() вызываются следующие функции:

- update_meshes_uv(): обновляет UV-координат мешей в соответствии с данными из JSON;

- spawn_instances(): порождает объекты в системе на основе данных;
- apply_basis(): вызывается в функции spawn_instances() и применяет базис координат к позиции объекта;
- build_material(): создаёт материал объекта на основе данных из JSON;
- actualize_mesh_handle(): вызывается в функциях update_meshes_uv() и spawn_instances(), обновляет обработанный меш в соответствии с данными;
- actualize_map_handle(): вызывается в функции build_material(), обновляет обработанные текстуры в соответствии с данными;

Функция actualize_mesh_handle() также вызывается в системе spawn_window(). Эта система принимает событие SpawnEventRes из плагина FloorPlugin и создаёт объекты окон с координатами и размерами, соответствующим проёмам для окон. Схема плагина представлена на рисунке 8.

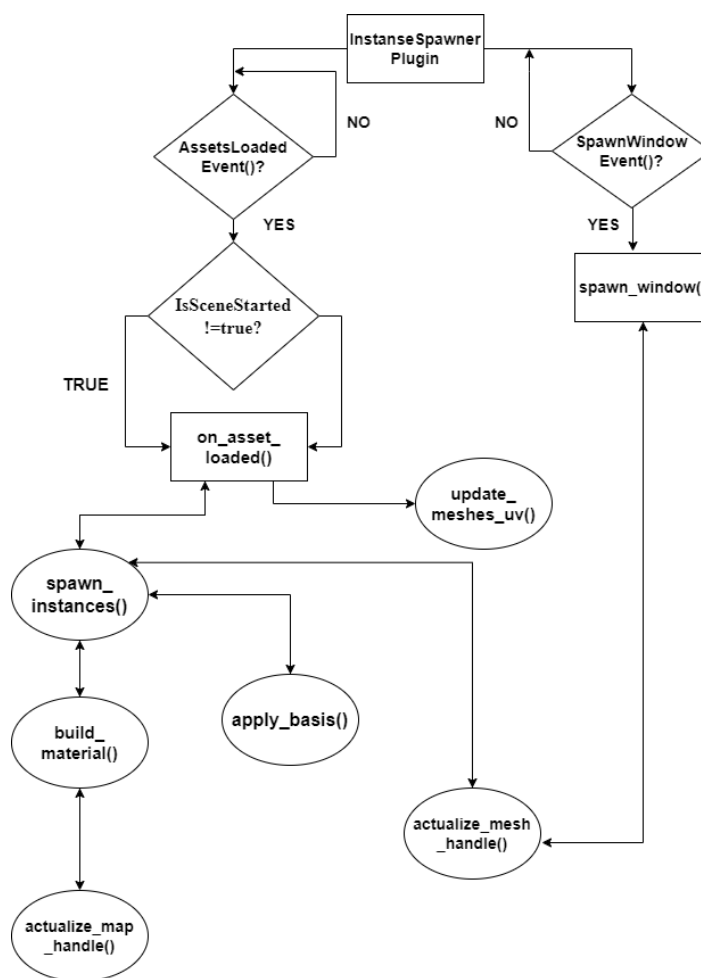


Рисунок 8-Схема InstanceSpawnerPlugin

1.9.3 Плагин FloorPlugin

Этот плагин представляет собой инструмент для создания и управления зданиями. Он используется для создания планировки этажей, размещения стен и окон. В данном плагине определены следующие структуры данных:

- FloorSenderRes: структура для отправки сообщений о зданиях с этажами через канал;
- FloorReceiverRes: структура для приёма сообщений о зданиях с этажами через канал;
- FloorState: содержит информацию о текущем состоянии этажа;
- SpawnWindowEvent: событие для создания окон на стенах;
- Opening: структура, представляющая проём в стене;
- CurrentBox: координаты прямоугольника для создания стен и пола.

Данный плагин содержит следующие системы:

- spawn_building(): создание здания и его этажей на сцене;
- load_building_and_floors(): загрузка информации о здании и его этажах.

Система load_building_and_floors() отвечает за загрузку данных о зданиях и его этажах, а также отправку этих данных в другую часть программы для последующего использования. Эта функция сначала проверяет, есть ли дополнительный параметр в данных инициализации сцены и если параметр существует, он используется в качестве идентификатора здания. С помощью wasm_bindgen_futures::spawn_local() запускается асинхронная задача. Затем, выполняется асинхронный запрос для получения данных здания и его этажей с использованием идентификатора. Запрос осуществляется методом BuildingStore::get_with_floors_by_id(id). Если запрос успешен, то полученные данные отправляются через копию отправителя канала local_sender.send(). Это позволяет другой части программы получить эти данные и использовать их для построения зданий. Код данной функции:

```
fn load_building_and_floors(sender: Res<FloorSenderRes>,  
scene_data: Res<SceneInitDataRes>) {  
    let id_str = if scene_data.extra_param.is_some() {
```

```

        scene_data.extra_param.clone().unwrap()
    } else {
        "408e43c0-8a3d-11eb-a13f-3b448741dae2".to_string()
//default
    };

    let local_sender = sender.0.lock().unwrap().clone();
    wasm_bindgen_futures::spawn_local(async move {
        let id = Uuid::parse_str(&id_str).unwrap();
        let resp =
BuildingStore::get_with_floors_by_id(id).await;
        if resp.clone().is_err() {
            info!(
                "Cannot fetch periodic json {:?} ",
                resp.clone().err().unwrap()
            );
        } else {
            let resp = resp.clone().unwrap();
            local_sender.send(resp).unwrap();
        }
    })
}

```

В системе `spawn_building()` вызывается функция `spawn_floor()`, которая отвечает за построение конкретного этажа. В ней с помощью функции `spawn_wall_segment()` происходит преобразование координат и создаётся участок стены. В качестве меша передаётся возвращаемое значение из функции `current_box_func()`. Она генерирует сегмент в форме параллелепипед. Перед вызовом функции `spawn_floor()` вызывается функция `actualize_map_handle()` для поиска и обновления параметров текстур. Эта функция была определена в плагине `InstanseSpawnerPlugin`. Схема данного плагина представлена на рисунке 9.

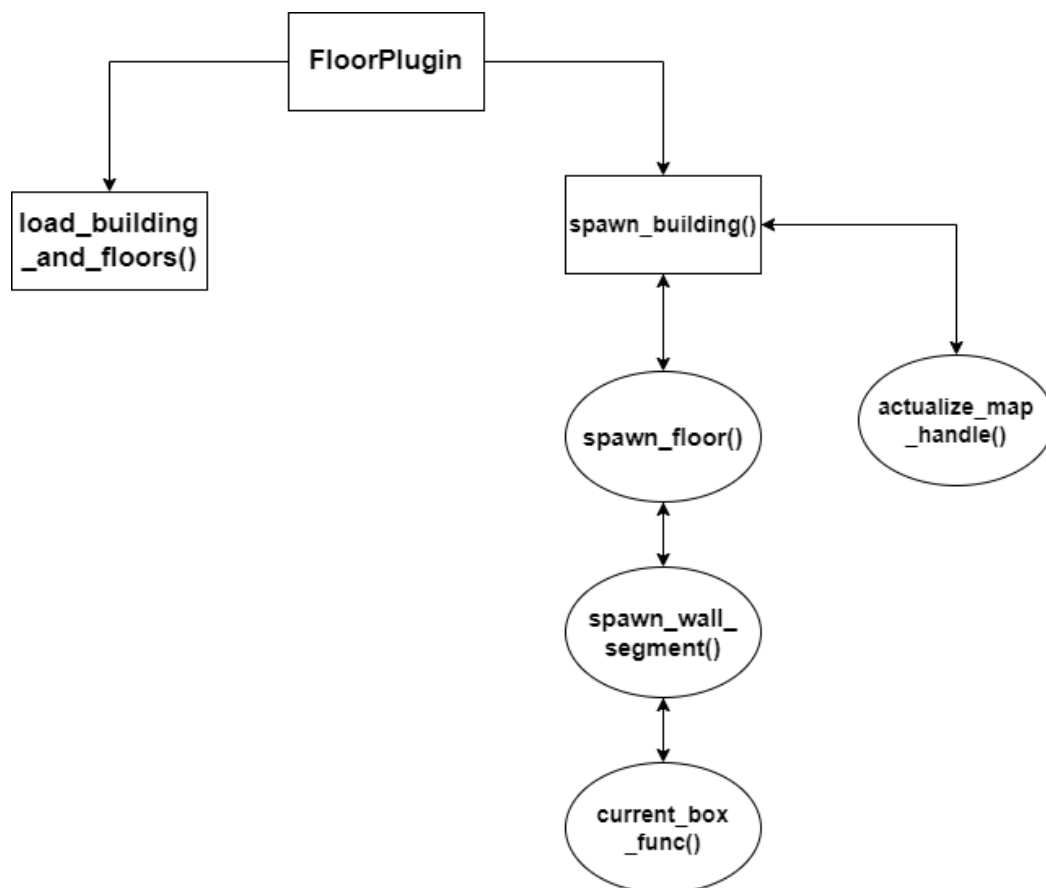


Рисунок 9-Схема FloorPlugin

1.9.4 Плагин FloorDefaultPlugin

Этот плагин обеспечивает базовую структуру для сцены, добавляя в неё освещение и поверхность по умолчанию. Он предназначен для использования в качестве отправной точки при создании игровой среды, позволяя быстро создать освещенную сцену с поверхностью. Также здесь происходит создание игрового персонажа и настройка его параметров с помощью структуры из плагина FpsControllerPlugin. Основные поля структуры:

- Collider: тип коллайдера, который определяет форму объекта;
- Friction: коэффициент трения для объекта;
- Restitution: коэффициент упругости (отскока) для объекта;
- ActiveEvents: перечисление, которое определяет, какие события физического мира должны быть активны для данного объекта;
- Velocity: вектор скорости объекта;

- Sleeping: управление возможностью спящего режима для объекта;
- AdditionalMassProperties: дополнительные свойства массы объекта;
- GravityScale: воздействие гравитации на объект;
- Ccd: включает/отключает у объекта непрерывное обнаружение столкновений ;
- TransformBundle: положение объекта;
- LockedAxes: определение осей, которые заблокированы для вращения объекта;
- Rigidbody: тип тела объекта в физическом мире.

Код создания структуры:

```

let logical_entity = commands
    .spawn((
        Collider::capsule(Vec3::Y * 0.5, Vec3::Y * 1.5,
0.5),
        Friction {
            coefficient: 0.0,
            combine_rule: CoefficientCombineRule::Min,
        },
        Restitution {
            coefficient: 0.0,
            combine_rule: CoefficientCombineRule::Min,
        },
        ActiveEvents::COLLISION_EVENTS,
        Velocity::zero(),
        Rigidbody::Dynamic,
        Sleeping::disabled(),
        LockedAxes::ROTATION_LOCKED,
        AdditionalMassProperties::Mass(1.0),
        GravityScale(0.0),
        Ccd { enabled: true }, // Prevent clipping when
going fast
        TransformBundle::from_transform(Transform::from_xyz(60.0, 130.0,
60.0)),
        LogicalPlayer,
        FpsControllerInput {
            pitch: -TAU / 12.0,
            yaw: TAU * 5.0 / 8.0,
        },
        ..default()
    ),
    FpsController {
        key_crouch: KeyCode::KeyC,
    }

```

```

        ..default()
    },
))
.insert(CameraConfig {
    height_offset: 0.0,
    radius_scale: 0.75,
})
.id();

```

Также в данном плагине происходит создание и настройки системы частиц для генерации прототипа огня.

1.9.5 Плагин FullViewportPlugin

Этот плагин обеспечивает функциональность изменения размера окна приложения таким образом, чтобы область просмотра занимала всю доступную область окна браузера. Этот плагин содержит 2 системы: `setup_viewport_resize_system()` и `viewport_resize_system()`. Первая система настраивает слушателя событий изменения размера окна браузера. Сначала извлекается глобальный объект `window` из веб-браузера с помощью `web_sys::window()`. Затем, извлекается отправщик событий изменения размера окна из ресурса `EventSenderRes`. Мьютекс блокируется, чтобы предотвратить гонку данных при доступе к отправителю из разных потоков. Затем отправитель клонируется для использования внутри замыкания, после чего через него же, отправляется сообщение через изменения размера окна для инициализации размера окна приложения. Отправляется пустое сообщение, так как нам не нужны конкретные данные, только сигнал о том, что размер окна изменился. Далее создаётся слушатель событий изменения размера окна с использованием библиотеки `gloo`. Этот слушатель привязывается к глобальному объекту `window` и слушает событие `resize`. Когда происходит событие изменения размера окна, вызывается замыкание, которое снова отправляет сообщение через отправителя событий изменения размера окна Код данной функции:

```

fn setup_viewport_resize_system(resize_sender:
Res<EventSenderRes>) {
    let web_window = web_sys::window().expect("could not get
window");

```

					ПГУ 1.090301.34.001 ПЗ	Лист
						31
Изм.	Лист	№ докум.	Подпись	Дата		

```

        let local_sender = resize_sender.0.lock().unwrap().clone();
        local_sender.send(()).unwrap();

        gloo_events::EventListener::new(&web_window, "resize", move
|_event| {
            local_sender.send(()).unwrap();
        })
        .forget();
    }

```

Система `viewport_resize_system()` выполняется каждый кадр и отвечает за изменение размеров окна при изменении размеров окна браузера. Сначала мы пытаемся получить сообщение о изменении размера окна из приемника событий. Если сообщение успешно получено, это означает, что размер окна браузера изменился, и мы можем продолжить обработку. Далее мы пытаемся получить доступ к компоненту `Window`, ассоциированному с основным окном `Bevy`. Затем, мы получаем новые размеры окна браузера с помощью функции `get_viewport_size()`, которая возвращает ширину и высоту окна и устанавливаем новое разрешение основного окна `Bevy`, чтобы обеспечить соответствие размера области просмотра размерам окна браузера. Код данной функции:

```

fn viewport_resize_system(
    mut window_query: Query<&mut Window, With<PrimaryWindow>>,
    resize_receiver: Res<EventReceiverRes>,
) {
    if resize_receiver.0.lock().unwrap().try_recv().is_ok() {
        let mut primary_w = window_query.get_single_mut();
        if primary_w.is_err() {
            return;
        }
        let size = get_viewport_size();
        // info!("Trying set resolution {},{}",size.0,size.1);
        primary_w.as_mut().unwrap().resolution =
WindowResolution::new(size.0, size.1);
    }
}

```

Структура данного плагина изображена на рисунке 10.

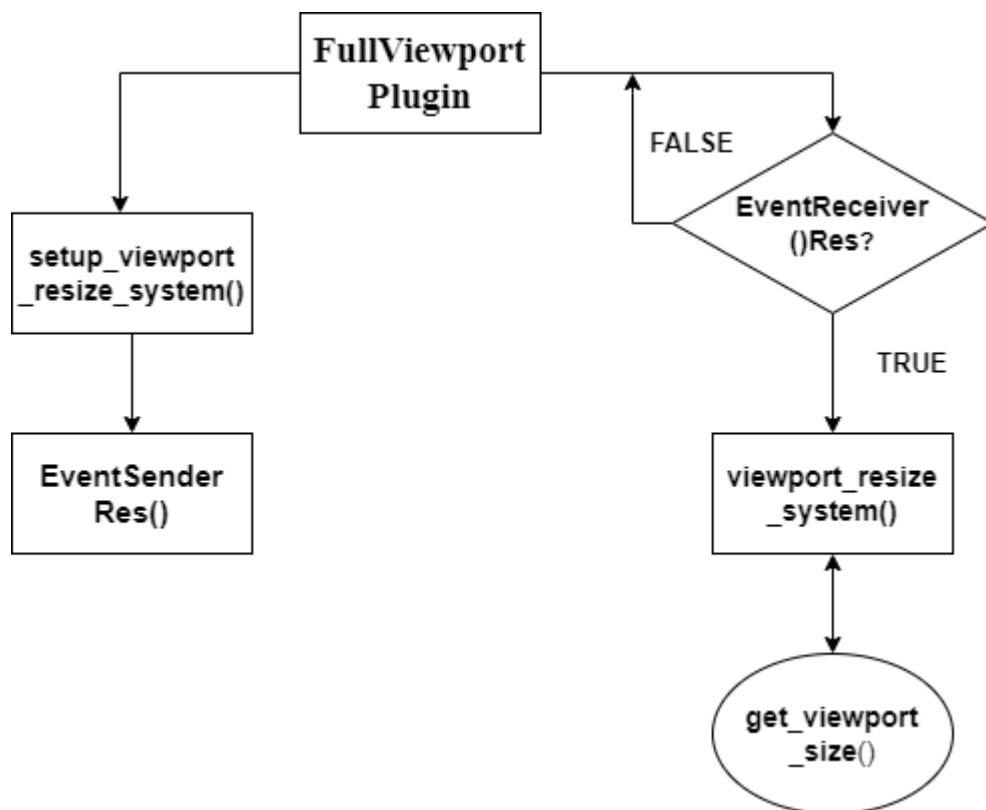


Рисунок 10-Схема FullViewPortPlugin

1.9.6 Плагин CrosshairPlugin

Этот плагин предназначен для добавления прицела в центр экрана веб-приложения. При запуске плагина создаётся начальный прицел в центре экрана. За это отвечает `crosshair_startup_system()`. Затем, при изменении размера окна браузера отправляется сообщение в канал, которое затем обрабатывается и вызывает обновление прицела, чтобы он оставался в центре экрана независимо от изменений размеров окна. Всё это происходит в системе `resize_listener()`, которая вызывает функцию `update_crosshair()`.

Структура плагина приведена на рисунке 11.

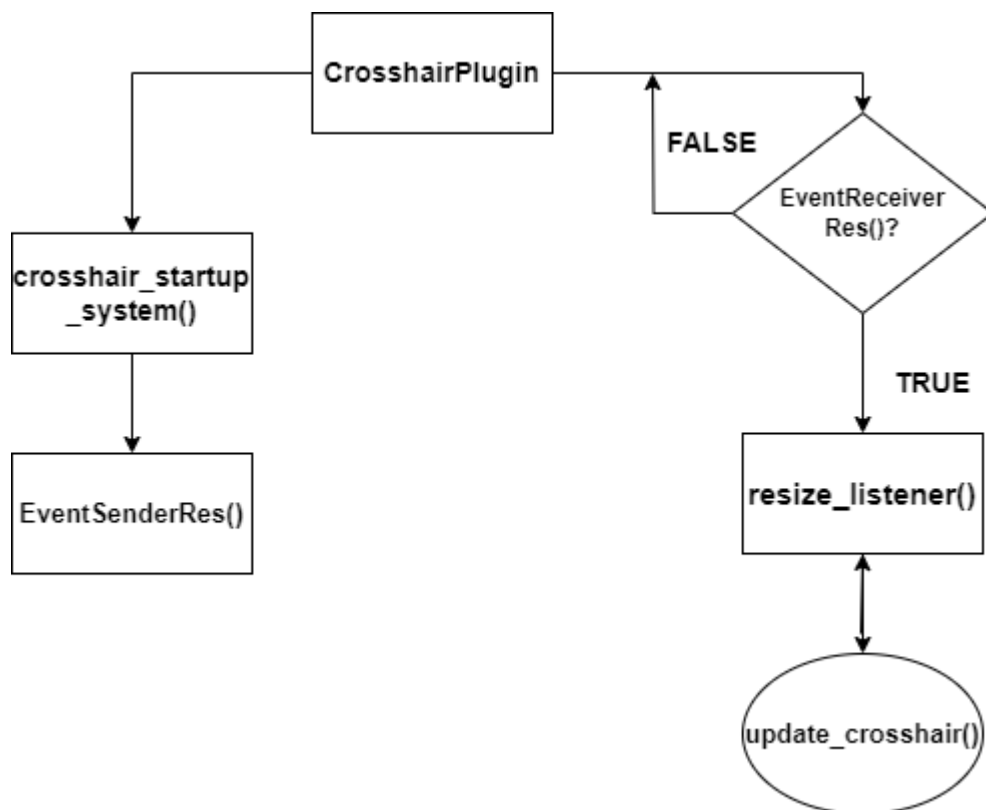


Рисунок 11-Схема CrosshairPlugin

1.9.7 Внешние плагины.

