РЕФЕРАТ

Выпускная квалификационная работа: 64 страницы, 3 части, 57 рисунков, 4 таблицы, 25 источников.

ВИРТУАЛЬНАЯ ЛАБОРАТОРИЯ, АРХИТЕКТУРА АППАРАТНЫХ СРЕДСТВ, КОД, C#, UNITY, BLENDER, СКРИПТ.

Целью исследования является разработка виртуальной лаборатории по дисциплине «Архитектура аппаратных средств» на примере Рубцовского института (филиала) АлтГУ.

Объектом исследования является процесс освоения дисциплины «Архитектура аппаратных средств».

Предметом исследования является геймификация процесса освоения дисциплины «Архитектура аппаратных средств».

Методы решения поставленных задач: системный анализ, разработка программного обеспечения, программирование.

Результатом выпускной квалификационной работы является разработанная виртуальная лаборатория по дисциплине «Архитектура аппаратных средств» (на примере Рубцовского института (филиал) АлтГУ).

СОДЕРЖАНИЕ

[ИЕ

Архитектура аппаратных сре это дисциплина, которая изучает организацию, проектирование и действие основных компонентов компьютерных систем. Она охватывает как аппаратную часть, так и связи с программным обеспечением, которое работает поверх неё.

Актуальность выпускной квалификационной работы обусловлена тем, что на данный момент процесс обучения по дисциплине «Архитектура аппаратных средств» проходит в стандартном формате: лекции, презентации, практические и лабораторные занятия. Такой подход снижает интерес учащихся и мотивацию к освоению материала. Внедрение современных подходов к обучению с использованием геймификации, позволит значительно повысить вовлеченность и интерес студентов.

Объектом исследования является процесс обучения по дисциплине «Архитектура аппаратных средств»

Предметом исследования я икация процесса обучения по дисциплине «Архитектура аппар .

Целью исследования является разработка виртуальной лаборатории по дисциплине «Архитектура аппарат едств» на примере Рубцовского института (филиала) АлтГУ.

Для достижения поставлен но решить следующие задачи:

- провести анализ предметной области;
- выработать проектные решения по обеспечивающим системам;
- реализовать проектные решения разрабатываемой виртуальной лаборатории по дисциплине «Архитектура аппаратных средств»;
- рассчитать экономические затраты на разработку виртуальной лаборатории по дисциплине «Архитектура аппаратных средств».

1 Аналитическая часть

1.1 Технико-экономическая характеристика предметной области

Рубцовский институт (филиал) федерального государственного бюджетного образовательного учреждения высшего образования «Алтайский государственный университет». Сокращённое название: Рубцовский институт (филиал) АлтГУ.

Адрес филиала организации, осуществляющей образовательную деятельность: 658225, Алтайский край, г. Рубцовск, проспект Ленина, д. 200б

Создание образовательной организации:

Приказом Государственного комитета РФ по высшему образованию от 22 февраля 1996г. № 324 г. был создан филиал Алтайского государственного университета в г. Рубцовске».

Вуз образован в соответствии с постановлением Совета Министров СССР от 27 марта 1973 г. № 179 и приказом Министра высшего и среднего специального образования РСФСР от 24 мая 1973 г. № 229 как Алтайский государственный университет. Приказом Государственного комитета РФ по высшему образованию от 22 февраля 1996г. № 324 г. был создан филиал Алтайского государственного университета в г. Рубцовске».

Приказом Министерства образования и науки Российской Федерации от 21 апреля 2016 г. № 457: Рубцовский институт (филиал) федерального государственного бюджетного образовательного учреждения высшего профессионального образования «Алтайский государственный университет» переименован в Рубцовский институт (филиал) федерального государственного бюджетного образовательного учреждения высшего образования «Алтайский государственный университет» [1].

Временная государственная аккредитация, срок действия, действующей

аккредитации: 26.09.2023 по 26.09.2025

Организационная структура управления Рубцовского института (филиала) АлтГУ представленная на рисунке 1.1.

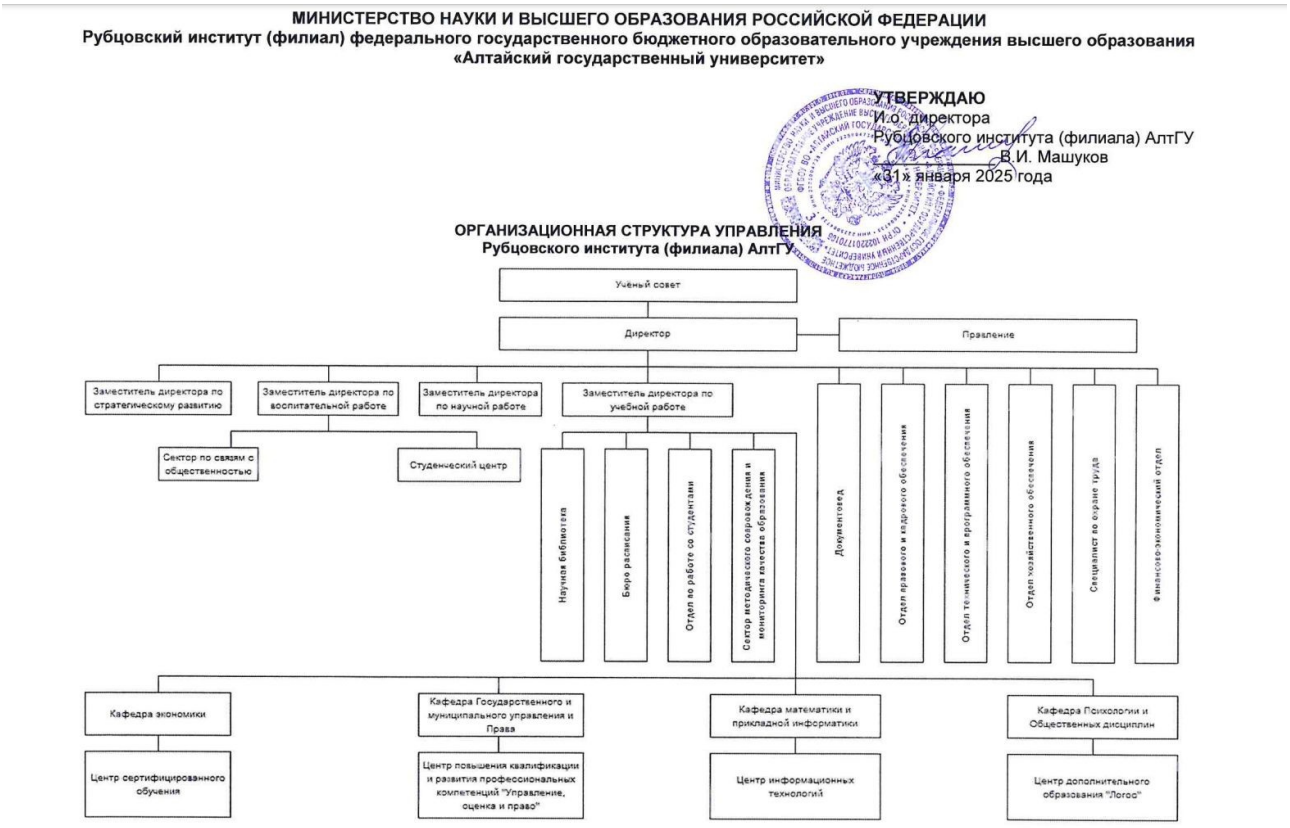

Рисунок 1.1 – Организационная структура управления

Техническое оснащение представляет собой, оснащение аудиторий и лабораторий, общее количество учебных аудиторий: 52

Аудитории распределены по двум адресам:

1. Проспект Ленина, 243 – 21 аудитория
2. Проспект Ленина, 200б – 31 аудитория

Учебные аудитории и лаборатории оснащены современным оборудованием, обеспечивающим комфортный образовательный процесс и практическую подготовку студентов.

Основные элементы оснащения:

1. Специализированная мебель:

– учебные столы и стулья;

– оборудованные места для преподавателей (кафедры, трибуны);

– доски (меловые, маркерные);

– лабораторные столы и шкафы для хранения оборудования.

2. Технические средства обучения:

– мультимедийное оборудование (проекторы, экраны);

– персональные компьютеры и ноутбуки с доступом в интернет и электронную образовательную среду;

– лингафонные системы (для изучения иностранных языков);

– телевизоры, аудиосистемы, магнитофоны.

3. Специализированные лабораторий:

– криминалистические лаборатории;

– компьютерные классы и ИТ-лаборатории;

– геодезическая лаборатория;

– лаборатория БЖД и первой медицинской помощи;

– стрелковый тир.

4. Оборудование для инклюзивного образования:

– пандусы;

– тактильные пиктограммы;

– широкие дверные проёмы;

– компьютерная техника со спецвозможностями.

Учебные помещения оснащены для проведения лекций, семинаров, лабораторных работ и практических занятий.

Программные средства, используемые в организации:

1. Электронные библиотечные системы (ЭБС)

– ЭБС Юрайт;

– ЭБС Академия;

– ЭБС Издательство Лань;

– и другие ЭБС.

2. Система дистанционного обучения:

- Moodle (Система управления образовательными курсами).

3. Корпоративные ресурсы

- официальный сайт https://rb.asu.ru/;
- корпоративная почта https://mail.rb.asu.ru/.

Сведения об оборудованных учебных кабинетах, аудиториях, лабораториях представлены в таблице 1.1.

Таблица 1.1 – Сведения об оборудованных учебных кабинетах филиала

Адрес	Номер кабинета	Оснащенность
658225, Алтайский край, город Рубцовск, проспект Ленина, дом 200б	Аудитория 108, 111, 205, 204, 215, 305, 307, 311,304	Учебная мебель, оборудованные места для обучения, место преподавателя, набор оборудования для демонстрирования учебного материала, компьютерная техника с обеспеченным доступом в интернет, мультимедийное оборудование
658225, Алтайский край, город Рубцовск, проспект Ленина, дом 200б	317, 316, 314, 310, 309, 306, 303, 301, 214, 213, 208, 106,	Мультимедийное оборудование, оборудованное место преподавателя, технические средства обучения, специальная аудиторная мебель
658227, Алтайский край, город Рубцовск, проспект Ленина, дом 243	108, 205, 207, 209, 211	Мультимедийное оборудование, оборудованное место преподавателя, технические средства обучения, специальная аудиторная мебель, переносной ноутбук, мультимедийное оборудование
658227, Алтайский край, город Рубцовск, проспект Ленина, дом 243	222 – стрелковый тир	Интерактивный лазерный тир «РУБИН», ИЛТ-110 «Кадет» (лазерная камера, лазерный автомат Калашникова, пневматический лазерный пистолет), специализированная мебель, оборудованное место для преподавателя, проектор

1.2 Анализ функционирования объекта исследования

Перед началом разработки программного обеспечения необходимо проанализировать, как работает система в настоящее время. Что бы это сделать, необходимо построить модель IDEF0 «Как есть» (рисунок 1.2) [2].

Модель позволяет систематизировать протекающие в данный момент процессы, а также используемые информационные объекты. На основе этого

выявляются «узкие места» в организации и взаимодействии бизнес-процессов, определяется необходимость тех или иных изменений в существующей структуре.

Рисунок – 1.2 Диаграмма функционирования процесса обучения по дисциплине «Архитектура аппаратных средств»

Декомпозиция бизнес-процессов объекта исследования представлена в приложении А (рисунок А1).

После создания декомпозиции была выявлена проблема, которая связана с процессом обучения по дисциплине «Архитектура аппаратных средств»: учебный материал подаётся линейно и недостаточно интересно. Чтобы повысить заинтересованность учащихся, необходимо оптимизировать формат подачи информации и сделать обучение более увлекательным используя методы геймификации.

Геймификация (от англ. gamification) – это использование игровых элементов и механик в неигровом контексте, чтобы повысить вовлеченность, мотивацию и интерес людей. В учебном процессе геймификация может значительно повысить мотивацию, сделать обучение менее скучным и гораздо более увлекательным [3].

Почему это работает:

– люди любят играть, даже взрослые;

- игровые элементы вознаграждают и дают чувство прогресса;
- видимый успех мотивирует;
- помогает превратить рутинное дело в интересное.

1.3 Определение цели и задачи проектирования ИС

Целью создания программного обеспечения виртуальной лаборатории является углубление вовлеченности учащихся в учебный процесс и повышение их интереса к дисциплине «Архитектура аппаратных средств». Внедрение виртуальной лаборатории способствует улучшению качества образования, делая обучение более наглядным, интерактивным и увлекательным, а также стимулирует самостоятельное изучения материала и активное участие в учебной деятельности.

Задачи, которые должна выполнить виртуальная лаборатория:

1. Обеспечить возможность взаимодействия с комплектующими персонального компьютера.
2. Обеспечить визуальное сопровождение.
3. Обеспечить текстовую и звуковую информацию.
4. Обеспечить контроль знаний в виде функционального взаимодействия с ПК.
5. Обеспечить контроль знаний в виде тестирования по пройденному материалу.

1.4 Обзор и анализ существующих технологий проектирования

1.4.1 Unreal Engine

Это набор инструментов для разработки игр, имеет широкие возможности в разработке: от создания двухмерных игр на мобильных

устройствах, до AAA-проектов для консольных устройств и персональных компьютеров. В качестве языка программирования используется C++, помимо этого, мощного и быстрого языка есть возможность использовать язык визуального программирования Blueprints, он позволяет создавать программы из специальных визуальных оболочек и связей между ними, без написания программного кода.

К недостаткам Unreal Engine можно отнести достаточно низкий спрос в России в отличии от более распространённого Unity среди начинающих разработчиков, также Unreal Engine нацелен в основном на высокобюджетные проекты, официальной документации порой недостаточно для решения существующих проблем, пятая версия Unreal Engine пока не может похвастаться стабильной работой [4]. Интерфейс Unreal Engine представлен на рисунке 1.3.

Рисунок 1.3 – Интерфейс Unreal Engine

1.4.2 Unity

Это среда разработки компьютерных игр в которой объединены различные программные средства, используемые при создании ПО – текстовый редактор, компилятор, отладчик и так далее. Unity делает создание программного обеспечения максимально простым и комфортным, а мультиплатформенность движка позволяет охватить как можно большее

количество платформ и операционных систем для разработки. Основные преимущества Unity:

- не требует особых знаний;
- использует компонентно-ориентированные подход разработки;
- наличие огромной библиотеки ассетов и плагинов;
- многие ассеты и плагины доступны бесплатно;
- поддержка большого количества платформ, технологий, API;
- созданное ПО легко портируется между различными ОС.

При всех своих достоинствах, движок имеет свои недостатки, например, если команде разработчиков нужно разработать что-то более-менее сложное, то ей придётся искать хорошего программиста на C#, который сможет написать скрипты и компоненты, чтобы внедрить их в приложение. Помимо потребности в программисте, Unity довольно медлительный, создание масштабных сцен с множеством объектов может негативно повлиять на производительность приложения [5]. Интерфейс Unity представлен на рисунке 1.4.

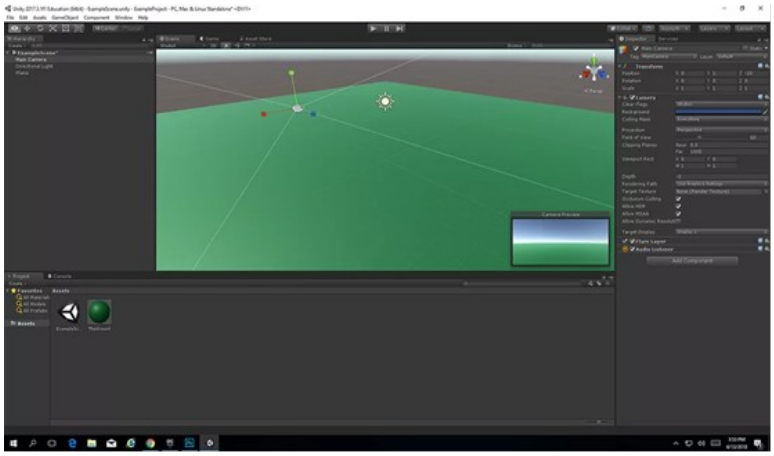

Рисунок 1.4 – Интерфейс Unity

1.4.3 Blender

Это бесплатный графический 3D редактор, предназначенный для моделирования, создания анимации и рендеринга.

Основные достоинства – это бесплатность и открытость, помимо этого Blender имеет следующие преимущества:

- многофункциональность;
- активное сообщество и поддержка;
- богатый набор инструментов;
- постоянное обновление и развитие;
- кроссплатформенность.

Blender имеет серьёзные достоинства, но и серьёзные недостатки, один из которых – это высокая сложность обучения, Blender имеет крутую кривую обучения, имеет довольно сложный интерфейс, множество инструментов в которых легко запутаться. Если быть кратким основная проблема – это время обучения в Blender, требуется много времени и терпения [6]. Другие недостатки:

- низкая производительность в больших проектах;
- не подходит для узконаправленных задач;
- сложный интерфейс и эргономика.

Интерфейс Blender представлен на рисунке 1.5.

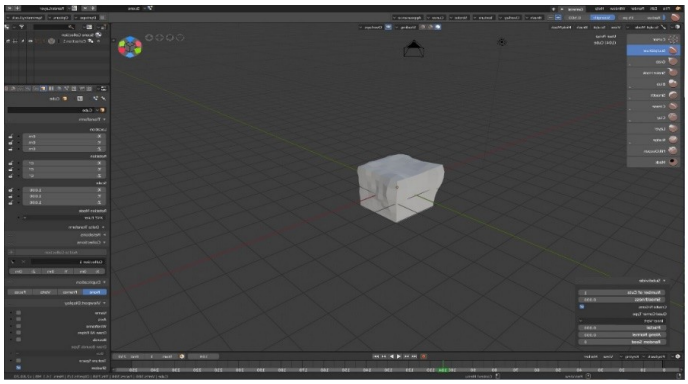

Рисунок 1.5 – Интерфейс Blender

1.4.4 Maya

Это программа для 3D моделирования, создания анимации, визуализации и рендеринга. Содержит универсальный набор инструментов для моделирования, симуляция, рендеринга и др [6].

В отличии от Blender, Maya имеет более «дружелюбный» интерфейс, помимо этого, имеет следующие достоинства:

- больше заточен под анимацию;
- может быть настроен полностью под конкретного пользователя;
- прост в управлении;
- более просто решает нестандартные задачи.

Имея прекрасные достоинства Maya имеет два ключевых недостаток для работы:

- отсутствие русскоязычной версии;
- имеет пробную версию в 30 дней, после придётся приобрести лицензию.

Интерфейс Maya представлен на рисунке 1.6.

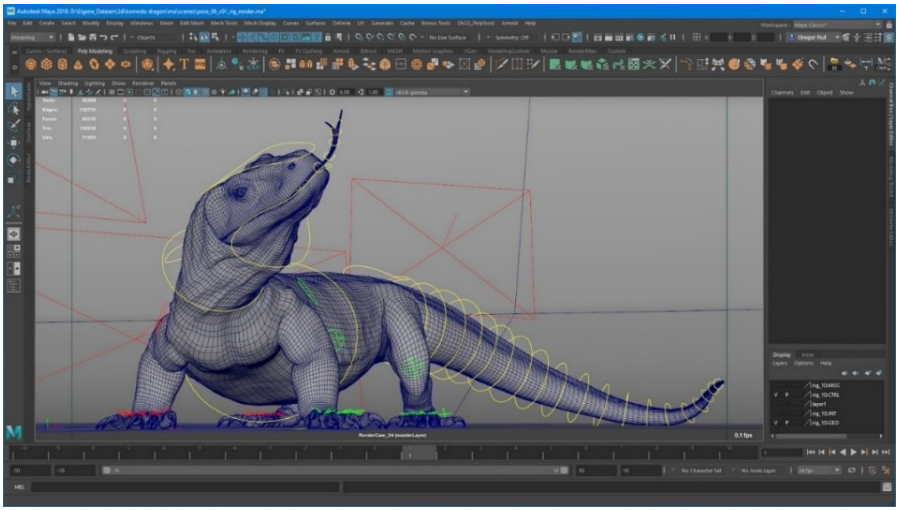

Рисунок 1.6 – Интерфейс Maya

1.5 Обоснование проектных решений

В Рубцовском институте (филиале) АлтГУ учебный процесс в лабораториях и аудиториях предлагает изучение технического оборудования, принцип работы которого зачастую невозможно понять без прямого взаимодействия, а именно «традиционный» подход, связанный с механической разборкой устройств что сопряжено с рядом существенных

рисков и ограничений. Работа с реальным оборудованием может привести к травмам и порче дорогостоящего оборудования что повлечёт за собой экономические издержки, помимо этого, некоторое оборудование имеет ограниченный доступ из-за его ремонта, замены или отсутствия.

В условиях таких ограничений становится актуальной задача поиска альтернативных методов обучения. Одним из направлений является разработка и внедрение специализированного программного обеспечения, которое позволит имитировать взаимодействие с технических оборудованием в виртуальной среде.

Разработка программного средства будет осуществляется при помощи кроссплатформенной среды разработки компьютерных игр Unity (с использованием языка программирования C#), и профессионального свободного и открытого программного обеспечения для создания трёхмерной компьютерной графики Blender 3D.

Данная среда разработки была выбрана по причине свободного доступа и большому количеству информации в открытых источниках, язык программирования C# был выбран по причине более полного и полноценного взаимодействия с Unity нежели других языков программирования, этот язык программирования является обязательным, многие функции и инструменты прописаны именно на нём, написание кода будет осуществляться в программной среде Microsoft Visual Studio.

Blender был выбран для создания трёхмерной графики, основные преимущества которого это, бесплатность, анимация и спецэффекты и самое главное обширное сообщество и ресурсы.

