ОГЛАВЛЕНИЕ

Перечень условных обозначений и сокращений

В настоящей пояснительной записке применяются следующие термины, обозначения и сокращения.

КТ – Компьютерный тренажер.
ОС – Операционная система.
ПК – Персональный компьютер.
ПО – Программное обеспечение.
ЭВМ – Электронная вычислительная машина.

GE (*Game Engine*) – основа платформы для разработки игр.
*IDEF*0 – методология функционального моделирования и графическая нотация, предназначенная для формализации и описания бизнес-процессов.
SDK (*Software development kit*) – набор средств разработки.

ВВЕДЕНИЕ

Тренажерные технологии сегодня – это сложные комплексы, системы моделирования и симуляции, системы визуализации, компьютерные программы и физические модели, специальные методики, создаваемые для того, чтобы подготовить человека к принятию качественных и быстрых решений в критических ситуациях.

Развитие современного общества делает процесс подготовки и постоянного повышения квалификации специалистов все дороже и дороже. На первое место выходят как проблемы доучебного тестирования и отсева кандидатов, так и удорожание процесса подготовки при сохранении приемлемой эффективности. В мире, где быстро меняется оборудование, приходится быстро менять и тренажеры. Поэтому экономичнее создать виртуальный тренажер, который будет гораздо легче модернизировать, не отставая от техники сегодняшнего дня, что позволит сэкономить время и финансы компаний.

Реализация современных тренажеров стала возможна в связи с бурным развитием и удешевлением компьютерной техники и прогрессом в области создания технологий виртуальной реальности, машинного зрения, систем искусственного интеллекта и т. п. На базе этих технологий разработаны многочисленные тренажеры для различных сфер жизни.

По мере развития и удешевления, тренажерные технологии начинают проникать из важных областей, где ошибка человека может привести к серьезным проблемам, и в другие отрасли: хозяйство, авто и судовождение, школьное и вузовское обучение и прочее. Тренажерные системы к настоящему времени сформировались в успешно развивающуюся отрасль мировой индустрии.

Путем создания тренажера становиться возможным проведение подготовки и повышения квалификации персонала на предприятии для улучшения производительности труда на предприятии, а также для оценки эффективности работоспособности сотрудников. Также это обеспечивает получение большого количества дополнительной информации о поведении человека в различных ситуациях, что является значимым дополнением к информации о сотруднике.

При создании тренажера с использованием технологий виртуальной реальности становится возможным улучшение процесса подготовки сотрудников из-за создания «естественных» рабочих условий с большим погружением и вовлечением в происходящее, чем с использованием классических тестирующих систем.

Разработанный в ходе дипломной работы программно комплекс позволит решить вышеперечисленные вопросы и улучшить процесс обучения сотружников на предприятии.

1 СОВРЕМЕННЫЕ СРЕДСТВА ПОДГОТОВКИ ПЕРСОНАЛА НА ПРЕДПРИЯТИЯХ

1.1 Компьютерные тренажеры для подготовки персонала

Важной частью эффективного обучения являются различного рода тренинги, поскольку люди по статистике запоминают только 20% из того, что они видят, 40% из того, что видят и слышат и 70%, если видят, слышат и делают [1], рисунок 1.1. Тренажер, от английского *train* – обучать, готовить, тренировать, является программно-аппаратным средством тренировки и контроля при обучении профессии или выработке практических профессиональных навыков. Тренажеры находят широчайшее применение во многих сферах деятельности – в образовательном процессе для получения практических навыков по изучаемому материалу, в промышленности для отработки режимов управления технологическими объектами и процессами, во всех видах транспорта: авиационном, водном, железно-дорожном, автомобильном для обучения в реальном времени управлению сложной современной техникой.

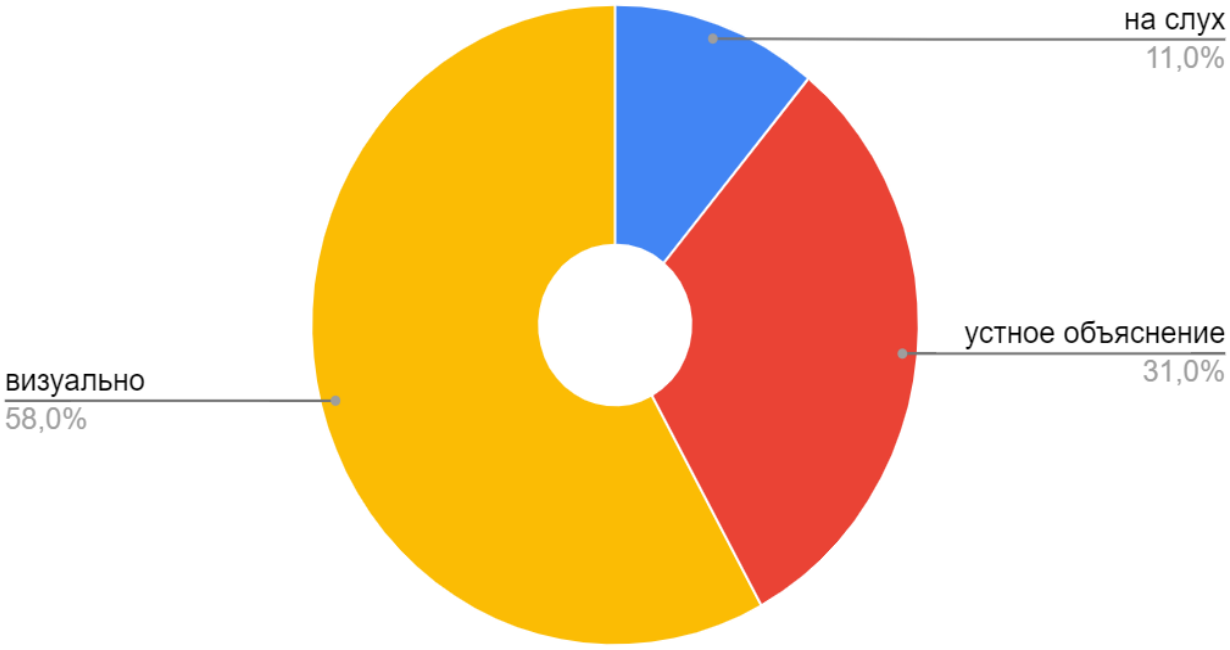

Рисунок 1.1 – Статистика предпочтений способов получения информации

Особое значение использование тренажеров имеет при подготовке персонала в отраслях, где ущерб от ошибочных действий может приводить к необратимым последствиям, таких как энергетика, нефтехимия, металлургическое производство и многое другое.

В этом случае при обучении специалиста тренажер выступает как имитационное средство профессиональной подготовки персонала, представляющее собой специализированный комплекс технических и программных средств, реализующий интерфейсные и математические модели технической и физической сущности сложной системы «объект-среда-оператор», а также все необходимые информационно-эргономические взаимосвязи в этой системе.

Тренажер предназначен для формирования и совершенствования у обучаемых профессиональных навыков и умений, необходимых для управления сложными технологическими объектами в штатных, нештатных и аварийных ситуациях. В названном классе выделяют два вида: компьютерные тренажеры и аппаратно-программные комплексы (стенды). При рассмотрении компьютерных тренажеров с точки зрения заложенной в них функциональности [2] можно выделить следующие группы:

Электронный экзаменатор представляют достаточно простой программный продукт, заменяющий живого экзаменатора в строго регламентированных областях (техника безопасности различных производств, правила дорожного движения и т.п.). Как правило, такой экзаменатор содержит набор вопросов, предлагаемых в случайном порядке, и ряд неправильных и один правильный ответ на каждый вопрос. Развитый экзаменатор должен обеспечивать такие возможности, как показ рисунков и анимации в кадре вопроса; распознавание ответа, представленного формулами; предварительное обучение (показ правильных ответов); режим редактирования вопросов и ответов.

Статические тренажеры, основная особенность которых заключается в отсутствии физико-математической модели процессов, происходящих в оборудовании, проверяют определенный порядок действий, который жестко задан (например, тренажер по оперативным переключениям в электрических сетях), а в более сложных случаях предусматриваются разветвления в цепочке действий, что обеспечивается логическими функциями. Недостатком является трудность программирования динамических эффектов (даже простого изменения показаний приборов). Область применения таких тренажеров ограничена дискретными системами управления и не включает моделирование сложных физических процессов.

Динамические тренажеры имеют в своей основе математическую модель реальных физических процессов. Как правило, сложны для разработки и реализации, требуют больших вычислительных мощностей.

Интеллектуальные тренажеры составляют особый класс обучающих систем, соединяющий в себе обычный тренажер с системой, имитирующей деятельность инструктора. Может содержать базу знаний экспертной системы с набором технологических правил по управлению объектом.

Современные компьютерные технологии позволяют создавать тренажеры, включающие мультимедийные компоненты – компьютерную мультипликацию, аудио и видеоэффекты. Использование этих средств усиливает ощущение реальности при работе с тренажером и открывает новые возможности в процессе обучения. Выбор тех или иных программных средств определяется целями обучения. Так, для контроля персонала по выполнению определенных правил можно использовать простые экзаменаторы. Общее ознакомление с устройством и обучение определенному порядку действий можно выполнять средствами статических тренажеров. Для проведения экспериментов, изучения физических основ и способов функционирования устройств, для проблемного обучения, противоаварийных тренировок и анализа аварий следует использовать динамические тренажеры для осуществления комплексного обучения операторов с применением математических и имитационных моделей сложных технологических объектов необходимо использовать интеллектуальные обучающие тренажеры.

Если же рассматривать стенды-тренажеры, сочетающие компьютерную модель с аппаратной частью, то можно отметить, что в этом классе принято различать тренажеры полномасштабные и локальные.

Полномасштабные воспроизводят органы управления объектом в реальном виде и служат как для изучения физики процесса, так и для отработки моторных навыков управления им. К локальным же тренажерам не предъявляют требования полного соответствия объекту и его системе управления. Они могут воспроизводить лишь одну из технологических подсистем и служат для изучения процессов, происходящих в технологическом объекте. Их преимущество – низкая стоимость и возможность использования на обычном компьютере. Компьютер в данном случае заменяет реальный управляемый объект; здесь, как правило, требуется хорошая динамическая модель. Недостатком стендов-тренажеров является сложность их модернизации при изменении оборудования. Грамотно спроектированный и реализованный интеллектуальный тренажер является более предпочтительным, поскольку его можно легко перенастроить. Перспективным представляется направление по разработке системоболочек (или систем-конструкторов) для создания программных тренажеров технологических объектов, настраиваемых на различные предметные области.

Аналогом полномасштабных виртуальных тренажеров являются тренажеры использующие технологии дополненной и виртуальной реальности. Они представляют особый вид человеко-машинного интерфейса с применением специальных устройств для динамической визуализации и прямого манипулирования объектами. Они как нельзя лучше подходят для разработки тренажеров в различных сферах деятельности. Среда виртуальной реальности относится к тем областям деятельности, которые в последние годы часто обсуждаются и рассмат-

риваются к использованию в образовательных целях [3]. Также реализации виртуальной реальности повсеместно встречаются в рекламах туристического направления, презентациях исторических элементов или в сфере развлечений.

Развитие и распространение современных технологии позволяют расширить использование реализаций виртуальной реальности на все сферы деятельности человека. На рисунке 1.2 описываются основные современные сферы использования технологий виртуальной реальности.