В проекте также используют плагины, импортирующиеся из библиотек в cargo.toml. Среди них:

- FPSConrollerPlugin: плагин, вдохновленный движением движка Source, реализует движение, подходящее для игр от 1-ого лица. Он обеспечивает управление персонажам в сцене, его движение, прыжки, приседания, взаимодействие с окружающими объектами;

- RapierDebugRenderingPlugin: плагин отвечает за рендеринг того, что Rapier «видит» при выполнении физического моделирования. Это нужно для проверки правильности согласования между коллайдерами и собственными визуальными ресурсами;

- RapierPhysicsPlugin: плагин, отвечающий за физику объектов;

- ParticleSystemPlugin: плагин, реализующий эффект возгорания;

1.9.8 Запуск приложения

За запуск приложения в браузере отвечают два главных файла main.rs и lib.rs. В файле main.rs объявлена функция `pub async fn start_scene()` с атрибутом `#[wasm_bindgen(start)]`. Атрибут указывает, что эта функция будет вызвана автоматически при загрузке WebAssembly модуля. Внутри функции инициализируются переменные `scene_id` для хранения идентификатора конкретной сцены и `extra` для получения информации из БД по конкретному зданию. Затем, происходит получение объекта окна браузера и текущей URL страницы. URL разбирается с использованием библиотеки `url` и извлечённые параметры сохраняются в переменные `scene_id` и `extra`. Затем происходит вызов асинхронной функции `enginelib::start()` с переданными параметрами. Код функции `start_scene()`:

```
#[wasm_bindgen(start)]
pub async fn start_scene() {
    let mut scene_id = "".to_string();
    let mut extra = "".to_string();

    //parse url params
    let window = web_sys::window();
    if window.is_none() {
        console_log!("Error handling web_sys::Window");
        return;
    }
    let url_str = window.unwrap().location().href();
    if url_str.is_err() {
        console_log!("Error handling
web_sys::Location.href()");
        return;
    }
    let url = Url::parse(url_str.unwrap().as_str());
    if url.is_err() {
        console_log!("Error parsing URL");
        return;
    }
    let url2 = url.unwrap();
    let params = url2.query_pairs();
    for (key, value) in params.into_iter() {
        if key.as_ref() == "id" {
            scene_id = value.as_ref().to_string();
        }
    }
}
```

```

    if key.as_ref() == "extra" {
        extra = value.as_ref().to_string();
    }
}

match engine::start(scene_id.clone(), extra).await {
    Ok(_) => {
        console_log!("Scene started with id {:?}",
scene_id)
    }
    Err(_err) => {
        // Обработка ошибки
        console_log!("Scene failed with id {:?}", scene_id)
    }
}
}

```

Файл lib.rs содержит основную функцию start(), которая запускает приложение и загружает сцену с сервера по идентификатору scene_id. В этой функции регистрируются и добавляются основные плагины и компоненты и происходит запуск приложения с помощью метода run(). Код данной функции:

```

pub async fn start(scene_id: String, extra: String) ->
Result<SceneResponse, String> {
    console_log!("id from URL param: {}",
scene_id.to_string().as_str());
    //get from the server
    let id = Uuid::parse_str(&scene_id).unwrap();
    let resp = SceneStore::get_by_id(id).await;

    //read a local file
    // let resp =
fetch_by_local_path:<SceneResponse>("floor.json".to_string()).await
;

    if resp.clone().is_err() {
        console_log!("{:?}", resp.clone().err().unwrap());
        return resp;
    }

    console_log!("res {:?}", resp.clone().unwrap());
    let scene_data = resp.clone().unwrap().data.unwrap();

    let mut app = App::new();

    app.register_type:<GameObject>()
        .insert_resource(AssetMetaCheck::Never)

```

```

        .add_plugins((WebAssetPlugin::default(),
DefaultPlugins))
        .add_plugins(AssetsLoadPlugin {
            asset_load_algorithm:
scene_data.clone().assets_loading_algorithm,
        })

        .add_plugins(SceneLibPlugin {
            scene_lib_name: scene_data.clone().scene_lib_name,
        })
        .add_plugins(InstanceSpawnerPlugin)
        .insert_resource(SceneInitDataRes {
            scene_response: resp.clone().unwrap(),
            extra_param: if extra != "" { Some(extra) } else {
None },
        })
    })
    .run();
return resp;
}

```

1.10 Тестирование работы приложения

С помощью программы Dbeaver в базу данных были введены произвольные значения координат стен, окон и дверей для каждого из этажей. В данном случае для примера использовались 2 этажа. Введённые данные представлены на рисунках 12,13.

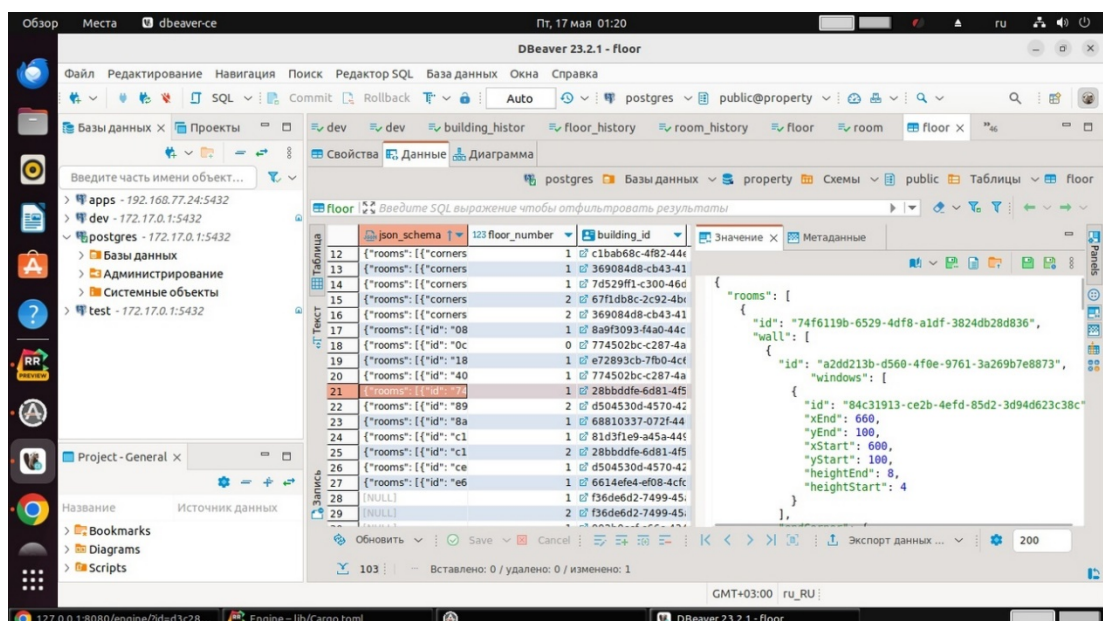


Рисунок 12-данные о первом этаже в Dbeaver

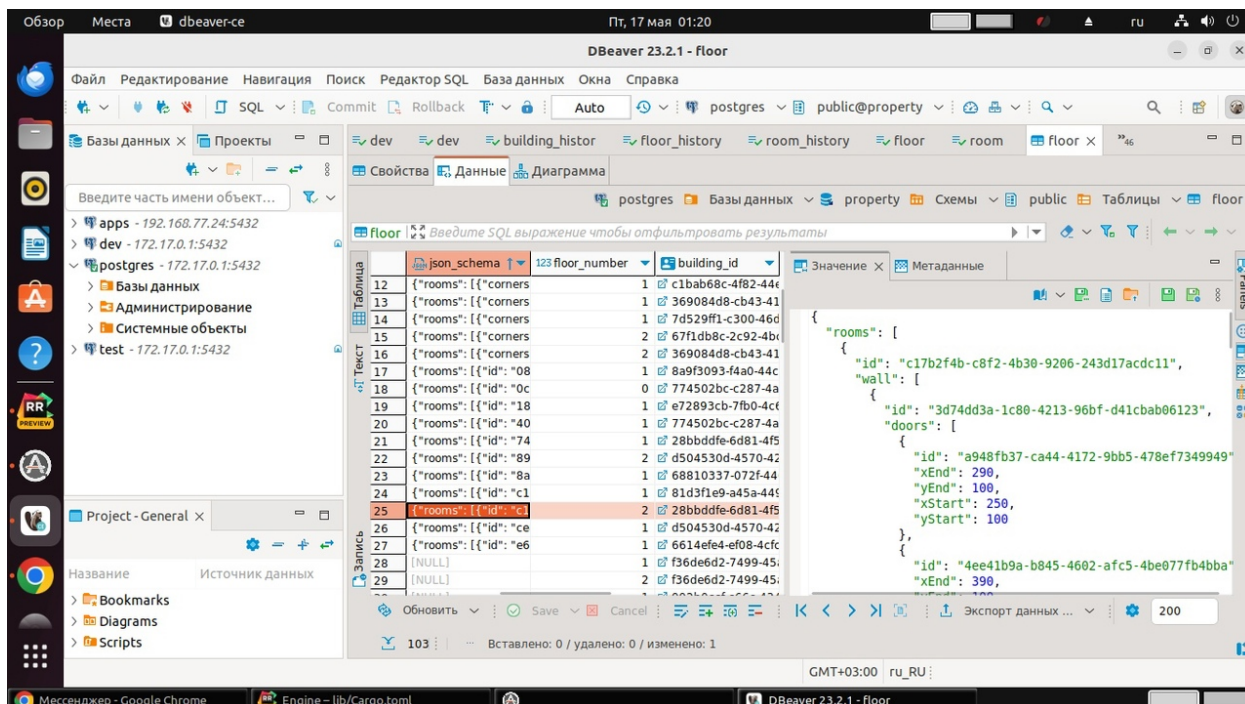


Рисунок 13-данные о втором этаже в Dbeaver

После запуска приложения в консоль были выведены промежуточные значения параметров, необходимых для генераций объектов. Вывод в консоль представлен на рисунке 14.

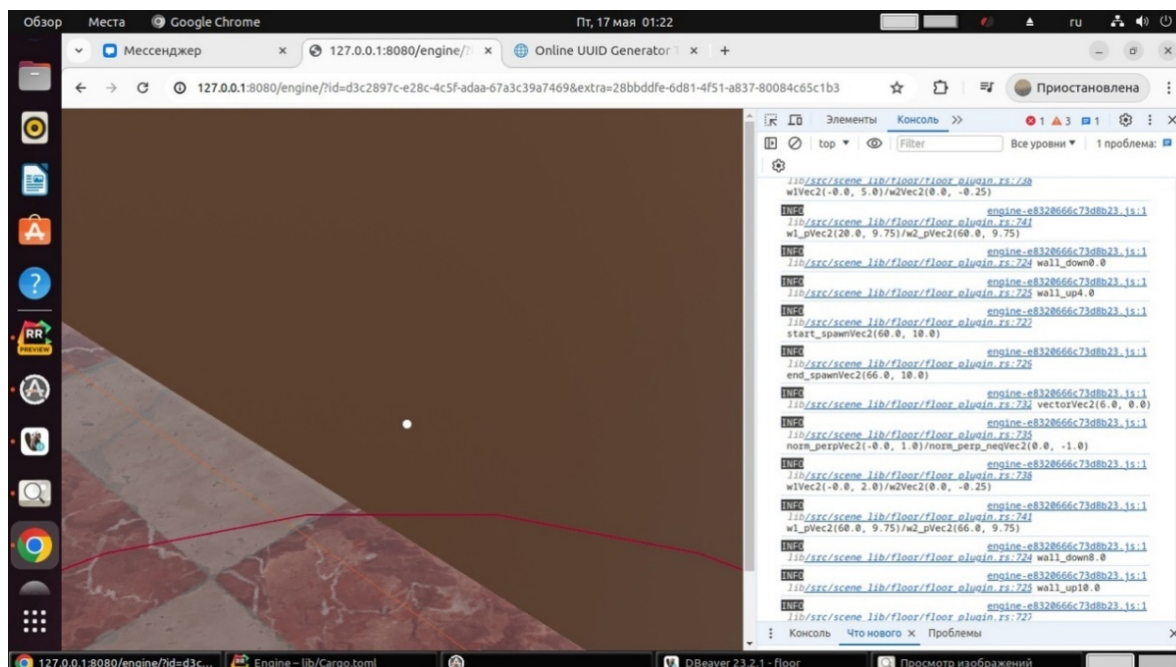


Рисунок 14-Консоль

Значения полностью совпали с полученным результатом. Был произведён «внутриигровой» обход здания для удостоверения правильности введённых данных по второму этажу. Демонстрация полученного результата представлена на рисунках 15 – 18.

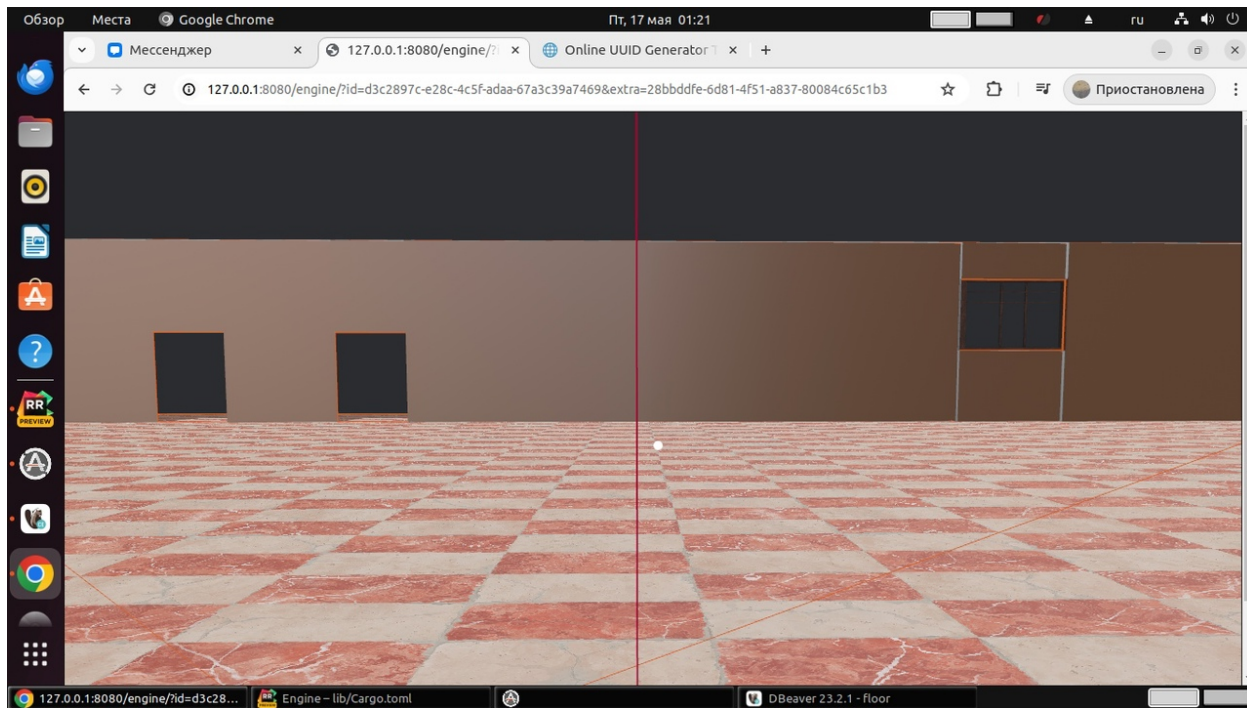


Рисунок 15-Второй этаж, 3D-симуляция.

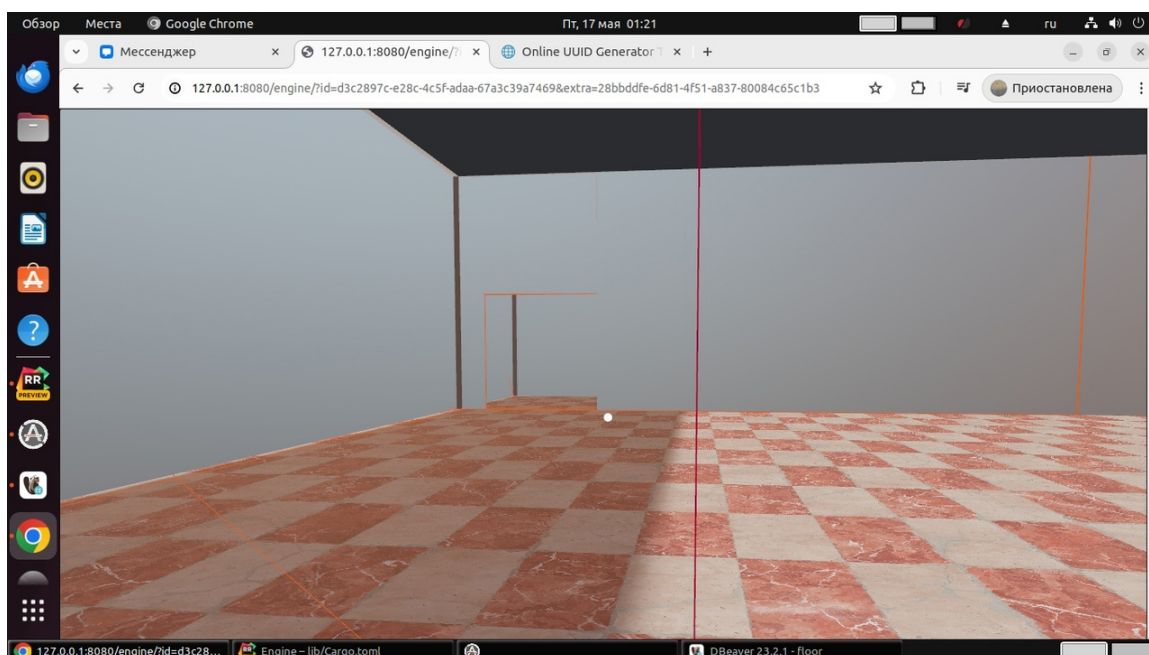


Рисунок 16-Второй этаж, 3D-симуляция.