Unity – это игровой движок и платформа для разработки видеоигр, в которую встроена интегрированная среда разработки (IDE) а именно Visual Studio Code для написания кода и работы с проектом. Можно создавать объекты, задавать траекторию для их движения, настраивать анимацию, добавлять эффекты, работать с физикой и проектировать интерфейсы. Кроме

прочего, в этой среде можно писать и проверять код, что позволяет задавать логику происходящего [7].

Возможности Unity:

- кроссплатформенность: PC (Windows, macOs, Linux), мобильные (iOS, Android), консоли (PlayStation, Xbox, Nintendo Switch), VR/AR – устройства, WebGL;
- профилирование, возможность анализа производительности проекта, оптимизация рендеринга для больших сцен, динамическая погрузка ресурсов;
- написания кода на языках программирования C#, JavaScript, Boo, I ronPython, Rust, Lua, C/C++, возможен и визуальный скриптинг;
- технологии графики: Рендеринг, шейдеры, эффекты, Ray Tracing, VFX Graph, Ассеты;
- реалистичная физика и взаимодействия благодаря технологиям Ragdoll и разрушения, PhysX (NVIDIA), а также возможность кастомизировать физику под специфические задачи транспорта или жидкостей;
- система анимация с поддержкой скелетов, создание кат-сцен и сложных сценариев, анимация мимики;
- встроенные элементы интерфейса (кнопки, слайдеры) и поддержка кастомизации, создание собственных адаптивных интерфейсов.

Пользовательский интерфейс Unity представлен на рисунке 1.7.

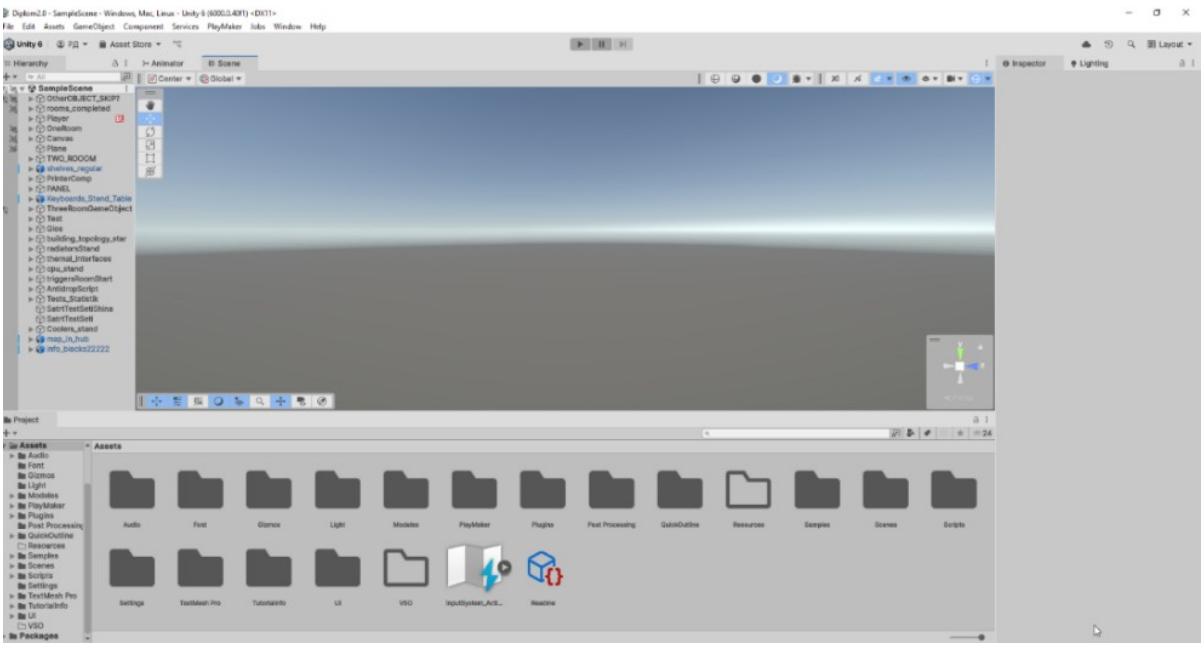

Рисунок 1.7 – Пользовательский интерфейс Unity

Microsoft Visual Studio – представляет собой полноценную IDE с широким набором инструментов для разработки, отладки и тестирования приложений. Это включает в себя редактор кода, визуальные дизайнеры, мощные средства отладки и профилактики [8]. Пользовательский интерфейс Microsoft Visual Studio представлен на рисунке 1.8.

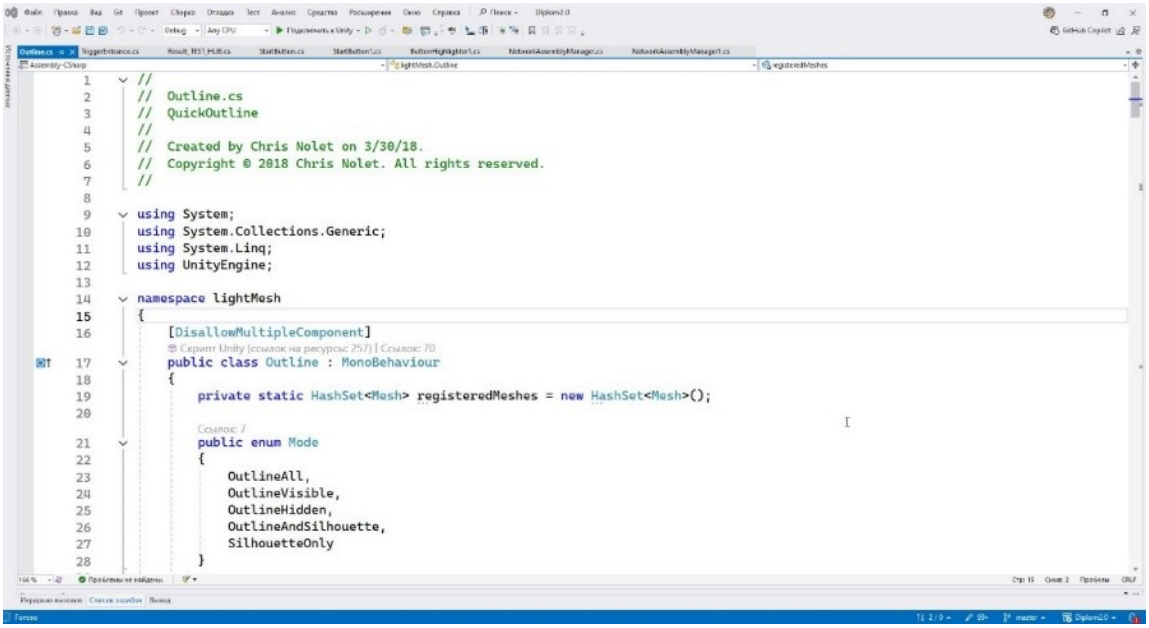

Рисунок 1.8 – Пользовательский интерфейс Microsoft Visual Studio

Blender 3D – это бесплатное ПО для создания 2D и 3D графики а также анимаций. Имеет открытый исходный код, могут использовать как любители, так и профессионалы [9].

Основные направления Blender 3D:

- моделирование 2D и 3D объектов;
- создание текстур;
- анимирование объектов и персонажей;
- рендеринг изображений и видео.

Пользовательский интерфейс Blender 3D представлен на рисунке 1.9.

Рисунок 1.9 – Пользовательский интерфейс Blender 3D

2 Проектная часть

2.1 Разработка функционального обеспечения

С учётом проведённого анализа предметной области и контекстной диаграммы IDEF0 «Как есть» по дисциплине «Архитектура аппаратных средств» была построена диаграмма IDEF0 «Как будет» отражающая предполагаемую модель функционирования системы. Диаграмма IDEF0 «Как будет» представлена на рисунке 2.1.

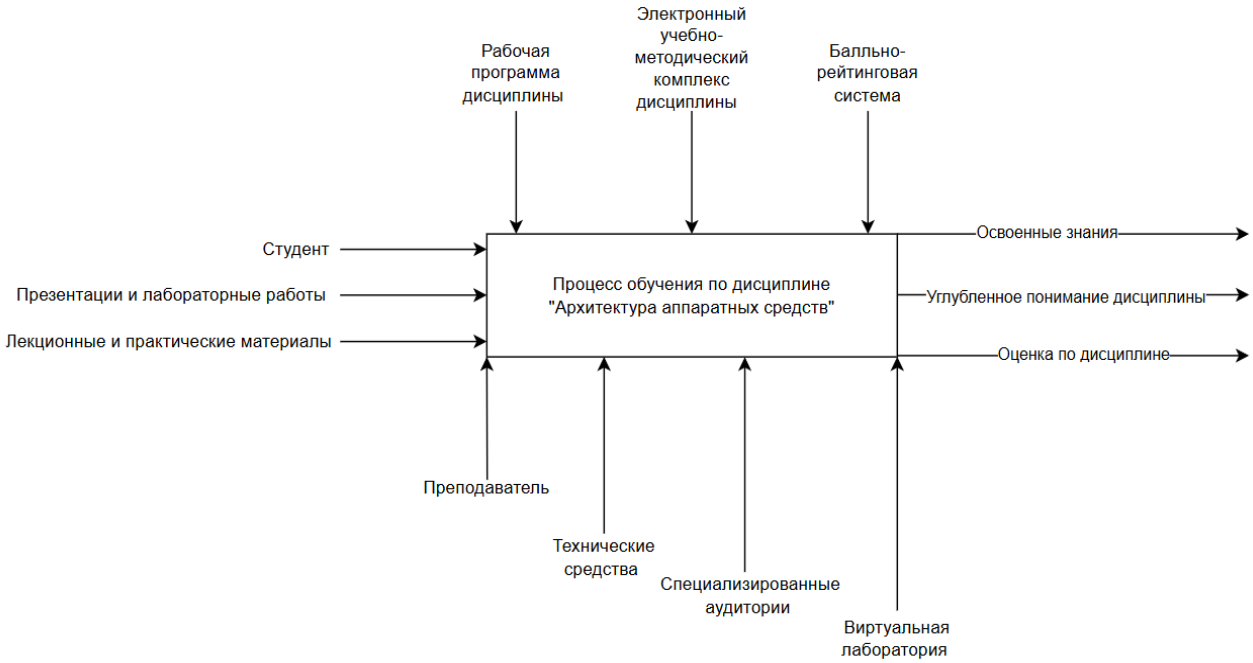

Рисунок 2.1 – Диаграмма IDEF0 «Как будет»

Разрабатываемая виртуальная лаборатория выступает одним из механизмов обучения, способствующим не только передаче теоретических знаний, но и формирование практических навыков студентов. Интерактивны формат работы с лабораторией позволяет существенно повысить вовлеченность и мотивацию обучающихся, сделать процесс освоения материала более увлекательным и понятным. Декомпозиция диаграммы IDEF0 «Как будет» представлена в приложении А (рисунок А2).

2.2 Разработка информационного обеспечения

Перед реализацией программного обеспечения нужно создать 3D-объекты для взаимодействия с ними, всего было создано около 60 полноценных 3D-объектов, не считая копий. Если посчитать все объекты, похожие объекты или немного видоизменённые объекты то количество экземпляров превысит 100. Демонстрация небольшой части 3D-объектов: графический ускоритель (рисунок 2.2), материнская плата (рисунок 2.3), центральный процессор (рисунок 2.4), оперативная память (рисунок 2.5), системы охлаждения (рисунок 2.6), кулеры (рисунок 2.7).

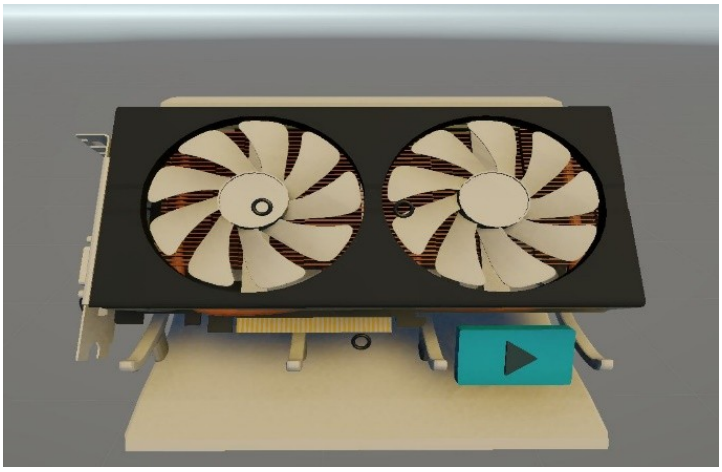

Рисунок 2.2 – Графический ускоритель

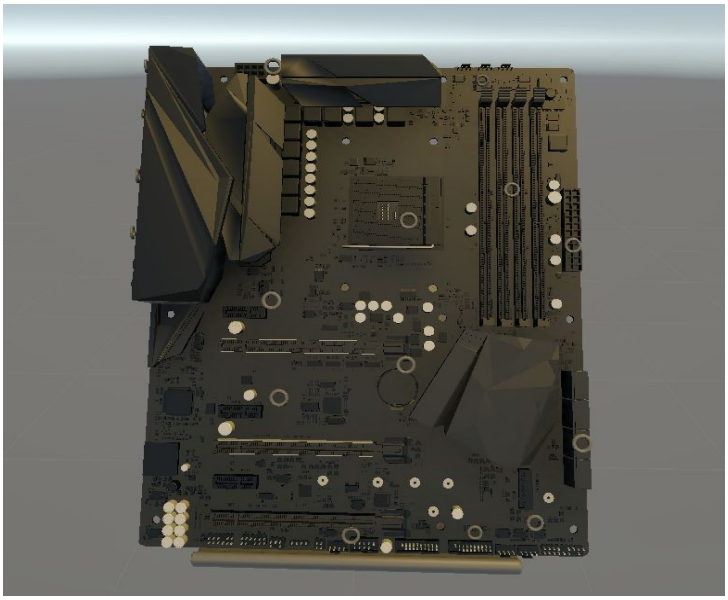

Рисунок 2.3 – Материнская плата

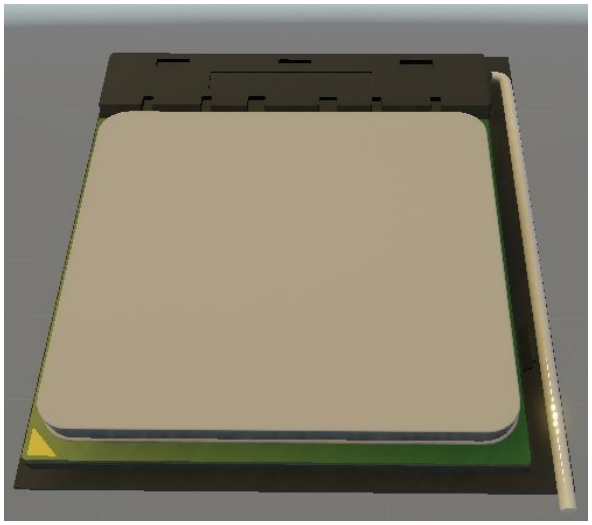

Рисунок 2.4 – Центральный процессор

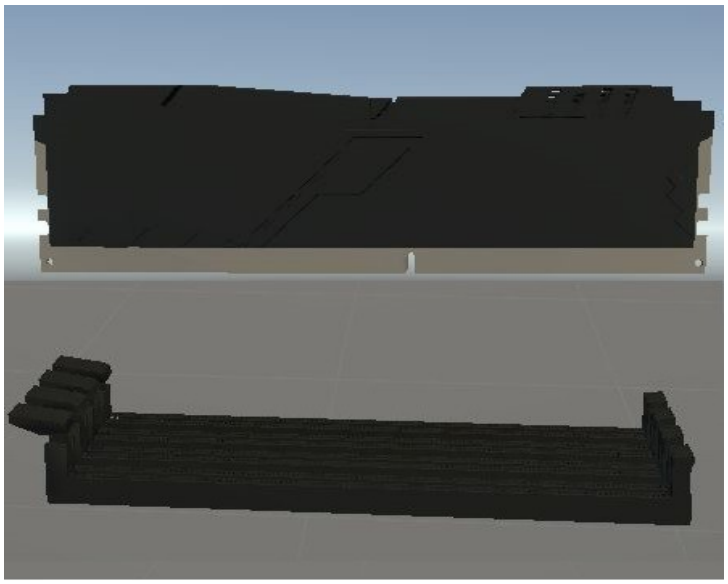

Рисунок 2.5 – Оперативная память

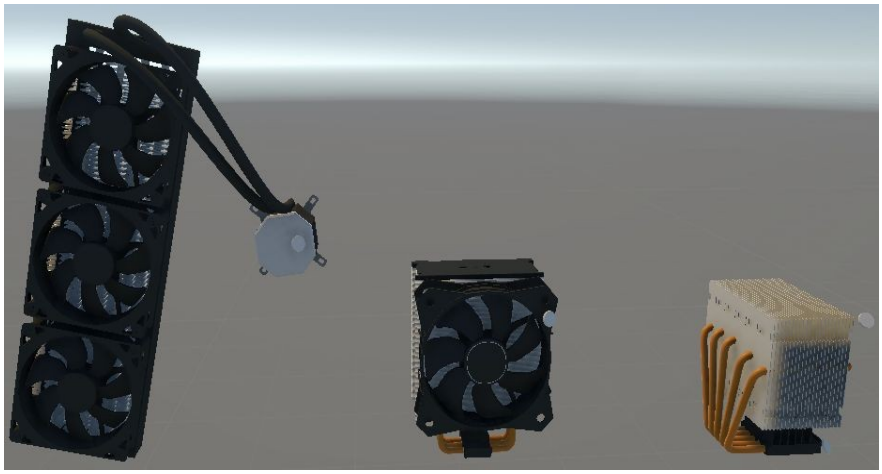

Рисунок 2.6 – Системы охлаждения

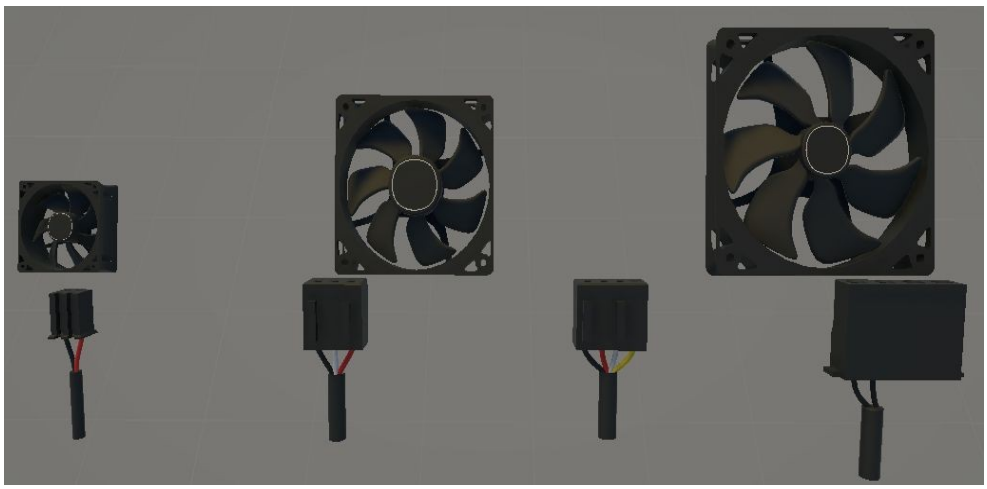

Рисунок 2.7 – Кулеры

Все 3D-объекты являются цифровыми копиями из реальной жизни и не являются художественными интерпретациями, они выполнены с высокой точностью копируя оригинальные предметы, некоторые предметы увеличены в масштабе для большей наглядности. Демонстрация всех моделей займёт десятки страниц, для понимания общей картины, нужно продемонстрировать весь масштаб созданных моделей [10]. Демонстрация всех моделей представлена на рисунке 2.8.

Рисунок 2.8 – Демонстрация всех объектов

2.4 Создание анимации

Анимация – искусственное представление движения в кино, на телевидении или в компьютерной графике путем отображения последовательности рисунков или кадров с частотой, при которой обеспечивается целостное зрительное восприятие образов [11].

Анимация является неотъемлемой частью проекта, важность анимации сложно переоценить, демонстрация правильной установки видеокарты, процессора или оперативной памяти не показать простым, мгновенным перемещением в пространстве, важна плавность и точность, анимированные объекты помогают лучше понять принцип работы объекта, понять, как следует обращаться с некоторым оборудованием в жизни, чтобы не допустить критических ошибок.

Создание анимации на первый взгляд кажется простым процессом: выбираются нужные объекты, устанавливаются ключи на таймлайне (рисунок 2.9), перемещаются объекты, процесс повторяется, но, на самом деле это длительный и трудоёмкий процесс, важно правильно всё настроить и «подогнать», чтобы анимация выглядела не «рвано» и точно [12]. Кроме того, сложность анимации зависит от количества объектов, легко создать анимацию из 2-3 объектов, а что, если их десятки, тогда процесс создания анимации затягивается на часы, а иногда даже на дни. В рамках проекта анимация создавалась в Blender.