Рисунок 1.2 – Области применения *VR* технологий

В принципиальном плане рассмотренные характеристики связываются с тремя обязательными компонентами КТ:

– функциональность – с набором инструкторских функций (модель обучения);

– точность – с полнотой, связностью и адекватностью тренажерной модели (модель ТП);

– масштабность – с полнотой и точностью реализации среды управления (информационная модель и модель системы управления).

Базовая функциональность КТ предусматривает реализацию функций обучаемого персонала и инструктора.

Функциональность обучающегося должна представлять реальные возможности персонала на предприятии, связанные с непосредственно занимаемой должностью. Графический интерфейс должен с достаточной степенью подобия соответствовать реальному окружению. В этот случае удается достичь наибольшего реализма и погружения сотрудника в рабочие процессы на предприятии.

Функциональность инструктора должна обеспечить все возможности для контроля и управления процессом обучения: инициализация условий виртуального мира, отслеживание статистики прохождения для оценки действий пользователя и успешности обучения, создание и хранение сценариев, прохождений испытания.

1.2 Системы разработки приложений виртуальной реальности

В настоящее время наиболее известными *GE* для разработки игр являются: *Unity*, *CryEngine* 4, *Unreal Engine* 4, *Creation Engine*, *Frostbite*, *Source*. Сегодня они связаны со всеми компонентами игры, начиная от рендеринга, физики, звукового оформления, кода, создания ИИ и заканчивая сетевыми аспектами. Также многие среды поддерживают импорт или доработку кода под целевой проект или платформу.

1.2.1 *Unity* позволяет создавать игры под большинство популярных платформ. Приложения запускаются на персональных компьютерах (работающих под *Windows*, *MacOS*, *Linux*), на смартфонах и планшетах (*iOS*, *Android*, *Windows Phone*), на игровых консолях (*PS*, *Xbox*, *Wii*). Позволяет создавать 2*D*, 3*D*, а также игры под *VR*.

Ценовая политика и политика распространения достаточно проста, в стандартной версии можно разрабатывать коммерческие игры и сразу под все платформы. Ограничениями в этом случае служат демонстрирует лого *Unity* перед запуском игры, а проект, созданный с ее помощью, не должен приносить разработчику больше $100 тысяч в год. Улучшенная версия стоит всего 1500$ и предоставляет дополнительные функции, в первую очередь, связанные с упрощением взаимодействия с элементами среды и инструменты для разработки графики.

Разработка приложений может вестись как с использованием методики *Drag* & *Drop* и компонентно-ориентированный подходом, в рамках которого разработчик создает объекты и к ним добавляет различные компоненты (например, визуальное отображение персонажа и способы управления им). На рисунке 1.3 изображается пример рабочего окна данной среды.

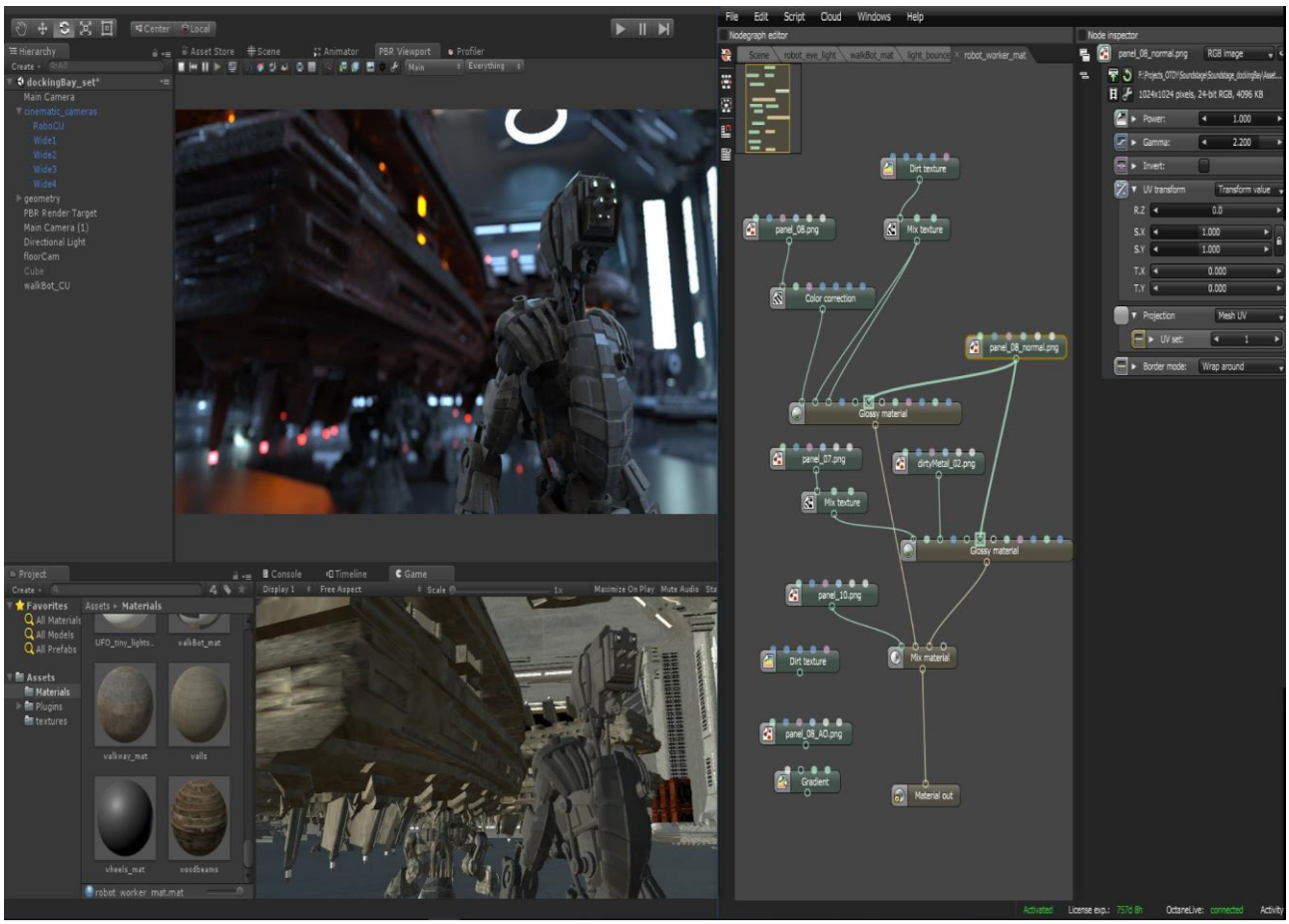

Рисунок 1.3 – Окно разработки среды *Unity*

Имеет ограниченный функционал для создания многопользовательских сетевых приложений.

Среда отлично подходит для разработки инди-игр с упрощенной графикой. Иначе приложения начинают занимать много дискового пространства ПК, а также требовать больших мощностей.

1.2.2 *CryEngine* 4 разрабатываемый компанией *Crytek*. Сейчас компания планирует выпуск новой версии среды и в отличие от предыдущих релизов компании, новый *CryEngine* не будет идентифицироваться номерами версий.

Сама среда и *SDK* распространяются бесплатно для использования в образовательных и некоммерческих целях. Но для коммерческого распространения продуктов, основанных на *CryEngine* нужно приобрести лицензию. На *GDC* 2014 *Crytek* заявили, что с мая 2014-го года инди-разработчики смогут пользоваться *CryEngine* по подписке за 9.90 долларов/евро на одного пользователя в месяц, без отчислений каких-либо роялти [4].

Игры с использованием *CryEngine* разрабатываются не только студией, создавшей его. Изначально его могли лицензировать сторонние компании за фиксированную плату, а образовательные учреждения могли использовать его бесплатно, но на некоммерческой основе – только для обучения студентов.

Но начиная с 2016 года среда и *SDK* (набор средств разработки) распространяются бесплатно для всех желающих, но с условием выплаты *Crytek* 5% прибыли при доходах, превышающих 5000 долларов/евро (начиная с версии 5.5, на более ранних версиях стоимость за рецензирование не выплачивается). На рисунке 1.4 изображается рабочая область данной среды.

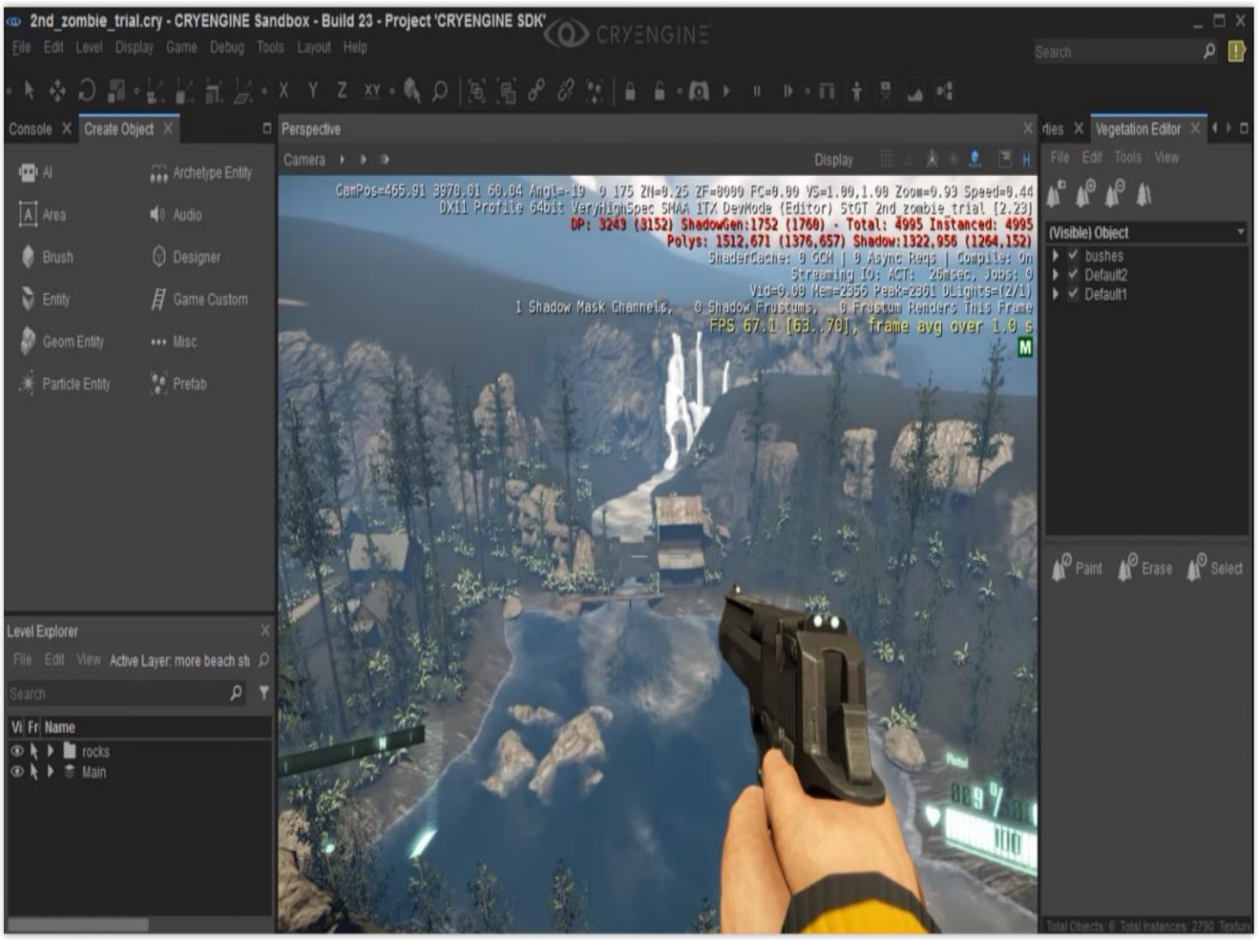

Рисунок 1.4 – Окно рабочего пространства среды *CryEngine*

Среда отличается продвинутыми возможностями по разработке видеоигр и поддержкой самых передовых технологий, включая *DirectX* 12, *Vulkan API*, *VR*, написание скриптов на *C#*, попиксельное освещение в реальном времени, карты отражений, детализированные текстуры, туман, поверхности с бликами, реалистичную физику, продвинутую анимацию и многое, многое другое.