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.34.001 ПЗ

Лист

39

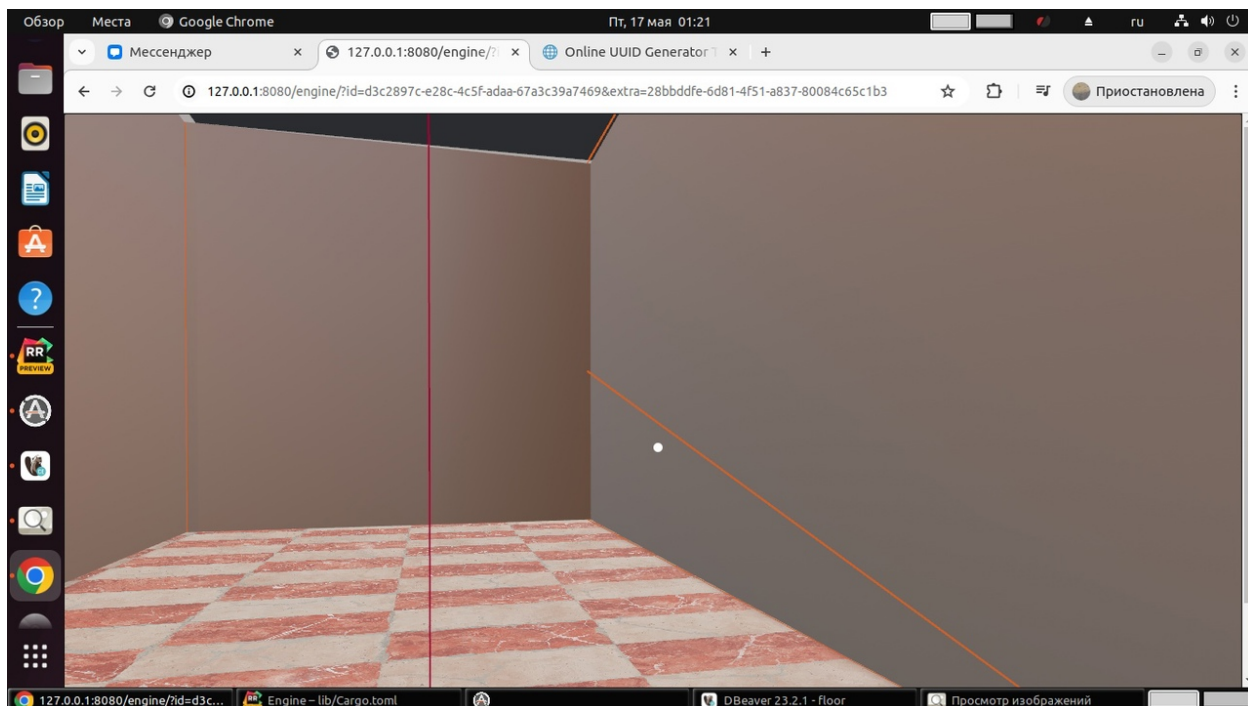


Рисунок 17-Второй этаж, 3D-симуляция.

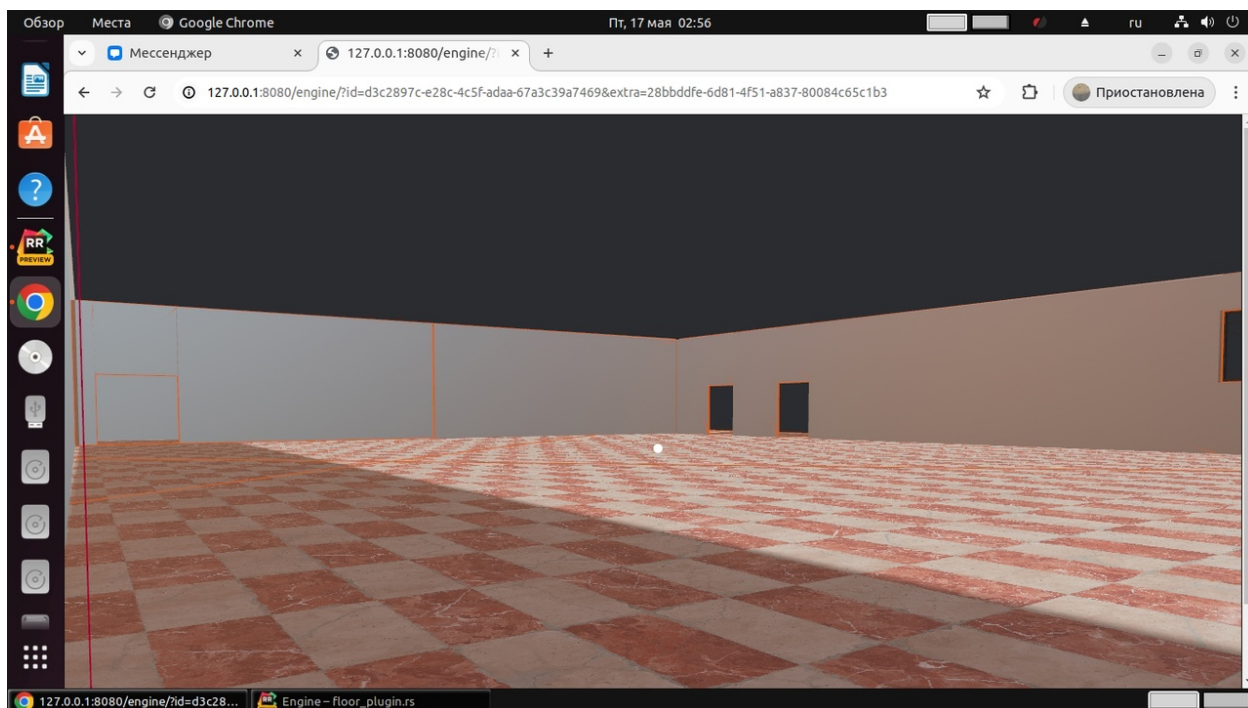


Рисунок 18-Второй этаж, 3D-симуляция.

Также был произведён обход по первому этажу. Результат представлен на рисунках 19–21.

					ПГУ 1.090301.34.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		40

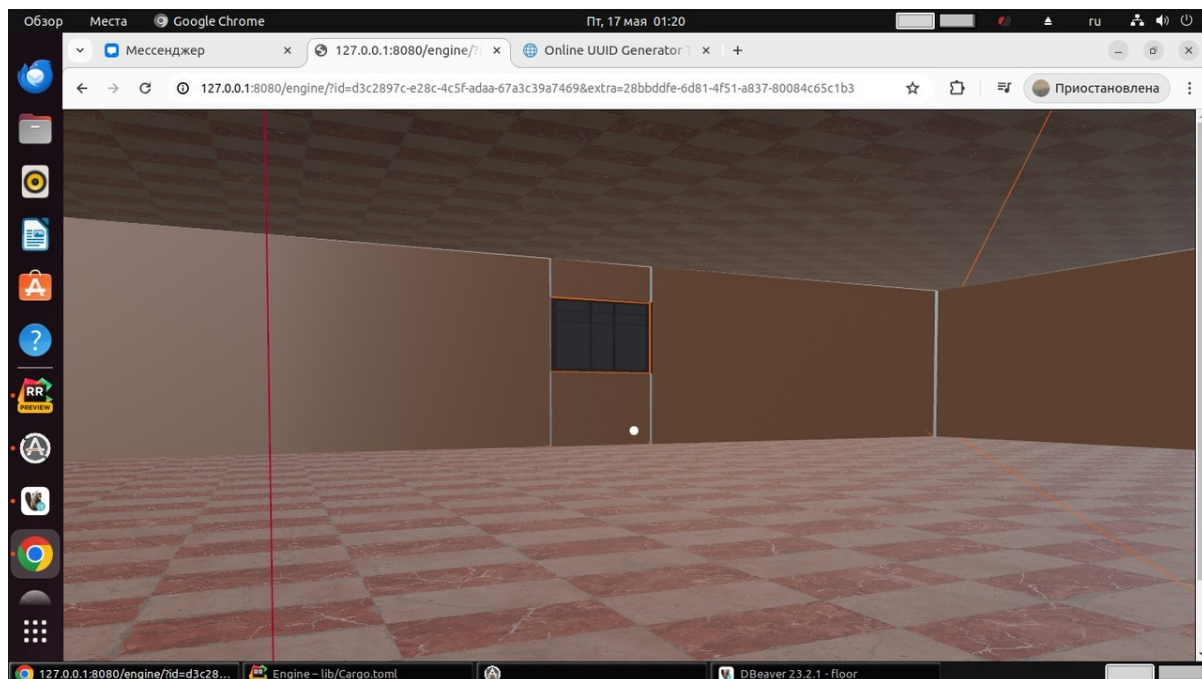


Рисунок 19-Первый этаж.3D-симуляция.

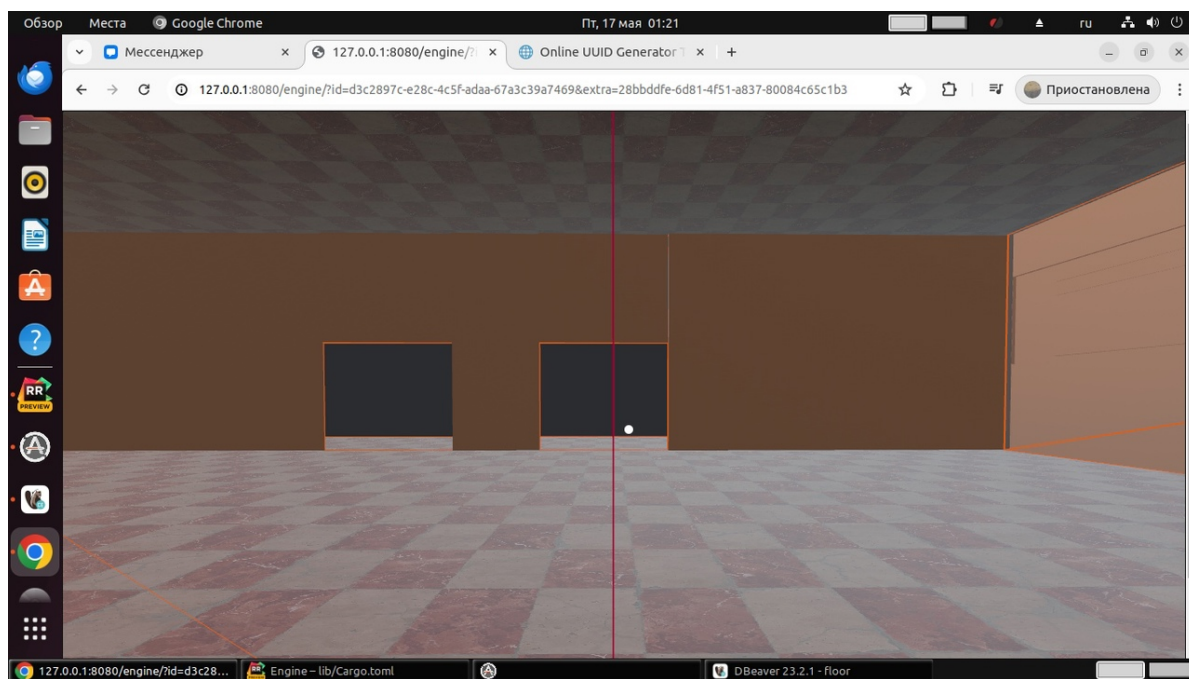


Рисунок 20-Первый этаж.3D-симуляция.

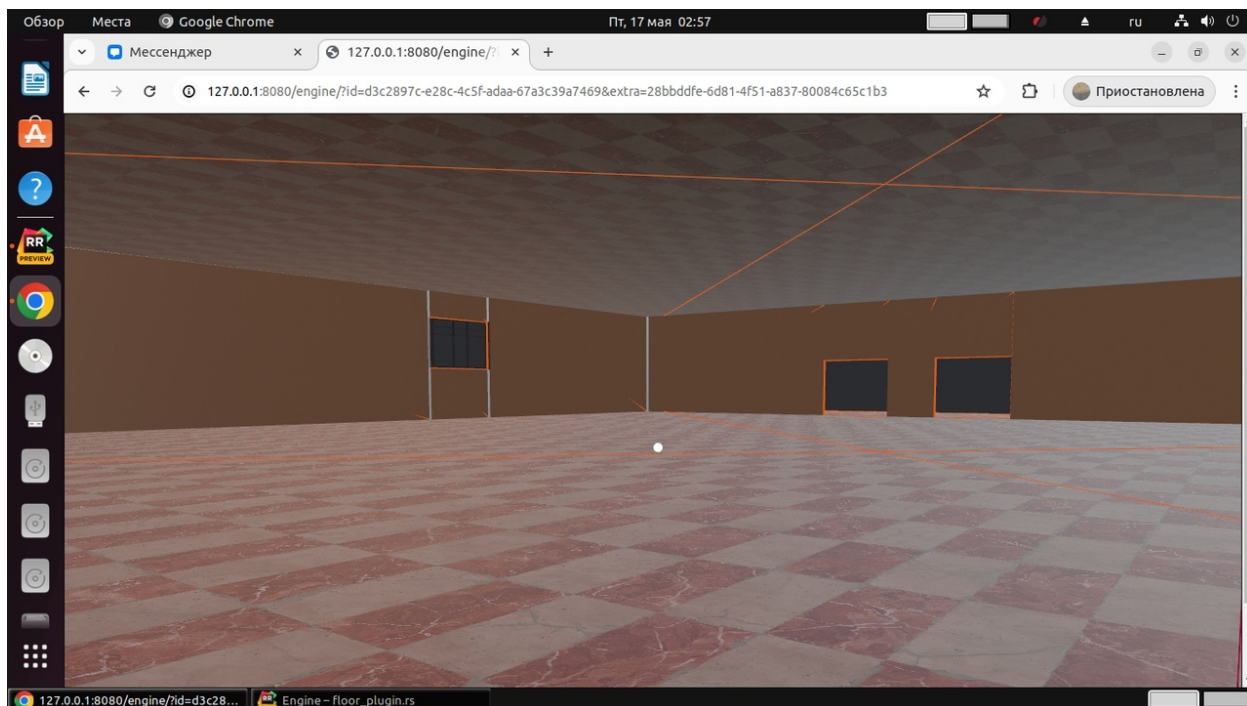


Рисунок 21-Первый этаж.3D-симуляция.

Также была произведена проверка функционала управлением персонажем. Было протестировано передвижение, ускорения, взаимодействие с окружающими объектами, прыжки и приседания. Поведение персонажа при подпрыгивании и приседании представлено на рисунках 22, 23.

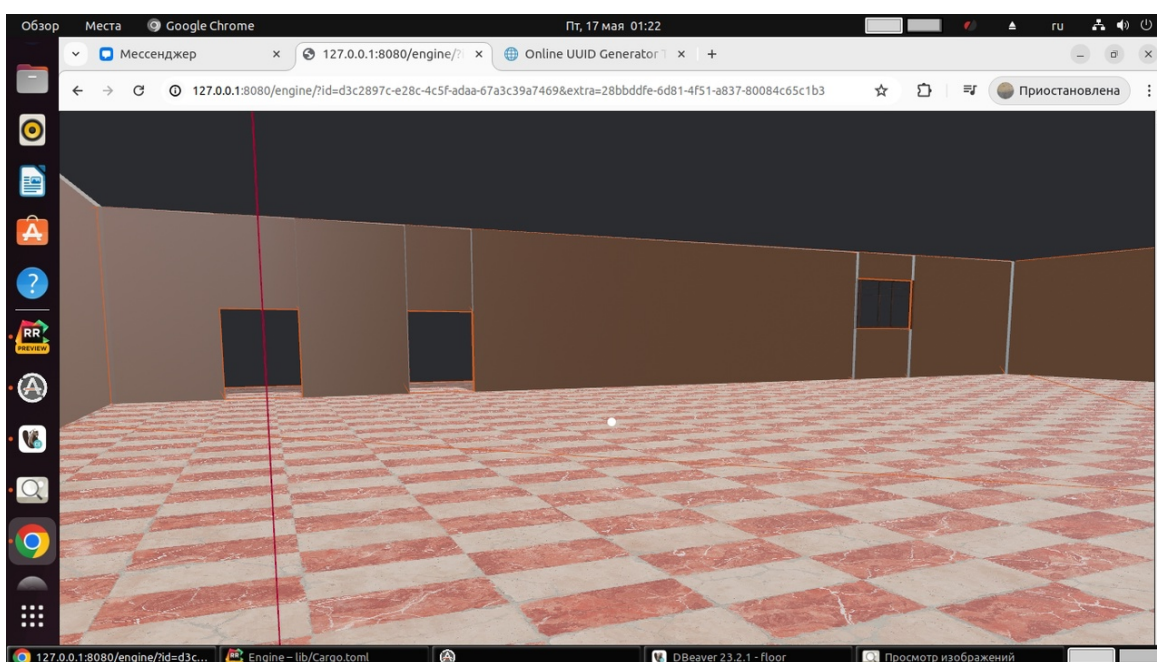


Рисунок 22-прыжок персонажа.

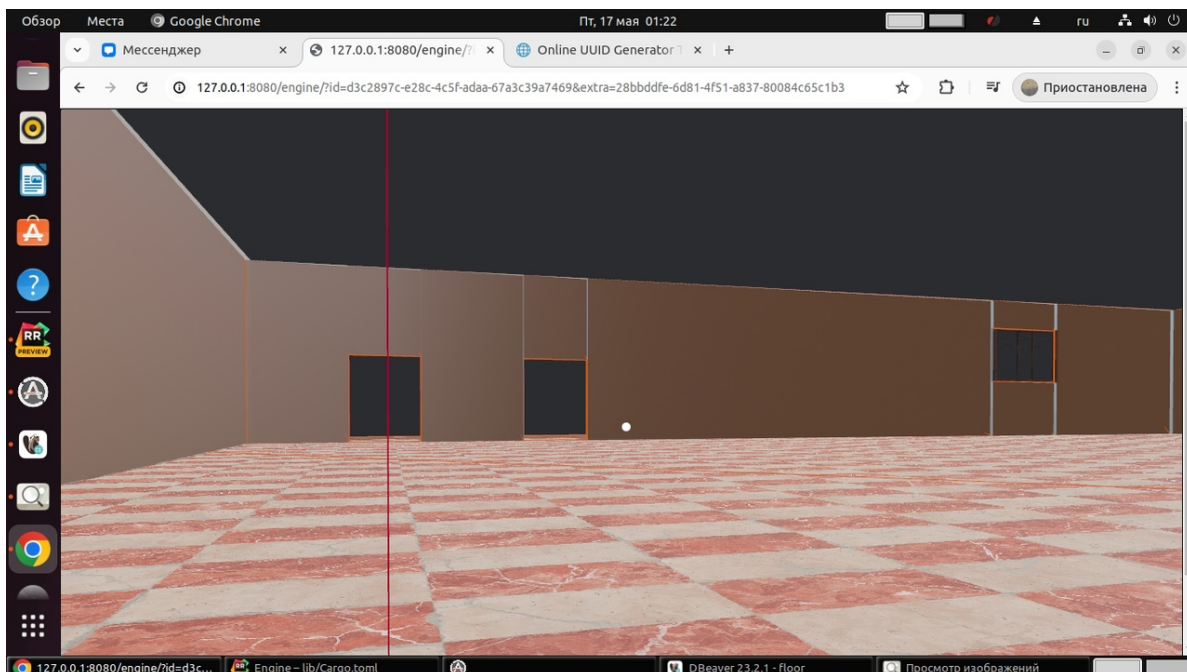


Рисунок 23-приседание персонажа.

Была протестирована система частиц для визуализации огня. Демонстрация приведена на рисунках 24, 25.