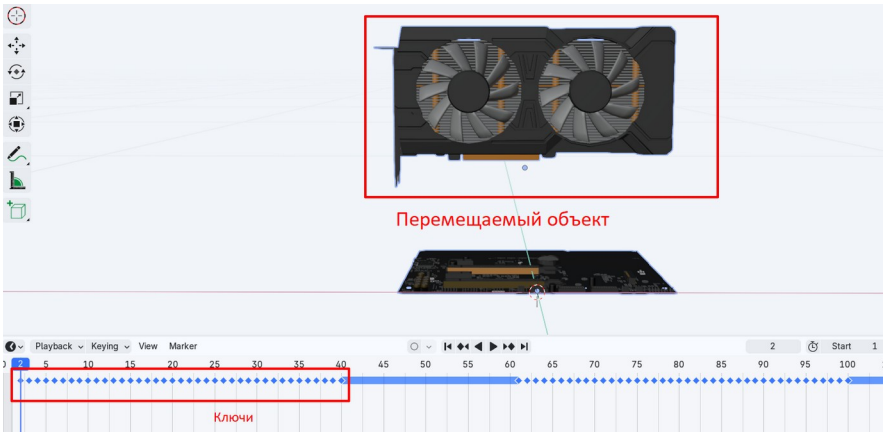

Рисунок 2.9 – Размещение ключей на таймлайне

После установки ключа фиксируются текущие координаты объекта. Затем объект перемещается на нужное расстояние, устанавливается следующий ключ. Этот процесс повторяется до тех пор, пока объект не окажется в нужной позиции. Важно соблюдать баланс в количестве устанавливаемых ключей, если анимация достаточно простая, как представлено на рисунке 2.9, установка видеокарты в материнскую плату, всё что нужно это установить видеокарту в правильном положении, благодаря этой простоте количество ключей не столь велико. Например, если в данной анимации количество ключей будет в 2 раза выше, то анимация станет «рваной» т.к. Программа не будет успевать сгладить переходы, помимо «рваной» анимации возникает проблема редактирование такой анимации.

Важно отметить, если требуется постоянно повторяющаяся анимация, необходимо создать зеркальное расположение ключей на таймлайне, чтобы анимация повторялась бесконечно, переходя из точки А в точку Б и обратно. Таймлайн с зеркальным расположением ключей представлен на рисунке 2.10.

Рисунок 2.10 – Таймлайн с зеркальным расположением ключей

Рассуждая о трудоёмкости анимации, представлен таймлайн с 9 перемещаемыми объектами. Таймлайн стенда установки разъёмов передней панели представлен на рисунке 2.11.

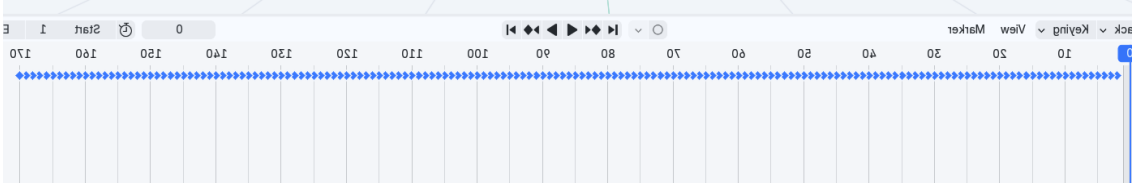

Рисунок 2.11 – Таймлайн стенда установки разъёмов передней панели

Обращая внимание на рисунок 2.11, видно количество установленных

ключей – их десятки. На первый взгляд может показаться, что анимация установки одного объекта из точки А в точку Б так проста, но, когда это нужно сделать 9 и более раз, трудоёмкость процесса значительно возрастает. И это лишь в случае прямолинейного перемещения. Если же объект необходимо переместить из или через несколько точек, и даже самая простая работа с анимацией превращается в рутинную и длительную работу.

Основные факторы, влияющие на плавность анимации:

- интерполяция между ключами;
- частота кадров;
- правильная расстановка ключей.

Интерполяций – это способ, с помощью которого программа автоматически рассчитывает и создаёт промежуточные кадры между двумя ключевыми кадрами. Например, если объект был в точке А на кадре 1 и в точке Б на кадре 10, интерполяция создаст движение между А и Б [13].

Типы интерполяций:

- liner (линейная) – движение с постоянной скоростью;
- ease in/out: - плавный старт или замедление;
- bezier – можно настраивать кривые вручную для сложного движения.

В проекте использовался в основном тип интерполяции Bezier.

Частота кадров в секунду или FPS (frames per second) – это количество кадров, которое показывается каждую секунду анимации, стандартным количеством кадров является 24 FPS, но в современных проектах в основном в играх используются 60 FPS [14].

Правильная расстановка ключей – один из наиболее важных моментов в движении анимации, правильно расставленные ключи отражают основные положения передвигаемого объекта, его поза, его поворот, его скорость или замедление. Если ключей слишком мало – движение будут слишком грубыми.

Если слишком много и в неправильных местах – анимация может быть «рваной» и трудной в исправлении [15].

Всего в проекте было создано 20 анимаций. Объекты с анимациями в стартовом положении представлены на рисунке 2.12.

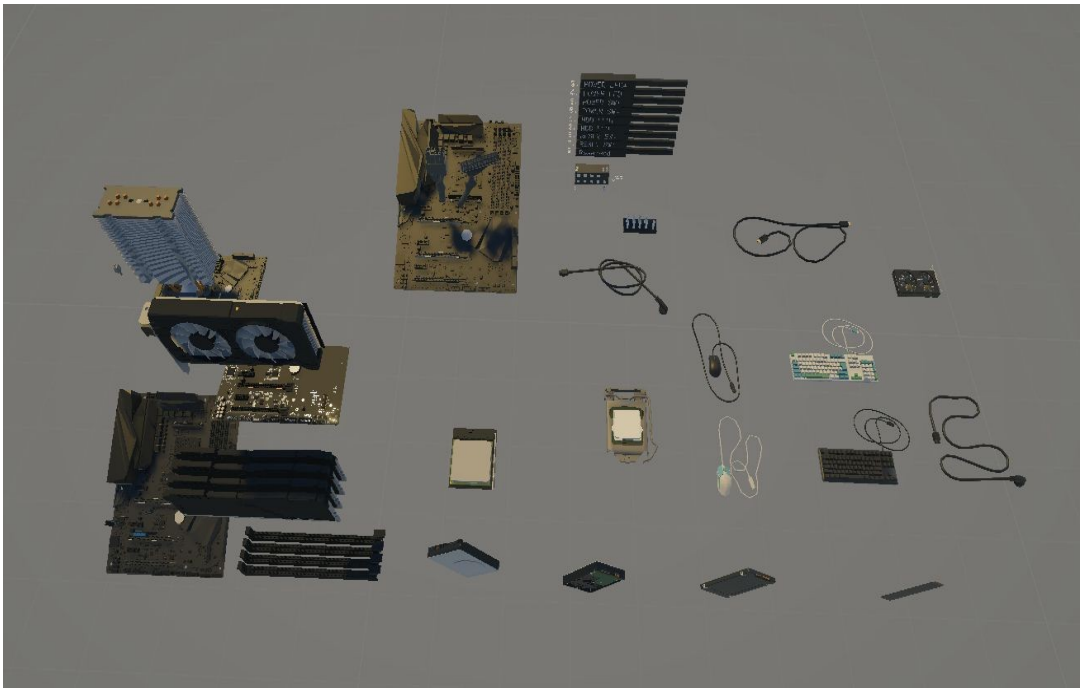

Рисунок 2.12 – Объекты с анимациями в стартовом положении

Объекты в конечном положении анимации представлены на рисунке 2.13.

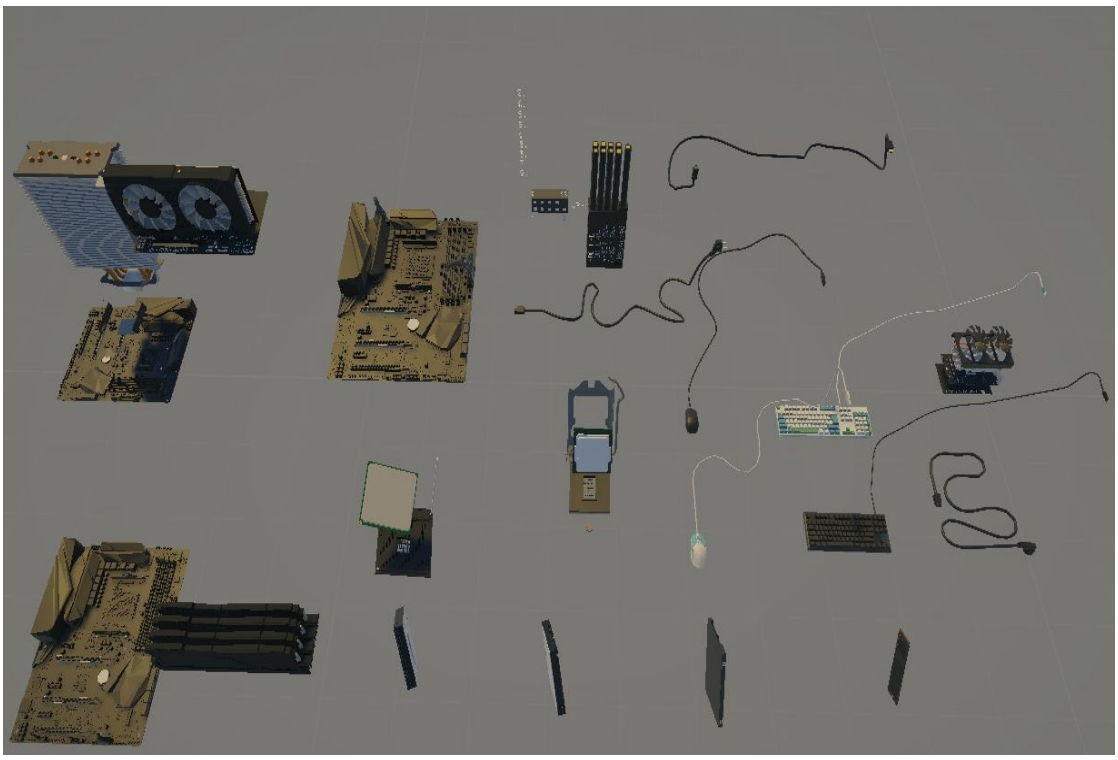

Рисунок 2.13 – Объекты в конечном положении анимации

2.5 Иерархии и компоненты объектов в Unity

Прежде чем начать создавать скрипты для взаимодействия с предметами, первым делом создадим игрока, камеру и руку.

Игрок (Player) – объект в сцене который, способен передвигаться и взаимодействовать с окружающими предметами.

Камера (Main Camera) – своего рода «глаза» игрока, благодаря камере можно создавать разные сцены или просто наблюдать за происходящим на экране.

Рука (Hand) – чаще всего представляет собой пустой объект, выполняющий функцию перемещения и взаимодействия с объектами, когда они находятся в руке.

Создать объекты мало, важно правильно их расположить. В Unity используется система иерархий: один объект может быть «материнским», а другой – «дочерним». Например, создав объект 1 и поместив в него объект 2, объект 2 будет повторять все манипуляции за объектом 1, передвижения, повороты, увеличение или уменьшение объекта, включение и выключение.

Все манипуляции, применённые к родителю, автоматически повлияют на дочерний объект. Учитывая это важно правильно организовать структуру созданных объектов, начнём с материнского объекта, материнским объектом будет игрок(Player). Его дочерним объектом будет выступать камера (Main camera), поскольку она должна повторять все движения игрока. Рука(Hand), в свою очередь, будет дочерним объектом камеры, так как взаимодействием с предметами должно находиться в поле зрения игрока [16]. Иерархия объектов представлена на рисунке 2.14.

Рисунок 2.14 – Иерархия объектов

Иерархия настроена, теперь необходимо правильно расположить объекты на сцене, чтобы они могли корректно взаимодействовать друг с другом. Расположение объектов в сцена представлено на рисунке 2.15.

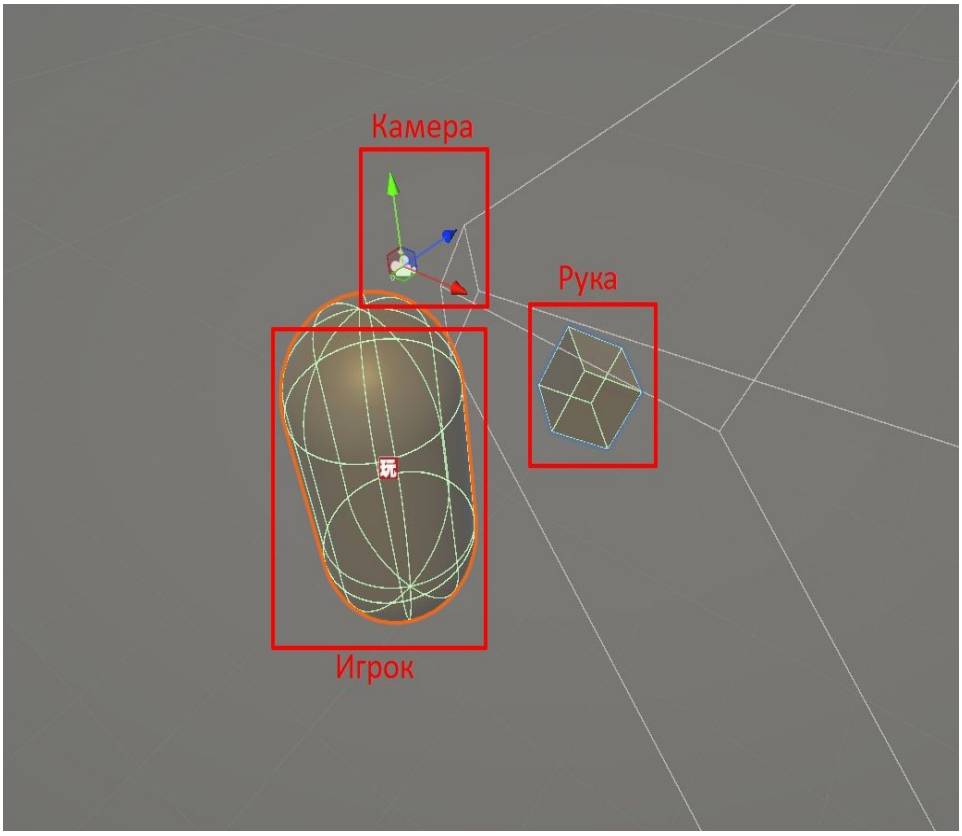

Рисунок 2.15 – Расположение объектов в сцене

Обращая внимание на рисунок 2.15, игрок представлен в виде капсулы, камера – это невидимым объект, как и рука, которая для наглядности представлена в виде куба. От камеры исходят белые линии и образуют конусовидную область – это область видимости. Все объекты, попадающие в эту зону, будут отображаться на экране и восприниматься пользователем.

Капсула имеет зелёные линии по периметру капсулы, это зона компонента объекта, называемая Capsule Collider, или просто «коллайдер». Коллайдеры нужны для того, чтобы объекты могли сталкиваться, реагировали на физику и могли взаимодействовать друг с другом, например, падение объекта на пол, если пол и падающий объект имеют коллайдеры, то падающий объект не пролетит сквозь пол, а упадёт на пол. Существует 4 основных вида коллайдеров для 3D-объектов:

- box collider;
- capsule collider;
- mesh collider;
- terrain collider.

Box collider – это один из наиболее часто используемых коллайдеров в Unity, из названия можно понять что это коллайдер имеет форму параллелепипеда (коробки или куба). Его нельзя превратить в другую геометрическую фигуру, он всегда остаётся прямоугольным объёмом с ровными гранями. Тем не мене, можно изменить его размер по трём осям (ширина, высота, глубина), чтобы он подходил под форму объекта [17]. Свойства Box collider представлены на рисунке 2.16.

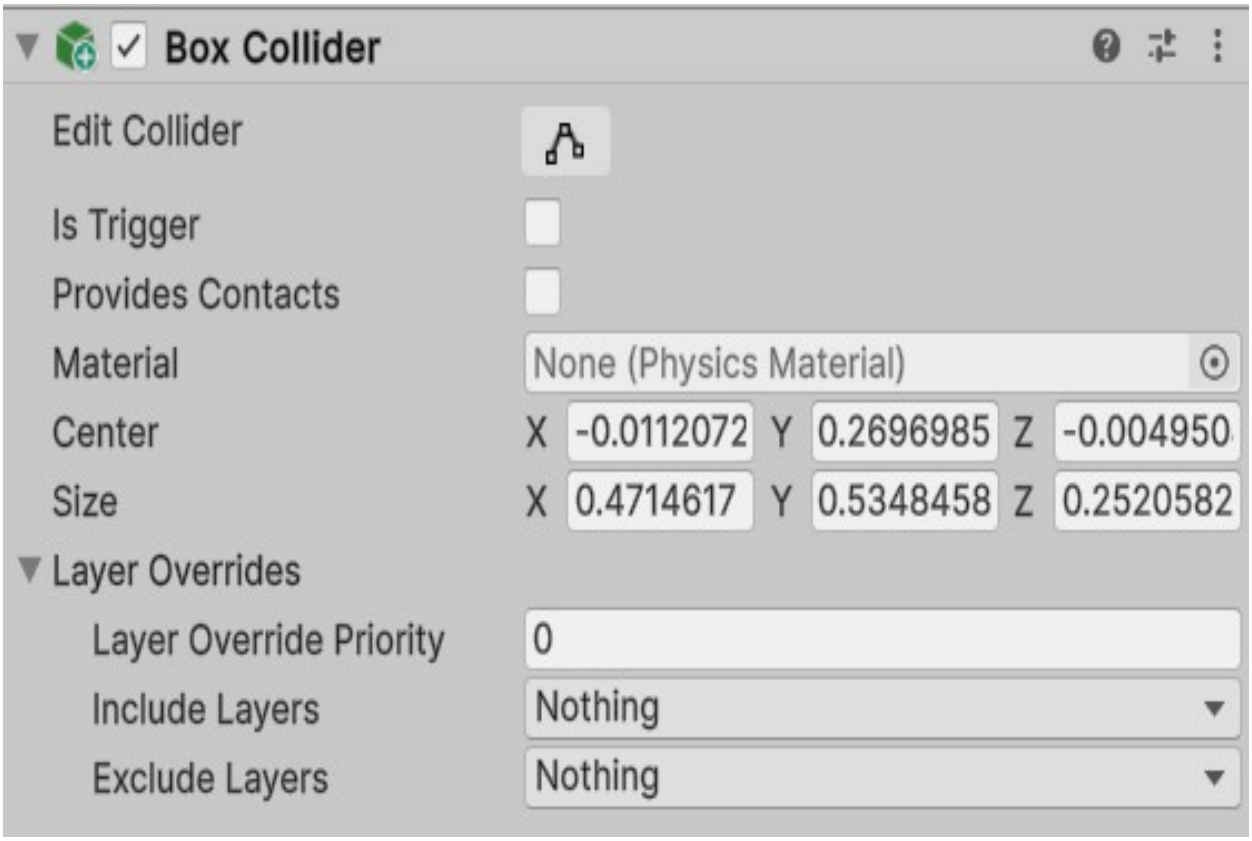

Рисунок 2.16 – Свойства Box collider

Edit Collider – позволяет редактировать коллайдер, вручную менять размер и положение коллайдера в пространстве.

Is Trigger – триггер, если включён, объект не сталкивается с другими, но сможет вызывать события.

Provides Contacts – обычно используется для сложных сценариев с физикой. Определяет, будут ли генерироваться данные о контактах, даже если объект – триггер.

Material – позволяет назначать физический материал, который задаёт свойства, такие как трение и упругость.

Center – задаёт смещение центра коллайдера относительно объекта.

Size – задаёт размер коллайдера.

Layer Override Priority – приоритет переопределения слоя.

Include Layers – список слоёв, которые учитываются этим коллайдером.

Exclude Layers – список слоёв, которые игнорируются коллайдером.

Свойства других коллайдеров схожи и имеют небольшие отличия.

Capsule collider – это коллайдер имеющий форму капсула с цилиндрами на концах. Его часто используют для персонажей в компьютерных играх, так как он хорошо подходит под форму стоящего человека. Его размеры, высоту и радиус можно изменять, но форма всегда останется капсульной.

Mesh collider – это коллайдер который повторяет форму 3D-модели (меша). Подходит для объектов, имеющих сложную геометрию, является наиболее сложным и ресурсоёмким коллайдером.

Terrain collider – это коллайдер, предназначенный специально для объектов типа Terrain (ландшафт). Он автоматически повторяет форму рельефа, созданного в редакторе Unity.

Ещё одним важнейшим компонентом для объектов является Rigidbody, он позволяет Unity рассчитывать:

- гравитацию;
- силу;
- ускорение;
- трение;
- столкновения;
- отскоки.

С компонентом Rigidbody объект уже не просто висит в пространстве, а становится участником физики. Свойства Rigidbody представлены на рисунке 2.17.