CryEngine позволяет создавать игры с почти фотореалистичной графикой. При должном умении проекты, разработанные с его помощью, превосходят по качеству картинки любые игры на *Unreal Engine* 4 или *Unity*. К тому же, движок содержит функциональный *realtime renderer*, позволяющий быстро испытать только что созданный уровень или сцену.

Crytek разработала собственную технологию трассировки лучей на движке, которая работает на видеокартах *AMD* и *Nvidia* и не требует мощности

графических чипов *RTX*. Наконец, обязательно стоит упомянуть *GameSDK* – инструмент, на основе которого можно быстро создавать собственные игры, используя в том числе ассеты с официального сайта *Crytek*.

Из недостатков многие разработчики отмечают трудности при работе с ним, возникающие из-за сложности сборки билда, наличия багов в редакторе, скромного выбора ассетов, ограничений для разработки сетевой игры, а также отсутствия хорошей техподдержки и активного комьюнити, в результате чего часто приходится решать проблемы методом проб и ошибок, не имея возможности посоветоваться с опытными коллегами.

При всей своей мощности, *CryEngine* довольно сложен в освоении, так что необходимо обладать обширными познаниями в области разработки, чтобы создавать с его помощью игры.

1.2.3 *Unreal Engine* 4 очень гибок и универсален. В отличие от *Unity*, который требует установки множества плагинов (часто – платных), *UE*4 уже «из коробки» снабжен всеми необходимыми инструментами разработки. Пользователи, знающие *C++*, освоят движок быстрее, ведь он использует именно этот язык программирования. Тем, кто не знаком с *C++*, *Unreal Engine* тоже поддастся без проблем – все благодаря визуальному редактору *Blueprints*, который позволяет создавать скрипты и размещать объекты, что в значительной степени ускоряет разработку приложений [5].

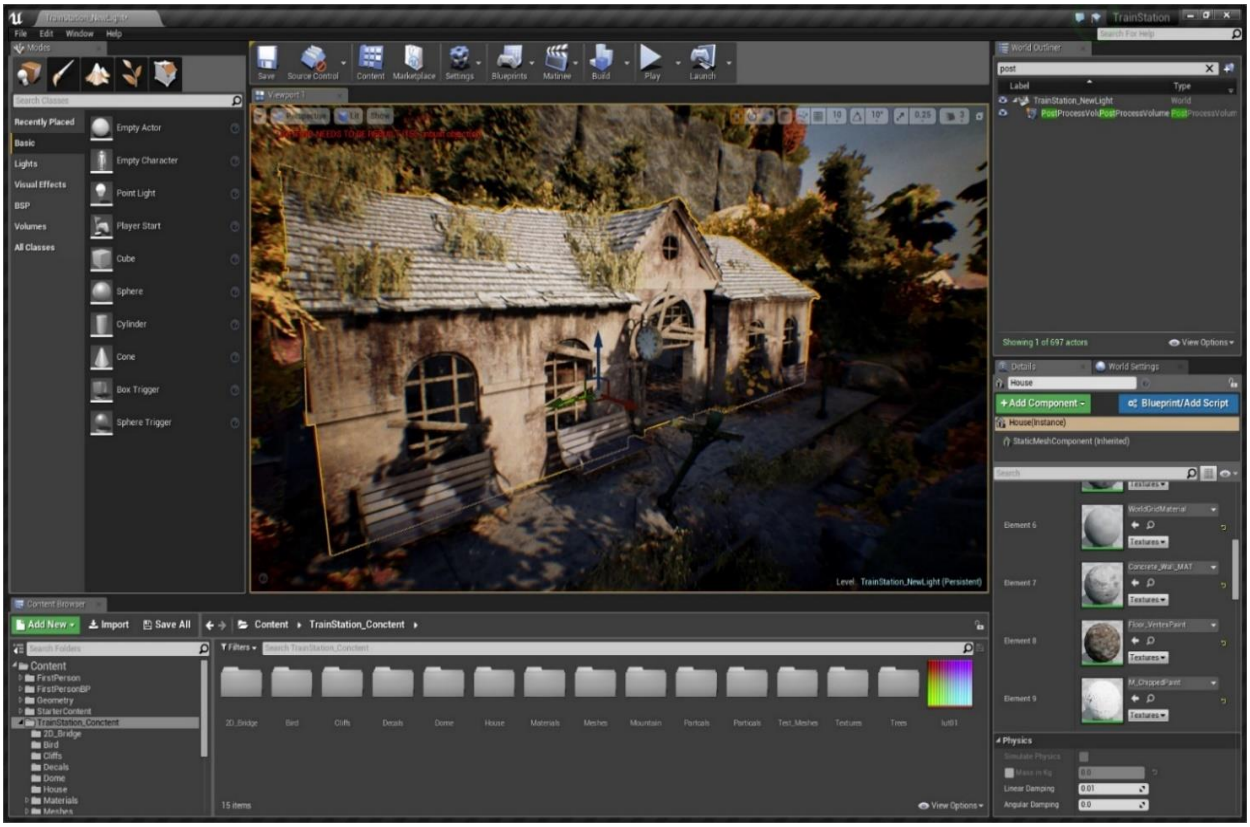

Рисунок 1.5 – Окно рабочего пространства среды *Unreal Engine* 4

Поддерживает множество форматов текстур, точно передает физические свойства материалов, позволяет изменять объекты в реальном времени, задавать для них функции и комментарии, автоматически выбрать источники освещения, добавить туман и другие эффекты, и так далее. Имеет большую коллекцию ассетов (платных и бесплатных) можно использовать при разработке игр, а открытый исходный код игрового ядра дает возможность вносить в него изменения при необходимости. Ядро гибко подстраивается под платформу разработки, что позволяет оптимизировать игры под консоли, мобильные гаджеты и ПК.

Компания-разработчик *Epic Games* стабильно держит лидерские позиции в области передовых технологий, поэтому *Unreal Engine* 4 имеет максимальное количество современных новинок, обеспечивая, при должном мастерстве разработчиков, впечатляющую визуальную составляющую с качественным освещением (трассировка лучей в реальном времени с недавнего времени также поддерживается), мягкими тенями, честными отражениями, достоверной анимацией персонажей и прочими эффектами [6].

За счет качественной работы с текстурами и высокой детализацией мира среда также используется в создании компьютерной графики в киноиндустрии – например, его силами был создан дроид *K*-2*S*O для фильма «Изгоя-1» сцены из мультфильма «В поисках Дори» и сериала «Мандалорец». Из-за его популярности для создания реалистичных фонов фильмов компания планирует выпустить отдельную «упрощенную» среду *Twinmotion*, которая позволит упростить разработку фонов сцен.

1.2.4 *Creation Engine* – игровое ядро, разработанное американской компанией *Bethesda Game Studios* для использования в собственных проектах. Первая компьютерная игра, построенная с использованием этой среды разработки – *The Elder Scrolls V*: *Skyrim* 2011 года.

Графическое ядро создавалось так, чтобы объекты освещались более правдоподобно, чем раньше, а визуализация воды была более качественной. Система автоматически, в зависимости от местности, генерирует нужное количество снега для деревьев, камней и кустов.

За искусственный интеллект персонажей, которые встречаются в играх, отвечает собственная система – *Radiant AI*. В новой версии среды авторы существенно усовершенствовали искусственный интеллект, стремясь создать иллюзию жизни горожан игры: они завтракают, идут на работу, заходят в паб и т.п.

Система управления сюжетом, названная *Radiant Story*, позволяет разработчикам смешивать созданные вручную задания с заданиями, которые могут быть сгенерированы случайно из различных условий, кроме того задания могут появляться в разном порядке и отличаться в зависимости от стиля прохождения игр пользователя, позволяет создавать вариативные диалоги в играх. На рисунке 1.6 изображается рабочее пространство данной среды.

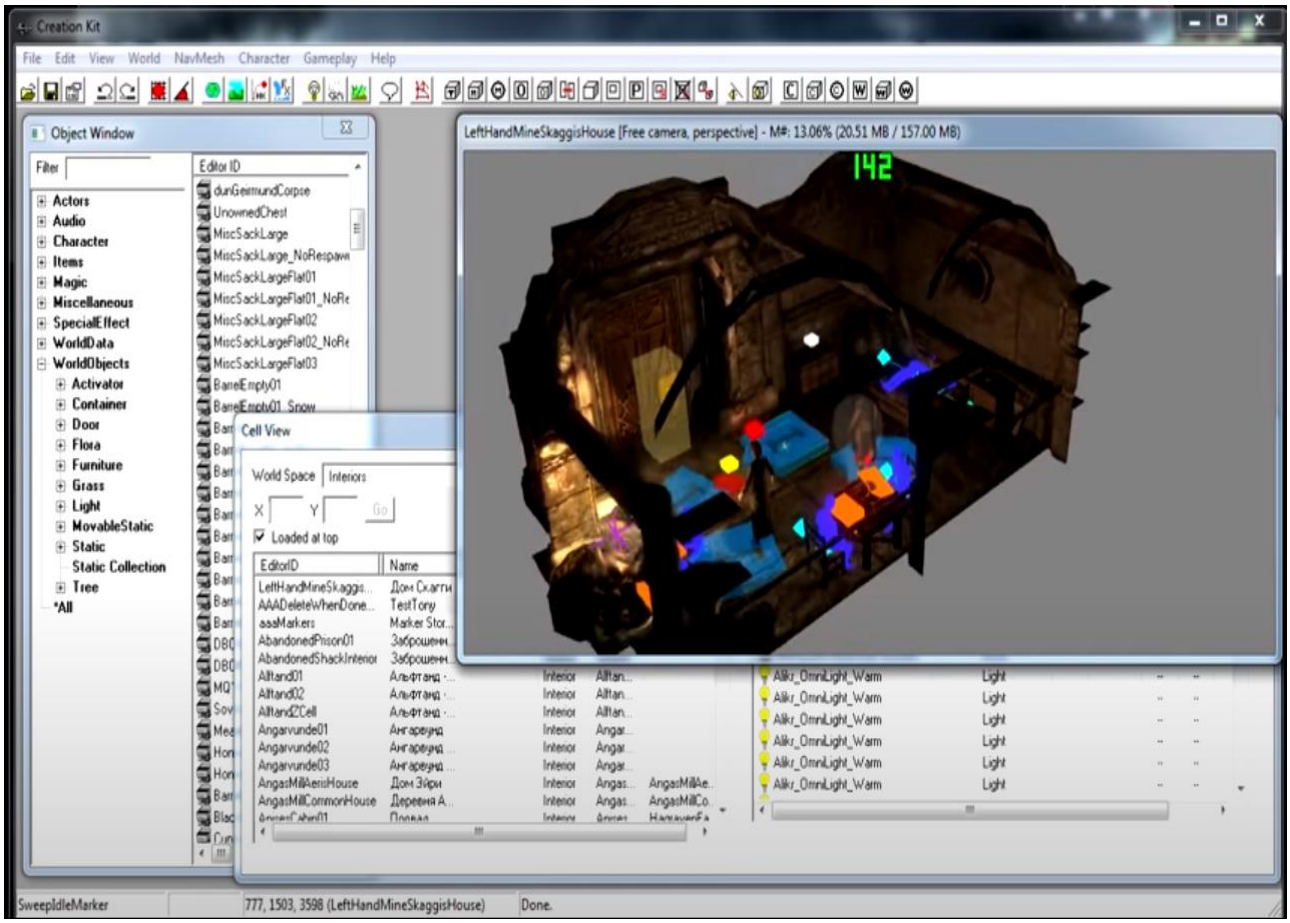

Рисунок 1.6 – Окно рабочего пространства среды *Creation Engine*

В данный момент идет 3 цикл обновлений для среды, однако многие разработчики утверждают, что среда морально устарела, ввиду того что многие функциональные возможности остались на уровне проектов 2002 года. Это относится как к физике, свету и графике игрового мира, так и к самой методологии разработки игр.