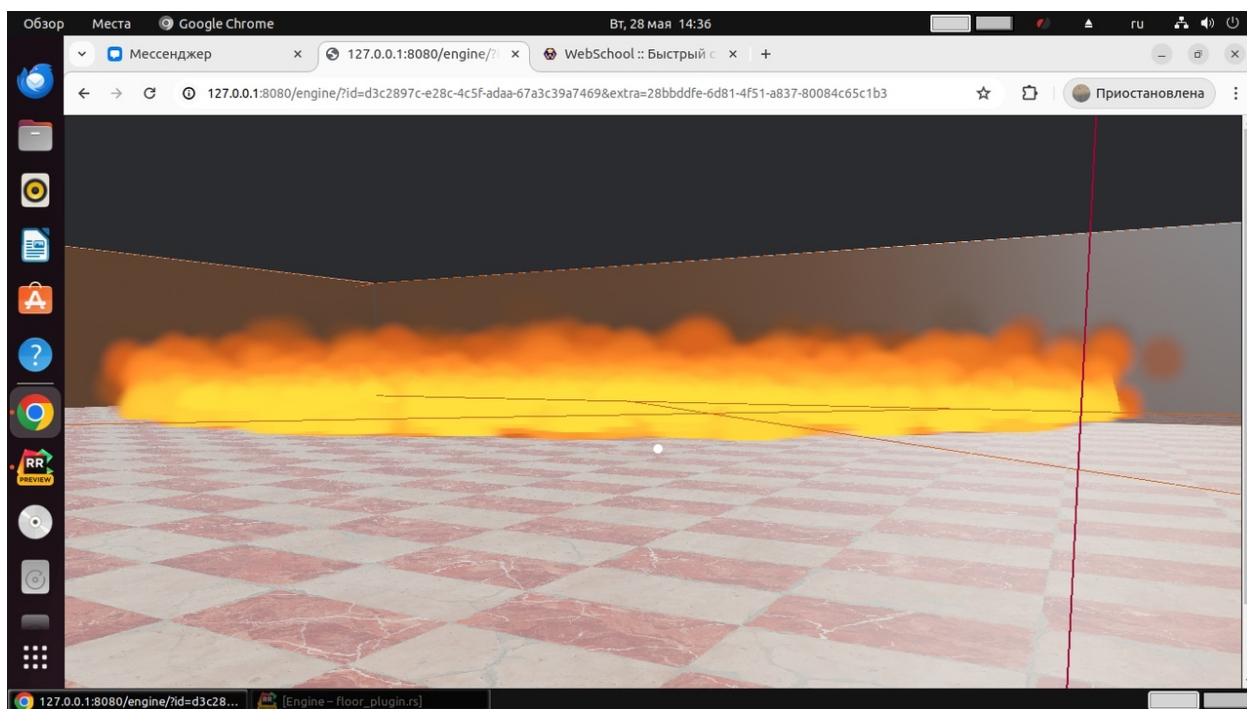


Рисунок 24-Эффект огня.

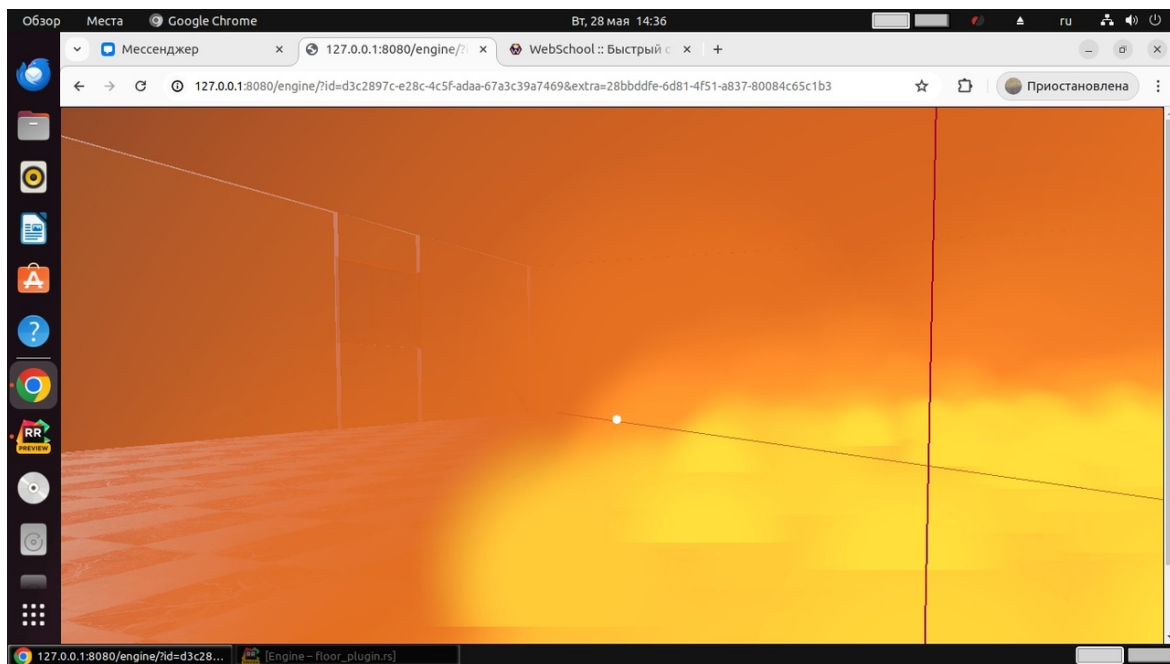


Рисунок 25-Эффект огня внутри.

Процесс тестирования прошел успешно, все функции приложения работают правильно, ожидаемый результат совпал с действительным, тестирование данных функций прошли успешно.

2 Безопасность жизнедеятельности

При взаимодействии с компьютером, необходимо соблюдать меры безопасности, чтобы не было неприятных последствий для здоровья после использования. Если пользователь ПК будет пренебрегать техникой безопасности, то проблемы со здоровьем будут неизбежны.

2.1 Вред здоровью при работе с ПК

Проблемы со здоровьем в процессе взаимодействия с компьютером имеют свойство накапливаться, то есть здоровье ухудшается не сразу, а постепенно. Наиболее опасными факторами являются:

- нагрузка на суставы. При неправильной посадке за компьютером, на суставы будет идти большая нагрузка, а именно кистей рук, из-за чего может произойти даже онемение рук;

- нагрузка на спину. Опять же, если принять неправильно положение осанки при работе за компьютером, может появиться искривление позвоночника и даже грыжа межпозвоночного диска;

- нагрузка на органы зрения. При долгой работе с мониторами, будет ухудшаться зрение, впоследствии могут появиться синдромы «красного глаза» и «сухого глаза»;

- нагрузка на психику и нервную. При долгом взаимодействии с ПК, у пользователя могут возникать проблемы с памятью, а также может появиться депрессия и даже бессонница.

2.2 Снижение вреда для здоровья при работе с ПК

При соблюдении простых правил можно избежать серьезных проблем со здоровьем, для этого нужно:

– соблюдать необходимое расстояние до монитора. Правильным расстоянием до дисплея является расстояние, при котором пользователь может коснуться указательным пальцем до верхнего края монитора;

– соблюдать оптимальный угол экрана. Оптимальным углом является наклон монитора в 30 градусов;

– клавиатуру ставить в нужное положение. Клавиатура должна находиться на расстоянии 20-30 сантиметров от края стола;

– делать перерыв. Примерно каждый час необходимо делать разминку для органов зрения, а именно оторваться от монитора и делать гимнастику для глаз: рисовать различные фигуры глазами, фокусировать внимание на дальние объекты и переводить фокус с одних предметов на другие. Также примерно каждые полтора-два часа необходимо выполнять зарядку;

– выбрать правильный стул или кресло. При выборе данной мебели, необходимо приобрести такую, которая будет обеспечивать прямую осанку, при которой спина будет опираться в спинку.

2.3 Техника безопасности при работе с компьютером

При работе с компьютером необходимо соблюдать следующие правила, которые связаны с техникой безопасности:

- не оставлять компьютер без присмотра на длительное время;
- не использовать компьютер, который работает от питания без заземления;
- не работать за ПК, у которого есть повреждение в системном блоке или в его периферийных устройствах;
- не включать компьютер, если в помещении слишком высокая влажность;
- не класть на корпус компьютера или монитора посторонние предметы.

Заключение

В ходе выполнения данной выпускной квалификационной работы были получены навыки написания 3D-приложения для тренировки эвакуации из здания с поддержкой запуска из браузера. Результатом работы является разработанный интерактивный 3D-тренажёр «Эвакуация из здания», а также отчёт, в котором представлены результаты тестирования разработанного программного обеспечения, описание разработки и подробный разбор его основных функциональных особенностей.

Разработанное приложение позволяет пользователю прямо в браузере увидеть прототип созданного здания в 3D, перемещаться по нему от первого лица и выполнять различные действия. После проведения тестирования было выяснено, что разработанное приложение соответствует всем поставленным в начале работы требованиям и работает корректно, не вызывая критических ошибок в системе.

В дальнейшем приложение можно усовершенствовать путем добавления окружающих людей, декораций, а также возможности стоять здания в форме окружности. Разработанный интерфейс пользователя можно улучшить путем добавления инвентаря и шкалы жизни для реализации более лучшей реалистичности.

Список использованных источников

1. VR ОБЖ // Varwin. – URL: <https://varwin.com/ru/education/vr-obzh/> (дата обращения 20.04.2024)
2. Fire Safety Lab VR // Steam. – URL: https://store.steampowered.com/app/1250170/Fire_Safety_Lab_VR/ (дата обращения 23.04.2024)
3. Unity (игровой движок) // Википедия. – URL: https://ru.wikipedia.org/wiki/Unity_/ (дата обращения 25.04.2024)
4. Unreal Engine – Features // Unreal Engine. – URL: <https://www.unrealengine.com/en-US/features/> (дата обращения 27.04.2024)
5. Introduction // Bevy Engine – URL: <https://bevyengine.org/learn/quick-start/introduction/> (дата обращения 28.04.2024)
6. Foreword // Rust Programming Language. – URL: <https://doc.rust-lang.org/book/title-page.html/> (дата обращения 30.04.2024)

Приложение А

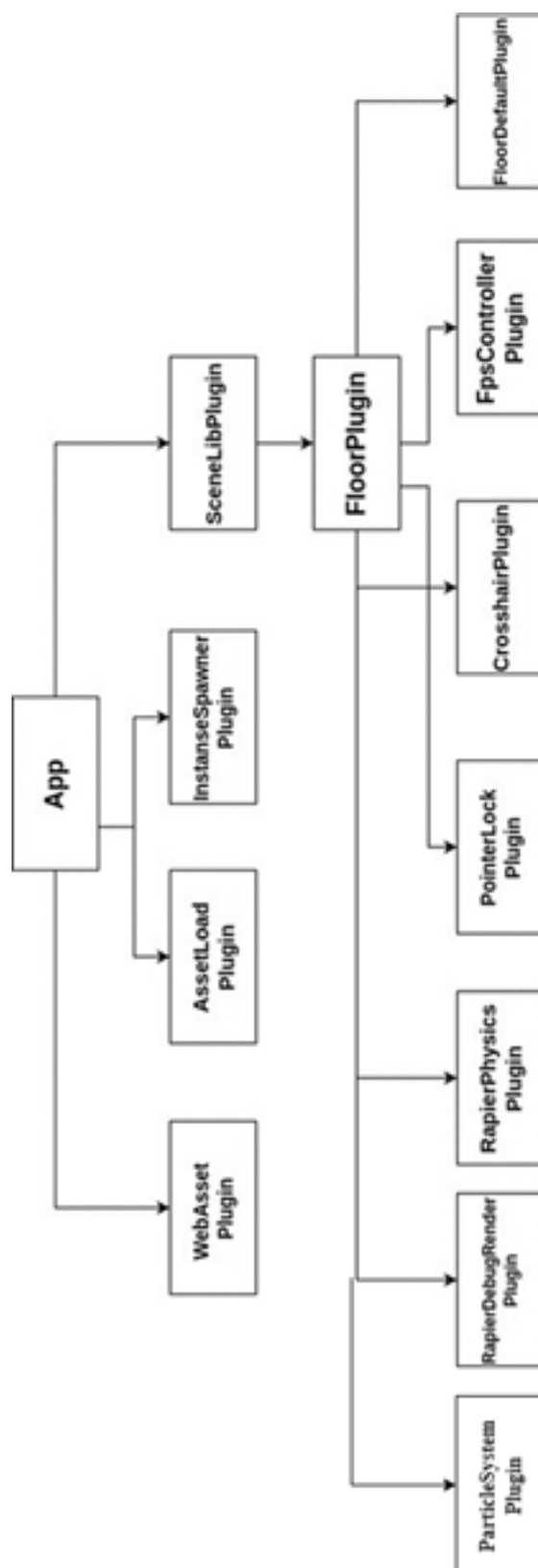
(Обязательное)

Интерактивный 3D тренажёр «Эвакуация из здания»

UML-диаграммы

					ПГУ 1.090301.34.001 ПЗ	Лист
						49
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение А.1-Диаграмма компонентов

[illegible]

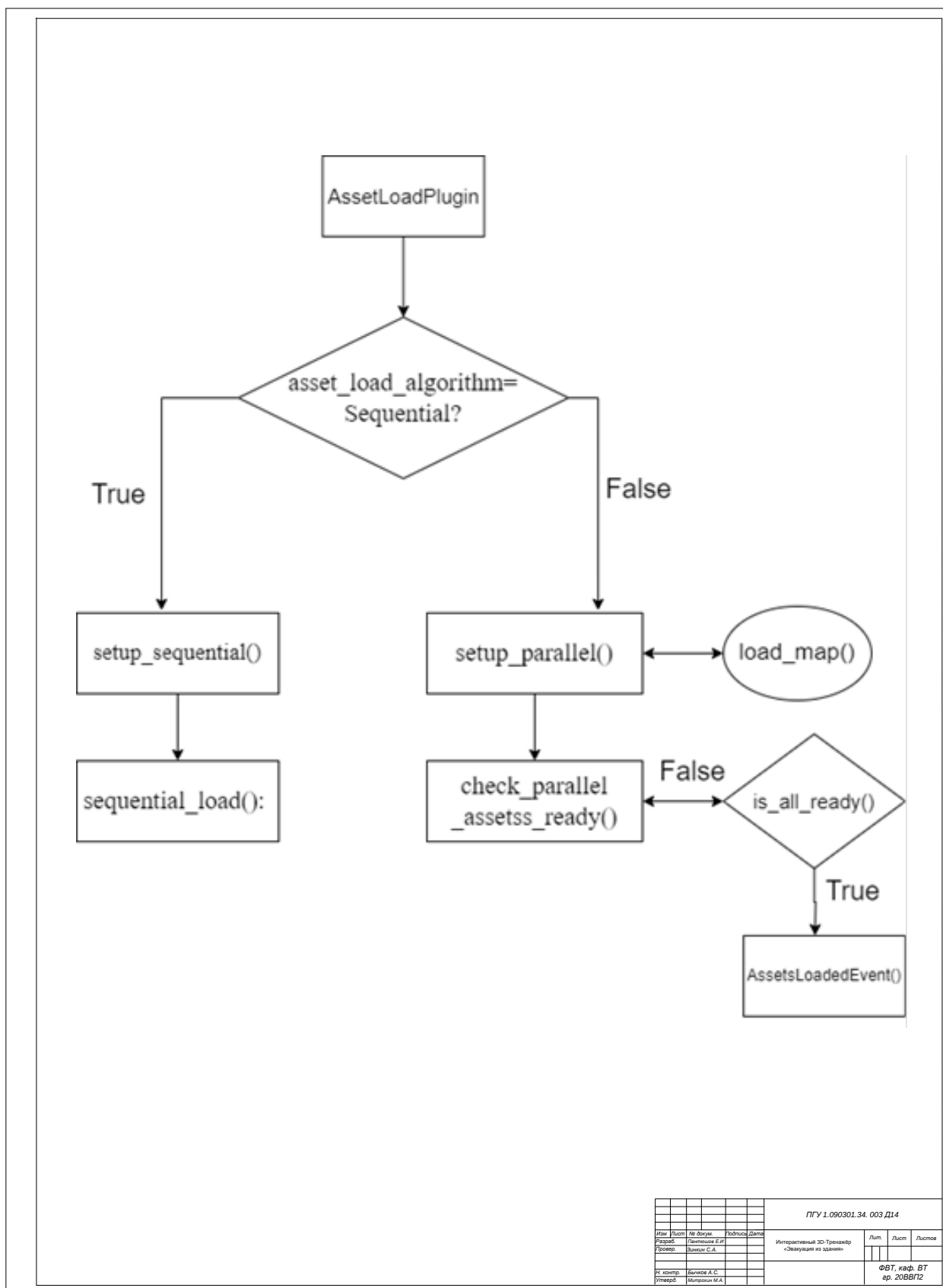
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.34.001 ПЗ

Лист

50

Приложение А.2 – Диаграмма последовательностей плагина AssetLoadPlugin



				ПГУ 1.090301.34. 003 Д14			
Изм.	Лист	№ докум.	Подпись	Дата	Изм.	Лист	Листов
Разраб.	Пычешова Е.И.				Интерактивный 3D-Тренажер «Эвакуация из здания»		
Провер.	Виноков С.А.						
И. вып.	Виноков А.С.				ФБТ, каф. ВТ гр. 20ВВВТ2		
Утверд.	Митрохин М.А.						