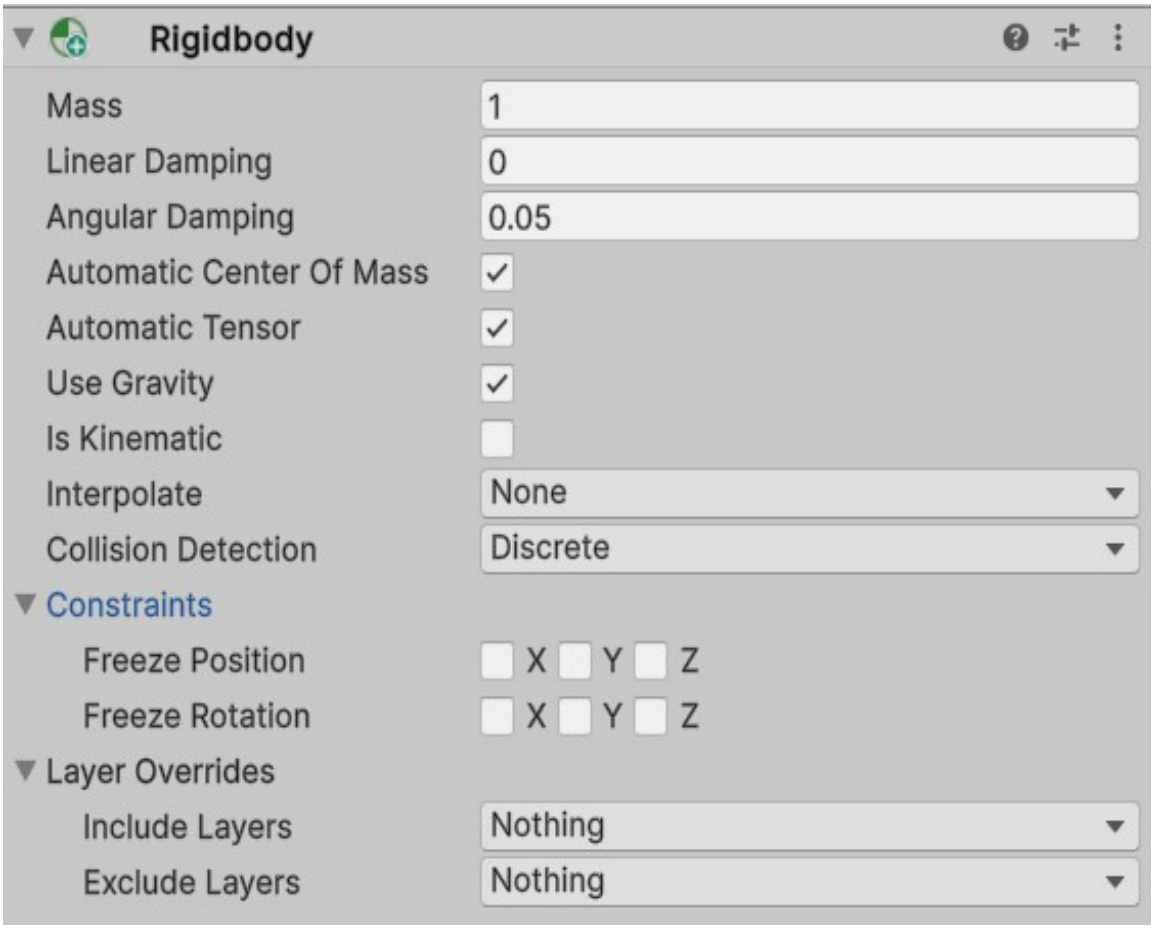

Рисунок 2.17 – Свойства Rigidbody

Mass – масса объекта, которая влияет на силу столкновения и ускорение.

Liner Damping – линейное демпфирование. Сопротивление движению объекта (чем больше, тем быстрее он прекращает крутится)

Angular Damping – вращательное демпфирование. Сопротивление вращению объекта.

Automatic Center Of Mass – автоматический центр массы, Unity автоматически рассчитывает момент инерции.

Automatic Tensor – автоматический расчёт инерции.

Use Gravity – использование гравитации.

Is Kinematic – режим кинематики, объект не учувствует в физике (не падает, не сталкивается), но можно передвигать его вручную.

Interpolate – сглаживание движения.

Collision Detection – обнаружение столкновений.

Freeze Position – замораживает позицию по выбранным осям.

Freeze Rotation – замораживает вращение по выбранным осям.

Include Layers – указывает, какие слои учитывать при взаимодействии.

Exclude Layers – исключает указанные слои.

Box Collider и Rigidbody являются основными компонентами объекта для взаимодействия с ним [17].

2.6 Написание скриптов

После создания 3D-объектов и анимации идёт наиболее важная часть проекта, написание скриптов.

Скрипт – это программный файл, содержащий команды или инструкции, которые выполняют необходимые команды одну за другой [18]. Скрипты используют для написания логики в играх, веб-разработке и других сферах. Помимо скриптов важно добавить компоненты для объектов взаимодействия Box Collider и Rigidbody.

Для передвижения и поворота камеры был использован визуальный скриптинг.

Визуальный скриптинг – это программирование с использованием графического интерфейса, а не текстового кода. Вместо написания кода, нужно соединять блоки или узлы, которые представляют определённые действия, условия, переменные и т.д. Для визуального скриптинга использовался плагин PlayMaker.

PlayMaker – это инструмент в Unity который позволяет создавать скрипты без необходимости программирования. Он предоставляет готовые инструменты для создания игровых элементов. Интерфейс PlayMaker представлен на рисунке 2.18.

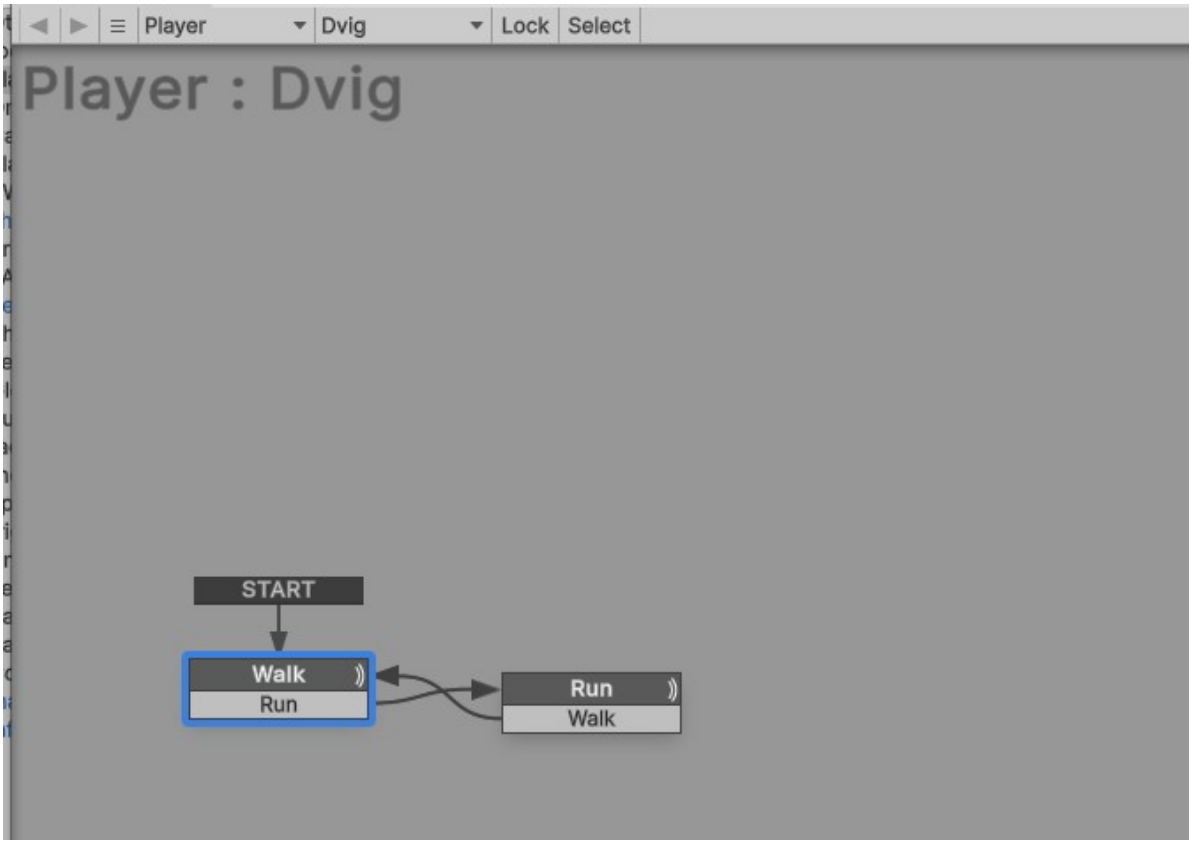

Рисунок 2.18 – Интерфейс PlayMaker

Создание скрипта для передвижения игрока.

Скрипт состоит из двух логических блоков: Walk и Run.

- блок walk отвечает за стандартное перемещение игрока, ходьбу;
- блок run отвечает за ускоренное продвижение игрока, бег.

Выбрав блок Walk и перейдя на вкладку State создаётся 4 блока, Get Axis Vector, Set Float Value, Controller Simple Move, Get Key Up.

Get Axis Vector имеет следующие параметры:

- horizontal axis / vertical axis – стандартные оси в unity (например, wasd или стрелки клавиатуры);
- multiplier – множитель, усиливающий или ослабляющий результат;
- map to plane – движение по горизонтальной плоскости;

– relative to – направление будет относительным к объекту player (например, если игрок повернут, движение будет соответствовать его направлению);

– store vector – сохраняет получившийся вектор движения в переменную, которая используется позже.

Set Float Value имеет следующие параметры:

– float variable – имя переменной;

– float value – фиксированное значение скорости (единиц в секунду);

– every frame – обновление каждый кадр.

Controller Simple Move имеет следующие параметры:

– game object – используется объект, к которому прикреплён fsm;

– move vector – направление движения, ранее полученное;

– speed – переменная, определяющая скорость;

– space – движение происходит в глобальном пространстве.

Get Key Up имеет следующие параметры:

– key – отслеживание отпускания назначенной кнопки;

– send event – когда клавиша отпущена, запускается событие

Важные параметры блока Walk:

– set float value – параметра float value установлен на 4 единицам скорости;

– get key up – параметр send event, принимает событие run, которое запускает блок run.

блок run имеет те же параметры во вкладке state, только с изменёнными данными:

– set float value – параметр float value установлен на 10 единицам скорости;

– get key up – параметр send event, принимает событие walk, которое запускает блок walk.

События работают следующим образом, когда пользователь нажимает на клавиши клавиатуры («W», «A», «S», «D») объект к которому закреплён данный скрипт, в нашем случае капсула игрока, начинает движение в зависимости от нажатых клавиш, при нажатии на клавишу - «left shift», объект увеличивает скорость передвижения, при повторном нажатии на клавишу, объект возвращается к прошлой скорости передвижения. Наглядное представление параметров блока Walk и Run представлено на рисунке 2.19.

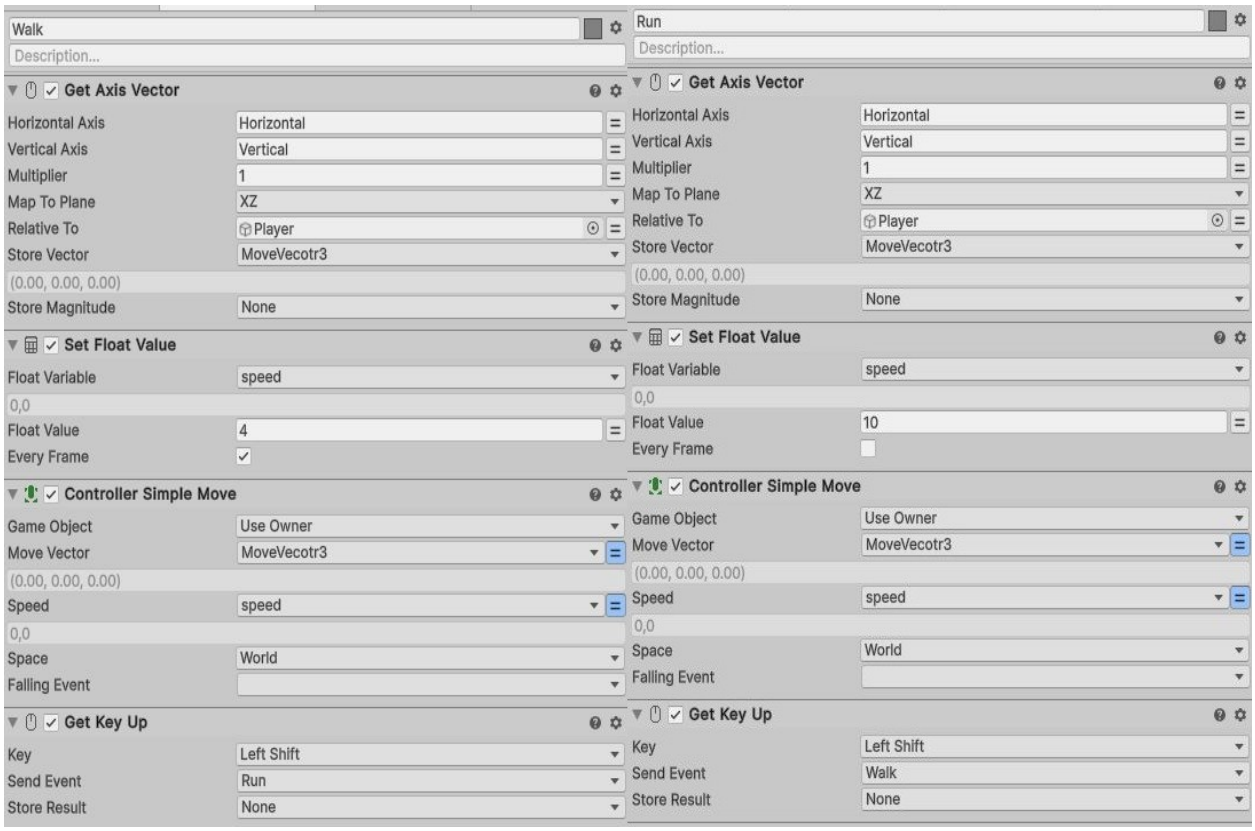

Рисунок 2.19 – Параметры блоков Walk и Run

Передвижение, управляемое только с клавиатуры, может быть неудобным, поэтому необходимо создать дополнительный скрипт, отвечающий за управление движением с помощью компьютерной мыши. Используя Play Maker, создаём ещё один файл и называем его «Mouse».

Mouse содержит единственный блок State 1, данный блок содержит 3 блока:

- mouse look – отвечает за движение мыши по оси y;
- mouse look – отвечает за движение мыши по оси x;

– set mouse cursor – используется для скрытия и блокировки курсора мыши windows.

Теперь пользователь может полноценно передвигаться по сцене, используя клавиатуру для передвижения по пространству, а мышь для указания направления движения [19]; [20]. Визуальный скриптинг использовался только для передвижения персонажа. Параметры блока State 1 представлены на рисунке 2.20.

Одна из основных задач – это симулировать взаимодействия с оборудованием. Персональный компьютер является одним из ключевых объектов для изучения оборудования, описание его составных частей и важность правильной, поэтапной сборки. Цель первого написанного скрипта – обеспечить корректную последовательность сборки ПК.

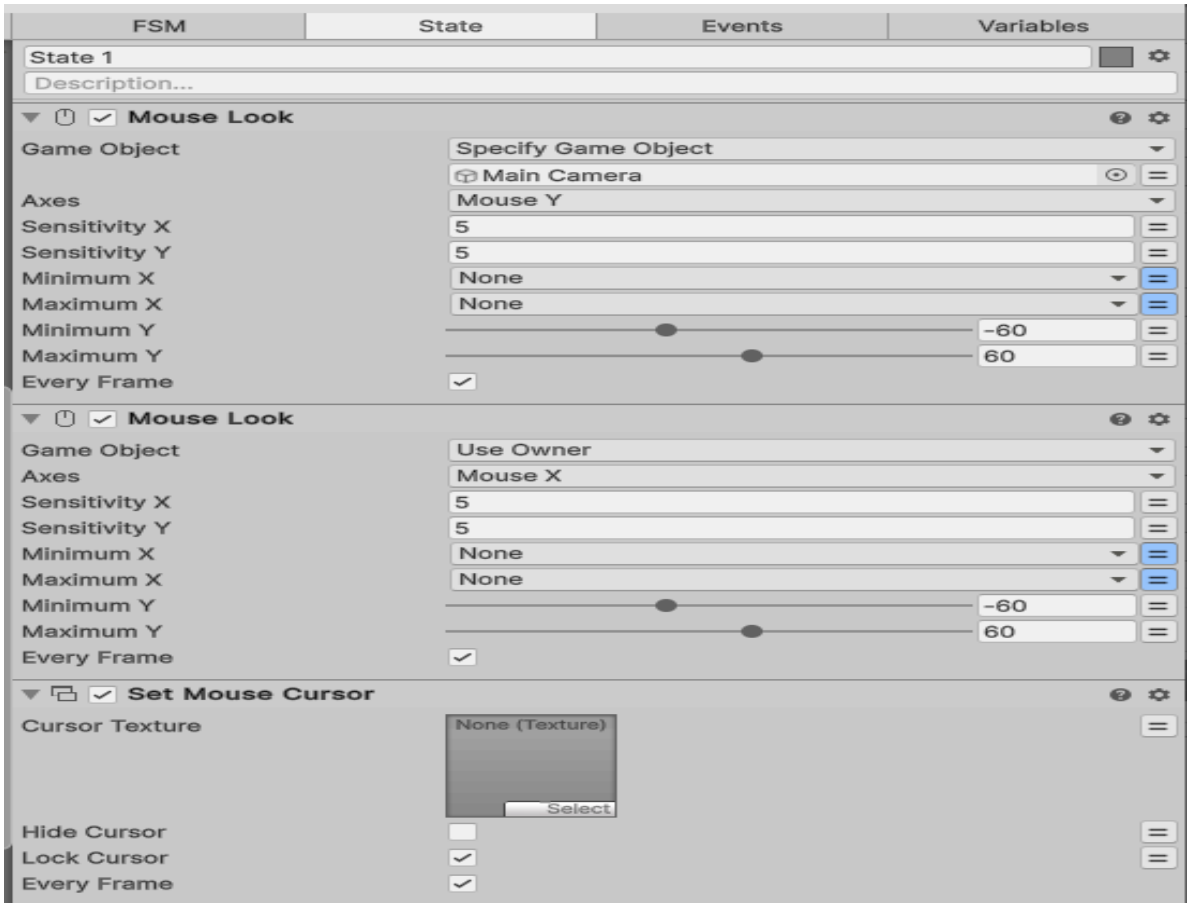

Рисунок 2.20 – Параметры блока State 1

Скрипт для сбора ПК называется PC_SBOR_NEW, основные функции скрипта:

– обнаружить искомый объект;
– поднять при взаимодействии;
– подсветить место установки;
– установить при взаимодействии с подсвечиваемой целью;
– собрать ПК поэтапно.

Программный код для сборки ПК представлен в приложении Б (Рисунок Б1).

В коде присутствуют два основных списка, placeholders – список мест установки, items – список поднимаемых предметов. При взаимодействии с предметом, нажав клавишу E, предмет поднимается и перемещается в руку персонажа, а место установки предмета подсвечивается зелёным цветом, зелёный цвет придаётся при помощи материала, используемого в Unity. Было написано 16 методов, назначение каждого метода:

– start – отключает эффекты подсветки и скрывает места установки;
– update – основной игровой цикл, все методы вызываются здесь;
– tryPickup – попытка подобрать предмет;
– hold – установка предмета в нужную позицию;
– tryDrop – отцепляет предмет от руки при нажатии Q;
– dropItem – придаёт предмету физику откидывания;
– tryPlace – проверяет условия установки;
– startPlacement – запускает анимацию установки;
– smoothPlacement – отвечает за плавное перемещение предметов;
– finishPlacement – фиксирует предмет установки;
– placeAt – удаляет компоненты объекта (например, коллайдер);
– setItemPositionAndRotation – задаёт индивидуальную позицию/ поворот для конкретного предмета;
– showPlaceholdersFor – показывает место установки;
– hideAllPlaceholders – скрывает точку установки;

– allItemsPlaced – проверяет установлены ли все предметы.

Помимо методов присутствует один класс Placeholder, выполняет следующие условия:

– назначает точку установки;

– хранит ссылки на допустимый предмет;

– отслеживает состояние предмета, установлен или нет;

– проверяет условие активации;

– хранит и показывает предупреждения;

– управляет видимостью объекта;

– управляет внешним видом объекта [21].

Параметры скрипта сборки ПК представлено на рисунке 2.21.

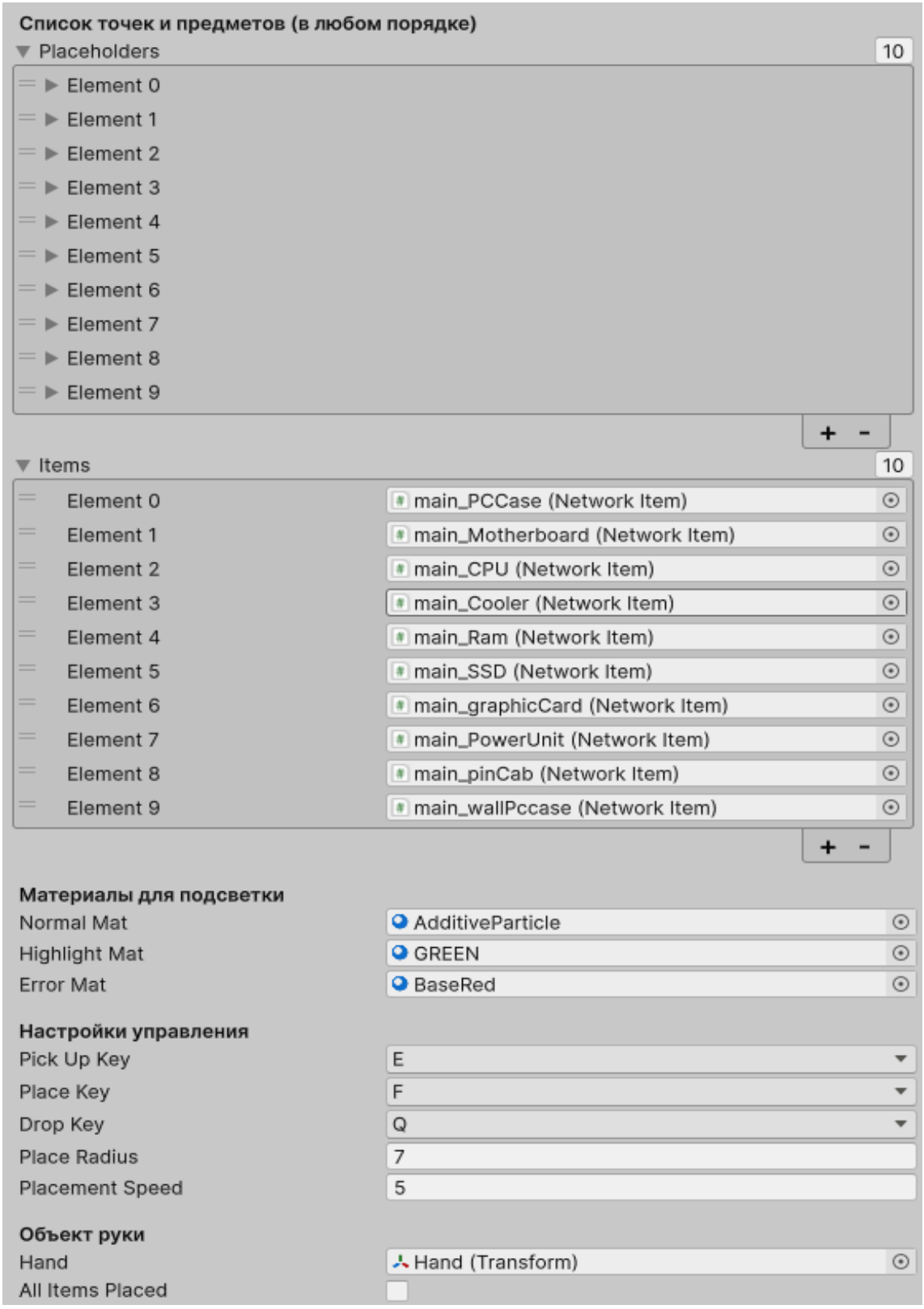

Рисунок 2.21 – Параметры скрипта сборки ПК

Объекты для сборки ПК представлены на рисунке 2.22.