1.2.5 *Frostbite* – продукт, разработанный компанией *EA Digital Illusions CE*; применяется как в собственных разработках, так и проектах других филиалов *Electronic Arts*. Относится к типу подпрограммного обеспечения и представляет собой связку нескольких компонентов, таких как графическое ядро, звуковое ядро и т. д. В операционной системе *Windows* поддерживает отображение графики при помощи *Mantle* начиная с версии 3, *DirectX* 10.1, а начиная с версии 1.5 – и *DirectX* 11. Одной из заявленных особенностей является оптимизация для работы на многоядерных процессорах.

Технология способна обрабатывать разрушаемость ландшафта и окружения (например, построек, деревьев, автомобилей). Поддерживается динамическое освещение и затенение с функцией *HBAO*, процедурный «шейдинг», различные пост-эффекты (например, *HDR* и *depth of field*), система частиц и техники

текстурирования, такие, как «бамп-маппинг». Максимальный размер локации составляет ограничение в 32×32 километра отображаемой площади и 4×4 километра игрового пространства. Помимо этого, по утверждению создателей, максимальная дистанция прорисовки позволяет увидеть уровень вплоть до горизонта. Также встроен собственное звуковое ядро, не требующий использования специализированных средств, подобных *EAX*.

Среда комплектуется игровым редактором *FrostED*, написанном на языке программирования *C#*. Программа предназначена для создания уровней, а также работы с сетками, «шейдерами» и объектами. На рисунке 1.7 изображается рабочее пространство данной среды.

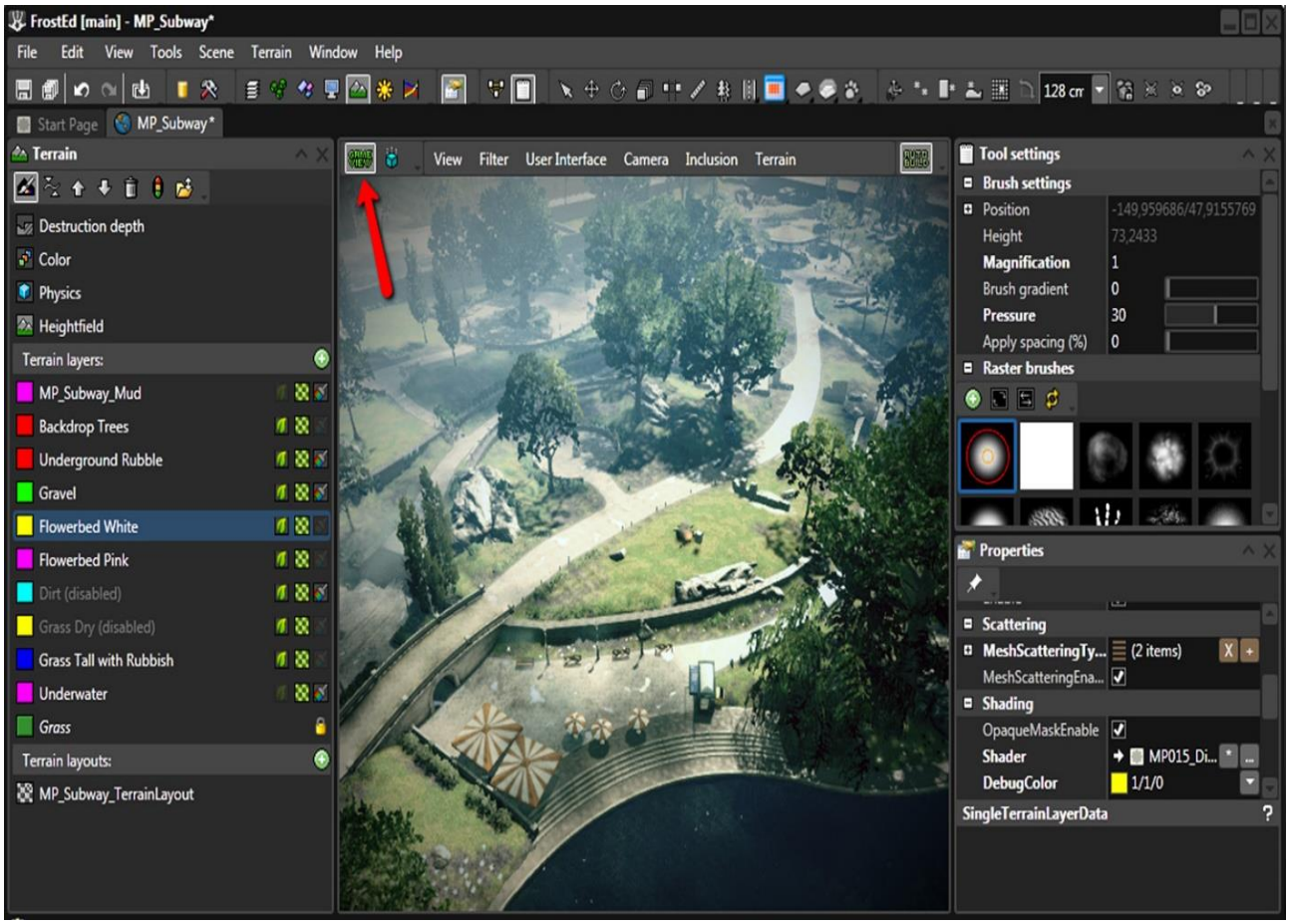

Рисунок 1.7 – Окно рабочего пространства среды *Frostbite*

В данный момент активно развивается компанией-разработчиком. Постоянно обновляется с выходом новых игровых технологий. Лицензия проприетарная: не лицензируется – только внутреннее использование.

1.2.6 *Source* – среда, разработка которого началась еще в 1998 году, когда *Valve* подходила к финальному этапу создания своей первой игры. Тогда разработчики захотели реализовать свои идеи, но не хотели рисковать, внося изменения в почти уже готовый проект. В результате было решено разделить код на две

части: *GoldSource* и *Source*. Соответственно, первая часть осталась за *Half-Life*, а вторая начала вбирать в себя инновации, постепенно оформляясь в полноценное игровое ядро, на котором *Valve* позже выпустит вторую часть игры.

Инструментарий *Source SDK* доступен бесплатно всем пользователям *Steam*, и включает в себя редактор карт, программы для импорта/экспорта и просмотра моделей и проработки лицевой анимации, а также утилиты для распаковки файлов и файлы исходного кода библиотек для некоторых игр *Valve*. Данный набор позволяет быстро начать разработку собственного проекта имея базу наработок компании.

Среда получила крайне высокую популярность для создания модификаций и отдельных игр, создания различного типа видео проектов. Разработка с использованием *Source* (если это не создание какой-нибудь модификации) требует хорошего знания *C++*. На рисунке 1.8 изображается рабочее пространство данной среды.

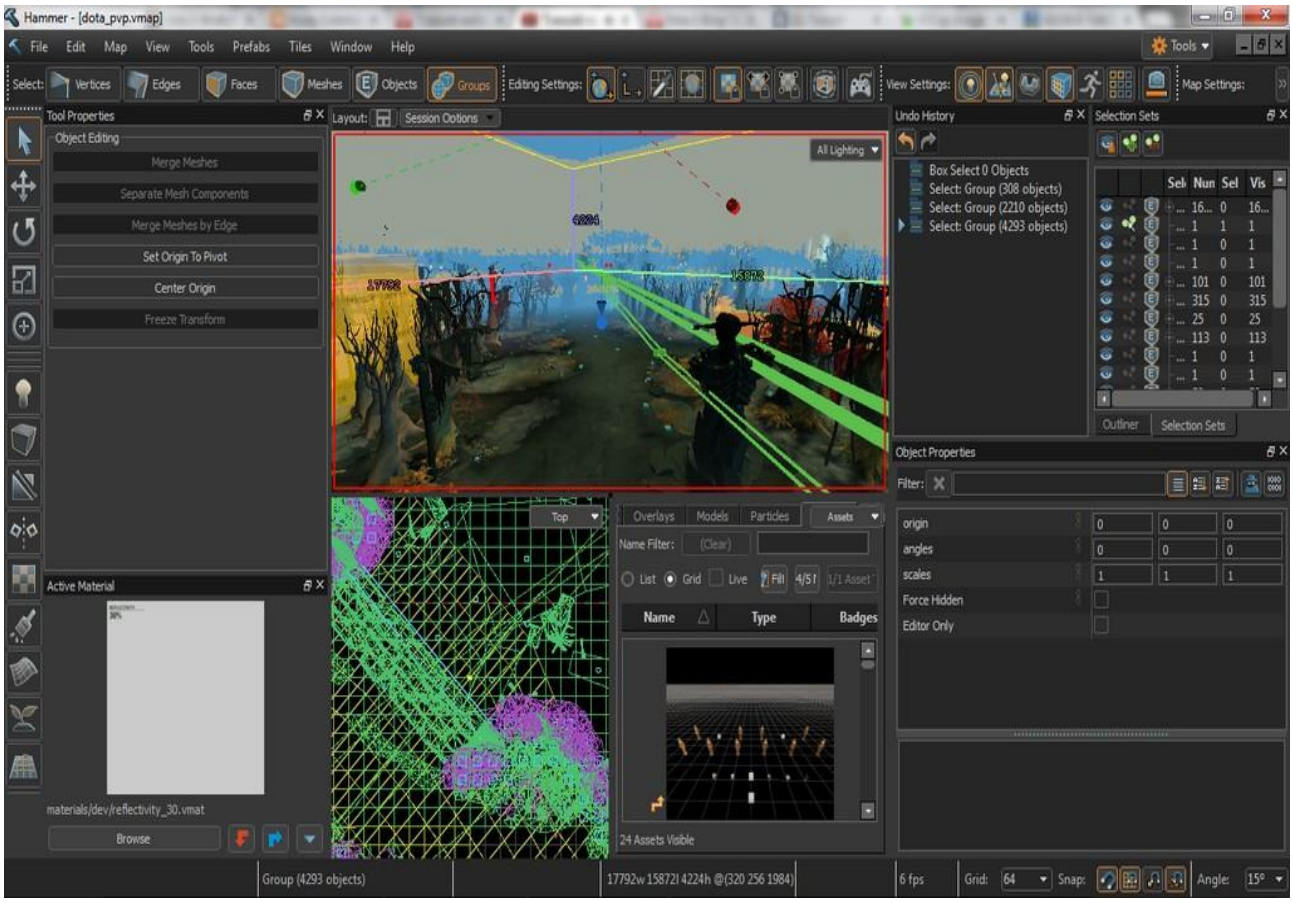

Рисунок 1.8 – Окно рабочего пространства среды *Source*

Несмотря на всю гибкость и доступность *Source*, среда не может конкурировать с инструментами от других компаний в плане насыщенности технологиями. Даже самые новые игры от опытных студий (например, *Titanfall* 2) выглядят блекло по сравнению с проектами, выполненными на *Unreal Engine* 4 и

CryEngine. Вторая версия среды позволила значительно подтянуть графическую составляющую, также добавила возможность создания полноценных приложений виртуальной реальности.