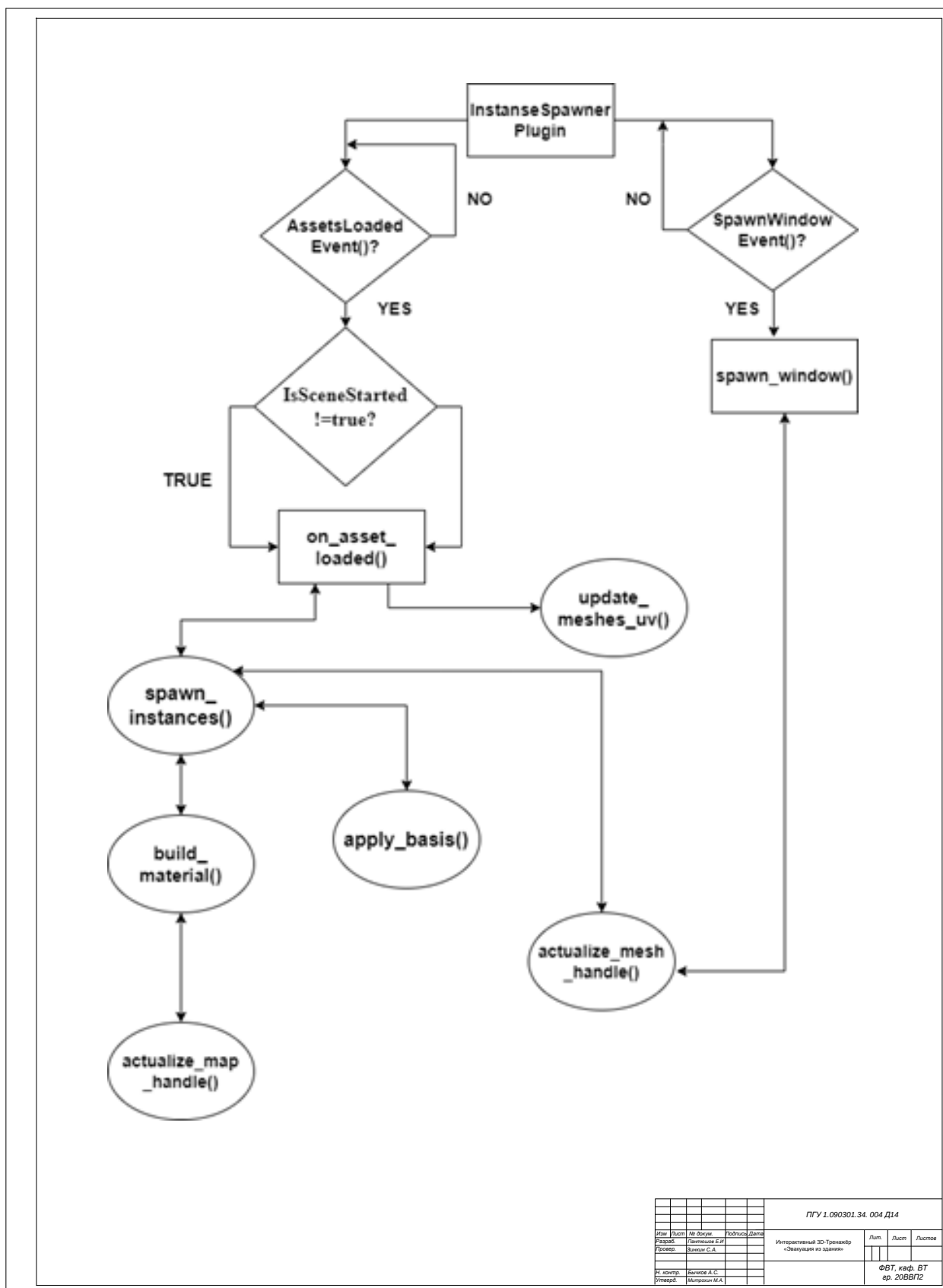
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.34.001 ПЗ

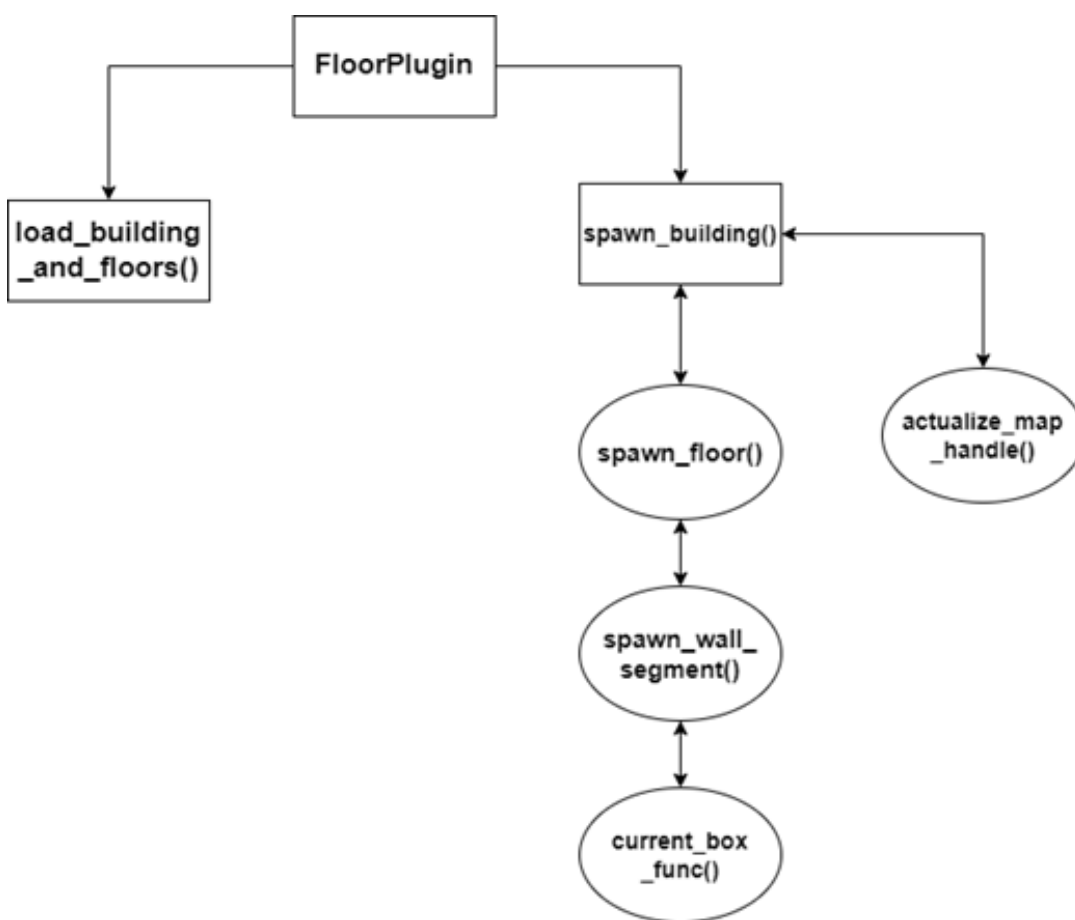
Лист

51

Приложение А.3 – Диаграмма последовательностей плагина InstanseSpawnerPlugin



Приложение А.4-Диаграмма последовательностей плагина FloorPlugin



ПГУ 1.090301.34. 005 Д14				
Изм.	Лист	№ докум.	Подпись	Дата
Разраб.	Лыткин Е.И.			
Провер.	Винкин С.А.			
И. номер	Исход. А.С.			
Итератор	Митрохин М.А.			

Информационный 3D-Тренажер
«Эвакуация из здания»

Лит. Листов

ФБТ, каф. ВТ
гр. 20БВТ2

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.34.001 ПЗ

Лист

53

Приложение Б

(Обязательное)

Интерактивный 3D-тренажёр «Эвакуация из здания»

Листинг программы

					ПГУ 1.090301.34.001 ПЗ	Лист
						54
Изм.	Лист	№ докум.	Подпись	Дата		

Файл main.rs

```
use url::Url;
use wasm_bindgen::prelude::*;

#[wasm_bindgen]
extern "C" {
    #[wasm_bindgen(js_namespace = console)]
    fn log(s: &str);
}
macro_rules! console_log {
    ($($t:tt)*) => (log(&format_args!($($t)*).to_string()))
}

fn main() {
    println!("Hello, world!");
}

#[wasm_bindgen(start)]
pub async fn start_scene() {
    let mut scene_id = "".to_string();
    let mut extra = "".to_string();

    //parse url params
    let window = web_sys::window();
    if window.is_none() {
        console_log!("Error handling web_sys::Window");
        return;
    }
    let url_str = window.unwrap().location().href();
    if url_str.is_err() {
        console_log!("Error handling
web_sys::Location.href()");
        return;
    }
    let url = Url::parse(url_str.unwrap().as_str());
    if url.is_err() {
        console_log!("Error parsing URL");
        return;
    }
    let url2 = url.unwrap();
    let params = url2.query_pairs();
    for (key, value) in params.into_iter() {
        if key.as_ref() == "id" {
            scene_id = value.as_ref().to_string();
        }
        if key.as_ref() == "extra" {
            extra = value.as_ref().to_string();
        }
    }

    match enginelib::start(scene_id.clone(), extra).await {
```

Изм.	Лист	№ докум.	Подпись	Дата

```

        Ok(_) => {
            console_log!("Scene started with id {:?}",
scene_id)
        }
        Err(_err) => {
            // Обработка ошибки
            console_log!("Scene failed with id {:?}", scene_id)
        }
    }
}

```

Файл lib.rs

```

use bevy::asset::AssetMetaCheck;
use bevy::prelude::*;

use wasm_bindgen::prelude::*;
// use bevy_hanabi::prelude::*;
use appslib::store::scene_store::SceneStore;
use bevy_web_asset::WebAssetPlugin;
use uuid::Uuid;

use crate::component::game_object::GameObject;
use appslib::dto::scene_response::SceneResponse;
// use crate::network::network::{fetch_by_local_path};
use crate::util::ang_to_rad;

//use
crate::asset_loader::custom_asset_io::CustomAssetReaderPlugin;

use crate::plugin::assets_load_plugin::AssetsLoadPlugin;
use
crate::plugin::instance_spawner_plugin::InstanceSpawnerPlugin;
use crate::scene_lib::scene_lib_plugin::SceneLibPlugin;

pub mod component;
pub mod network;
pub mod plugin;
pub mod scene_lib;
pub mod util;

#[wasm_bindgen]
extern "C" {
    #[wasm_bindgen(js_namespace = console)]
    fn log(s: &str);
}
macro_rules! console_log {
    ($($t:tt)*) => (log(&format_args!($($t)*).to_string()))
}

#[derive(Resource)]
pub struct SceneInitDataRes {

```

```

        pub scene_response: SceneResponse, //http response
        pub extra_param: Option<String>,    //some additional param
passed via URL or LocalStorage
    }

    pub async fn start(scene_id: String, extra: String) ->
Result<SceneResponse, String> {
        console_log!("id from URL param: {}",
scene_id.to_string().as_str());

        // 8b4de5b7-a992-445d-baa9-af3be4effc8a
        //Historix 1c3e872a-4bf5-4916-8e2d-7d9e4a5d8dd2
        //glb periodic 3dfd682d-a812-4d36-b423-87e982d745cf

        //get from the server
        let id = Uuid::parse_str(&scene_id).unwrap();
        let resp = SceneStore::get_by_id(id).await;

        //read a local file
        // let resp =
fetch_by_local_path:::<SceneResponse>("floor.json".to_string()).await
;

        if resp.clone().is_err() {
            console_log!("{:?}", resp.clone().err().unwrap());
            return resp;
        }

        console_log!("res {:?}", resp.clone().unwrap());
        let scene_data = resp.clone().unwrap().data.unwrap();

        let mut app = App::new();
        app.register_type:::<GameObject>()
            .insert_resource(AssetMetaCheck::Never)
            .add_plugins((WebAssetPlugin::default(),
DefaultPlugins))
            //
            .add_plugins(DefaultPlugins.set(ImagePlugin::default_nearest()).disa
ble:::<AssetPlugin>())
            // .add_plugins(CustomAssetReaderPlugin)
            // .add_plugins(HanabiPlugin)

            .add_plugins(AssetsLoadPlugin {
                asset_load_algorithm:
scene_data.clone().assets_loading_algorithm,
            })
            .add_plugins(SceneLibPlugin {
                scene_lib_name: scene_data.clone().scene_lib_name,
            })
            .add_plugins(InstanceSpawnerPlugin)
            //
            .add_plugins(DefaultPlugins.build().disable:::<LogPlugin>()) //

```

```

disable LogPlugin so that you can pipe the output directly into `dot
-Tsvg`
        .insert_resource(SceneInitDataRes {
            scene_response: resp.clone().unwrap(),
            extra_param: if extra != "" { Some(extra) } else {
None },
        })
        .run();

// bevy_mod_debugdump::print_main_schedule(&mut app);
return resp;
}

```

Файл floor_plugin.rs

```

use crate::plugin::crosshair_plugin::CrosshairPlugin;
use crate::plugin::full_viewport_plugin::FullViewportPlugin;
use crate::plugin::game_character_plugin::GameCharacterPlugin;
use crate::plugin::object_grab_plugin::ObjectGrabPlugin;
use crate::plugin::pointer_lock_plugin::PointerLockPlugin;

use std::ops::{Neg, Sub};

use crate::plugin::fly_camera_plugin::FlyCameraPlugin;

use crate::plugin::assets_load_plugin::AssetsLoadedRes;
use crate::plugin::instance_spawner_plugin::IsSceneStarted;
use bevy_rapier3d::prelude::*;

use bevy::render::mesh::Indices;
use bevy::render::mesh::VertexAttributeValues;
use bevy_fps_controller::controller::*;
use earcutr::Error;

use
crate::plugin::instance_spawner_plugin::actualize_map_handle;
use bevy::prelude::*;
use bevy::render::mesh::PrimitiveTopology;
use
propertylib::dto::building_with_floors_response::BuildingWithFloorsR
esponse;
use propertylib::store::building_store::BuildingStore;
use std::sync::{
    mpsc::{Receiver, Sender},
    Mutex,
};
use std::borrow::BorrowMut;
use bevy::render::camera::Exposure;
use bevy::render::render_asset::RenderAssetUsages;

use bevy_rapier3d::prelude::*;
use propertylib::dto::floor_response::FloorResponse;

```

```

use uuid::Uuid;

use
crate::scene_lib::floor::component::floor_wall_part::FloorWallPart;
use
crate::scene_lib::floor::floor_default_plugin::FloorDefaultPlugin;
use crate::SceneInitDataRes;
use propertylib::jsonb::structure::corner::Corner;
use propertylib::jsonb::structure::door::Door;
use propertylib::jsonb::structure::wall::Wall;
use propertylib::jsonb::structure::window::Window;
//Channel for waiting for loading
type FloorsSender = Sender<BuildingWithFloorsResponse>;
type FloorsReceiver = Receiver<BuildingWithFloorsResponse>;
#[derive(Resource)]
pub struct FloorSenderRes(Mutex<FloorsSender>);
#[derive(Resource)]
pub struct FloorReceiverRes(Mutex<FloorsReceiver>);

#[derive(Resource)]
pub struct FloorState {
    floors: Option<Vec<FloorResponse>>,
    floor_index: usize,
    floors_count: usize,
}

#[derive(Event)]
pub struct SpawnWindowEvent {
    pub position: Vec3,
    pub quat:Vec3,
    pub scale:Vec2
}

pub struct Vertex {
    position: Vec3,
}

#[derive(Debug)]
pub struct Opening {
    pub id: Uuid,
    pub x_start: i32,
    pub y_start: i32,
    pub x_end: i32,
    pub y_end: i32,
    pub height_start: f32,
    pub height_end: f32,
}

impl Opening {
    // Implement the new function
    fn new(
        id: Uuid,

```

```

        x_start: i32,
        y_start: i32,
        x_end: i32,
        y_end: i32,
        height_start: f32,
        height_end: f32,
    ) -> Self {
        Opening {
            id,
            x_start,
            y_start,
            x_end,
            y_end,
            height_start,
            height_end,
        }
    }
}

pub struct CurrentBox {
    pub min_x: f32,
    pub max_x: f32,

    pub min_y: f32,
    pub max_y: f32,

    pub min_z: f32,
    pub max_z: f32,
}

pub struct FloorPlugin;

impl Plugin for FloorPlugin {
    fn build(&self, app: &mut App) {
        info!("Floor planner lib enabled!");

        let channel = std::sync::mpsc::channel();
        let f_sender: FloorsSender = channel.0;
        let f_receiver: FloorsReceiver = channel.1;

        app.add_plugins(FullViewportPlugin)
            .add_plugins(PointerLockPlugin)
            //.add_plugin(WorldInspectorPlugin::new())
            .add_plugins(CrosshairPlugin)
            // .add_plugins(GameCharacterPlugin)
            // .add_plugins(FlyCameraPlugin)
            // .add_plugins(ObjectGrabPlugin::default())
            //physics
            .add_event::<SpawnWindowEvent>()
            .add_plugins(FpsControllerPlugin)

        .add_plugins(RapierPhysicsPlugin::<NoUserData>::default())
    }
}

```

```

        .add_plugins(RapierDebugRenderPlugin::default())
        .add_plugins(FloorDefaultPlugin)

    .insert_resource(FloorSenderRes(Mutex::new(f_sender)))
        .insert_resource(RapierConfiguration::default())

    .insert_resource(FloorReceiverRes(Mutex::new(f_receiver)))
        .insert_resource(FloorState {
            floor_index: 0,
            floors_count: 0,
            floors: None,
        })
        .add_systems(Startup, load_building_and_floors)
        .add_systems(Update, setup_environment)
        .add_systems(Update, spawn_building);
    }
}

fn setup_environment(
    _commands: Commands,
    _meshes: ResMut<Assets<Mesh>>,
    _materials: ResMut<Assets<StandardMaterial>>,
    // mut effects: ResMut<Assets<EffectAsset>>
    _query_camera: Query<&mut Transform, With<Camera>>,

    _is_scene_started: Res<IsSceneStarted>,
    _has_setuped: Local<bool>,
) {
    // if *has_setuped || !is_scene_started.0 {
    //     return;
    // }
    //
    // for mut camera_transform in query_camera.iter_mut(){
    //     camera_transform.translation = Vec3::new(-
54.0, 70.0, 40.0);
    //     camera_transform.look_at(Vec3::new(50.0, 0.0, 50.0),
Vec3::Y);
    // }
    // *has_setuped = true;
}

fn load_building_and_floors(sender: Res<FloorSenderRes>,
scene_data: Res<SceneInitDataRes>) {
    let id_str = if scene_data.extra_param.is_some() {
        scene_data.extra_param.clone().unwrap()
    } else {
        "408e43c0-8a3d-11eb-a13f-3b448741dae2".to_string()
    }
    //default
    };

    let local_sender = sender.0.lock().unwrap().clone();
    wasm_bindgen_futures::spawn_local(async move {

```

```

        let id = Uuid::parse_str(&id_str).unwrap();
        let resp =
BuildingStore::get_with_floors_by_id(id).await;
        if resp.clone().is_err() {
            info!(
                "Cannot fetch periodic json {:?}",
                resp.clone().err().unwrap()
            );
        } else {
            let resp = resp.clone().unwrap();
            local_sender.send(resp).unwrap();
        }
    })
}

fn face_normal(v1: Vec3, v2: Vec3, v3: Vec3) -> Vec3 {
    let edge1 = v2 - v1;
    let edge2 = v3 - v1;

    // Проверяем, что векторное произведение не равно нулевому
    вектору
    let cross_product = edge1.cross(edge2);
    if cross_product.length() == 0.0 {
        // Если длина вектора равна 0, возвращаем вектор (0, 1,
    0) или другой вектор по вашему выбору
        // Это предотвращает NaN при нормализации нулевого
    вектора
        return Vec3::new(0.0, 1.0, 0.0);
    }

    cross_product.normalize().into()
}

fn generate_coating(building_corners: Vec<Corner>, wall_down:
f32) -> Mesh {
    let wall_height = wall_down + 1.0;
    let mut building_vertices: Vec<Vec2> = vec![];
    let mut combined_vertices: Vec<Vec3> = Vec::new();
    let _normals: Vec<Vec3> = Vec::new();
    for i in 0..building_corners.len() {
        let corner = &building_corners[i];
        let corner_p = Vec2::new(corner.x as f32 / 10.0,
corner.y as f32 / 10.0);
        building_vertices.push(corner_p);
    }

    let flattened_vertices: Vec<f64> = building_vertices
        .iter()
        .flat_map(|v| vec![v.x as f64, v.y as f64])
        .collect();
    let indices: Result<Vec<usize>, Error> =
earcutr::earcut(&flattened_vertices, &vec![], 2);