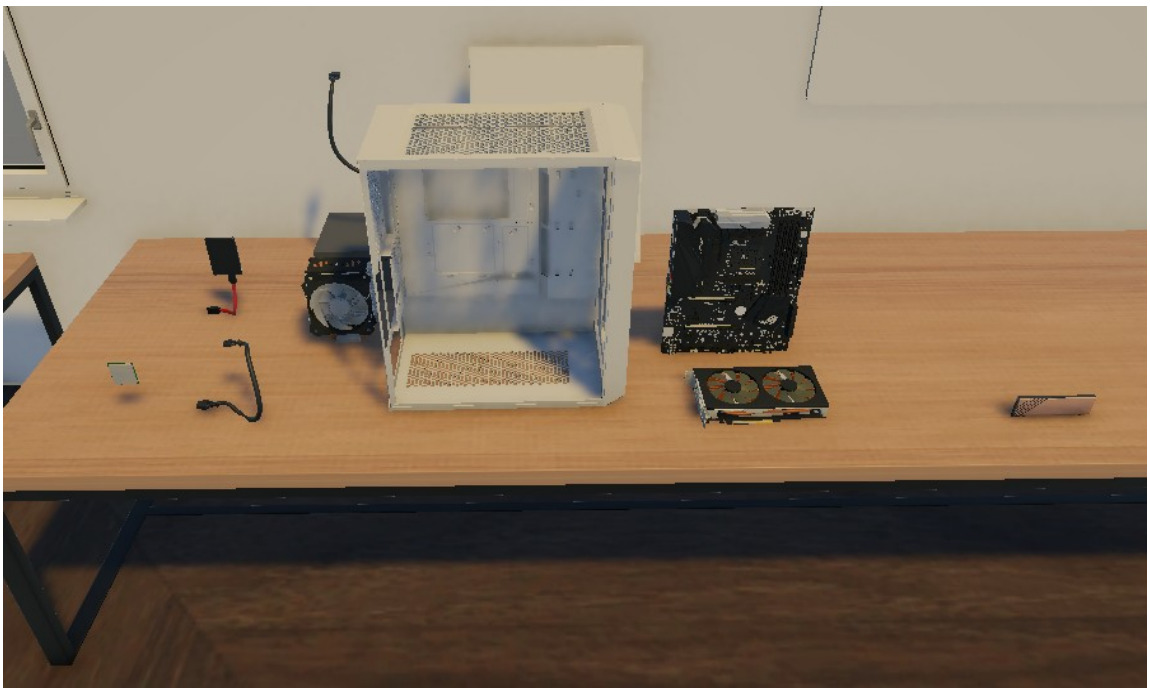

Рисунок 2.22 – Объекты для сборки ПК

Устанавливаемый объект для сборки ПК представлен на рисунке 2.23.

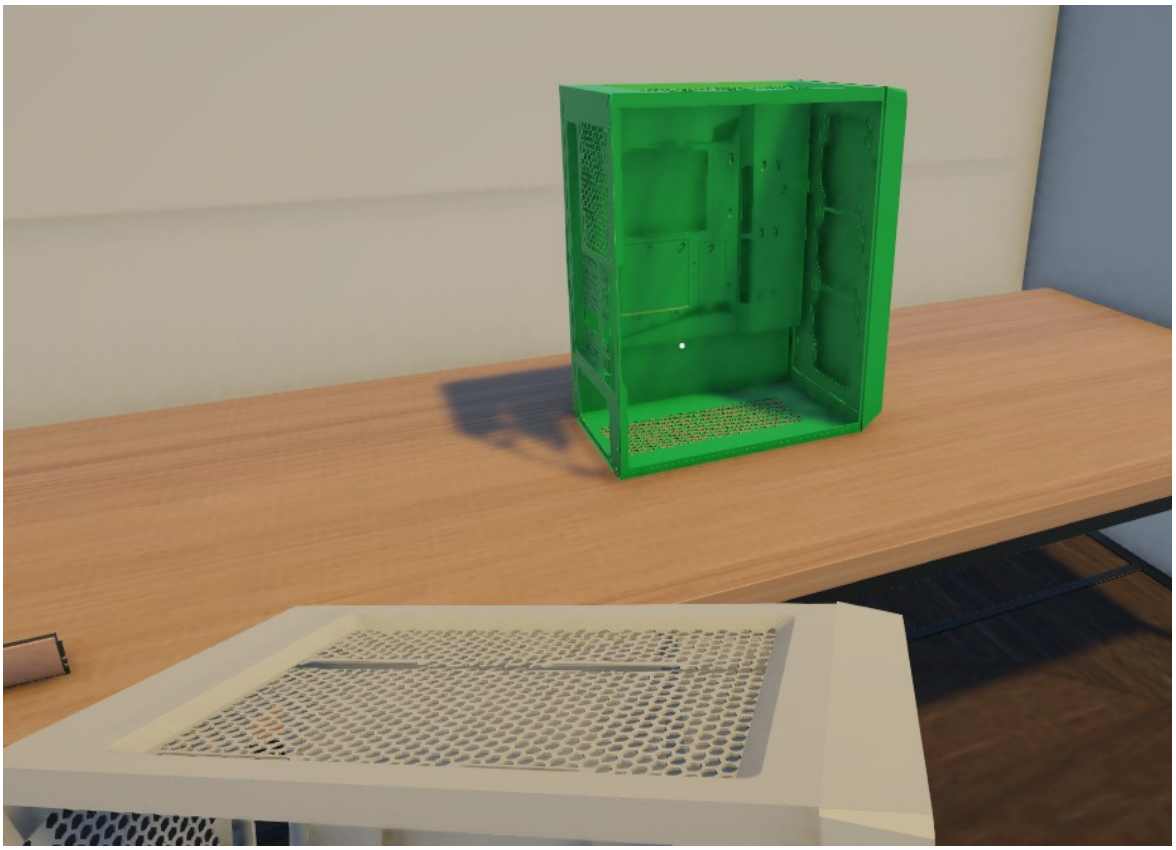

Рисунок 2.23 – Устанавливаемый объект для сборки ПК

ПК в полном сборе представлен на рисунке 2.24.

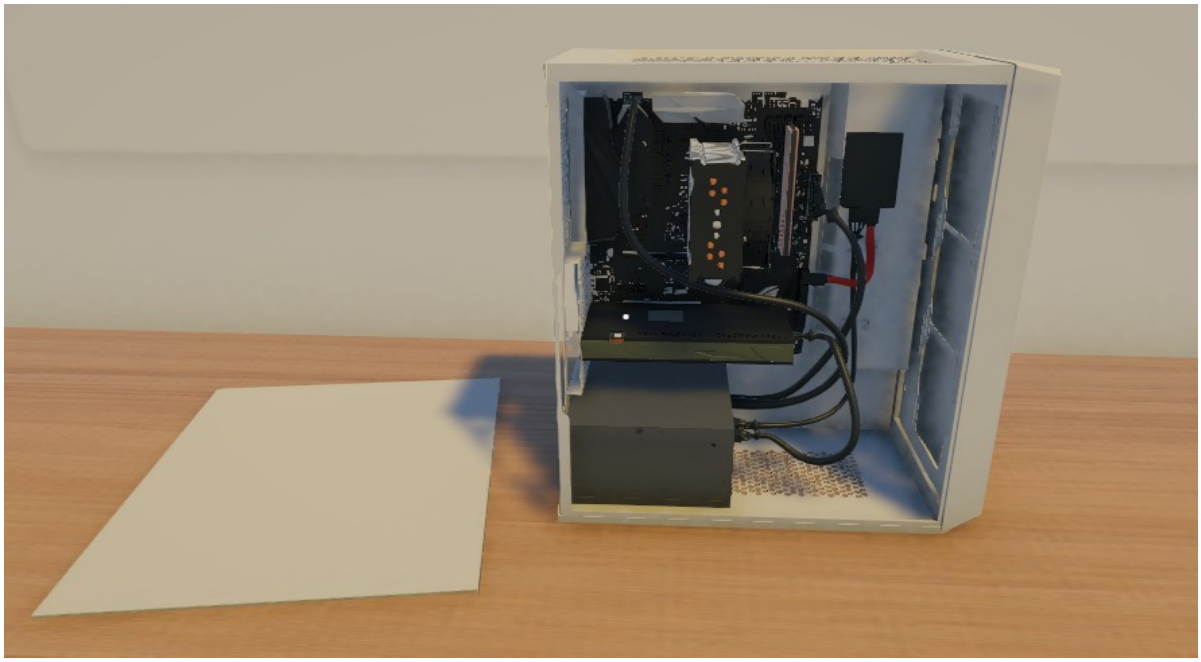

Рисунок 2.24 – ПК в полном сборе (без стенки корпуса)

Для работы следующего стенда, стенда для детального просмотра видеокарта и материнской платы создано 4 скрипта:

- secondcamera;
- stendsgm;
- cameralookonly;
- videocartaandmotherscriptrot.

SecondCamera – довольно крупный скрипт в рамках проекта, отвечает за интерактивное взаимодействие пользователя с элементами стенда. Его основная задача заключается в том, чтобы, используя луч, исходящий из камеры, как бы из взгляда игрока, определять, на какой объект в данный момент направлен взгляд игрока. Когда нужный объект оказывается в поле зрения игрока и на него попадает луч, скрипт использует визуальное выделение объекта (подсветку) и проявляет текст у данного объекта, описание объекта или его части. Кроме того, скрипт может запускать определённую анимацию и взаимодействовать с кнопками. Программный код скрипта SecondCamera представлен на рисунке 2.25.

```
    Сообщение Unity | Ссылок: 0
    void Start()
    {
        currentCamera = GetComponent<Camera>();

        // Инициализация прозрачности текстов
        foreach (var textObject in textObjects)
        {
            if (textObject != null && textObject.GetComponent<Renderer>() != null)
            {
                Renderer renderer = textObject.GetComponent<Renderer>();
                renderer.material.color = new Color(1f, 1f, 1f, 0f);
                renderer.enabled = false;
                textAlphaValues[textObject] = 0f;
            }
        }

    }
```

Рисунок 2.25 – Программный код скрипта SecondCamera (метод Start)

В методе Start используется цикл foreach, который перебирает все элементы списка текстов textObjects. Используется для отключения текста в стартовом положении и, если луч камеры не попадает на нужный объект. Продолжение программного кода скрипта SecondCamera представлено на рисунке 2.26.

```
// Проверка Raycast
if (Physics.Raycast(currentCamera.transform.position, currentCamera.transform.forward, out hit, range))
{
    if (hit.collider.CompareTag("marker") && IS_trigger_markerObjects.Contains(hit.collider.gameObject))
    {
        hoveredObject = hit.collider.gameObject;
    }
    // Проверяем, попадает ли луч на animGPUStart или animGPUStop
    if (hit.collider.gameObject == buttonGPUStart || hit.collider.gameObject == buttonGPUStop)
    {
        hoveredObject = hit.collider.gameObject;
    }
}
// Изменение позиции статичного стенда видеокарты
if (Input.GetMouseButtonDown(0) && hit.collider.gameObject == buttonGPUStart)
{
    upDownLock = true;// Флаг для разрешения передвижения вверх, вниз

    var (positionGPUPassiv, rotationGPUPassiv) = posAndRot.PositionAddAndRotationAdd(-0.21f, 9f, 1.129f, 0, -90, 0);
    standGPUPassiv.transform.SetLocalPositionAndRotation(positionGPUPassiv, rotationGPUPassiv);
    var (positionGPUAnim, rotationGPUAnim) = posAndRot.PositionAddAndRotationAdd(-0.1f, 0.7f, 1.129f, 0, -90, 0);
    standGPUAnim.transform.SetLocalPositionAndRotation(positionGPUAnim, rotationGPUAnim);

    if (animGpuAciv != null)
    {
          animGpuAciv.enabled = true;
    }
    if (currentlyHoveredObject != hoveredObject)
    {
        ResetPreviousHover();
        currentlyHoveredObject = hoveredObject;

        if (currentlyHoveredObject != null)
        {
            ActivateHover(currentlyHoveredObject);
        }
    }
}
if (Input.GetMouseButtonDown(0) && hit.collider.gameObject == buttonGPUStop)
{
    upDownLock = false;
    var (positionGPUPassiv, rotationGPUPassiv) = posAndRot.PositionAddAndRotationAdd(-0.21f, 0.4458656f, 1.129f, 0, -90, 0);
    standGPUPassiv.transform.SetLocalPositionAndRotation(positionGPUPassiv, rotationGPUPassiv);
    var (positionGPUAnim, rotationGPUAnim) = posAndRot.PositionAddAndRotationAdd(-0.1f, 9f, 1.129f, 0, -90, 0);
    standGPUAnim.transform.SetLocalPositionAndRotation(positionGPUAnim, rotationGPUAnim);
}
    // Если объект поменялся
    if (currentlyHoveredObject != hoveredObject)
{
    ResetPreviousHover();
    currentlyHoveredObject = hoveredObject;

    if (currentlyHoveredObject != null)
    {
```

Рисунок 2.26 – Программный код скрипта SecondCamera (метод Update)

В методе Update используется 2 основных условия:

1. Условие определения, попадает ли луч на нужный объект: с помощью Physics.Raycast проверяется, пересикает ли луч из камеры объект с тегом «marker» (при этом он должен быть в списке IS_trigger_markerObjects), либо кнопки под названием buttonGPUStart.

2. Условие, проверяющее нажатие левой кнопки мыши: при нажатии выполняются определённые действия:

– Усли нажата кнопка buttonGPUStart, меняется положение стендов, активируется анимация и обновляется подсветка;

– Если нажата кнопка buttonGPUStop, положение стендов возвращается назад, а флаг upDownLock сбрасывается.

В конце метода идёт обновление текущего подсвеченного объекта (если он имзенился), обработка текста и обновление контуров подсветки [22].

Продолжение программного кода скрипта SecondCamera представлено на рисунке 2.27.

```
Ссылок: 1
void ProcessTextObjects()
{
    // Сначала обрабатываем все тексты для fade out
    foreach (var textObject in textObjects)
    {
        if (textObject == null) continue;

        Renderer renderer = textObject.GetComponent<Renderer>();
        if (renderer == null) continue;

        // Если это не текущий активный текст, делаем fade out
        if (textObject != currentTextObject)
        {
            float currentAlpha = textAlphaValues.ContainsKey(textObject) ? textAlphaValues[textObject] : 0f;
            float newAlpha = Mathf.Lerp(currentAlpha, 0f, OutSpeed * Time.deltaTime);

            if (newAlpha <= 0.01f)
            {
                newAlpha = 0f;
                renderer.enabled = false;
            }
            else
            {
                renderer.enabled = true;
            }

            renderer.material.color = new Color(1f, 1f, 1f, newAlpha);
            textAlphaValues[textObject] = newAlpha;
        }
    }

    // Затем обрабатываем текущий активный текст (если есть)
    if (currentTextObject != null)
    {
        Renderer renderer = currentTextObject.GetComponent<Renderer>();
        if (renderer != null)
        {
            float currentAlpha = textAlphaValues.ContainsKey(currentTextObject) ? textAlphaValues[currentTextObject] : 0f;
            float newAlpha = Mathf.Lerp(currentAlpha, 1f, InSpeed * Time.deltaTime);

            renderer.enabled = true;
            renderer.material.color = new Color(1f, 1f, 1f, newAlpha);
            textAlphaValues[currentTextObject] = newAlpha;
        }
    }
}
```

Рисунок 2.27 – Программный код скрипта SecondCamera (метод ProcessTextObj)

Метод ProcessTextObjects обрабатывает все текстовые объекты, контролируя их палвное появление и исчезновение. В цикле пребираются все объекты из списка. Если текст не является активным, происходит плавное уменьшение прозрачности используя функцию Mathf.Lerp, которая постепенно снижает альфа-канал текстуы и отключает видимость объекта по достижению определённого уровня альфа-канала. Если объекта активен, то его прозрачность начинает увеличиваться, чтобы текст стал полностью видимым. Продолжение программного кода скрипта SecondCamera представлено на рисунке 2.28.

```
185     void ResetPreviousHover()
186     {
187         if (currentlyHoveredObject != null)
188         {
189             int index = IS_trigger_markerObjects.IndexOf(currentlyHoveredObject);
190
191             // Проверяем, что объект существует в списке и не является null
192             if (index >= 0 && index < Is_object_complectObjects.Count && Is_object_complectObjects[index] != null)
193             {
194                 GameObject complectObject = Is_object_complectObjects[index];
195                 if (complectObject != null && activeOutlines.ContainsKey(complectObject))
196                 {
197                     activeOutlines[complectObject].enabled = false;
198                     activeOutlines.Remove(complectObject);
199                 }
200             }
201
202             // Сбрасываем подсветку для animGPUStart и animGPUStop
203             if (buttonGPUStart != null && activeOutlines.ContainsKey(buttonGPUStart))
204             {
205                 activeOutlines[buttonGPUStart].enabled = false;
206                 activeOutlines.Remove(buttonGPUStart);
207             }
208             if (buttonGPUStop != null && activeOutlines.ContainsKey(buttonGPUStop))
209             {
210                 activeOutlines[buttonGPUStop].enabled = false;
211                 activeOutlines.Remove(buttonGPUStop);
212             }
213         }
214
215         // Сбрасываем текущий текст при смене объекта
216         currentTextObject = null;
217     }
218
        Ссылок: 2
219     void ActivateHover(GameObject markerObject)
220     {
221         int index = IS_trigger_markerObjects.IndexOf(markerObject);
222
223         if (index >= 0 && index < Is_object_complectObjects.Count)
224         {
225             GameObject complectObject = Is_object_complectObjects[index];
226             if (complectObject != null)
227             {
228                 lightMesh.Outline outline = complectObject.GetComponent<lightMesh.Outline>();
229                 if (outline == null)
230                 {
231                     outline = complectObject.AddComponent<lightMesh.Outline>();
232                 }
233
234                 if (outline != null && !activeOutlines.ContainsKey(complectObject))
235                 {
236                     outline.enabled = true;
237                     outline.OutlineColor = HexToColor(color);
238                     outline.OutlineWidth = 0f; // Начинаем с 0 для плавного появления
239                     activeOutlines[complectObject] = outline;
240                 }
241             }
242             // Устанавливаем текущий текст
243             if (index < textObjects.Count)
244             {
245                 currentTextObject = textObjects[index];
246             }
247         }
248         // Добавляем подсветку для animGPUStart и animGPUStop
249         if (markerObject == buttonGPUStart || markerObject == buttonGPUStop)
250         {
251             lightMesh.Outline outline = markerObject.GetComponent<lightMesh.Outline>();
252             if (outline == null)
253             {
254                 outline = markerObject.AddComponent<lightMesh.Outline>();
255             }
256
257             if (outline != null && !activeOutlines.ContainsKey(markerObject))
258             {
259                 outline.enabled = true;
260                 outline.OutlineColor = HexToColor(color);
261                 outline.OutlineWidth = 0f; // Начинаем с 0 для плавного появления
262                 activeOutlines[markerObject] = outline;
```

Рисунок 2.28 – Программный код скрипта SecondCamera (метода ResetPreviousHover и ActivateHover)

Метод ResetPreviousHover отвечает за сброс подсветки и текста, если пользователь перестаёт наводить камеру на объект. Метод проверяет, есть ли в данный момент активный объект. Если такой объект есть, скрипт находит его индекс в списке IS_trigger_markerObjects и проверяет, существует ли связанный с ним объект из спика IS_object_complectObjects. Если всё в порядке, подсветка этого объекта отключается. Дополнительно метод проверяет попадает ли луч на кнопки.

Метод ActivateHover активирует подсветку и отображение текста для нового объекта, на который навёлся луч камеры. Метод ищет индекс объекта в

списке IS_trigger_markerObjects, находит связанный с ним объект из списка IS _object_complectObjects и и включает компонент подсветки (Outline).