Компания *Valve* приветливо относится к разработчикам модификаций и предоставляет свой инструментарий на бесплатной основе. Но если разработчики будут продавать модификации или игры на движке *Source*, то им придется заплатить за использование движка и всех его компонентов, включая отчисления *Havoc, Miles* и/или *Blink*, и обязательно выпустить игру в *Steam*.

Сравнение существующих систем для разработки виртуальной реальности приведено в таблице 1.1

Таблица 1.1 – Сравнение систем для разработки приложений

Среда	Доступность	Поддержка *VR*	Кроссплат-форменность	Сложность вхождения
Unity	есть	есть	есть	нет
CryEngine	есть	есть	нет	есть
Unreal Engine	есть	есть	есть	нет
Creation Engine	есть	нет	нет	есть
Frostbite	нет	нет	нет	есть
Source	есть	есть	нет	нет

Из проведённого сравнительного анализа следует, что существующие системы разработки приложений в незначительной мере отличаются друг от друга. Передовые решения обладают практически идентичным функционалом имея отличия лишь в подходе к изданию игр и реализации определенных инструментов. Ввиду использования технологий виртуальной реальности в проекте, данный пункт является одним из самых важных при выборе платформы разработки. Основными лидерами данного анализа можно выделить *Unity* и *Unreal Engine*. Ввиду более высокой гибкости среды *Unreal Engine*, она будет использована далее для разработки тренажера для подготовки персонала.

Использование технологий виртуальной реальности позволит усилить эффект присутствия пользователя, что положительно повлияет на качество обучения. Также применение технологий *VR* позволит добавить интерактивности процессу обучения. Применение новой диалоговой системы позволит расширить возможности создания диалогов тестов, позволяя создавать более ветвистые конструкци.

2 АРХИТЕКТУРА ПРОГРАММНОГО КОМПЛЕКА ДЛЯ ПОДГОТОВКИ ПЕРСОНАЛА С ИСПОЛЬЗОВАНИЕМ ИНТСРУМЕНТОВ *VR «VRTRAINER»*

2.1 Описание основных функций программного обеспечения

На рисунке 2.1 описывается общая схема проектируемого приложения с использованием многоуровневой модели проектирования ПО [7, с. 222].

Рисунок 2.1 – Схема проектируемого программного комплекса

Уровень представления представляет интерфейс пользователя, к которому относятся все элементы пользовательского интерфейса приложения-тренажера и сайта для просмотра статистики.

Уровень логики представляет описание процессов создания и редактирования сценариев, запись и скачивание видеофрагментов прохождений, прохождение тестирований, просмотр журнала прохождений.

Уровень доступа к данным описывает правила работы с данным приложений, методы для сохранения и загрузки видео.

Разрабатываемое программное обеспечение предназначено для образовательных учреждений, предприятий, занимающихся подготовкой и повышением квалификации персонала в различных сферах труда.

В программном комплексе должно осуществляться моделирование различных реальных ситуаций и проводится оценка ответов тестируемого, формирование сводных отчетов по результатам тестирования, визуализация рекомендаций, предоставление возможностей добавления и редактирования различных ситуаций и сценариев, возможность просмотра прошлых тестирований, возможность сохранения видео с прохождением сценария, хранение полную информации о прохождении пользователем сценария.

На основании полученной информации специалист сможет делать выводы об уровне сотрудника в определенной сфере труда. Это позволит улучшить работоспособность коллектива, выявить у пользователя недостаточную базу знаний в той или иной области для дальнейшей работы по улучшению качества знаний.

2.2 Функциональная модель программного комплекса

Функциональная модель – это система элементов, отражающих функциональные способности разрабатываемой системы и создающих упрощенное представление о ее реальном устройстве. Элементы системы состоят из диаграммы процесса и текстовой части, в которой выполняется описание данного процесса.

В качестве методологии для разработки функциональной модели разрабатываемой системы использовалась методология функционального моделирования *IDEF*0 [8, с. 141].

Контекстная диаграмма разрабатываемого программного комплекса в виде *IDEF*0 диаграммы приведена на рисунке 2.2.

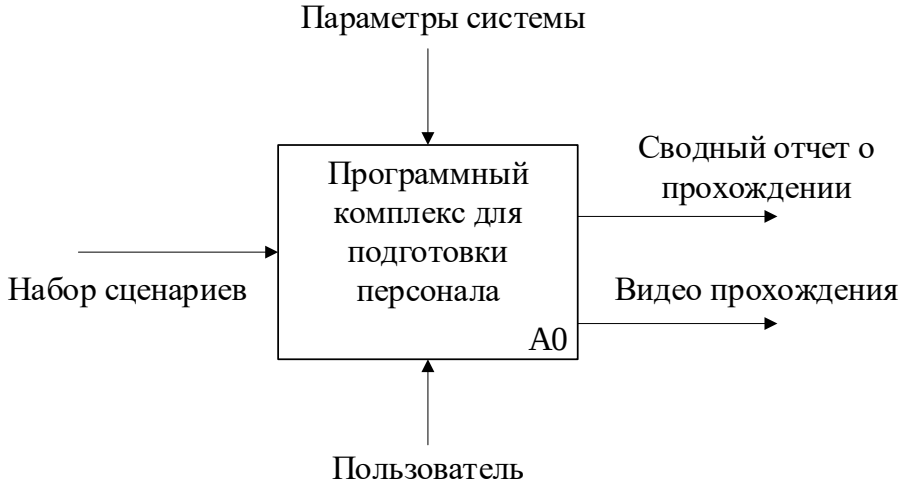

Рисунок 2.2 – Общая диаграмма программного комплекса

Входными параметрами на данной схеме будут являться наборы сценариев для определенных ситуаций.

Управляющая структура представлена параметрами для прохождения сценария.

Конечный пользователь данного программного комплекса будет являться его механизмом.

В результате работы данного приложения на выходе будут формироваться сводные отчеты о прохождении пользователем того или иного сценария, а также будет сохранятся видеофрагмент с непосредственным прохождением сценария пользователем.

На рисунке 2.3 приведена декомпозиция программного средства.

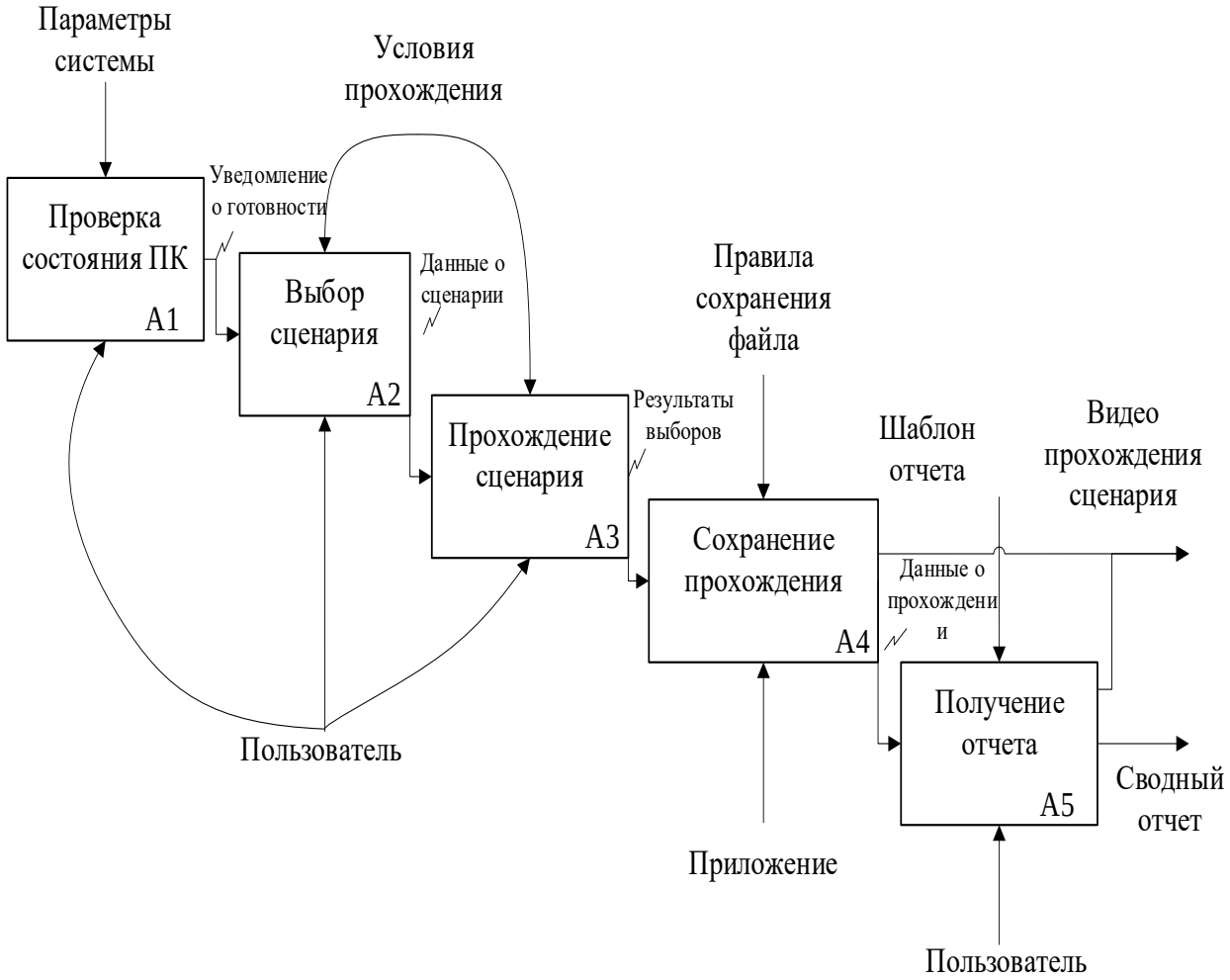

Рисунок 2.3 – Диаграмма декомпозиции программного средства

Диаграмма декомпозиции программного средства описывается пятью основными процессами разрабатываемого программного комплекса – A1, A2, A3, A4 и A5. Данные процессы описывают все основные разрабатываемые части программного комплекса.

Входными данными для процесса A1, описывающего этап подготовки и проверки условий для запуска приложения будет являться информация о текущем состоянии компьютера, его операционной системе, и подключенным к нему устройствам.

Перед запуском основного приложения требуется проведение ряда проверок соответствия операционной системы рекомендуемым требованиям. Операционная система должна иметь все последние обновления, присылаемые компанией *Microsoft*, в противном случае будет получена ошибка о том, что система устарела и запуск приложений с поддержкой *VR* невозможен.

Затем необходима проверка наличия установленного приложения *SteamVR*. Данное приложение полностью отвечает за корректную работу с различными устройствами виртуальной реальности такими как шлемы, контроллеры и дополнительные станции отслеживая движений.

После запуска разрабатываемого приложения должна производиться попытка подключения, поиска и инициализации доступных устройств *VR*, после всех проверок должно запутаться домашнее окружение приложения, будет производиться синхронизация контроллеров и устройства будут готовы к работе.