```

```

let indices = indices.expect("Earcut failed");
let _num_vertices = building_vertices.len();
let mut indices_res: Vec<u32> = vec![];
let mut normals: Vec<Vec3> = Vec::new();
for i in 0..indices.len() {
    let floor_vertex = Vec3::new(
        building_vertices[*&indices[i]].x,
        wall_down,
        building_vertices[*&indices[i]].y,
    );
    combined_vertices.push(floor_vertex);
    indices_res.push(i as u32);
}
for i in 0..indices.len() {
    let floor_vertex = Vec3::new(
        building_vertices[*&indices[i]].x,
        wall_height,
        building_vertices[*&indices[i]].y,
    );
    combined_vertices.push(floor_vertex);
    indices_res.push((i + indices.len()) as u32);
}
/* for i in 0..building_vertices.len() {
    let floor_vertex = Vec3::new(building_vertices[i].x,
wall_down, building_vertices[i].y);
    combined_vertices.push(floor_vertex);
}
for i in 0..building_vertices.len() {
    let floor_vertex = Vec3::new(building_vertices[i].x,
wall_height, building_vertices[i].y);
    combined_vertices.push(floor_vertex);
}
for i in 0..building_vertices.len() {
    let floor_vertex = Vec3::new(building_vertices[i].x,
wall_down, building_vertices[i].y);
    combined_vertices.push(floor_vertex);
}
for i in 0..building_vertices.len() {
    let floor_vertex = Vec3::new(building_vertices[i].x,
wall_height, building_vertices[i].y);
    combined_vertices.push(floor_vertex);
}*/
/* for i in 0..building_corners.len() {
    indices_res.push((i + indices.len()*2 )as u32);
    indices_res.push((i + indices.len()*2
+building_corners.len()) as u32);
    indices_res.push((i+indices.len()*2 +
building_corners.len()*2) as u32);
    indices_res.push((i+indices.len()*2 +
building_corners.len()) as u32);
    indices_res.push((i + indices.len()*2 +
building_corners.len()*3) as u32);
}

```

```

        indices_res.push((i + indices.len()*2 +
building_corners.len()*2) as u32);
    }*/
    /* for i in 0..num_vertices {
        let floor_vertex = Vec3::new(building_vertices[i].x,
wall_down, building_vertices[i].y);
        combined_vertices.push(floor_vertex);
    }
    for i in 0..building_corners.len() {
        let roof_vertex = Vec3::new(building_vertices[i].x,
wall_height, building_vertices[i].y);
        combined_vertices.push(roof_vertex);
    }*/

    /* for i in 0..indices.len() {
        indices_res.push(indices[i] as u32);
    }
    for i in 0..indices.len() {
        indices_res.push((indices[i] + building_corners.len())
as u32);
    }
    for i in 0..building_corners.len() {
        indices_res.push((indices[i] +
building_corners.len()*2)as u32);
        indices_res.push((indices[i] +
building_corners.len()*3) as u32);
        indices_res.push((indices[i+1] +
building_corners.len()*3) as u32);
        indices_res.push((indices[i+1] +
building_corners.len()*3) as u32);
        indices_res.push((indices[i] +
building_corners.len()*2) as u32);
        indices_res.push((indices[i+1] +
building_corners.len()*2) as u32);
    }*/

    let mut texture_coordinates: Vec<Vec2> = Vec::new();

    // Calculate texture coordinates based on vertex position
    for i in 0..combined_vertices.len() {
        let vertex = combined_vertices[i];

        // Example: Use the x and y components of the vertex as
texture coordinates
        let u = vertex.x / 100.0; // Adjust max_x_value based
on your needs
        let v = vertex.z / 100.0; // Adjust max_y_value based
on your needs

        texture_coordinates.push(Vec2::new(u, v));
    }

```

```

// Calculate normals for each face
/* for i in (0..combined_vertices.len()-2).step_by(3) {
    let i1 = indices_res[i ] as usize;
    let i2 = indices_res[i+ 1] as usize;
    let i3 = indices_res[i + 2] as usize;

    let v1 = combined_vertices[i1];
    let v2 = combined_vertices[i2];
    let v3 = combined_vertices[i3];

    // Calculate face normal
    let normal = face_normal(v1, v2, v3);

    // Assign the same normal to each vertex of the face
    normals.push(normal);
    normals.push(normal);
    normals.push(normal);

}*/
for _i in 0..combined_vertices.len() / 2 {
    normals.push([0.0, -1.0, 0.0].into());
}
for _i in 0..combined_vertices.len() / 2 {
    normals.push([0.0, 1.0, 0.0].into());
}
//normals.push([0.0,1.0,0.0].into());
// normals.push([0.,1.0,0.0].into());
info!("combined_vertices{:?}", combined_vertices);
info!("indices_res{:?}", indices_res);
info!("indices_res{:?}", normals);
let mut mesh = Mesh::new(
    PrimitiveTopology::TriangleList,
    RenderAssetUsages::MAIN_WORLD |
RenderAssetUsages::RENDER_WORLD,
);
mesh.insert_attribute(Mesh::ATTRIBUTE_NORMAL, normals);
mesh.insert_attribute(Mesh::ATTRIBUTE_POSITION,
combined_vertices);
mesh.insert_attribute(Mesh::ATTRIBUTE_UV_0,
texture_coordinates);
mesh.insert_indices(Indices::U32(indices_res));
mesh
}
fn spawn_building(
    mut commands: Commands,
    mut meshes: ResMut<Assets<Mesh>>,
    mut materials: ResMut<Assets<StandardMaterial>>,
    loaded_assets: Res<AssetsLoadedRes>,
    mut images: ResMut<Assets<Image>>,

    receiver: Res<FloorReceiverRes>,
    is_scene_started: Res<IsSceneStarted>,

```

```

mut floor_state: ResMut<FloorState>,
mut has_it_spawned: Local<bool>,
mut event:EventWriter<SpawnWindowEvent>
) {
    if *has_it_spawned || !is_scene_started.0 {
        return;
    }
    let channel_result = receiver.0.lock().unwrap().try_recv();
    if channel_result.is_err() {
        return;
    }
    let building =
channel_result.clone().unwrap().building.clone();
    let floors = channel_result.unwrap().floors.clone();
    if building.json_schema.clone().is_none()
        ||
building.json_schema.clone().unwrap().corners.is_none()
    {
        return;
    }
    if floors.is_some() {
        floor_state.floors = floors.clone();
        floor_state.floor_index = 0;
        floor_state.floors_count =
floors.clone().unwrap().len();
    }
    let mut wall_height = 10.0 * floor_state.floors_count as
f32;

    let _wall_width = 0.5;

    let _wall_material = materials.add(Color::GRAY);

    let floor_texture_path =
"https://staticapi.webschool.au/data/4e7fc989-2216-4759-aeb7-
9a719ae34f5f/3dmodels/textures/texturescom_marble_checkerboard2_1k_a
lbedo.png";
    let handle =
actualize_map_handle(floor_texture_path.to_string(), &loaded_assets,
&mut images);
    let floor_material = StandardMaterial {
        base_color_texture: Some(handle.unwrap()),
        ..StandardMaterial::default()
    };

    // --- spawn building ---
    // let building_corners =
building.json_schema.unwrap().corners.unwrap().clone();
    /*for i in 0..(building_corners.len()-1){
        let start = &building_corners[i];
        let end = &building_corners[i+1];
        let start_p= Vec2::new(start.x as f32 / 10.0, start.y
as f32 / 10.0);

```

```

    let end_p= Vec2::new(end.x as f32 / 10.0,  end.y as f32
/ 10.0);
    spawn_wall_segment(
        &mut commands,
        &mut meshes,
        false,
        start_p,
        end_p,
        wall_height,
        0.0,
        wall_width,
        wall_material.clone()
    );

}*/

/*let last = &building_corners[building_corners.len()-1];
let first = &building_corners[0];
spawn_wall_segment(
    &mut commands,
    &mut meshes,
    false,
    last,
    first,
    wall_height,
    wall_width,
    wall_material.clone()
);*/
let mut wall_down = 0.0;
wall_height = 10.0;
let mut ground_down = 0.0;
let mut ground_up = 0.001;

while floor_state.floor_index < floor_state.floors_count {
    let floor =
&floor_state.floors.clone().unwrap()[floor_state.floor_index];
    spawn_floor(
        &mut commands,
        &mut meshes,
        &mut materials,
        wall_height,
        wall_down,
        floor,
        event.borrow_mut()
    );

    let box_geometry = shape::Box::from_corners(
        Vec3::new(0.0, ground_down, 0.0),
        Vec3::new(100.0, ground_up, 100.0),
    );
    let mut mesh = Mesh::from(box_geometry);
    let uv_attr = mesh.attribute_mut(Mesh::ATTRIBUTE_UV_0);

```

```

        if uv_attr.is_some() {
            match uv_attr.unwrap() {
                VertexAttributeValues::Float32x2(ref mut uvs)
=> {
                    for mut uv in uvs {
                        uv[0] *= 10.0;
                        uv[1] *= 10.0;
                    }
                }
            }
        }
    }
}

commands
    .spawn(PbrBundle {
        mesh: meshes.add(mesh.clone()),
        material:
materials.add(floor_material.clone()),
        ..default()
    })
    .with_children(|child| {
        child.spawn(
            Collider::from_bevy_mesh(&mesh,
&ComputedColliderShape::TriMesh).unwrap(),
        );
    });

    floor_state.floor_index = floor_state.floor_index + 1;
    wall_height += 10.0;
    wall_down += 10.0;
    ground_down += 10.0;
    ground_up += 10.0;
}

*has_it_spawned = true;
}

fn spawn_floor(
    commands: &mut Commands,
    meshes: &mut ResMut<Assets<Mesh>>,
    materials: &mut ResMut<Assets<StandardMaterial>>,
    wall_up: f32,
    wall_down: f32,
    floor: &FloorResponse,
    mut event:&mut EventWriter<SpawnWindowEvent>
) {
    let wall_width = 0.5;
    let wall_material = materials.add(Color::GRAY);
    let mut built_walls: Vec<Uuid> = Vec::new();
    //let mut pairs = Vec::new();
    if floor.json_schema.is_none() ||
floor.json_schema.clone().unwrap().rooms.is_none() {

```

```

        return;
    }
    for room in
floor.json_schema.clone().unwrap().rooms.unwrap().iter() {
        if room.wall.is_none() {
            continue;
        }

        let wall = room.wall.clone().unwrap();
        for i in 0..(wall.len()) {
            let current_wall = &wall[i];
            let mut openings: Vec<Opening> = Vec::new();
            info!("current_wall{:?}", current_wall.id);
            let mut doors: Vec<Door> = Vec::new();

            let mut doors: Vec<Window> = Vec::new();

            if let Some(d) = &current_wall.doors {
                info!("Yes, this wall include walls",);
                for door in d {
                    let opening = Opening::new(
                        door.id,
                        door.x_start,
                        door.y_start,
                        door.x_end,
                        door.y_end,
                        wall_down,
                        wall_up - 5.0,
                    );
                    openings.push(opening);
                }
            }
            if let Some(w) = &current_wall.windows {
                for window in w {
                    let opening = Opening::new(
                        window.id,
                        window.x_start,
                        window.y_start,
                        window.x_end,
                        window.y_end,
                        wall_down + window.height_start as f32,
                        wall_down + window.height_end as f32,
                    );
                    let x_glb=2.63092;
                    let y_glb=2.63431;
                    let scale_width = (window.x_end-
window.x_start)as f32/10.0 / x_glb ;
                    let scale_height = (wall_down +
window.height_end as f32-wall_down - window.height_start as f32) /
y_glb ;

```

					ПГУ 1.090301.34.001 ПЗ	Лист
						69
Изм.	Лист	№ докум.	Подпись	Дата		

```

        let
scale=Vec2::new(scale_width,scale_height);
        let
wall_direction_x=current_wall.end_corner.x.clone()-
current_wall.start_corner.x.clone();
        let
wall_direction_y=current_wall.end_corner.y.clone()-
current_wall.start_corner.y.clone();
        let
transform=Vec3::new(((window.x_end+window.x_start)/20.0 as i32)as
f32,wall_down + window.height_start as
f32,((window.y_end+window.y_start)/20.0 as i32) as f32);
        event.send(SpawnWindowEvent{
            position:transform,
            quat:Vec3::new(wall_direction_x as
f32,0.0,wall_direction_y as f32),
            scale:scale
        });
        openings.push(opening);
    }
}

if built_walls.contains(&current_wall.id) {
    info!("YESSSS!!!!!!{:?}", current_wall.id);
    continue;
}

//let mut end_p = Vec2::new((start.x as f32 /
10.0+end.x as f32 / 10.0)/2.0, (start.y as f32/10.0 + end.y as f32 /
10.0)/2.0);
if openings.is_empty() {
    spawn_wall_segment(
        commands,
        meshes,
        true,
        current_wall.start_corner.x.clone(),
        current_wall.end_corner.x.clone(),
        current_wall.start_corner.y.clone(),
        current_wall.end_corner.y.clone(),
        wall_up,
        wall_down,
        wall_width,
        wall_material.clone(),
    );
    built_walls.push(current_wall.id);
} else {
    let mut start_x =
current_wall.start_corner.x.clone();
    let mut start_y =
current_wall.start_corner.y.clone();

```

```

        openings.sort_by(|a, b|
a.x_start.cmp(&b.x_start));

        info!("openings{:?}", openings);
        built_walls.push(current_wall.id);
        for opening in &openings {
            spawn_wall_segment(
                commands,
                meshes,
                true,
                start_x,
                opening.x_start,
                start_y,
                opening.y_start,
                wall_up,
                wall_down,
                wall_width,
                wall_material.clone(),
            );

            spawn_wall_segment(
                commands,
                meshes,
                true,
                opening.x_start,
                opening.x_end,
                opening.y_start,
                opening.y_end,
                opening.height_start as f32,
                wall_down,
                wall_width,
                wall_material.clone(),
            );

            spawn_wall_segment(
                commands,
                meshes,
                true,
                opening.x_start,
                opening.x_end,
                opening.y_start,
                opening.y_end,
                wall_up,
                opening.height_end as f32,
                wall_width,
                wall_material.clone(),
            );
            start_x = opening.x_end;
            start_y = opening.y_end;
        }
        spawn_wall_segment(
            commands,

```

Изм.	Лист	№ докум.	Подпись	Дата

```

        meshes,
        true,
        start_x,
        current_wall.end_corner.x.clone(),
        start_y,
        current_wall.end_corner.y.clone(),
        wall_up,
        wall_down,
        wall_width,
        wall_material.clone(),
    );
    }
}
info!("all_walls{:?}", built_walls);
}

fn current_box_func(sp: CurrentBox, start_p: Vec2, end_p: Vec2)
-> Mesh {
    // suppose Y-up right hand, and camera look from +z to -z

    let vertices = &[
        // Front
        ([start_p.x, sp.min_y, start_p.y], [0., 0., 1.0], [0.,
0.]),
        ([sp.min_x, sp.min_y, sp.min_z], [0., 0., 1.0], [1.0,
0.]),
        ([sp.min_x, sp.max_y, sp.min_z], [0., 0., 1.0], [1.0,
1.0]),
        ([start_p.x, sp.max_y, start_p.y], [0., 0., 1.0], [0.,
1.0]),
        // Back
        ([sp.max_x, sp.max_y, sp.max_z], [0., 0., -1.0], [1.0,
0.]),
        ([end_p.x, sp.max_y, end_p.y], [0., 0., -1.0], [0.,
0.]),
        ([end_p.x, sp.min_y, end_p.y], [0., 0., -1.0], [0.,
1.0]),
        ([sp.max_x, sp.min_y, sp.max_z], [0., 0., -1.0], [1.0,
1.0]),
        // Right
        ([sp.min_x, sp.min_y, sp.min_z], [1.0, 0., 0.], [0.,
0.]),
        ([sp.min_x, sp.max_y, sp.min_z], [1.0, 0., 0.], [1.0,
0.]),
        ([sp.max_x, sp.max_y, sp.max_z], [1.0, 0., 0.], [1.0,
1.0]),
        ([sp.max_x, sp.min_y, sp.max_z], [1.0, 0., 0.], [0.,
1.0]),
        // Left
        ([end_p.x, sp.min_y, end_p.y], [-1.0, 0., 0.], [1.0,
0.]),
    ]
}

```

```

0.]),
    ([end_p.x, sp.max_y, end_p.y], [-1.0, 0., 0.], [0.,
1.0]),
    ([start_p.x, sp.max_y, start_p.y], [-1.0, 0., 0.], [0.,
[1.0, 1.0]]),
    // Top
    ([sp.max_x, sp.max_y, sp.max_z], [0., 1.0, 0.], [1.0,
0.]),
    ([sp.min_x, sp.max_y, sp.min_z], [0., 1.0, 0.], [0.,
0.]),
    ([start_p.x, sp.max_y, start_p.y], [0., 1.0, 0.], [0.,
1.0]),
    ([end_p.x, sp.max_y, end_p.y], [0., 1.0, 0.], [1.0,
1.0]),
    // Bottom
    ([sp.max_x, sp.min_y, sp.max_z], [0., -1.0, 0.], [0.,
0.]),
    ([end_p.x, sp.min_y, end_p.y], [0., -1.0, 0.], [1.0,
0.]),
    ([sp.min_x, sp.min_y, sp.min_z], [0., -1.0, 0.], [1.0,
1.0]),
    ([start_p.x, sp.min_y, start_p.y], [0., -1.0, 0.], [0.,
1.0]),
];

    let positions: Vec<_> = vertices.iter().map(|(p, _, _)|
*p).collect();
    let normals: Vec<_> = vertices.iter().map(|(_, n, _)|
*n).collect();
    let uvs: Vec<_> = vertices.iter().map(|(_, _, uv)|
*uv).collect();

    let indices = Indices::U32(vec![
        0, 1, 2, 2, 3, 0, // front
        4, 5, 6, 6, 7, 4, // back
        8, 9, 10, 10, 11, 8, // right
        12, 13, 14, 14, 15, 12, // left
        16, 17, 18, 18, 19, 16, // top
        20, 21, 22, 22, 23, 20, // bottom
    ]);

    let mut mesh = Mesh::new(
        PrimitiveTopology::TriangleList,
        RenderAssetUsages::MAIN_WORLD |
RenderAssetUsages::RENDER_WORLD,
    );
    mesh.insert_attribute(Mesh::ATTRIBUTE_POSITION, positions);
    mesh.insert_attribute(Mesh::ATTRIBUTE_NORMAL, normals);
    mesh.insert_attribute(Mesh::ATTRIBUTE_UV_0, uvs);
    mesh.insert_indices(indices);
    mesh

```

```

}
fn spawn_wall_segment(
  commands: &mut Commands,
  meshes: &mut ResMut<Assets<Mesh>>,

  is_floor_wall: bool,
  start_corner_x: i32,
  end_corner_x: i32,
  start_corner_y: i32,
  end_corner_y: i32,
  wall_up: f32,
  wall_down: f32,
  wall_width: f32,
  material_handle: Handle<StandardMaterial>,
) {
  info!("wall_down{:?}", wall_down);
  info!("wall_up{:?}", wall_up);
  let start_p = Vec2::new(start_corner_x as f32 / 10.0,
start_corner_y as f32 / 10.0);
  info!("start_spawn{:?}", start_p);
  let end_p = Vec2::new(end_corner_x as f32 / 10.0,
end_corner_y as f32 / 10.0);
  info!("end_spawn{:?}", end_p);

  let vector = end_p.sub(start_p);
  info!("vector{:?}", vector);
  let norm_perp = vector.perp().normalize(); //perp 1
  let norm_perp_neg = norm_perp.neg(); //perp -1
  info!("norm_perp{:?}/norm_perp_neg{:?}", norm_perp,
norm_perp_neg);
  let w1 = norm_perp.clamp_length(wall_up / 2.0, wall_up /
2.0);
  let w2 = norm_perp_neg.clamp_length(wall_width / 2.0,
wall_width / 2.0);
  info!("w1{:?}/w2{:?}", w1, w2);
  let w1_p = Vec2::new(w2.x + start_p.x, w2.y + start_p.y);
  let w2_p = Vec2::new(w2.x + end_p.x, w2.y + end_p.y);
  info!("w1_p{:?}/w2_p{:?}", w1_p, w2_p);
  let vertices = vec![
    Vec3::new(w1_p.x, wall_down, w1_p.y),
    Vec3::new(w1_p.x, wall_up, w1_p.y),
    Vec3::new(w2_p.x, wall_down, w1_p.y),
    Vec3::new(w2_p.x, wall_up, w1_p.y),
    Vec3::new(w2_p.x, wall_down, w2_p.y),
    Vec3::new(w2_p.x, wall_up, w2_p.y),
    Vec3::new(w1_p.x, wall_down, w2_p.y),
    Vec3::new(w1_p.x, wall_up, w2_p.y),
  ];
  let indexes = vec![
    [0 as u32, 1 as u32, 2 as u32],
    [2 as u32, 3 as u32, 1 as u32],
    [2 as u32, 4 as u32, 5 as u32],
  ]
}