В целом, эти два метода работают в паре, один отвечает за сброс предидущего состояния, драгой – за его активацию. Продолжение программного кода скрипта SecondCamera представлено на рисунке 2.29.

```
267     void UpdateActiveOutlines()
268     {
269         foreach (var kvp in activeOutlines.ToList())
270         {
271             GameObject complectObject = kvp.Key;
272             lightMesh.Outline outline = kvp.Value;
273 
274             bool isHovering = (currentlyHoveredObject != null &&
275                 IS_trigger_markerObjects.IndexOf(currentlyHoveredObject) ==
276                 Is_object_complectObjects.IndexOf(complectObject));
277 
278             if (isHovering)
279             {
280                 outline.OutlineColor = Color.Lerp(outline.OutlineColor, HexToColor(color), OutlineFadeInSpeed * Time.deltaTi
281                 outline.OutlineWidth = Mathf.Lerp(outline.OutlineWidth, 5f, OutlineFadeInSpeed * Time.deltaTime);
282             }
283             else
284             {
285                 outline.OutlineColor = Color.Lerp(outline.OutlineColor, new Color(0, 0, 0, 0), OutlineFadeOutSpeed * Time.de
286                 outline.OutlineWidth = Mathf.Lerp(outline.OutlineWidth, 0f, OutlineFadeOutSpeed * Time.deltaTime);
287 
288                 if (outline.OutlineWidth <= 0.1f)
289                 {
290                     outline.enabled = false;
291                     activeOutlines.Remove(complectObject);
292                 }
293             }
294         }
295     }
296 
        Ссылок: 3
297     private Color HexToColor(string hex)
298     {
299         hex = hex.Replace("#", "");
300         byte r = byte.Parse(hex.Substring(0, 2), System.Globalization.NumberStyles.HexNumber);
301         byte g = byte.Parse(hex.Substring(2, 2), System.Globalization.NumberStyles.HexNumber);
302         byte b = byte.Parse(hex.Substring(4, 2), System.Globalization.NumberStyles.HexNumber);
303 
304         return new Color(r / 255f, g / 255f, b / 255f);
```

Рисунок 2.29 – Программный код скрипта SecondCamera (метода UpdateActiveOutlines и HexToColor)

Метод UpdateActiveOutlines отвечает за обновление состояния подсветки контуров у всех активных объектов. Этот метод вызывается каждый кадр, чтобы сделать появление и исчезновение подсветки более плавной.

Метод HexToColor преобразует строку в формат HEX в Unity-цвет, разделяя компоненты на R, G и B. Этот метод используется, чтобы задавать целевой цвет подсветки для контуров [22]; [23]. Параметры скрипта SecondCamera представлены на рисунке 2.30.

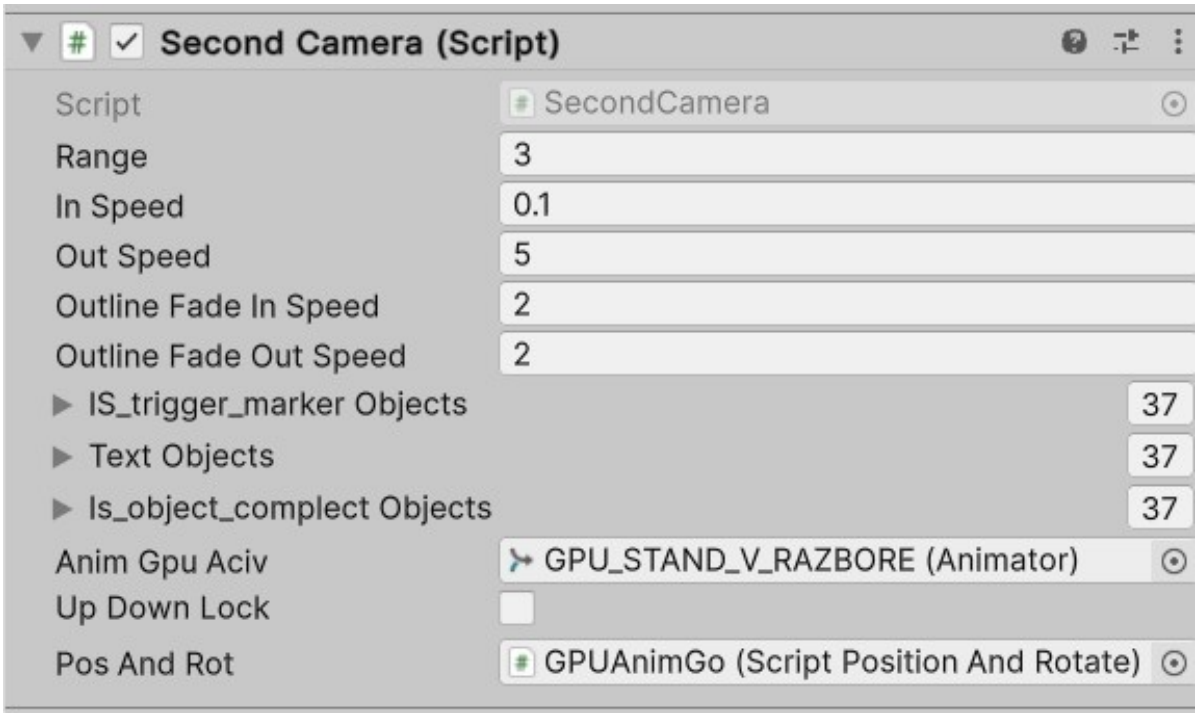

Рисунок 2.30 – Параметры скрипта SecondCamera

stendsGM – скрипт, позволяющий переключиться на другую камеру, а также обеспечивающий взаимодействие с этой камерой. Переключение между камерами происходит посредством взаимодействия пользователя со стендом, путем нажатия клавиши E. После этого действия камера игрока переключиться на камеру стенда. Камера стенда ограничена в пространстве и может передвигаться на ограниченное расстояние.

Важно отметить, что скрипт SecondCamera начинает работать только после переключения на вторую камеру. Соответственно, взаимодействии со стендом становится доступным только в этом режиме. Программный код скрипта stendsGM представлен на рисунке 2.31.

```
    void Update()
    {
        GameObject rotScriptObj = GameObject.Find("GPU_STAND");
        videoCard = rotScriptObj.GetComponent<BoxCollider>();

        if (pick == null)
        {
            RaycastHit hit;
            if (Physics.Raycast(Camera.main.transform.position, Camera.main.transform.forward, out hit, intRange))
            {
                if (hit.collider.CompareTag("gpuMotherStend"))
                {
                    if (Input.GetKeyDown(KeyCode.E))
                    {
                        if (cameraGpu != null && !cameraGpu.enabled)
                        {

                            cameraGpu.enabled = true;
                        }

                        if (otherScript != null && !otherScript.enabled)
                        {
                            otherScript.enabled = true;

                            if (rotScript != null && !rotScript.enabled && rotScriptObj != null)
                            {

                                rotScript.enabled = true;
                                videoCard.enabled = false;
                            }

                            ShowText("Кнопки для перемещения AA,DD\r\nНажмите EE что бы выйти из режима просмотра \r\n*Используйте колесо мыши");

                        }
                    }
                }
            }

        }

        if (isDisplaying)
        {
            timer -= Time.deltaTime;
            if (timer <= 0f)
            {
                HideText();
            }
        }
    }
    Ссылок 1
    public void ShowText(string message)
    {
        if (textElement != null)
        {
            textElement.text = message;
            textElement.gameObject.SetActive(true);
            timer = displayDuration;
            isDisplaying = true;
            key_A.SetActive(true);
            key_D.SetActive(true);
            key_E.SetActive(true);
        }
    }
    Ссылок 1
    public void HideText()
    {
        if (textElement != null)
        {
            textElement.gameObject.SetActive(false);
            isDisplaying = false;
            key_A.SetActive(false);
            key_D.SetActive(false);
```

Рисунок 2.31 – Программный код скрипта stendsGM

Принцип работы метода Update описан в описании данного скрипта. В скрипте также присутствует два метода ShowText и HideText.

Метод ShowText активирует элементы текста, устанавливает его содержимое, запускает таймер и показывает визуальные элементы. Когда таймер истекает в блоке if (isDisplaying) в методе Update, вызывается метод HideText, который скрывает текст и отключает визуальные элементы.

CameraLookOnly – скрипт, отвечающий за ограниченное перемещение камеры в пространстве и за стартовое положение камеры. Программный код скрипта CameraLookOnly представлен на рисунке 2.32.

```
Ссылок: 1
private void HandleCameraMovement()
{
    if (Input.GetKeyDown(KeyCode.D))
    {
        targetPosition = startPosition + new Vector3(2f, 0, 0); // Двигаемся вправо
        SwitchRotationScripts(false); // Выключаем GPU, включаем Motherboard
    }
    else if (Input.GetKeyDown(KeyCode.A))
    {
        targetPosition = startPosition; // Возвращаемся в исходное положение
        SwitchRotationScripts(true); // Включаем GPU, выключаем Motherboard
    }
    //перемещение камеры
    cameraGpu.transform.position = Vector3.Lerp(cameraGpu.transform.position, targetPosition, moveSpeed * Time.deltaTime);

    if (camerScript != null && camerScript.upDownLock == true)
    {
        if (Input.GetKeyDown(KeyCode.W))
        {
            targetPosition = startPosition + new Vector3(0, 0.3f, 0);
            /*SwitchRotationScripts(false); */
        }
        else if (Input.GetKeyDown(KeyCode.S))
        {
            targetPosition = startPosition;
            SwitchRotationScripts(true);
        }
    }
}
Ссылок: 3
private void SwitchRotationScripts(bool enableGPU)
{
    if (rotScriptGPU != null && rotScriptMotherboard != null)
    {
        rotScriptGPU.enabled = enableGPU;
        rotScriptMotherboard.enabled = !enableGPU;
    }
}
Ссылок: 1
private void HandleExitCamera()
{
    if (Input.GetKeyDown(KeyCode.E))
    {
        if (cameraGpu != null && cameraGpu.enabled == true)
        {
            cameraGpu.enabled = false;
        }
        if (myScript != null && myScript.enabled == true)
        {
            myScript.enabled = false;
            Debug.LogWarning("Выход из камеры 2 !");

            if (rotScriptGPU != null)
            {
                rotScriptGPU.enabled = false;
                Debug.LogWarning("Выкл поворот GPU");
            }
            if (rotScriptMotherboard != null)
            {
                rotScriptMotherboard.enabled = false;
                Debug.LogWarning("Выкл поворот Motherboard");
            }

            // Проверяем, что videoCard существует, и включаем его
            if (videoCard != null)
            {
                videoCard.enabled = true;
                Debug.Log("BoxCollider у видеокарты включен.");
            }
            else
```

Рисунок 2.32 – Программный код скрипта CameraLookOnly

Метод HandleCameraMovment отслеживает нажатие клавиш, когда игрок нажимает на клавишу D, камера смещается на 2 единицы расстояния от исходной позиции и вызывает метод который отключает возможность вращения GPU и включает вращение Motherboard. По нажатию клавиши A, камера возвращается в прежнее положение и обратно включает возможность вращать GPU. Кроме того, если активно свойство upDownLock, то при нажатии клавиши W, камера поднимается на 0.3 единицы расстояния вверх, а при S нажатии возвращается обратно.

Метод SwitchRotationScripts просто включает и выключает соответствующие скрипты вращения GPU и Motherboard в зависимости от переданного флага.

Метод HandleExitCamera отвечает за выход из режима камеры при нажатии на клавишу E.

VideocartaANDMotherScriptRot – скрипт, позволяющий поворачивать объект вокруг своей оси с помощью колёсика мыши. Скрипт работает если добавить его в виде компонента на объект [24]. Программный код скрипта VideocartaANDMotherScriptRot представлен на рисунке 2.33.

```
void Start()
{
    // Установка начальной ориентации объекта
    transform.rotation = Quaternion.Euler(0, currentYRot, 0);
}

Сообщение Unity | Ссылок: 0
void Update()
{
    rotaitComplect();
}

Ссылок: 1
void rotaitComplect()
{
    float scrollInput = Input.GetAxis("Mouse ScrollWheel");

    if (scrollInput != 0)
    {
        currentYRot += scrollInput * rotaiteSpeed;
    }

    if (currentYRot > maxYRot)
    {
        currentYRot = Mathf.MoveTowards(currentYRot, maxYRot, Time.deltaTime * 5f);
    }
    else if (currentYRot < minYRot)
    {
        currentYRot = Mathf.MoveTowards(currentYRot, minYRot, Time.deltaTime * 5f);
    }

    currentYRot = Mathf.Clamp(currentYRot, minYRot, maxYRot);
    transform.rotation = Quaternion.Euler(0, currentYRot, 0);
}
```

Рисунок 2.33 – Программный код скрипта VideocartaANDMotherScriptRot

Результат работы программного кода представлен на рисунке 2.34.

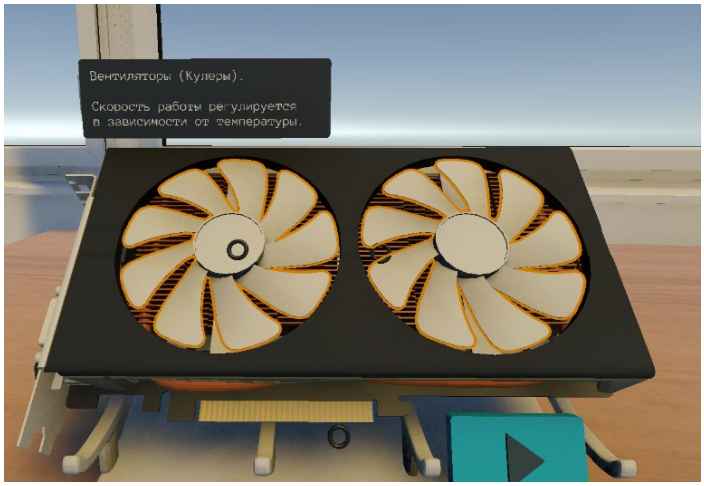

Рисунок 2.34 – Результат работы программного кода (GPU)

Наводя взгляд на специальные триггеры, можно заметить, что они представленны ввиде небольших чёрных круглых фигур. При наведении на триггер подсвечивается объект, к которому он относиться, а так же отображается описание этого объекта. Продолжение результата работы программного кода представленно на рисунке 2.35.

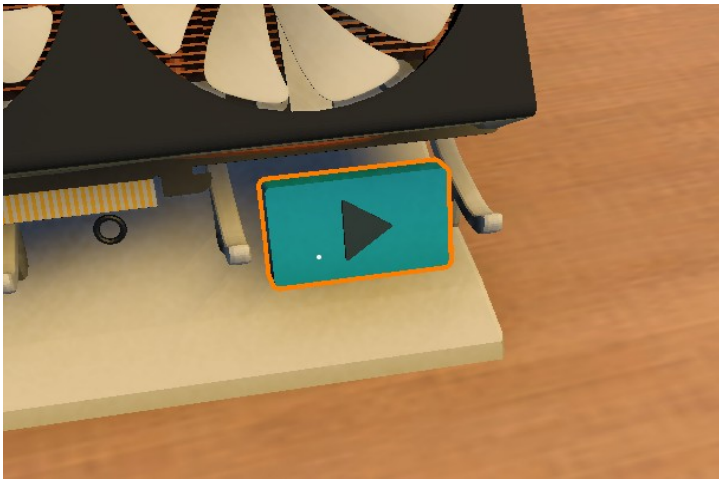

Рисунок – 2.35 Результат работы программного кода (кнопка для запуска анимации)

На рисунке 2.35 представлена кнопка запуска анимации, после нажатия данная модель исчезает и появляется другая, у которой есть анимация, анимация сразу запускается. Результат работы программного кода представлен на рисунке 2.36.

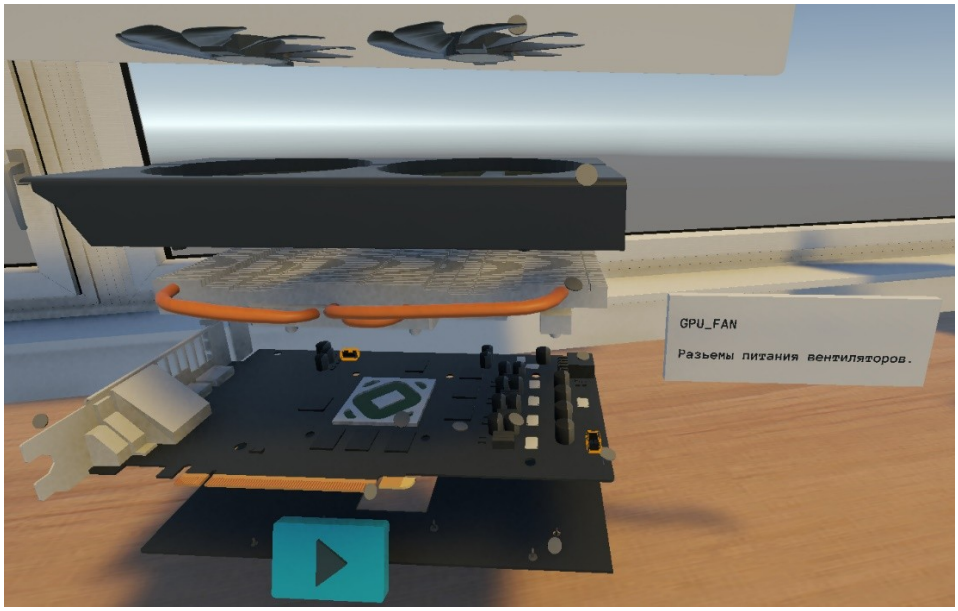

Рисунок 2.36 – Результат работы программного кода (GPU в разборе)

На рисунке 2.36 представлена видеокарта в разобранном виде. После нажатия на соответствующую кнопку (представленную на рисунке 2.35) запускается анимация разбора видеокарты, которая позволяет наглядно продемонстрировать последовательность отделения её компонентов. У разобранной видеокарты также присутствуют триггеры для подсвечивания и просмотра деталей видеокарты, они представлены в виде круглых фигур белого цвета. Кроме того, видеокарту так же, как и видеокарту в собранном состоянии можно поворачивать вокруг своей оси при помощи колёсика мыши, а также перемещать камеру вверх и вниз в ограниченном пространстве, для детального рассмотрения видеокарты. Продолжение результата работы программного кода представлено на рисунке 2.37.

Рисунок 2.37 – Результат работы программного кода (Motherboard)

При нажатии на кнопку D происходит перемещение к материнской плате и можно детально рассмотреть её, по тому же принципу что и видеокарту.

Следующий скрипт называется PANEL_CONNECT_Scripts. Данный скрипт отвечает за запуск анимации, всего можно запустить 10 анимаций, нажав на кнопку анимация запускается, нажав повторно на туже кнопку анимация ставится на паузу, если нажать вновь, анимация продолжает

проигрываться, это важно для полного понимания как взаимодействовать с предметом.

Анимации запускаемые скриптом:

- анимация установки процессора в сокет от компании AMD;
- анимация установки процессора в сокет от компании Intel;
- анимация подключения передней панели компьютера;
- анимация установки видеокарты в материнскую плату;
- анимация установки радиатора на материнскую плату;
- анимация подключения разъемов USB 3.0, 2.0;
- анимация подключения питания процессора и материнской платы;
- три анимации твердотельных накопителей.

Программный код скрипта PANEL_CONNECT_Scripts представлен на рисунке 2.38.

```
//луч
if (mainCamera != null)
{
    RaycastHit hit;

    // Проверка, попадает ли луч
    if (Physics.Raycast(Camera.main.transform.position, Camera.main.transform.forward, out hit, intRange))
    {
        // Проверка тега
        if (hit.collider != null && hit.collider.CompareTag("buttonStend"))
        {
            //компоненты Renderer и Outline
            Renderer renderer = hit.collider.GetComponent<Renderer>();
            Outline outline = hit.collider.GetComponent<Outline>();

            if (renderer != null && outline != null)
            {
                // Если это новый объект, отличный от предыдущего
                if (renderer != lastRenderer)
                {
                    ResetHighlight(); // Сбросить подсветку предыдущего объекта

                    // Подсветить текущий объект
                    lastRenderer = renderer;
                    lastOutline = outline;

                    // Устанавливаем цвет контура
                    lastOutline.OutlineColor = HexToColor("82D8FF");
                    lastOutline.OutlineWidth = 10f;
                    lastOutline.enabled = true;
                }
            }

            if (Input.GetMouseButtonDown(0) && hit.collider.gameObject == button_CPU)
            {
                AnimControll(animCPU, ref FlagsCPU);

            }
            if (Input.GetMouseButtonDown(0) && hit.collider.gameObject == button_CPU)...
            if (Input.GetMouseButtonDown(0) && hit.collider.gameObject == button_frontPanel)...
            if (Input.GetMouseButtonDown(0) && hit.collider.gameObject == button_GPU)...
            if (Input.GetMouseButtonDown(0) && hit.collider.gameObject == button_MotherAndCuller)...
            if (Input.GetMouseButtonDown(0) && hit.collider.gameObject == button_PSCCASEMother)...
            if (Input.GetMouseButtonDown(0) && hit.collider.gameObject == button_PSUCableMother)...
            if (Input.GetMouseButtonDown(0) && hit.collider.gameObject == button_RAM)...
            if (Input.GetMouseButtonDown(0) && hit.collider.gameObject == button_Crimper)...
            if (Input.GetMouseButtonDown(0) && hit.collider.gameObject == button_SASDISK)...

            if (Input.GetMouseButtonDown(0) && stateCPU == false && hit.collider.gameObject == button_SvapCPU)...
            else if (Input.GetMouseButtonDown(0) && stateCPU == true && hit.collider.gameObject == button_SvapCPU)...
        }
```

Рисунок 2.38 – Программный код скрипта PANEL_CONNECT_Scripts

Код работает по тому же принципу луча из камеры, луч попадает на объект с тегом buttonStend и по нажатию левой кнопки мыши на кнопку, запускается анимация. Параметры крипта PANEL_CONNECT_Scripts представлены на рисунке 2.39.