Декомпозиция процесса A1 описывается на рисунке 2.4.

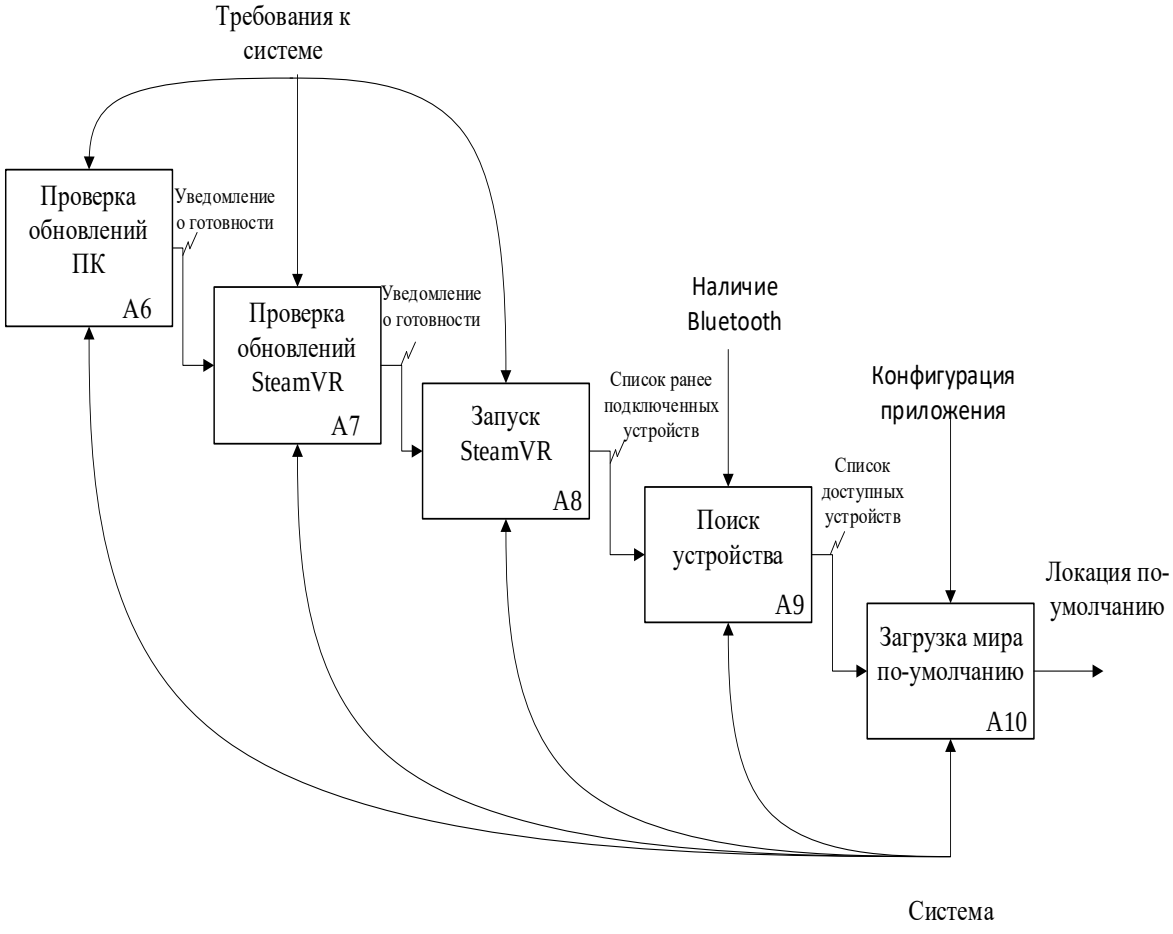

Рисунок 2.4 – Декомпозиция процесса инициализации параметров

На данном этапе может проводится настройка рабочего окружения системы, выбор параметров для работы с окружением (выбор возможности передвижения или использование в сидячем положении). Данный этап, в отличии от проверки обновлений может проводится единожды или при смене рабочего окружения.

Процесс А2 (взаимодействие со сценариями) описывает предоставляемый пользователю функционал для работы со сценариями.

Перед прохождением теста, пользователю необходимо будет воспользовавшись веб-приложением загрузить различные сценарий в приложение. Необходимо предоставлять возможность создания нового сценария, редактирования существующего или возможность импортирования уже готового сценария. После того как сценарий будет выбран и загружен пользователь сможет переходить к запуску приложения виртуальной реальности.

Декомпозиция процесса А2 описывается на рисунке 2.5.

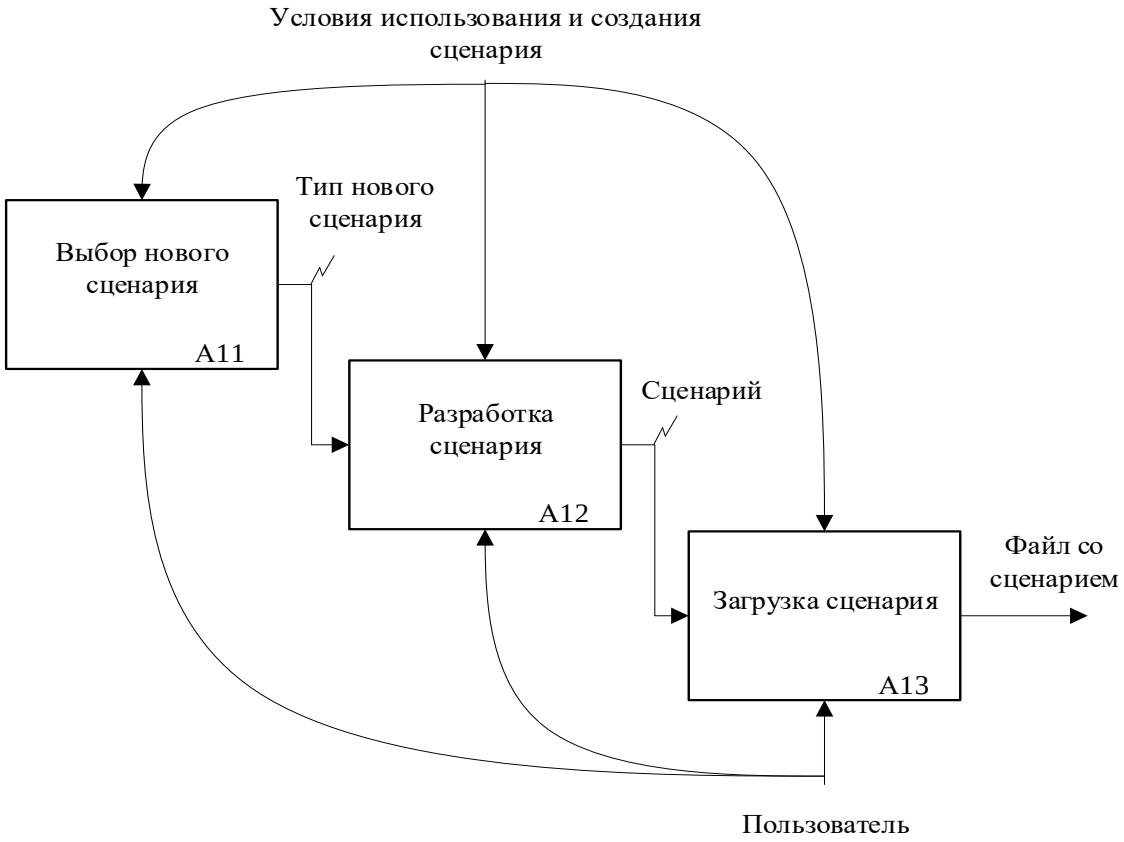

Рисунок 2.5 – Декомпозиция процесса взаимодействия со сценариями

При создании нового сценария или загрузке существующего необходимо реализовать проверку соответствия некоторым критериям:

– всегда имел возможность завершения диалога, недопустима возможность загрузки «бесконечных» сценариев;

– содержал корректное количество вариантов ответов (минимальное количество вариантов ответов один, а максимальное четыре);

– не нарушал структуру сценария, если этот сценарий был создан из вне приложения требуется реализовать проверку правильности структуры файла;

– фразы должны быть лаконичными, слишком большие вопросы или ответы будут осложнять прохождение тестирования.

Пользователь задает необходимые параметры запуска приложения, после чего запускается процесс А3, который представляет непосредственно загрузку игрового приложения и прохождение пользователем сценария.

Процесс А3 отвечает за запуск приложения с ранее установленными параметрами, производит инициализацию стартового мира, синхронизируется с устройствами *VR*, загружает сценарии. После этого пользователь может переходить выбору уровня.

Декомпозиция процесса А3 описывается на рисунке 2.6.

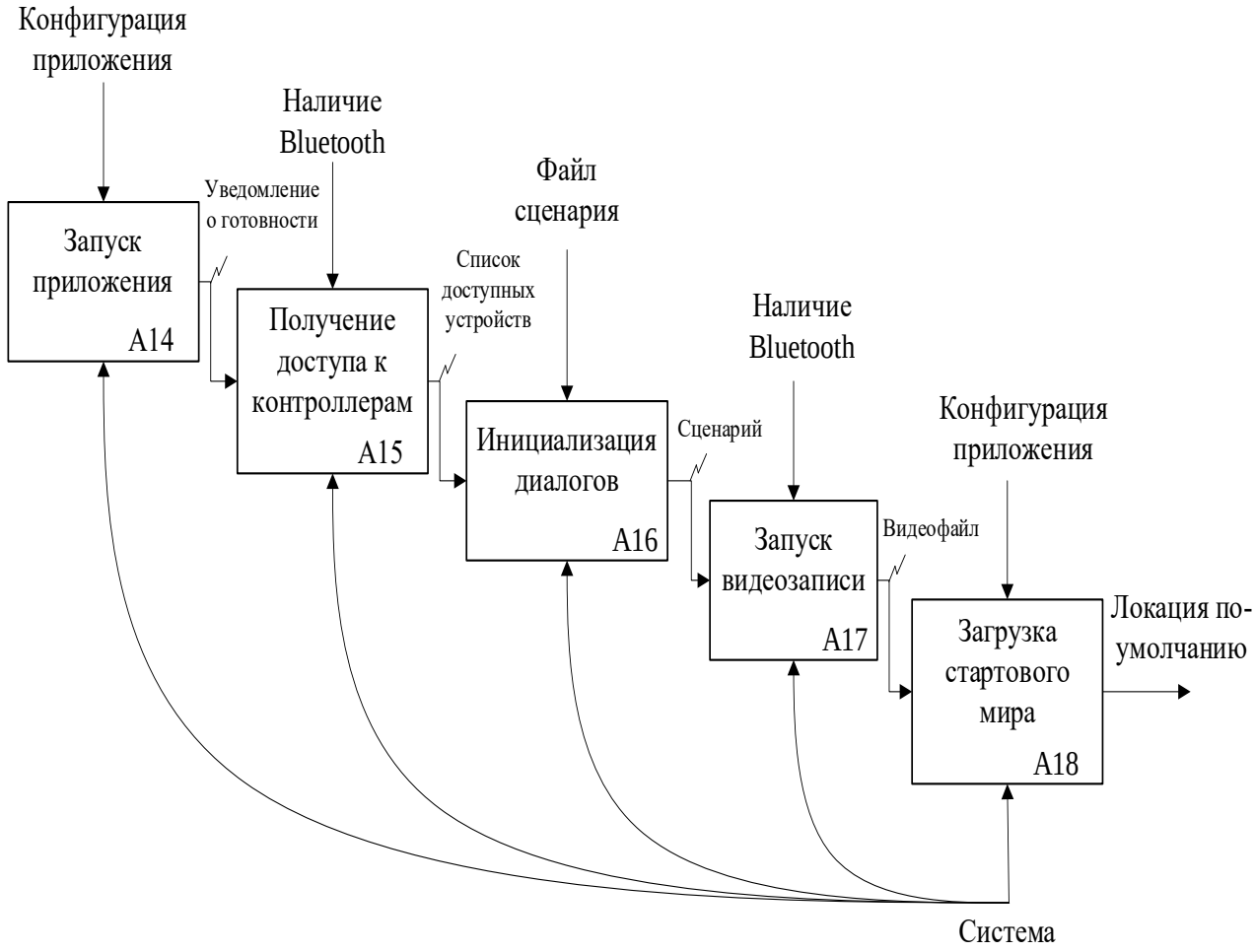

Рисунок 2.6 – Декомпозиция процесса запуска приложения

После выбора уровня будет запущен процесс инициализации основной локации, произведена загрузка всех объектов сцены. Также при загрузке мира будет запушен процесс записи видео с прохождением данного сценария. После

этого пользователь может приступить к непосредственному прохождению данного сценария.