```

```

        [5 as u32, 3 as u32, 2 as u32],
        [4 as u32, 6 as u32, 7 as u32],
        [7 as u32, 5 as u32, 4 as u32],
        [0 as u32, 6 as u32, 7 as u32],
        [7 as u32, 1 as u32, 0 as u32],
    ];
    let current_box = CurrentBox {
        min_x: w1_p.x,
        max_x: end_p.x,

        min_y: wall_down,
        max_y: wall_up,

        min_z: w1_p.y,
        max_z: end_p.y,
    };
    if is_floor_wall {
        commands
            .spawn(PbrBundle {
                mesh: meshes.add(current_box_func(current_box,
start_p, w2_p)),
                material: material_handle,
                ..default()
            })
            .with_children(|child| {
                child.spawn(Collider::trimesh(vertices,
indexes));
            })
            .insert(FloorWallPart);
    } else {
        commands
            .spawn(PbrBundle {
                mesh: meshes.add(current_box_func(current_box,
start_p, w2_p)),
                material: material_handle,
                ..default()
            })
            .with_children(|child| {
                child.spawn(Collider::trimesh(vertices,
indexes));
            })
    }

}

```

Файл asset_load_plugin.rs

```

use crate::SceneInitDataRes;
use appslib::enums::loading_algorithm::LoadingAlgorithm;
use bevy::asset::LoadState;

```

```

use bevy::asset::{UntypedAssetId, UntypedHandle};
use bevy::prelude::*;
use uuid::Uuid;

#[derive(Debug, Clone, PartialEq)]
pub enum AssetType {
    Mesh,
    TextureMap,
}

#[derive(Debug, Clone)]
pub struct AssetInQueue {
    pub asset_type: AssetType,
    pub id: Uuid,
    pub path: String,
    pub has_loaded: bool,
}

#[derive(Debug, Clone)]
pub struct AssetDescriptor {
    pub id: Uuid,
    pub name: Option<String>,
    pub path: String,
    pub asset_type: AssetType,
    pub handle: UntypedHandle,
}

//resource with information about loaded and loaded assets
#[derive(Resource, Default)]
pub struct AssetsLoadedRes(pub Vec<AssetDescriptor>);

#[derive(Resource, Default)]
pub struct AssetsLoadQueue(pub Vec<AssetInQueue>);

//event is triggered when all assets are loaded
#[derive(Event)]
pub struct AssetsLoadedEvent();

pub struct AssetsLoadPlugin {
    pub asset_load_algorithm: Option<LoadingAlgorithm>,
}

impl Plugin for AssetsLoadPlugin {
    fn build(&self, app: &mut App) {
        app.init_resource::()
            .add_event::();
        if self.asset_load_algorithm.is_some()
            && (self.asset_load_algorithm.clone().unwrap() ==
LoadingAlgorithm::Sequential)
        {
            //sequential load
            app.init_resource::()

```

```

        .add_systems(Startup, setup_sequential)
        .add_systems(Update, sequential_load);
    } else {
        //parallel load
        app.add_systems(Startup, setup_parallel)
            .add_systems(Update,
check_parallel_assets_ready);
    }
}

fn setup_sequential(scene_data: Res<SceneInitDataRes>, mut
to_load: ResMut<AssetsLoadQueue>) {
    to_load.0 = vec![];
    if scene_data.scene_response.clone().data.is_none() {
        return;
    }

    for model_data in scene_data
        .scene_response
        .data
        .as_ref()
        .unwrap()
        .models
        .as_ref()
        .unwrap()
        .iter()
    {
        let id = model_data.id;
        let path = model_data.clone().model_path.clone();
        to_load.0.push(AssetInQueue {
            id,
            asset_type: AssetType::Mesh,
            path,
            has_loaded: false,
        });
    }

    for material in scene_data
        .scene_response
        .data
        .as_ref()
        .unwrap()
        .materials
        .as_ref()
        .unwrap()
        .iter()
    {
        //diffuse
        if material.clone().texture_map_path.is_some() {
            to_load.0.push(AssetInQueue {
                id: Uuid::nil(),

```

```

        asset_type: AssetType::TextureMap,
        path:
material.clone().texture_map_path.unwrap().clone(),
        has_loaded: false,
    });
}
//normal
if material.clone().normal_map_path.is_some() {
    to_load.0.push(AssetInQueue {
        id: Uuid::nil(),
        asset_type: AssetType::TextureMap,
        path:
material.clone().normal_map_path.unwrap().clone(),
        has_loaded: false,
    });
}
//emissive
if material.clone().emissive_map_path.is_some() {
    to_load.0.push(AssetInQueue {
        id: Uuid::nil(),
        asset_type: AssetType::TextureMap,
        path:
material.clone().emissive_map_path.unwrap().clone(),
        has_loaded: false,
    });
}
}

fn sequential_load(
    server: Res<AssetServer>,
    mut to_load: ResMut<AssetsLoadQueue>,
    mut loaded: ResMut<AssetsLoadedRes>,

    mut loading_now: Local<Option<AssetDescriptor>>,
    mut event: EventWriter<AssetsLoadedEvent>,
    mut has_triggered: Local<bool>,
) {
    if *has_triggered {
        return;
    }

    if (*loading_now).is_some() {
        let status =
server.get_load_state(loading_now.clone().unwrap().handle.id());
        if status.is_some() {
            let status = status.unwrap().clone();
            if status == LoadState::Failed {
                info!("Error. Failed : {:?} ",
loading_now.clone().unwrap().path);
                *loading_now = None;
            }
        }
    }
}

```

```

        if status == LoadState::Loaded {
            let copy = loading_now.clone().unwrap();
            loaded.0.push(AssetDescriptor {
                id: copy.id,
                name: copy.name,
                path: copy.path,
                asset_type: copy.asset_type,
                handle: copy.handle,
            });
            *loading_now = None;
            let mut all_ready = true;
            'asset_cycle: for asset in to_load.0.iter() {
                // info!("Check status for {:?} status
{:?}", asset.clone().path, asset.clone().has_loaded);
                if !asset.has_loaded {
                    all_ready = false;
                    break 'asset_cycle;
                }
            }
            if all_ready {
                event.send(AssetsLoadedEvent());
                *has_triggered = true;
            }
        }
    } else {
        'asset_cycle2: for asset in to_load.0.iter_mut() {
            if !asset.has_loaded {
                // Загрузка asset_handle в зависимости от типа
                let handle = match asset.asset_type {
                    AssetType::Mesh => {
                        let path = asset.path.clone();
                        let path_pat =
Box::leak(path.clone().into_boxed_str());
                        let path_str = &*path_pat;
                        let handle: Handle<Mesh> =
server.load(path_str);
                        handle.untyped()
                    }
                    AssetType::TextureMap => {
                        let path = asset.path.clone();
                        let path_pat =
Box::leak(path.clone().into_boxed_str());
                        let _path_str = &*path_pat;
                        let handle: Handle<Image> =
server.load(path);
                        handle.untyped()
                    }
                };
                *loading_now = Some(AssetDescriptor {

```

```

        id: asset.id.clone(),
        name: None,
        path: asset.path.clone(),
        asset_type: asset.asset_type.clone(),
        handle: handle,
    });

    asset.has_loaded = true;
    break 'asset_cycle2;
    }
    }
}

fn setup_parallel(
    server: Res<AssetServer>,
    scene_data: Res<SceneInitDataRes>,
    mut loading: ResMut<AssetsLoadedRes>, //through this
resource end-of-load tracking
) {
    if scene_data.scene_response.clone().data.is_none() {
        return;
    }

    for model_data in scene_data
        .scene_response
        .data
        .as_ref()
        .unwrap()
        .models
        .as_ref()
        .unwrap()
        .iter()
    {
        //TODO format check here
        let id = model_data.id;
        let path = model_data.clone().model_path.clone();
        let name = model_data.clone().name.clone();
        let mesh_handle: Handle<Mesh> = server.load(format!(
            "{}#Mesh0/Primitive0",
            model_data.model_path.clone()
        ));
        loading.0.push(AssetDescriptor {
            id,
            name,
            path,
            asset_type: AssetType::Mesh,
            handle: mesh_handle.untyped(),
        });
    }

    for material in scene_data

```

```

        .scene_response
        .data
        .as_ref()
        .unwrap()
        .materials
        .as_ref()
        .unwrap()
        .iter()
    {
        //diffuse
        if material.clone().texture_map_path.is_some() {
            load_map(

material.clone().texture_map_path.unwrap().clone(),
                    &server,
                    &mut loading,
                );
        }
        //normal
        if material.clone().normal_map_path.is_some() {
            load_map(

material.clone().normal_map_path.unwrap().clone(),
                    &server,
                    &mut loading,
                );
        }
        //emissive
        if material.clone().emissive_map_path.is_some() {
            load_map(

material.clone().emissive_map_path.unwrap().clone(),
                    &server,
                    &mut loading,
                );
        }
    }
}

fn load_map(path: String, server: &AssetServer, loading: &mut
ResMut<AssetsLoadedRes>) {
    let path_pat = Box::leak(path.clone().into_boxed_str());
    let path_str = &*path_pat;
    let image_handle: Handle<Image> = server.load(path_str);
    loading.0.push(AssetDescriptor {
        id: Uuid::nil(),
        name: None,
        path,
        asset_type: AssetType::TextureMap,
        handle: image_handle.untyped(),
    })
}

```

```

//the method checks the state of all assets. When everyone is
loaded, triggers the AssetsLoadedEvent
fn check_parallel_assets_ready(
    server: Res<AssetServer>,
    loading: Res<AssetsLoadedRes>,
    _meshes: Res<Assets<Mesh>>,
    mut event: EventWriter<AssetsLoadedEvent>,
    mut has_triggered: Local<bool>, //preventing multiple
sending of the AssetsLoadedEvent event + loop optimization
) {
    if *has_triggered {
        return;
    }
    let mut test_vec: Vec<UntypedAssetId> = Vec::new();
    for el in loading.0.iter() {
        let handler_cp = el.clone().handle.clone();
        test_vec.push(handler_cp.id().clone());

        let status =
server.get_load_state(handler_cp.clone().id());
        if status.is_some() {
            let status = status.unwrap().clone();
            let path = el.handle.path();
            if status == LoadState::Failed {
                info!("Error. Failed this: {:?}" , path)
            }
        }
    }

    if is_all_ready(server, test_vec) {
        event.send(AssetsLoadedEvent());
        *has_triggered = true;
    }

    pub fn is_all_ready(
        server: Res<AssetServer>,
        handles: impl IntoIterator<Item = UntypedAssetId>,
    ) -> bool {
        for handle_id in handles {
            match server.get_load_state(handle_id) {
                Some(LoadState::Loaded) => {
                    info!("Asset succes loaded");
                    continue;
                }
                Some(LoadState::Loading) => {
                    info!("Asset loading");
                    return false;
                }
                Some(LoadState::Failed) => {
                    info!("Asset failed {:?}",
server.get_path(handle_id));
                    return false;
                }
            }
        }
    }
}

```

```

        }
        Some(LoadState::NotLoaded) => {
            info!("Asset not loaded {:?}",
server.get_path(handle_id));
            return false;
        }
        None => {
            info!("???");
            return false;
        }
    }
}
true
}
}
}

```

Файл instance_spawner_plugin.rs

```

use appslib::dto::scene_response::SceneResponse;
use appslib::enums::physics_b_type::PhysicsBType::*;
use appslib::enums::physics_shape::PhysicsShape::*;
use appslib::jsonb::instance::Instance;
use appslib::jsonb::scene_data::SceneData;
use bevy::pbr::PbrBundle;
use bevy::prelude::*;
use bevy::prelude::{App, Plugin, Res, ResMut};
use bevy::render::mesh::VertexAttributeValues;
use bevy::render::texture::ImageSamplerDescriptor;
use bevy::render::texture::{ImageAddressMode, ImageSampler};
use bevy_rapier3d::na::Matrix3;
use bevy_rapier3d::prelude::*;

use crate::{ang_to_rad, GameObject, SceneInitDataRes};
use uuid::Uuid;

use crate::plugin::assets_load_plugin::AssetsLoadedEvent;
use crate::plugin::assets_load_plugin::AssetsLoadedRes;
use crate::scene_lib::floor::floor_plugin::SpawnWindowEvent;
//this resource is set to =true when all assets are loaded and
all instances are spawned
#[derive(Resource)]
pub struct IsSceneStarted(pub bool);

pub struct InstanceSpawnerPlugin;

impl Plugin for InstanceSpawnerPlugin {
    fn build(&self, app: &mut App) {
        app.insert_resource(IsSceneStarted(false))
            .add_systems(Update, on_assets_loaded)
    }
}

```

```

        .add_systems(Update, spawn_window);

    }

}

fn on_assets_loaded(
    mut commands: Commands,
    asset_server: Res<AssetServer>,
    materials: ResMut<Assets<StandardMaterial>>,
    mut meshes: ResMut<Assets<Mesh>>,
    images: ResMut<Assets<Image>>,
    event: EventReader<AssetsLoadedEvent>, //listen to this
event from the plugin AssetsLoadPlugin
    is_scene_started: ResMut<IsSceneStarted>,
    scene_data: Res<SceneInitDataRes>,
    loaded_assets: Res<AssetsLoadedRes>,
) {
    if !event.is_empty() && !is_scene_started.0 {
        // info!("Scene init... meshes look through
{:?}]",meshes.len());
        //prepare meshes
        update_meshes_uv(
            &mut meshes,
            scene_data.scene_response.clone(),
            &asset_server,
            &loaded_assets,
        );

        //spawn
        spawn_instances(
            &mut commands,
            scene_data,
            loaded_assets,
            materials,
            meshes,
            images,
            asset_server,
        );
        commands.insert_resource(IsSceneStarted(true));
    }
}

fn spawn_instances(
    commands: &mut Commands,
    scene_data: Res<SceneInitDataRes>,
    loaded_assets: Res<AssetsLoadedRes>,
    mut materials: ResMut<Assets<StandardMaterial>>,
    mut meshes: ResMut<Assets<Mesh>>,
    mut images: ResMut<Assets<Image>>,
    asset_server: Res<AssetServer>,
) {
    if scene_data.scene_response.clone().data.is_none()

```

```

        || scene_data
            .scene_response
            .clone()
            .data
            .unwrap()
            .instances
            .is_none()
    {
        return;
    }
    //all object coordinates will be multiplied by this matrix.
    //this is necessary to use a different basis of a linear
space with different dimension [3] - XYZ, ZYX, (-Z)YX, (-Z)Y(2X) and
so on.
    //this matrix by default stores coordinates in the common
XYZ basis
    let mut scene_space_basis: Matrix3<f32> =
        Matrix3::new(1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0,
1.0);

    let scene_data =
scene_data.scene_response.clone().data.unwrap();
    if scene_data.space_basis.is_some() {
        let mat =
scene_data.clone().space_basis.unwrap().clone();
        scene_space_basis = Matrix3::new(
            mat.m11, mat.m12, mat.m13, mat.m21, mat.m22,
mat.m23, mat.m31, mat.m32, mat.m33,
        );
    };

    for instance in scene_data.clone().instances.unwrap() {
        if instance.model.is_none() {
            continue;
        };

        let id =
Uuid::parse_str(instance.model.clone().unwrap().as_str()).unwrap();
        let model_asset_descr =
loaded_assets.0.iter().find(|&asset| asset.id == id);
        if model_asset_descr.is_none() {
            continue;
        }

        let material = materials.add(build_material(
            instance.clone(),
            scene_data.clone(),
            &loaded_assets,
            &mut images,
        ));

        let mut pos: Vec3 = Vec3 {

```

```

        x: 0.0,
        y: 0.0,
        z: 0.0,
    };
    let mut scale: f32 = 1.0;
    let mut quat: Quat = Quat::default();

    if instance.clone().position.is_some() {
        pos = apply_basis(
            Vec3 {
                x:
instance.clone().position.unwrap().clone().x as f32,
                y:
instance.clone().position.unwrap().clone().y as f32,
                z:
instance.clone().position.unwrap().clone().z as f32,
            },
            scene_space_basis,
        )
    }

    if instance.scale.is_some() {
        scale = instance.scale.unwrap().clone()
    }

    if instance.quaternion.is_some() {
        quat = Quat::from_xyzw(
            instance.clone().quaternion.unwrap().clone().x
as f32,
            instance.clone().quaternion.unwrap().clone().y
as f32,
            instance.clone().quaternion.unwrap().clone().z
as f32,
            instance.clone().quaternion.unwrap().clone().w
as f32,
        )
    }

    if instance.euler.is_some() {
        let x1 =
ang_to_rad(instance.clone().euler.unwrap().clone().x as f32);
        let y1 =
ang_to_rad(instance.clone().euler.unwrap().clone().y as f32);
        let z1 =
ang_to_rad(instance.clone().euler.unwrap().clone().z as f32);
        quat = Quat::from_euler(EulerRot::XYZ, x1, y1, z1);
        //базис вращения Y-up
        //info!("Euler rad {:?} ", quat.clone());
    }

```

```

        let mut handle =
model_asset_descr.unwrap().handle.clone().typed::<Mesh>();
        handle = actualize_mesh_handle(handle, &asset_server,
&meshes);
        let mesh_mod = meshes.get_mut(&handle);
        if mesh_mod.is_some() {
            mesh_mod.unwrap().generate_tangents();
        }

// info!("Quat {:?}", quat.clone());
let mut transform = Transform {
    translation: pos,
    rotation: quat.clone(),
    scale: Vec3 {
        x: scale,
        y: scale,
        z: scale,
    },
};
// let mat = transform.compute_matrix();
transform.rotate_y(ang_to_rad(-90.0));

//-----SPAWN OBJECT
if instance.physics.is_some() {
    let bodyType =
instance.clone().physics.unwrap().clone().b_type.clone();

    let bundle = match bodyType {
        Fixed => RigidBody::Fixed,
        Dynamic => RigidBody::Dynamic,
        KinematicPositionBased =>
RigidBody::KinematicPositionBased,
        KinematicVelocityBased =>
RigidBody::KinematicVelocityBased,
    };

    let offset = instance
        .clone()
        .physics
        .unwrap()
        .clone()
        .offset
        .unwrap()
        .clone();
    let shape =
instance.clone().physics.unwrap().clone().shape.clone();
    let mass = instance
        .clone()
        .physics
        .unwrap()
        .clone()
        .mass