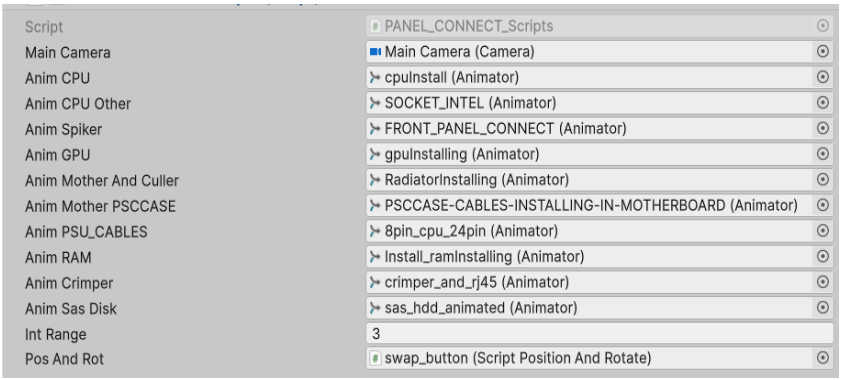

Рисунок 2.39 – Параметры крипта PANEL_CONNECT_Scripts

Следующий скрипт называется AudioScript. Данный скрипт отвечает за запуск аудиоинформации для определённого стенда или анимации. Звуковое сопровождение есть практически у каждой модели в виртуальной лаборатории. Всего было записано 30 аудиодорожек. Пример кода скрипта AudioScript представлен на рисунке 2.40.

```
        AudioPlayAndStop(SecondRoom_Keyboards_Stend_Membran_Audio, ref SecondRoom_Keyboards_Stend_Membran, SecondRoom_Keyboards_Stend_Membran_audio_button);
    }
    else if (IsClickMouseAnColliderWhy(Rom.hit.collider, glos_BIOS_PC))
    {
        if (AreAllAudioFlagsFalseExcept(is_glos_BIOS_PC))
        {
            Debug.Log("Глоссарий: BIOS ПК");
        }
        AudioPlayAndStop(audio_glos_BIOS_PC, ref is_glos_BIOS_PC, glos_BIOS_PC);
    }
    else if (IsClickMouseAnColliderWhy(Rom.hit.collider, glos_PcIe))
    {
        if (AreAllAudioFlagsFalseExcept(is_glos_PcIe))
        {
            Debug.Log("Глоссарий: PCIe");
        }
        AudioPlayAndStop(audio_glos_PcIe, ref is_glos_PcIe, glos_PcIe);
    }
    else if (IsClickMouseAnColliderWhy(Rom.hit.collider, glos_Lan))
    {
        if (AreAllAudioFlagsFalseExcept(is_glos_Lan))
        {
            Debug.Log("Глоссарий: LAN");
        }
        AudioPlayAndStop(audio_glos_Lan, ref is_glos_Lan, glos_Lan);
    }
    else if (IsClickMouseAnColliderWhy(Rom.hit.collider, glos_Tranz))
    {
        if (AreAllAudioFlagsFalseExcept(is_glos_Tranz))
        {
            Debug.Log("Глоссарий: Транзистор");
        }
        AudioPlayAndStop(audio_glos_Tranz, ref is_glos_Tranz, glos_Tranz);
    }
    else if (IsClickMouseAnColliderWhy(Rom.hit.collider, glos_Sistemoxl))
    {
        if (AreAllAudioFlagsFalseExcept(is_glos_Sistemoxl))
        {
            Debug.Log("Глоссарий: Системная плата");
        }
        AudioPlayAndStop(audio_glos_Sistemoxl, ref is_glos_Sistemoxl, glos_Sistemoxl);
    }
    else if (IsClickMouseAnColliderWhy(Rom.hit.collider, glos_ROM))
    {
        if (AreAllAudioFlagsFalseExcept(is_glos_ROM))
        {
            Debug.Log("Глоссарий: ПЗУ");
        }
        AudioPlayAndStop(audio_glos_ROM, ref is_glos_ROM, glos_ROM);
    }
```

Рисунок 2.40 – Пример кода скрипта AudioScript

На рисунке 2.40 представлены условия запуска аудиодорожки при нажатии на кнопку, при повторном нажатии аудиодорожка ставится на паузу. Основные методы скрипта AudioScript представлены на рисунке 2.41.

```
Ссылок: 66
public bool IsClickMouseAnColliderWhy(Collider hitCollider, GameObject targetObj)
{
    if (Input.GetMouseButtonDown(0))
    {
        if (hitCollider != null && hitCollider.gameObject == targetObj)
        {
            return true;
        }
    }
    return false;
}

Ссылок: 65
public void AudioPlayAndStop(AudioSource audio, ref bool audioFlag, GameObject obj)
{
    if (IsClickMouseAnColliderWhy(Rom.hit.collider, obj))
    {
        if (audio != null)
        {
            if (!audioFlag)
            {
                // Останавливаем все другие аудио
                StopAllAudio();

                // Включаем текущее аудио
                audio.enabled = true;
                if (!audio.isPlaying) audio.Play();
                audioFlag = true;
            }
            else
            {
                audio.Pause();
                audioFlag = false;
            }
        }
    }
}

Ссылок: 1
private bool AreAllAudioFlagsFalse()...
Ссылок: 64
private bool AreAllAudioFlagsFalseExcept(params bool[] excludedFlags)...
Ссылок: 1
private void StopAllAudio()...
```

Рисунок 2.41 – Основные методы скрипта AudioScript

Метод IsClickMouseAnColliderWhy проверяет действительно ли клик произошёл по нужному объекту.

Метод AudioPlayAndStop решает, нужно ли включить или приостановить аудио, если оно ещё не проигрывается, сначала вызывается StopAllAudio, чтобы остановить все остальные звуки, и только потом включает

новый звук.

Метод AreAllAudioFlagsFalse и AreAllAudioFlagsFalseExcept проверяют не активен ли аудиофлаг.

Метод StopAllAudio проходит по аудиодорожкам и ставит их на паузу. Кнопка запуска аудио информации представлена на рисунке 2.42.

Рисунок 2.42 – Кнопка запуска аудио информации

Параметры скрипта AudioScript представлены на рисунке 2.43.

Audio_glos_BIOS_PC	glos_BIOS_PC (Audio Source)
Audio_glos_Pc Ie	glos_Pcle (Audio Source)
Audio_glos_Lan	glos_Lan (Audio Source)
Audio_glos_Tranz	glos_Tranz (Audio Source)
Audio_glos_Sistemoxl	glos_Sistemoxl (Audio Source)
Audio_glos_ROM	glos_ROM (Audio Source)
Audio_glos_OS	glos_OS (Audio Source)
Audio_glos_RAM	glos_RAM (Audio Source)
Audio_glos_BP	glos_BP (Audio Source)
Audio_glos_Perf Ustr	glos_PerfUstr (Audio Source)
Audio_two Glos_fire Wall	twoGlos_fireWall (Audio Source)
Audio_two Glos_DNS	twoGlos_DNS (Audio Source)
Audio_two Glos_ETHER	twoGlos_ETHER (Audio Source)
Audio_two Glos_DHCP	twoGlos_DHCP (Audio Source)
Audio_two Glos_MA Cadres	twoGlos_MACadres (Audio Source)
Audio_two Glos_FTP	twoGlos_FTP (Audio Source)
Audio_two Glos_Ip Adres	twoGlos_IpAdres (Audio Source)
Audio_two Glos_LAN	twoGlos_LAN (Audio Source)
Audio_two Glos_OSI Model	twoGlos_OSIModel (Audio Source)
Audio_two Glos_Routing	twoGlos_Routing (Audio Source)
Audio_ohlad_main	play_main_ohlad (Audio Source)
Audio_Vodynka	play_button_vodyanka (Audio Source)
Audio_Standart Ohlad	play_button_vozduh_ohlad (Audio Source)
Audio_passiv Ohlad	play_passive_ohlad (Audio Source)
Audio_termoint	termo_button_audio (Audio Source)
Sas_disk	Disk_SAS_HDD_audio_button (Audio Source)
Topology_ring	topology_ring_button_audio_button (Audio Source)
Commutator_audio	commut_Audio_button (Audio Source)

Рисунок 2.43 – Параметры скрипта AudioScript

Следующий скрипт называется CombinedTestSystem. Данный скрипт отвечает за прохождение тестов в виртуальной лаборатории. Кнопка запуска теста представлена на рисунке 2.44.

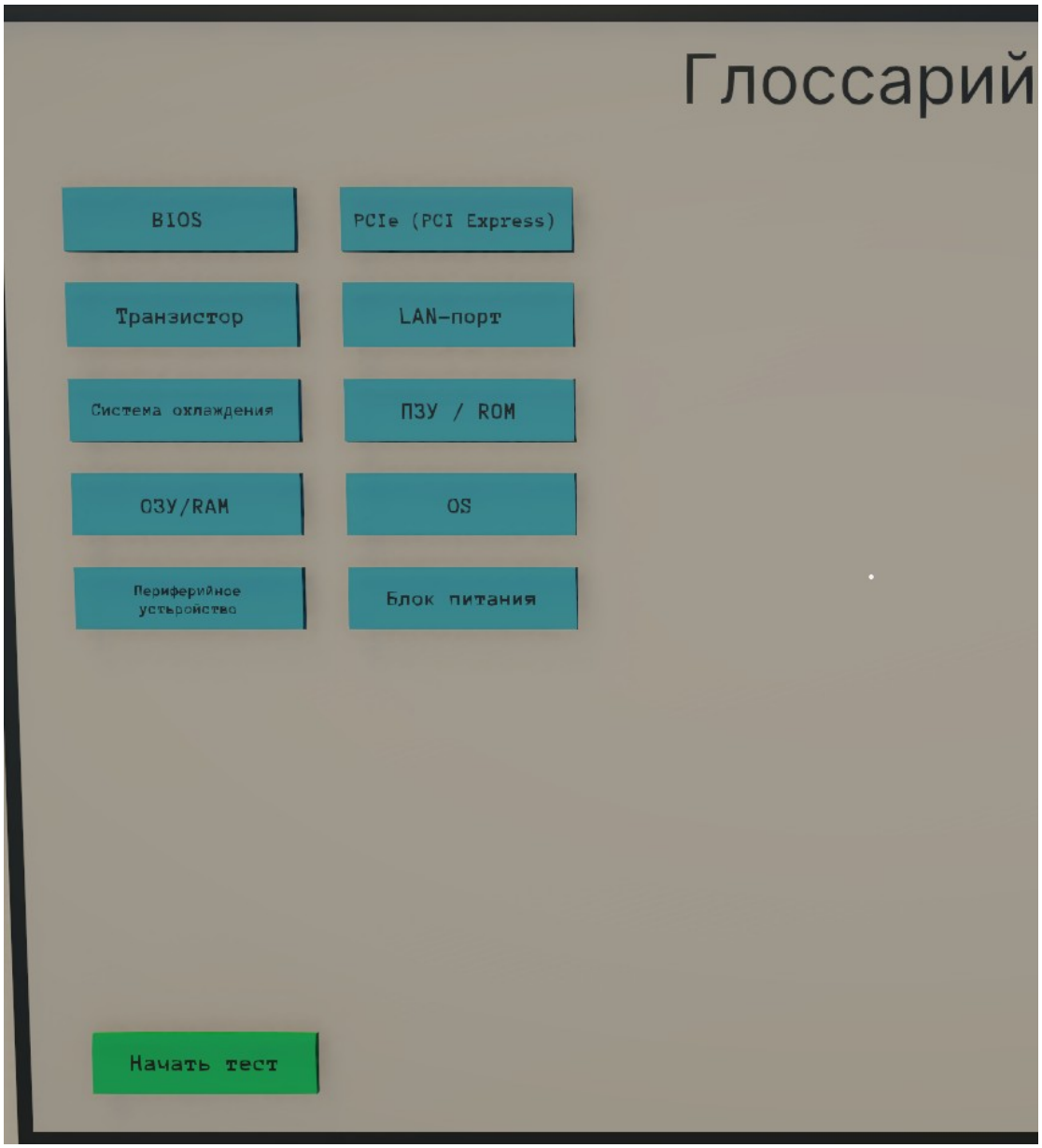

Рисунок 2.44 – Кнопка запуска теста

На рисунке 2.44 представлен глоссарий с кнопками. Кнопки голубого цвета отвечают за озвучку основных понятий, не озвученных у других стендов или объектов. Тест, запускается по нажатию на кнопку зелёного цвета. Тест оценивается по 100 бальной системе или в процентном соотношении, тест проходится посредством нажатия на номер ответа – номер клавиши, и нажатием клавиши Enter. По завершению теста рассчитывается текущие балл,

количество попыток и средний балл всех попыток [25]. Работа теста представлена на рисунке 2.45.

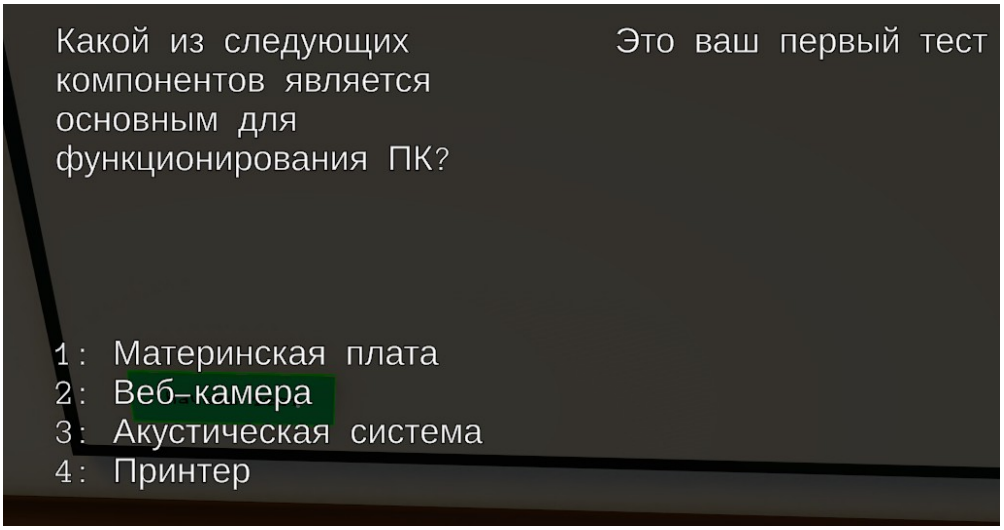

Рисунок 2.45 – Работа теста

Результаты прохождения теста представлены на рисунке 2.46.

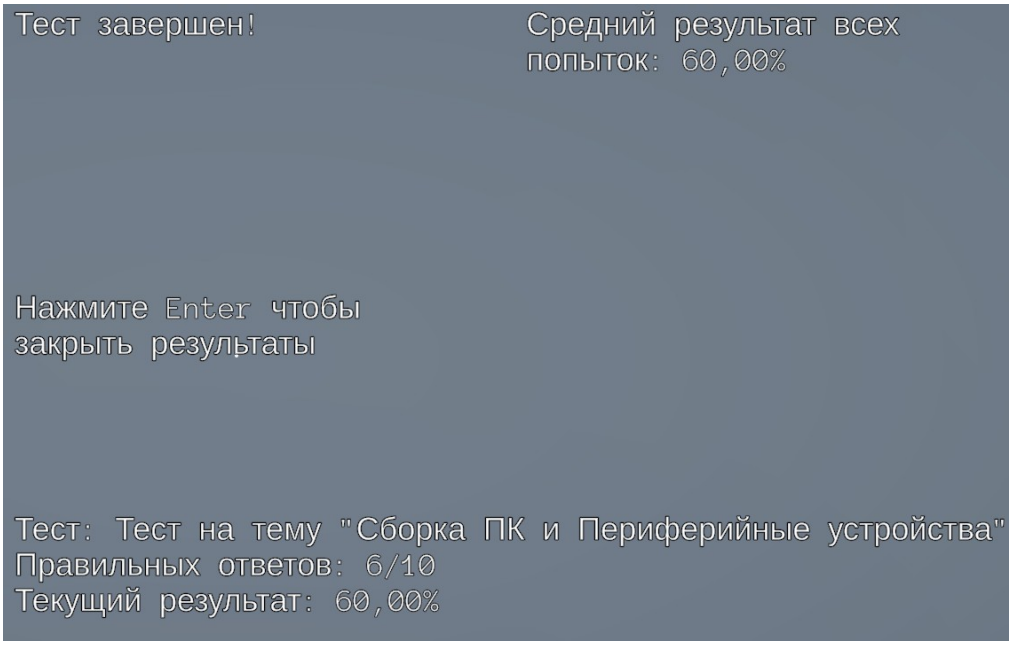

Рисунок 2.46 – Результаты прохождения теста

2.7 Расположение объектов в сцене Unity

После завершения всех подготовительных этапов – создания объектов, анимации и скриптов, важно правильно разместить объекты в сцене Unity,

чтобы обеспечить корректное взаимодействие с ними. Общий план размещения объектов представлен на рисунке 2.47.

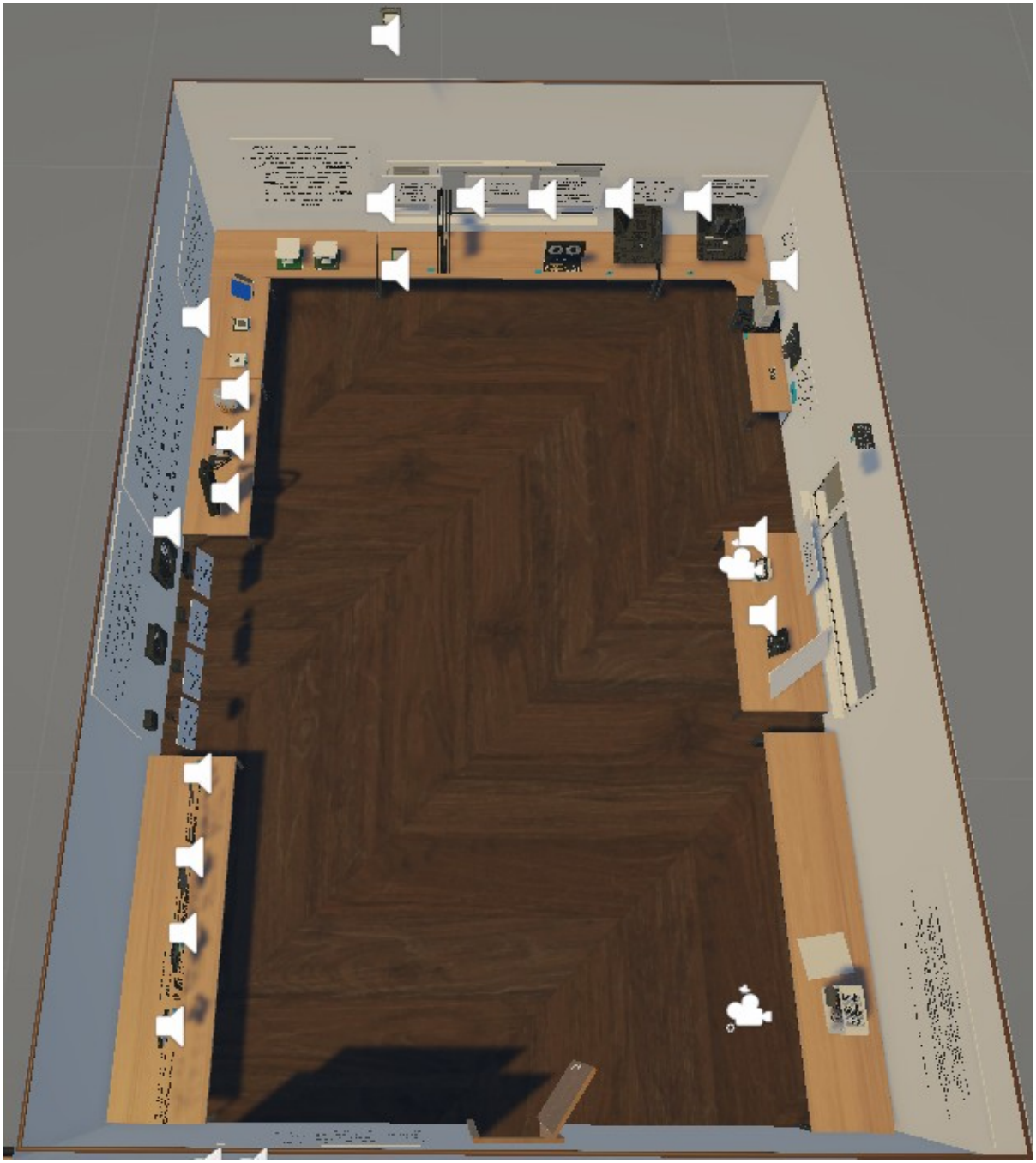

Рисунок – 2.47 Общий план размещения объектов в сцене

Объекты размещены в сцене по степени сложности реализации, слева направо: сначала расположены объекты, которые можно описать текстом и простой анимацией, затем – предметы с более сложной анимацией. Далее следует для детального осмотра видеокарты и материнской платы, а завершающим этапом является сборка ПК. У каждого объекта предусмотрено

два типа описания: текстовое и аудио. Детально расположение объектов в сцене представлено на рисунке 2.48.