При прохождении сценария пользователю необходимо будет взаимодействовать с интерактивными объектами мира, для успешного выполнения поставленной задачи. При этом пользователь неограничен во времени.

Если в процессе прохождения сценария будет потеряно соединение с одним из контроллеров приложение должно запускать процесс ожидания переподключения данного контроллера и при возобновлении соединения синхронизировать его состояние в игре, в этот момент пользователь сможет продолжить прохождение сценария, либо провести проверку для поиска причины отключения устройства, чаще всего такие проблемы могут возникать из-за разряда аккумуляторов контроллеров или помех в работе *Bluetooth*.

На протяжении прохождения определенного сценария, вся информация о выборах пользователя сохраняется для последующего формирования сводной информации о прохождении сценария, отправки на сервер и обработки системой. После прохождения сценария приложение должно сохранять на локальном диске ранее записанный видеофайл, а путь к файлу также отправлять на сервер.

Данные этапы описывает процесс А4. Декомпозиция процесса А4 описывается на рисунке 2.7.

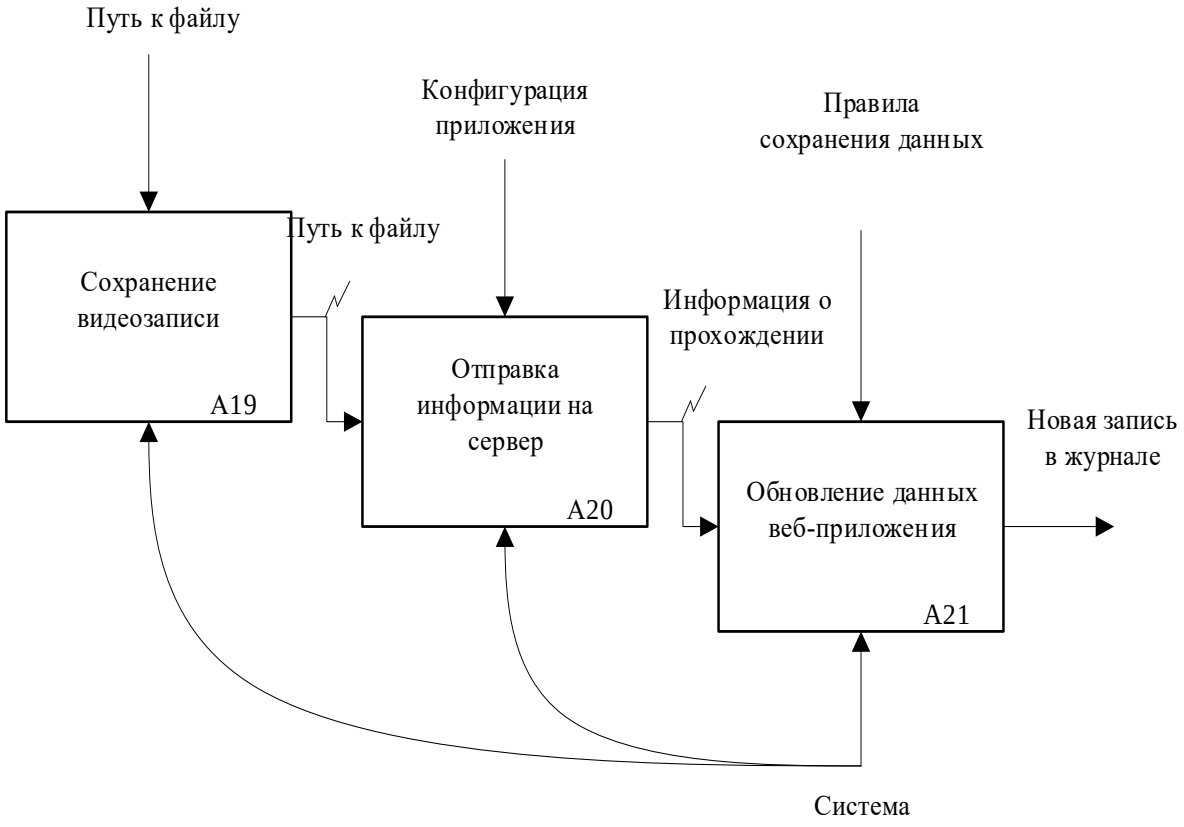

Рисунок 2.7 – Декомпозиция процесса передачи данных

Для передачи информации о прохождении сценария используется *post*-запрос, в теле которого будет передана вся собранная информация. После получения сервером новой информации о прохождении сценария, система должна осуществить проверку файл на соответствие шаблону, также будет проверено существование видеофайла с данным прохождением, если файл будет обнаружен, то на странице с информацией о данном прохождении будет работать возможность скачивания видео с тестом. после этого данные о прохождении будут занесены в общий журнал прохождений и станут доступны для просмотра.

Видео с прохождением будет иметь разрешение, соответствующее разрешению монитора, который был подключен к ПК в момент тестирования, в качестве кодека используется – *h.*264 *NVENC* [9], что позволит переложить нагрузку по кодированию и декодированию видео с центрального процессора на видеокарту.

Процесс А5 описывает возможности получения пользователем сводного отчета о результатах прохождения того или иного сценария.

На рисунке 2.8 описывается декомпозиция процесса А5.

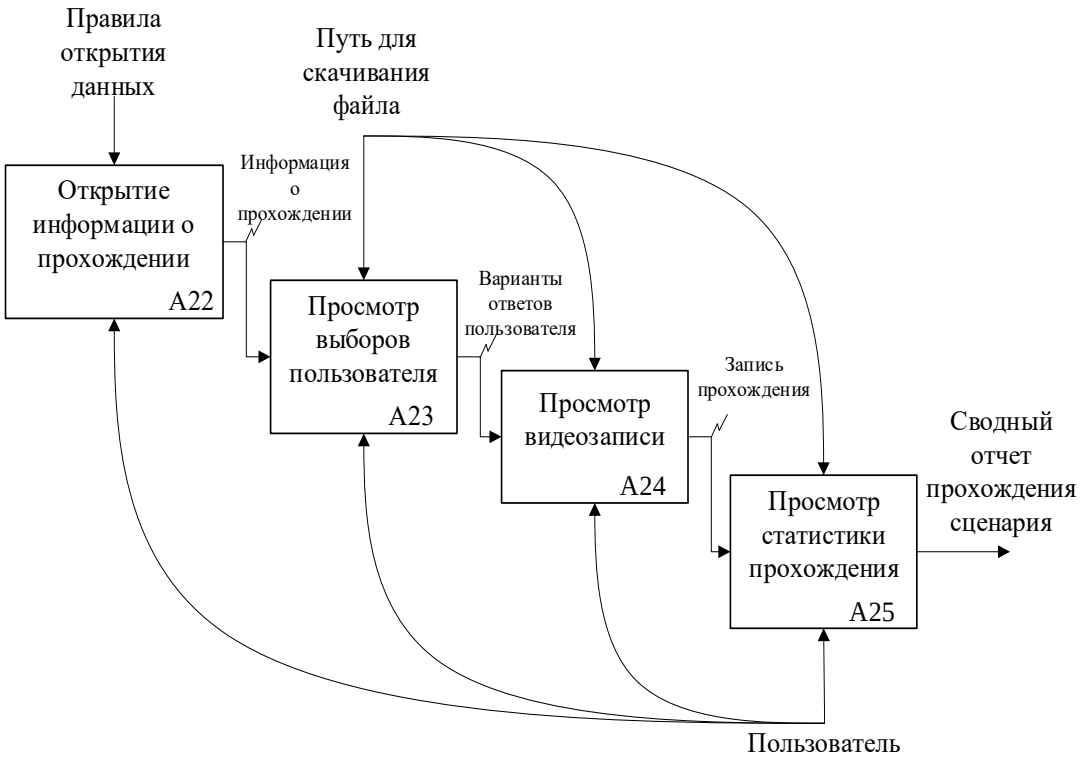

Рисунок 2.8 – Декомпозиция процесса обработки информации на сервере

После добавления прохождения в общий журнал пользователь может просмотреть все информацию о конкретном тесте:

– содержание сценария;
– правильные ответы;

– пояснения к ответам;
– выбранные ответы;
– видеоматериалы прохождения.

Результатом работы данного процесса будет являться сводный отчет о прохождении, информацию о котором при необходимости можно выгрузить в файлы и скачать.

2.3 Модель данных

Для реализации программного комплекса необходимо реализовать несколько основных моделей данных:
– модель, представляющая собой сценарий;
– модель, описывающая результат прохождения сценария;
– модель пользователя в игровом мире;
– модель игрового мира.

Все игровые сценарии будут храниться в *JSON* файлах, в которых содержится вся информация о текущем уровне. Пример структуры модели диалога

```
{
        "numofuestion":"2"
        "0":{
                "phrazes":
                {...
                },
                "comments":
                {...
                },
                "statuses":
                {...
                },
                "moves":
                {...
                },
                "ends":
                {...
                },
                "correct":2,
                "pick":1
        }
}
```

Рисунок 2.9 – Пример структуры диалога

После загрузки приложения и выбора уровня, система произведет загрузку определенного файла с диалогами, обработает их, структурирует и передаст в графический интерфейс диалоговой системы.

Данная модель будет содержать все необходимые поля для полноценного описания диалогов:

– набор всех фраз диалога, который содержит все используемые в данном сценарии фразу, как вопросы, так и ответы на них;

– набор вспомогательной информации диалога;

– набор элементов для перехода к следующему вопросу.

В результате прохождения сценария приложение сформирует новый *json* документ со всей необходимой информацией о прохождении и отправит данный файл на сервер. Данный файл будет содержать сценарий, ответы на вопросы и путь к видеозаписи прохождения.

Модель игрового персонажа, должна содержать все основные элементы и возможности пользователя в приложении. Необходимо дать пользователю возможность взаимодействия с интерактивными элементами мира, возможность передвижения. Класс пользователя состоит из нескольких основных частей: камера персонажа, элементы интерактивного взаимодействия (симуляция рук).

Модель локаций приложения. Приложение должна состоять из двух необходимых локаций:

– стартовой локация, где пользователь выбирает определенный сценарий;

– основной локации, где будет происходить непосредственное прохождение теста.

Исходя из описанной структуры приложения необходимо разработать окружение для двух локаций, связать их вместе создав возможность перехода из одной локации в другую. Провести разработку объектов наполнения виртуального мира для придания наибольшего эффекта присутствия и повышения реализма в приложении. Объектам следует создать реалистичные текстуры и дизайн.