```

```

        .unwrap()
        .clone();
    let rotation;
    if
instance.clone().physics.unwrap().clone().euler.is_some() {
        let euler = instance
            .clone()
            .physics
            .unwrap()
            .clone()
            .euler
            .unwrap()
            .clone();
        rotation = Quat::from_euler(
            EulerRot::default(),
            euler.x as f32,
            euler.y as f32,
            euler.z as f32,
        );
    } else {
        let quaternion = instance
            .clone()
            .physics
            .unwrap()
            .clone()
            .quaternion
            .unwrap()
            .clone();
        rotation = Quat::from_xyzw(
            quaternion.x as f32,
            quaternion.y as f32,
            quaternion.z as f32,
            quaternion.w as f32,
        );
    }

    let object = commands
        .spawn(PbrBundle {
            mesh: handle.clone(),
            material: material.clone(),
            transform: transform.clone(),
            ..PbrBundle::default()
        })
        .insert(GameObject {
            id_str: instance.id.to_string(),
        })
        .insert(bundle)
        // .insert(Velocity {
        //     linvel: Vec3::new(1.0, 2.0, 3.0),
        //     angvel: Vec3::new(0.2, 0.0, 0.0),
        // })
        .insert(GravityScale(1.0))

```

```

        .insert(Sleeping::disabled())
        // .insert(Ccd::enabled())
        .insert(ColliderMassProperties::Density(mass))
        .with_children(|child| {
            match shape {
                Ball => {
                    child
                        .spawn(Collider::ball(0.16))
                }
            }
        })
        .insert(AdditionalMassProperties::Mass(1.0))
        .insert(TransformBundle::from(
            Transform::from_xyz(
                offset.x as f32,
                offset.y as f32,
                offset.z as f32,
            )
            .with_rotation(rotation),
        ));
    }
    Cuboid => {
        child
            .spawn(Collider::cuboid(0.13,
0.13, 0.13))
    }
    .insert(AdditionalMassProperties::Mass(1.0))
    .insert(TransformBundle::from(
        Transform::from_xyz(
            offset.x as f32,
            offset.y as f32,
            offset.z as f32,
        )
        .with_rotation(rotation),
    ));
}

// Position the collider relative to the
rigid-body.
    })
    .id();

//axes lock
if instance
    .clone()
    .physics
    .unwrap()
    .clone()
    .axes_lock
    .is_some()
{
    let axesLock = instance
        .clone()

```

```

        .physics
        .unwrap()
        .clone()
        .axes_lock
        .unwrap()
        .clone();
        let axes_lock = match axesLock.as_str() {
            "rotationLockedX" =>
LockedAxes::ROTATION_LOCKED_X,
            "rotationLockedY" =>
LockedAxes::ROTATION_LOCKED_Y,
            "rotationLockedZ" =>
LockedAxes::ROTATION_LOCKED_Z,
            "rotationLocked" =>
LockedAxes::ROTATION_LOCKED,
            &_ => continue,
        };
        commands.entity(object).insert(axes_lock);
    }

    // .insert(Ccd::enabled())

    // .insert(Collider::cuboid(0.5, 0.5, 0.5));
    // Position the collider relative to the rigid-
body.

    // commands.spawn(PbrBundle {
    //     mesh: handle.clone(),
    //     material: material.clone(),
    //     transform: transform.clone(),
    //     ..PbrBundle::default()
    // })
    // .insert(GameObject {
    //     id_str: instance.id.to_string()
    // }).insert(RigidBody::Fixed)
    // // .insert(Velocity {
    // //     linvel: Vec3::new(1.0, 2.0, 3.0),
    // //     angvel: Vec3::new(0.2, 0.0, 0.0),
    // // })
    // .insert(GravityScale(0.1))
    // .insert(Sleeping::disabled())
    // // .insert(Ccd::enabled())
    // .with_children(|children| {
    //
children.spawn(TransformBundle::from(Transform::from_xyz(0.0, 0.15,
0.0)))

        // .insert(Collider::ball(0.8));
        // // Position the collider relative to the
rigid-body.

        // });
    } else {
        commands

```

```

        .spawn(PbrBundle {
            mesh: handle.clone(),
            material: material.clone(),
            transform: transform.clone(),
            ..PbrBundle::default()
        })
        .insert(GameObject {
            id_str: instance.id.to_string(),
        });
    }
    //----- OBJECT HAS BEEN SPAWNED
}

fn apply_basis(vec: Vec3, basis: Matrix3<f32>) -> Vec3 {
    Vec3::new(
        vec.x * basis.m11 + vec.y * basis.m12 + vec.z *
basis.m13,
        vec.x * basis.m21 + vec.y * basis.m22 + vec.z *
basis.m23,
        vec.x * basis.m31 + vec.y * basis.m32 + vec.z *
basis.m33,
    )
}

fn build_material(
    instance: Instance,
    scene_data: SceneData,
    loaded_assets: &Res<AssetsLoadedRes>,
    images: &mut ResMut<Assets<Image>>,
) -> StandardMaterial {
    //default material
    let default_color = Color::rgb(0.75, 0.75, 0.75);
    let mut material = StandardMaterial {
        base_color: default_color,
        ..StandardMaterial::default()
    };

    if instance.material.is_none() ||
scene_data.materials.is_none() {
        //there is no custom material, we use the standard
        return material;
    };

    let material_data_vec =
scene_data.materials.clone().unwrap();
    let id =
Uuid::parse_str(instance.material.clone().unwrap().as_str()).unwrap(
);
    let material_data = material_data_vec.iter().find(|&mat|
mat.id == id);

```

```

        if material_data.is_none() {
            return material;
        }

        let mut diffuse_color: Color = default_color.clone();
//TODO override from JSON parse from string
        let mut diffuse_map: Option<Handle<Image>> = Option::None;
        let mut normal_map: Option<Handle<Image>> = Option::None;
        let mut emissive_map: Option<Handle<Image>> = Option::None;
        let mut emissive_color: Color = Color::BLACK;
        let mut metallic: f32 = 1.00;
        let mut roughness: f32 = 1.00;
        let mut alpha_mode: AlphaMode = AlphaMode::Opaque;

        // MAPS
        if
material_data.unwrap().clone().texture_map_path.is_some() {
            diffuse_map = actualize_map_handle(

material_data.unwrap().clone().texture_map_path.unwrap(), //
"generated_texture".parse().unwrap(),
                loaded_assets,
                images,
            )
        }

        if material_data.unwrap().clone().normal_map_path.is_some()
{
            normal_map = actualize_map_handle(

material_data.unwrap().clone().normal_map_path.unwrap(),
                loaded_assets,
                images,
            )
        }

        if
material_data.unwrap().clone().emissive_map_path.is_some() {
            emissive_map = actualize_map_handle(

material_data.unwrap().clone().emissive_map_path.unwrap(),
                loaded_assets,
                images,
            );
            emissive_color = Color::WHITE;
        }
        //-----

        if material_data.unwrap().clone().metalness.is_some() {
            metallic =
material_data.unwrap().clone().metalness.unwrap();
        }

```

					ПГУ 1.090301.34.001 ПЗ	Лист
						92
Изм.	Лист	№ докум.	Подпись	Дата		

```

        if material_data.unwrap().clone().roughness.is_some() {
            roughness =
material_data.unwrap().clone().roughness.unwrap();
        }

        if material_data.unwrap().clone().opacity.is_some() {
diffuse_color.set_a(material_data.unwrap().clone().opacity.unwrap())
;
            alpha_mode = AlphaMode::Blend;
        }

        //there is no alpha map in the shader at all...
        material = StandardMaterial {
            alpha_mode: alpha_mode,
            base_color: diffuse_color,
            base_color_texture: diffuse_map,
            normal_map_texture: normal_map,
            emissive_texture: emissive_map,
            emissive: emissive_color,
            metallic: metallic,
            perceptual_roughness: roughness,
            ..StandardMaterial::default()
        };

        return material;
    }

    pub fn actualize_map_handle(
        map_path: String,
        loaded_assets: &Res<AssetsLoadedRes>,
        images: &mut ResMut<Assets<Image>>,
    ) -> Option<Handle<Image>> {
        let map_asset_descr = loaded_assets.0.iter().find(|&asset|
asset.path == map_path);
        return if map_asset_descr.is_some() {
            let handle =
map_asset_descr.unwrap().handle.clone().typed::<Image>();
            let image = images.get_mut(&handle);
            if image.is_some() {
                image.unwrap().sampler =
ImageSampler::Descriptor(ImageSamplerDescriptor {
                    address_mode_u: ImageAddressMode::Repeat,
                    address_mode_v: ImageAddressMode::Repeat,
                    address_mode_w: ImageAddressMode::Repeat,
                    ..Default::default()
                });
            }
            //
            image.as_ref().unwrap().texture_view_descriptor.as_ref().unwrap().ba
se_mip_level = 2;
        }
    }

```

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.34.001 ПЗ

Лист

93

```

        Option::from(handle)
    } else {
        Option::None
    };
}

//----- Public functions for reuse at runtime.
// Loading at runtime, looking for assets in resources to spawn
// (loaded from JSON) -----
// Handle received from the asset manager are temporary. To
// actualize the actual handle, use the following function:
pub fn actualize_mesh_handle(
    raw_handle: Handle<Mesh>,
    asset_server: &Res<AssetServer>,
    meshes: &ResMut<Assets<Mesh>>,
) -> Handle<Mesh> {
    let mut handle = raw_handle.clone();
    let path = handle.path();
    if path.is_some() {
        let path_cp = path.unwrap().clone();
        let path_str = path_cp.path().to_str();
        // search for a mesh in the scene resources (in the
        meshes resource).
        //this logic is needed because the asset_server moves
        the loaded assets to the appropriate resource in this case in
        ResMut<Assets<Mesh>>.
        'mesh_cycle: for (handle_id, _mesh) in meshes.iter() {
            let asset_of_mesh =
            asset_server.get_id_handle(handle_id);
            if asset_of_mesh.is_some() {
                let asset_of_mesh =
            asset_of_mesh.unwrap().clone();
                let path2 = asset_of_mesh.path();
                if path2.is_some() {
                    // info!("{:?} eq {:?}", handle.id,
            asset_of_mesh.id);

                    let path2 = path2.unwrap().clone();
                    let x = path_cp.source();
                    let y = path2.source();

                    if x == y {
                        handle = asset_of_mesh;
                    }

                    let path_str2 = path2.path().to_str();
                    if path_str.unwrap() == path_str2.unwrap()
                {
                    break 'mesh_cycle;
                }
            }
        }
    }
}

```

```

    }
    return handle;
}

fn update_meshes_uv(
    meshes: &mut ResMut<Assets<Mesh>>,
    scene_data: SceneResponse,
    asset_server: &Res<AssetServer>,
    loaded_assets: &Res<AssetsLoadedRes>,
) {
    if scene_data.data.is_none() ||
scene_data.data.clone().unwrap().models.is_none() {
        return;
    }

    for model in
scene_data.data.clone().unwrap().models.unwrap().iter() {
        if model.uv_tiling.is_some() {
            let model_asset_descr =
loaded_assets.0.iter().find(|&asset| asset.id == model.id);
            if model_asset_descr.is_none() {
                continue;
            }
            let mut handle =
model_asset_descr.unwrap().handle.clone().typed::<Mesh>();
            handle = actualize_mesh_handle(handle,
&asset_server, &meshes);
            let mesh = meshes.get_mut(&handle);
            if mesh.is_none() {
                continue;
            }
            let uv_attr =
mesh.unwrap().attribute_mut(Mesh::ATTRIBUTE_UV_0);
            if uv_attr.is_some() {
                match uv_attr.unwrap() {
                    VertexAttributeValues::Float32x2(ref mut
uvs) => {
                        for uv in uvs {
                            uv[0] *=
model.uv_tiling.clone().unwrap().x as f32;
                            uv[1] *=
model.uv_tiling.clone().unwrap().y as f32;
                        }
                    }
                    _ => {}
                }
            }
        }
    }
}

fn spawn_window(mut commands: Commands,

```

```

        loaded_assets: Res<AssetsLoadedRes>,
        mut materials:
ResMut<Assets<StandardMaterial>>,
        mut meshes: ResMut<Assets<Mesh>>,
        asset_server: Res<AssetServer>,
        mut event:EventReader<SpawnWindowEvent>,
        scene_data: Res<SceneInitDataRes>,
        mut images: ResMut<Assets<Image>>
    ) {
        for my_event in event.read() {
            let scene_data =
scene_data.scene_response.clone().data.unwrap();
            let path_window = "3b2f8af8-bad3-43aa-9eb2-
b39b4922c207";
            for instance in scene_data.clone().instances.unwrap() {
                if instance.model.clone().unwrap() != path_window {
                    continue;
                };
                info!("EVENT RECEIVE!");
                let id = Uuid::parse_str(path_window).unwrap();
                let model_asset_descr =
loaded_assets.0.iter().find(|&asset| asset.id == id);

                let material = materials.add(build_material(
                    instance.clone(),
                    scene_data.clone(),
                    &loaded_assets,
                    &mut images,
                ));
                let mut pos: Vec3 = Vec3 {
                    x: 0.0,
                    y: 0.0,
                    z: 0.0,
                };
                let mut scale: f32 = 1.0;
                let mut handle =
model_asset_descr.unwrap().handle.clone().typed::<Mesh>();
                handle = actualize_mesh_handle(handle,
&asset_server, &meshes);
                let mesh_mod = meshes.get_mut(&handle);
                if mesh_mod.is_some() {
                    mesh_mod.unwrap().generate_tangents();
                }

                let mut quat= Quat::from_rotation_arc(Vec3::new(0.0,
0.0, 1.0), my_event.quat.normalize());
                info!("my_event.quat {:?}", my_event.quat);
                info!("my_event.scale {:?}", my_event.scale);

                info!("quat {:?}", quat);
                let mut transform = Transform {

```

```

        translation: my_event.position,
        rotation: quat.clone(),
        scale: Vec3 {
            x: scale,
            y: my_event.scale.y,
            z: my_event.scale.x,
        },
    };

    commands
        .spawn(PbrBundle {
            mesh: handle.clone(),
            material: material.clone(),
            transform: transform.clone(),
            ..PbrBundle::default()
        })
        .insert(GameObject {
            id_str: instance.id.to_string()
        });
    }
}

```

Приложение В

(Обязательное)

Интерактивный 3D-тренажёр «Эвакуация из здания»

Презентация

					ПГУ 1.090301.34.001 ПЗ	Лист
						98
Изм.	Лист	№ докум.	Подпись	Дата		

Выпускная квалификационная работа на тему «Интерактивный 3D- тренажёр «Эвакуация из здания»»

Автор работы:

студент гр. 20ВВП2 Пантюшов Егор Игоревич

Научный руководитель:

д.т.н., профессор Зинкин Сергей Александрович

Пенза 2024

1

Цель работы

Основной целью данной работы является разработать интерактивный 3D-тренажёр, позволяющий пользователю пройти симуляцию эвакуации из здания при пожаре. Приложение должно создавать модель здания структуры, задаваемой самим пользователем. Также приложение должно позволять пользователю осуществлять процесс эвакуации от первого лица и поддерживать запуск в браузере.

2

					ПГУ 1.090301.34.001 ПЗ	Лист
						99
Изм.	Лист	№ докум.	Подпись	Дата		

Актуальность работы

Интерактивные 3D-тренажёры являются незаменимым продуктом в современном мире. Традиционные методы обучения эвакуационным процедурам, такие как лекции и практические занятия, не всегда позволяют в полной мере подготовить людей к реальным чрезвычайным ситуациям. Интерактивные 3D-тренажёры предлагают инновационный подход к обучению, который позволяет создать реалистичную и интерактивную среду. Это позволяет людям развивать навыки эвакуации в безопасной среде, учиться на своих ошибках без риска для жизни и здоровья и повышать свою уверенность и готовность к действиям в случае чрезвычайной ситуации.

3

Постановка задачи

Интерактивный 3D-тренажёр «Эвакуация из здания» должен иметь следующий функционал:

- создание трёхмерной модели здания с учетом параметров его конструкции, этажности и расположения выходов, окон и проёмов, заданных пользователем;
- возможность управления персонажем в трехмерном пространстве;
- возможность осуществлять различные действия, например прыжки, приседания;
- реализовывать эффект возгорания в определённых участках здания;
- поддержка запуска из браузера.

4

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.34.001 ПЗ

Лист

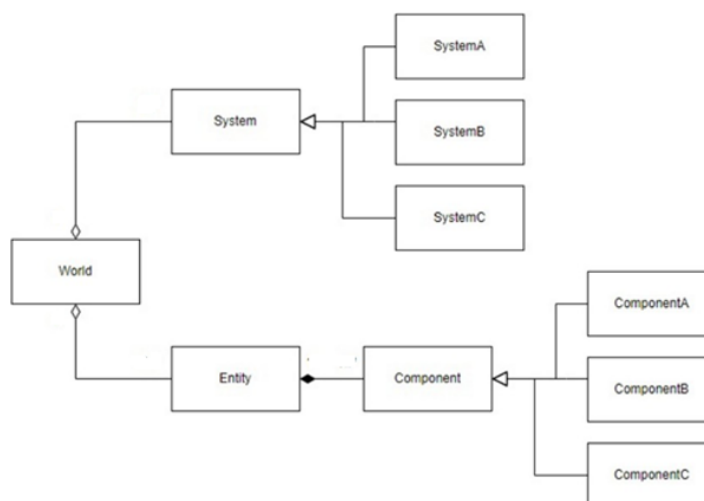
100

Используемые технологии



5

Архитектура приложения



6

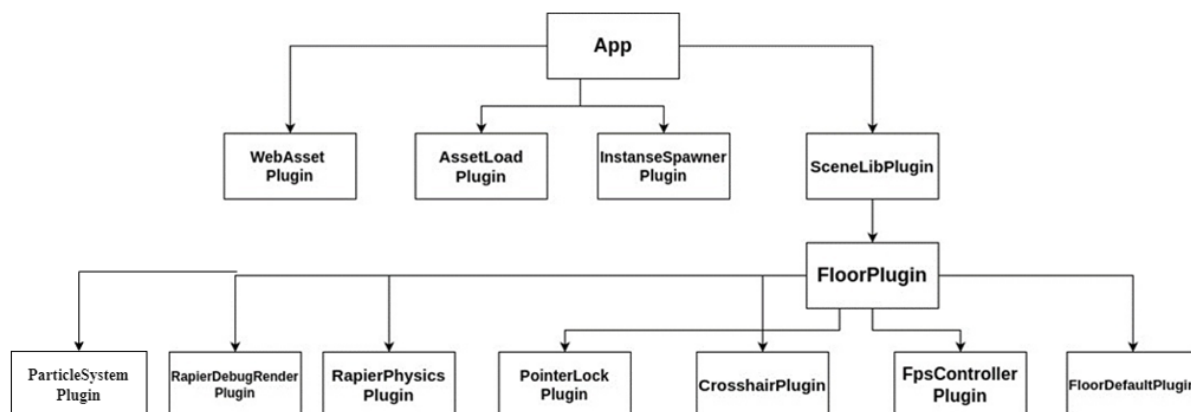
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.34.001 ПЗ

Лист

101

Схема плагинов



7

Демонстрация работы приложения

8

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.34.001 ПЗ

Лист

102

Результаты работы

Результатом работы является разработанный интерактивный 3D-тренажёр «Эвакуация из здания», а также отчёт, в котором представлены результаты тестирования разработанного программного обеспечения, описание разработки и подробный разбор его основных функциональных особенностей.

Разработанное приложение позволяет пользователю прямо в браузере увидеть прототип созданного здания в 3D, перемещаться по нему от первого лица и производить процесс эвакуации.

9

Спасибо за внимание

10

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.34.001 ПЗ

Лист

103