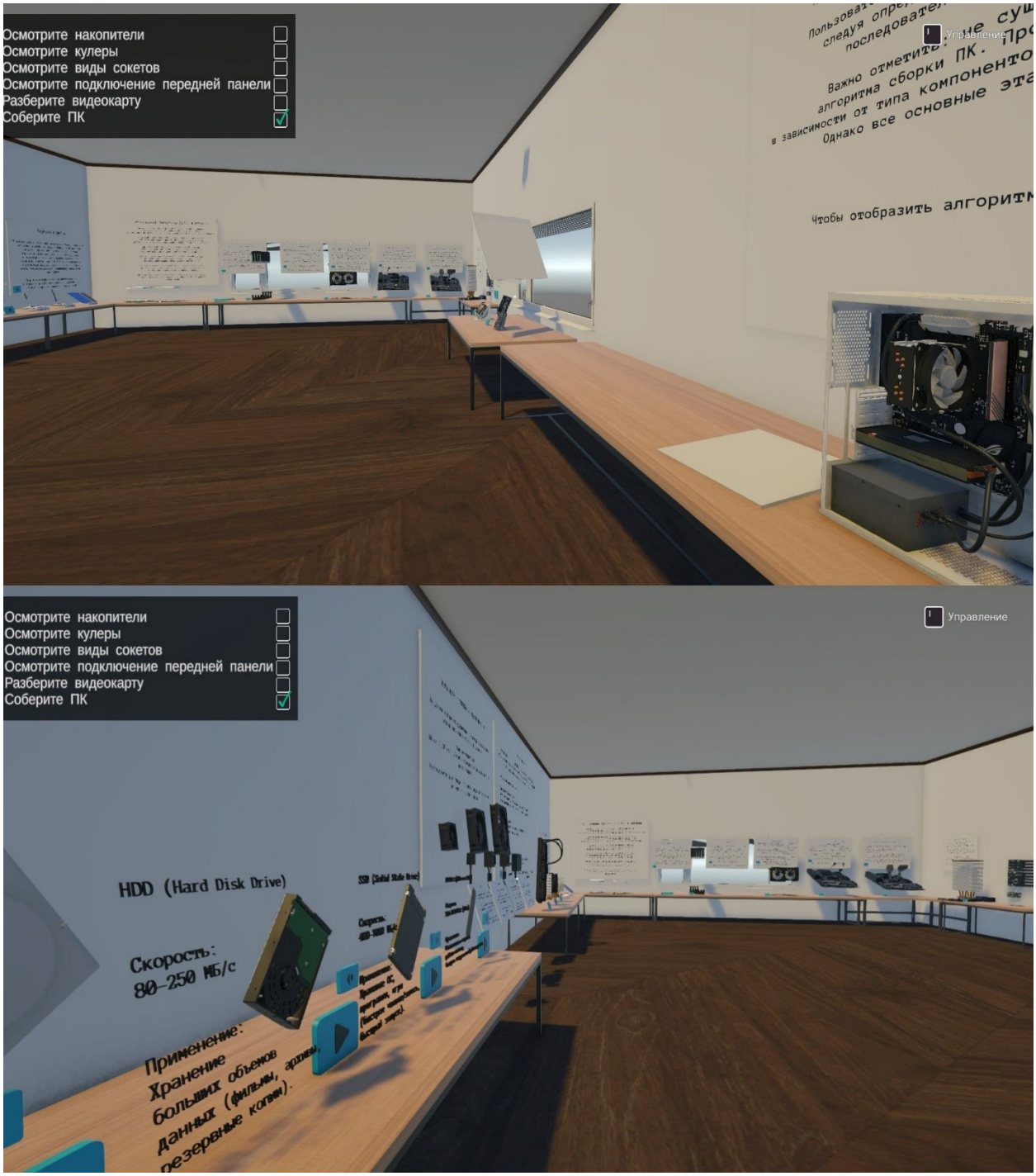

Рисунок – 2.48 Детальное расположение объектов в сцене

3 Оценка эффективности внедрения ИС

3.1 Расчёт экономических затрат

Состав сотрудников приведён в таблице 3.1

Месячный оклад сотрудников рассчитывается по формуле 3.1:

$$Ом = Omin \cdot Km \quad (3.1)$$

где, Ом – месячный оклад;

Omin – минимальная заработная плата в организации;

Кт – тарифный коэффициент (преподаватель – 1,43; ст. преподаватель – 1,51; доцент, к.н. – 2,94). Для руководителя: Ом = 40460*1,43 = 57857 руб. Для техника по ИС: Ом = 23548*1,01 = 23 783 руб.

Таблица 3.1 – Состав сотрудников

Наименование	Численность	Тарифный разряд	Месячный оклад, Руб.
Преподаватель	1	1,43	57857
Техник по ИС	1	1,01	23 783

В таблице 3.2 представлены необходимые трудовые затраты на разработку программного обеспечения.

Таблица 3.2 – Расчёт трудоёмкости выполнения работ

Наименование работ	Тмин	Тмакс	Тр	Преподаватель	Техник по ИС
Анализ предметной области	10	13	11,8	0	11,8
Изучение задания	8	12	10,4	2,4	8
Подбор и изучение списка литературы	6	8	7,2	0	7.2
Обоснование проектных решений	8	11	9,8	4	5,8
Работа в ПО 3D Blender	30	40	36	0	36
Создание анимации в 3D Blender	15	25	21	0	21
Изучение и разработка с помощью плагина Playmaker	15	20	18	0	18
Тестирование Playmaker	1	2	1,6	0	1,6

Написание скриптов	50	60	56	0	56

Продолжение таблицы 3.2

Расстановка объектов в сцене	2	4	3,2	0	3,2
Отладка, тестирование, корректировка, обнаружение и исправление ошибок	5	10	8	3	5
Ожидание выполнения программы (за всё время)	1	3	2,2	1,1	1,1
Оформление пояснительной записки проекта	15	20	18	0	18
Всего	166	228	203,2	15,1	192,7

Трудоёмкость выполнение работ следует рассчитывать, используя формулу (3.2):

$$Tp = (3 \cdot tmin + 2 \cdot tmax)/5 \quad (3.2)$$

где, Тр – расчётная трудоёмкость выполнения работы;

t min – минимальное время, необходимое для выполнение работы;

t max – максимальное время, необходимое для выполнение работы.

Расчётная трудоёмкость выполнения работы = (3*166+2*228)/5 = 190,8

Таблица 3.3 – Сумма затрат

№	Наименование стати затрат	Буквенное обозначение	Формула	Сумма, руб.
1	Зарплата техника по ИС	Зптех	Тр*Ом /(21*8)	27279,24
2	Зарплата руководителя	Зпрук	Тр*Ом /(21*8)	3616,76
3	Итого выплаты	ЗПпо	∑ всех з/п	30896
4	Премия	П	ЗПпо*0,22	6797,12
5	Выплаты по районному коэффициенту	Врк	(ЗПпо+П)*0,15	5663,968
6	Общий фонд оплаты труда	ФОТоб	ЗПпо+П+Врк	43347,088
7	Единый социальный налог	ЕСН	ФОТоб*0,356	15431,56333
8	Накладные расходы	НР	ЗПпо*1,2	36075,2
9	Итого затрат	Зпо	ФОТоб+ЕСН+НР	95853,85133
10	Затраты, связанные с Работой компьютера при разработке	Зком	Тр*оклад/(8*21)	27010,125
11	Прочие затраты связанные с разработкой ПО	Зпр	Зпо*0,35	12976,32
12	Итого затрат на разработку ПО	Зрп	Зпо+Зком+Зпр	135840,2963
13	Налоги, включаемые в	Нсп	ФОТоб*0,1	4334,7088

	затраты			

Продолжение таблицы 3.3

14	Затраты на оформление программного обеспечения	Зоф	Зпр*0,15	1946,488
15	Всего затрат на создание ПО	Зсп	Зрп+Нсп+ЗоФ	142121,4531

Проведя расчёт суммы затрат на разработку виртуальной лаборатории (на примере Рубцовского Института (филиала) АлтГУ) примерная сумма затрат составит 142121 руб.

3.2 Оценка педагогической эффективности

Внедрение разрабатываемой виртуальной лаборатории по дисциплине «Архитектура аппаратных средств» может обеспечить педагогическую эффективность. Благодаря лаборатории преподаватель сможет повысить эффективность изучения материала, за счёт встроенных испытаний и модуля контроля знаний в лаборатории. Модуль контроля знаний проверяет усвоенный материал студентами через систему тестирования, оценка за которую будет одним из определённых этапов обучения.

Ожидаемые педагогические эффекты:

- повышенный интерес к изучению дисциплины;
- углубление в дисциплину за счет механик геймификации;
- сокращение времени на преподавание за счёт материала в лаборатории;
- положительные результаты выполнения лабораторных работ.

3.3 Оценка социальной эффективности

Предполагается, что внедрение разрабатываемой виртуальной лаборатории по дисциплине «Архитектура аппаратных средств» позволит привлечь интерес, вовлечь студентов в образовательный процесс.

Внедрение элементов геймификации в образовательный процесс положительным образом влияет на качество обучения. Обучающийся с использованием элементов геймификации лучше понимает и запоминает предоставленную информацию, поскольку информация преподносится в более дружелюбном и простом ключе, игровом формате.

Ярким примером успешного процесса геймификации в образовательной среде является браузерное приложение под названием «CodeMonkey».

Данное браузерное приложение использует игровые элементы для изучения программирования, изучение проходит посредством перемещения игрового персонажа к точкам интереса используя языки программирования.

ЗАКЛЮЧЕНИЕ

Результатом выпускной квалификационной работы является разработанная виртуальная лаборатория по дисциплине «Архитектура аппаратных средств» (на примере Рубцовского института (филиал) АлтГУ).

Предметом являлась геймификация процесса обучения по дисциплине «Архитектура аппаратных средств».

Целью являлась разработка виртуальной лаборатории по дисциплине «Архитектура аппаратных средств» на примере Рубцовского института (филиала) АлтГУ.

Для достижения поставленной цели были решены следующие задачи:

- проведен анализ предметной области;
- выработаны проектные решения по обеспечивающим системам
- реализованы проектные решения разрабатываемой виртуальной лаборатории по дисциплине «Архитектура аппаратных средств»;
- рассчитаны экономические затраты на разработку виртуальной лаборатории по дисциплине «Архитектура аппаратных средств».

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Рубцовский институт (филиал) АлтГУ: Сведения об образовательной организации [Электронный ресурс]. – Режим доступа: https://rb.asu.ru/_- Загл. С экрана.

2. IDEF0. Знакомство с нотацией и пример использования [Электронный ресурс]. – Режим доступа: https://kurl.ru/IdLZe - Загл. С экрана.

3. Везиров, Т. Г. Курс лекций по дисциплине «Визуализация и геймификация в образовании»: учебное пособие / Т. Г. Везиров. – Махачкала: ДГПУ, 2024. – 72 с. – Текст: электронный // Лань: электронно-библиотечная система. – URL: https://e.lanbook.com/book/442670 – Режим доступа: для авториз. пользователей.

4. Что такое Unreal Engine: история развития, преимущества и ключевые функции [Электронный ресурс]. – Режим доступа: https://skillbox.ru/media/gamedev/chto–takoe-unreal-engine-istoriya-razvitiya-preimushchestva-i-klyuchevye-funktsii/ - Загл. С экрана.

5. Движок Unity: особенности, преимущества и недостатки [Электронный ресурс]. – Режим доступа: https://cubiq.ru/dvizhok-unity/ - Загл. С экрана.

6. Blender против Maya: 7 основных моментов, которые следует учитывать [Электронный ресурс]. – Режим доступа: https://kurl.ru/Mswdf - Загл. С экрана.

7. Unity: Вперёд к разработке [Электронный ресурс]. – Режим доступа: https://unity.com/ru_- Загл. С экрана.

8. Visual Studio: Бесплатная версия Visual Studio [Электронный ресурс]. – Режим доступа: https://visualstudio.microsoft.com/ru/ - Загл. С экрана.

9. Blender: Blender 4.4 [Электронный ресурс]. – Режим доступа: https ://www.blender.org/ - Загл. С экрана.

10. Читайло, К. С. Трехмерное моделирование в программной среде

Blender: учебное пособие / К. С. Читайло. – Новокузнецк: КГПИ КемГУ, 2024. – 122 с. – Текст: электронный // Лань: электронно-библиотечная система. – URL: https://e.lanbook.com/book/451982 – Режим доступа: для авториз. пользователей.

11. Анимация персонажа: учебное пособие / составитель Н. А. Саблина; под редакцией Н. Я. Безбородова, Н. В. Стюфляева. – Липецк: Липецкий ГПУ, 2018. – 55 с. – Текст: электронный // Лань: электронно-библиотечная система. – URL: https://e.lanbook.com/book/115019 – Режим доступа: для авториз. пользователей.

12. Хэсс, Ф. Практическое пособие. Blender 3.0 для любителей и профессионалов. Моделинг, анимация, VFX, видеомонтаж: учебное пособие / Ф. Хэсс. – Москва: СОЛОН-Пресс, 2022. – 300 с. – Текст: электронный // Лань: электронно-библиотечная система. – URL: https://e.lanbook.com/book/322268 – Режим доступа: для авториз. пользователей.

13. Лаборатория линуксоида: F-кривые анимации в Blender. Интерполяция [Электронный ресурс]. – Режим доступа: https://younglinux.info/animation/f-curve - Загл. С экрана.

14. Introduction to 2D-animation working practice: frames per second [Электронный ресурс]. – Режим доступа: https://booksite.elsevier.com/samplechapters/9780240520544/9780240520544.pdf - Загл. С экрана.

15. Яндекс практикум | Блог: как делать анимацию в Blender [Электронный ресурс]. – Режим доступа: https://practicum.yandex.ru/blog/kak-delat-animaciyu-v-blender/ - Загл. С экрана.

16. Хабр: Издательство дом «Питер». Базовые концепции Unity для программистов [Электронный ресурс]. – Режим доступа: https://habr.com/ru/companies/piter/articles/529648/ - Загл. С экрана.

17. Хабр: Создание игр на Unity: с чего начать? [Электронный ресурс]. – Режим доступа: https://habr.com/ru/articles/828302/- Загл. С экрана.

18. Разработка 3D-игр в Unity: руководство / Э. Дэвис, Т. Батист, Р.

Крейг, Р. Станкел; перевод с английского П. М. Бомбаковой. – Москва: ДМК Пресс, 2023. – 298 с. – Текст: электронный // Лань: электронно-библиотечная система. – URL: https://e.lanbook.com/book/456614 – Режим доступа: для авториз. пользователей.

19. UNITY PLAYMAKER ПО-РУССКИ: Обзор интерфейсов PlayMaker [Электронный ресурс]. – Режим доступа: http://unityplaymakers.ru/interfejs-playmaker/ - Загл. С экрана.

20. Тренинг GameHub: Разработка компьютерных игр в Unity 3D. Основы визуального программирования в PlayMaker Unity 3D [Электронный ресурс]. – Режим доступа: https://gamehub-cbhe.deusto.es/wp-content/uploads/2016/04/GameHub_ONPU_Local_Training_GameDevelopmentinUnity3D_Part2_Presentation.pdf - Загл. С экрана.

21. Unity Documentation: Create witch Unity – Tags [Электронный ресурс]. – Режим доступа: https://docs.unity3d.com/2023.2/Documentation/Manual/Tags.html - Загл. С экрана.

22. UnityHub: Представление цветов RGBA [Электронный ресурс]. – Режим доступа: https://unityhub.ru/scripting/Color - Загл. С экрана.

23. UnityHub: Редактирование свойств [Электронный ресурс]. – Режим доступа: https://unityhub.ru/manual/EditingValueProperties - Загл. С экрана.

24. Кудрина, Е. В. Основы алгоритмизации и программирования на языке C#: учебник для вузов / Е. В. Кудрина, М. В. Огнева. – Москва: Издательство Юрайт, 2025. – 322с. – (Высшее образование). – Текст: электронный // Образовательная платформа Юрайт [сайт]. – URL: https://urait.ru/bcode/565466

25. Хабр: Книга «Unity в действии. Мультиплатформенная разработка на C#. 3-е межд. Издание» [Электронный ресурс]. – Режим доступа: https://habr.com/ru/companies/piter/articles/747578/ - Загл. С экрана.

ПРИЛОЖЕНИЕ А

Диаграмма «как есть»

Рисунок А1 – Декомпозиция бизнес-процессов «как есть»

Диаграмма «как будет»

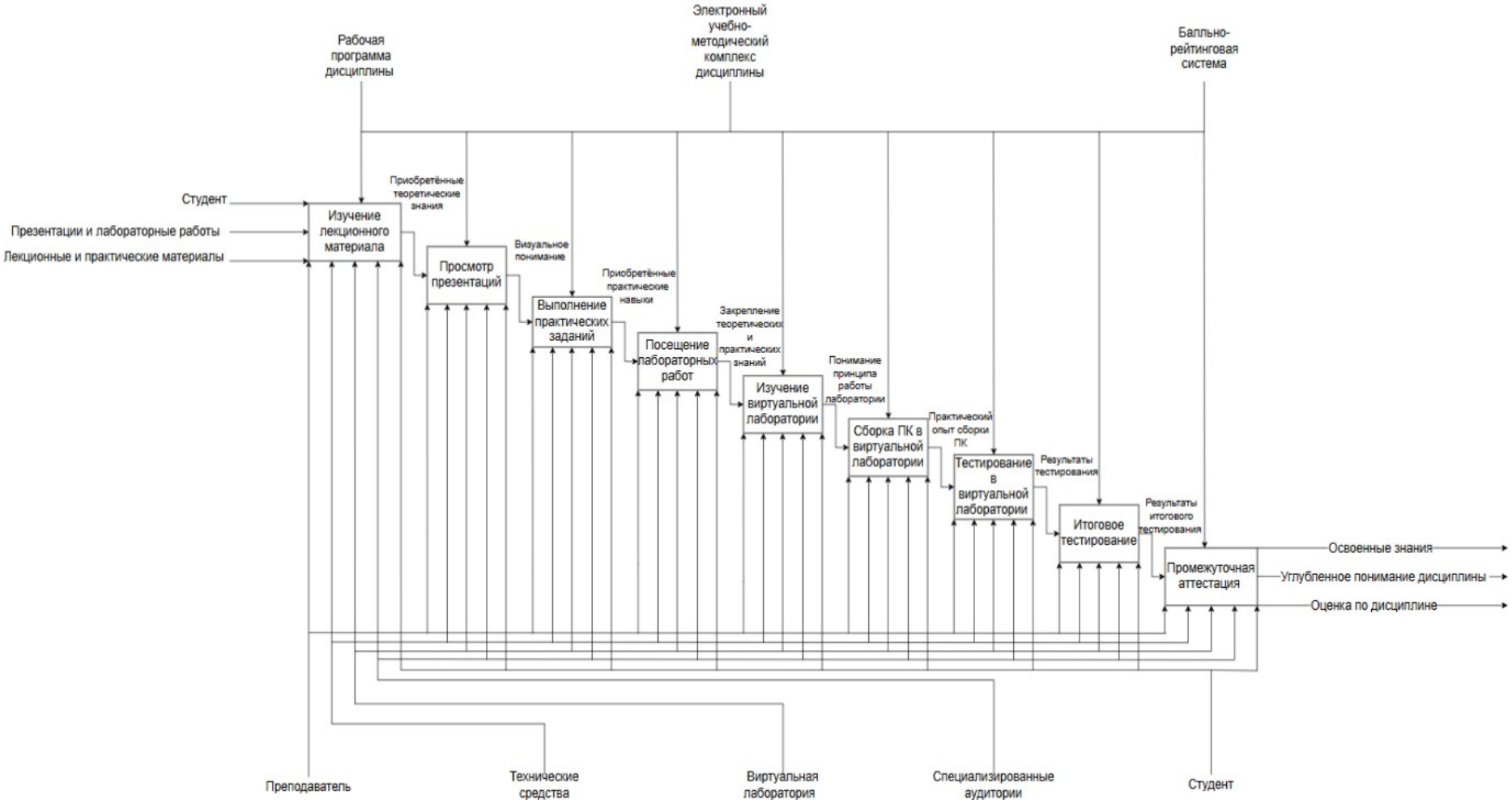

Рисунок А2 – Декомпозиция бизнес-процессов «как будет»

ПРИЛОЖЕНИЕ Б

Программный код сборка ПК

```
[Header("Список точек и предметов (в любом порядке)")]
public Placeholder[] placeholders;
public NetworkItem[] items;

[Header("Материалы для подсветки")]
public Material normalMat;
public Material highlightMat;
public Material errorMat;

[Header("Настройки управления")]
public KeyCode pickUpKey = KeyCode.E;
public KeyCode placeKey = KeyCode.E;
public KeyCode dropKey = KeyCode.Q;
public float placeRadius = 1f;
public float placementSpeed = 5f; // Скорость плавного перемещения

[Header("Объект руки")]
public Transform hand;

NetworkItem heldItem = null;
Vector3 holdOffset;
public bool allItemsPlaced = false;

// Для плавного перемещения
private bool isPlacing = false;
private Placeholder targetPlaceholder;
private Vector3 placementStartPosition;
private Quaternion placementStartRotation;
private float placementProgress;

Сообщение Unity | Ссылок: 0
void Start()...

Сообщение Unity | Ссылок: 0
void Update()...

Ссылок: 1
void UpdateHeldItemPosition()...
Ссылок: 1
void TryPickUp()...
Ссылок: 1
void Hold(NetworkItem itm)...
Ссылок: 1
void TryDrop()...
Ссылок: 1
void DropItem()...
Ссылок: 10
void SetItemPositionAndRotation(NetworkItem item, string name, float xV, float yV, float zV, float xQ, float yQ, float zQ)
Ссылок: 1
void TryPlace()...
Ссылок: 1
void StartPlacement(Placeholder placeholder)...
Ссылок: 1
void SmoothPlacement()...
Ссылок: 1
void FinishPlacement()...
Ссылок: 1
void PlaceAt(Placeholder placeholder)...
Ссылок: 1
bool AllItemsPlaced()...
Ссылок: 2
void HideAllPlaceholders()...
Ссылок: 1
void ShowPlaceholdersFor(NetworkItem item)...

[System.Serializable]
Ссылок: 4
public class Placeholder...
```

Рисунок Б1 – Программный код для сборки ПК