3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ «*VRTRAINER*»

3.1 Структура программного комплекса

На рисунке 3.1 описывается схема разработанного программного комплекса.

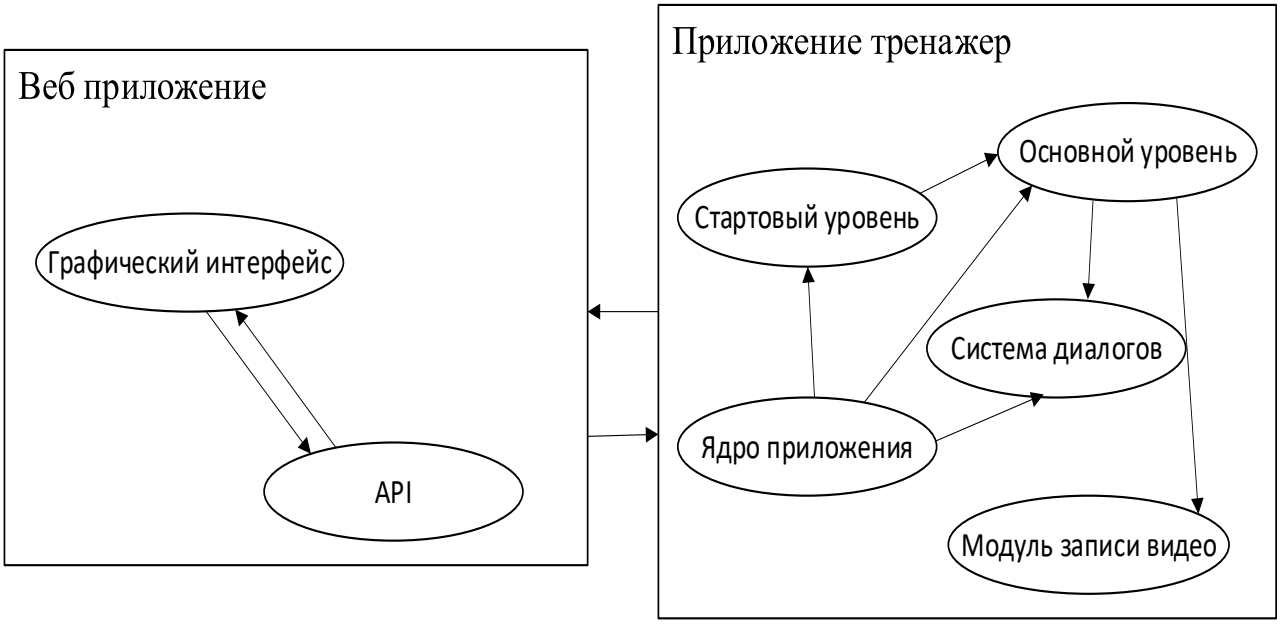

Рисунок 3.1 – Взаимодействие элементов программного комплекса

Программный комплекс состоит из двух независимых приложений, первое представляет собой – приложение виртуальной реальности, для непосредственного прохождения сценариев, второе – веб приложение, для создания, изменения и управления прохождениями и сценариями.

Главное меню приложения должно предоставлять выбор одного из уровней для прохождения, далее следует реализация перехода к одному из четырех уровней, после прохождения тестирования, данные будут сохраняться и передаться для обработки второму приложению. Главное меню должно представлять собой отдельную сцену, которая содержит четыре элемента *Button*, в каждом из которых будут размещены *Image* и *TextBox*. Сцена должна содержать один экземпляр класс *MenuActor*, экземпляр класса *LVLPicker*.

Класс *MenuActor* отвечает за реализацию логики контекстного меню, содержит методы:

– метод *InitializeMap* – метод, отвечающий за инициализацию визуальной части меню;

– метод *OpenLVL* – метод, запускающий загрузку выбранного уровня.

Класс *LVLPicker* представляет логику взаимодействия пользователя с элементами мира.

Второй сценой является основная локация приложения, в которой непосредственно и проходит работа сценария. Данная сцена содержит несколько основных классов *Actor*, а именно классы, отвечающие за передвижение и взаимодействия пользователя с элементами виртуального мира, управление диалогами [10, с. 147]. На рисунке 3.2 изображается список всех элементов, размещенных на данном уровне.

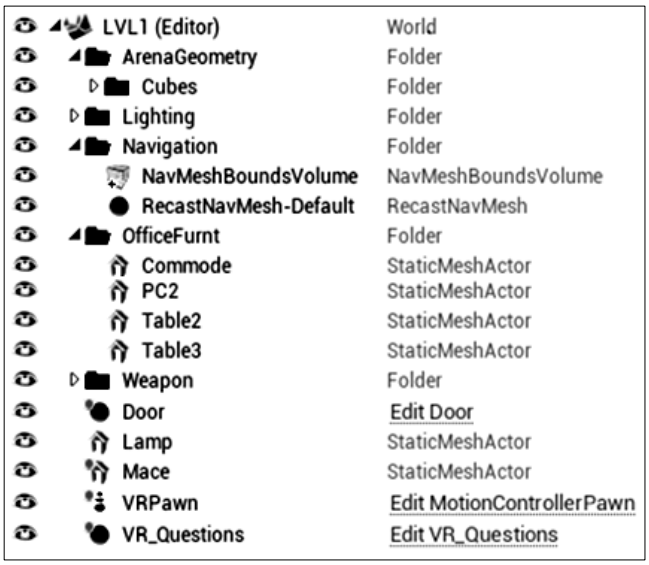

Рисунок 3.2 – Иерархия основной сцены приложения

Основными классами здесь выступают *VRPawn* и *VR_Questions*.

Класс *VRPawn* отвечает за взаимодействие пользователя с объектами, размещенными на текущем уровне, а также функции перемещения. На рисунке 3.3 изображается список методов данного класса.

Рисунок 3.3 – Методы класса *VRPawn*

Класс *VRPawn* содержит следующие методы:

– метод *ExecuteTeleportation* – метод, отвечающий за телепортационное перемещение пользователя. Метод производит просчет траектории, вычисляет

возможность перемещения на новое место, осуществляет смещение камеры на новое место;

– метод *InputActionGrabLeft* – метод, выполняющий симуляцию взаимодействия левого контроллера с виртуальными объектами;

– метод *InputActionGrabRight* – метод, выполняющий симуляцию взаимодействия правого контроллера с виртуальными объектами;

– метод *InputActionTeleportLeft* – метод, осуществляет перемещение пользователя на новую позицию, считывая направление левого контроллера;

– метод *InputActionTeleportRight* – метод, осуществляет перемещение пользователя на новую позицию, считывая направление левого контроллера;

– метод *SpawnActorController* – метод, инициализирующий объект персонаж в виртуальном пространстве, синхронизирует все задействованные устройства (шлем, два контроллера), производит первичную установку положения камеры в пространстве;

– метод *SetTrackingOrigin* – метод, занимающийся установкой модели и версии шлема виртуальной реальности, далее инициализирует методы отслеживания движений пользователя.

На рисунке 3.4 описывается список всех элементов входящих в класс *VRPawn*, в него также входят элемент *Camera* и *VROrign*, отвечающие за визуальные эффекты данного класса.

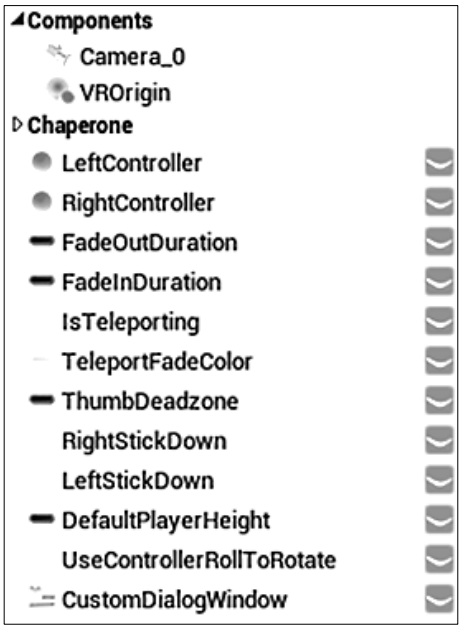

Рисунок 3.4 – Переменные класса *VRPawn*

Список используемых в данном классе переменных представлен ниже:

– переменная *LeftController* – переменная класса *MotionController*, симулирует левую руку пользователя в виртуальной реальности;

– переменная *RightController* – переменная класса *MotionController*, симулирует левую руку пользователя в виртуальной реальности;

– переменная *FadeOutDuration* – переменная типа *Float*, устанавливающая показатель прозрачности вектора перемещения персонажа до перемещения;

– переменная *FadeInDuration* – переменная типа *Float*, устанавливающая показатель прозрачности вектора перемещения персонажа после перемещения;

– переменная *IsTeleporting* – переменная типа *Boolean*, указывает на проходящее в данный момент перемещение в пространстве;

– переменная *TeleportFadeColor* – переменная класса *Color*, описывает цвет анимации вектора перемещения персонажа;

– переменная *ThumbDeadzone* – переменная типа *Float*, регулирующая показатель мертвой зоны вращения с использованием стиков контроллера во время телепортации;

– переменная *RightStickDown* – переменная типа *Boolean* указывает на нажатие правого стика, участвует в инициализации перемещения;

– переменная *LeftStickDown* – переменная типа *Boolean* указывает на нажатие левого стика, участвует в инициализации перемещения;

– переменная *DefaultPlayerHeight* – переменная типа *Float* указывает высоту персонажа по умолчанию;

– переменная *UseControllerRollToRotate* – переменная типа *Boolean* указывает на возможность поворота камеры с использованием трек площадки контроллера, зависит от модели устройства.

Класс *VR_Questions* представляет набор логики, а также визуальных элементов диалоговой системы приложения. На рисунке 3.5 изображены основные методы данного класса. Визуальная часть представляет собой *UI* виджет, на котором размещено четыре кнопки с вариантами ответа, на каждую установлен обработчик нажатия, вызывающий цепочку действий, направленных на переход и сохранение состояния диалога.

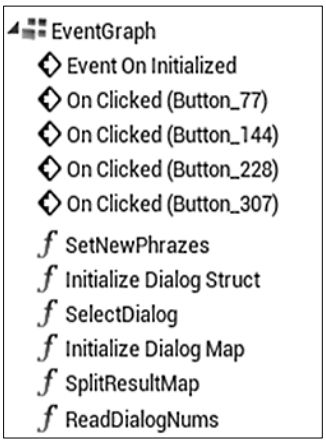

Рисунок 3.5 – Методы и события класса *VR_Questions*

Класс *VR_Questions* содержит следующие методы:

– метод *SetNewPhrazes* – метод, управляющий обновлением текущих отображаемых диалогов на виджете;

– метод *InitializeDialogStruct* – метод, инициализирующий начальное состояние диалога;

– метод *SelectDialog* – метод, обрабатывающий выбор пользователя в приложении, проводит сохранение выбора пользователя;

– метод *InitializeDialogMap* – метод проводит считывание *json*-файла, с последующей инициализацией глобальной структуры сценария;

– метод *SplitResultMap* – метод, обрабатывающий часть *json*-файла, формирует одну итерацию диалога.

На рисунке 3.6 описывается список всех элементов входящих в класс *VR_Questions*.

Рисунок 3.6 – Переменные класса *VR_Questions*

Список используемых в данном классе переменных представлен ниже:

– переменная *CurrentDialogArray* – переменная представляет строковый массив, содержащий текущее состояние диалога;

– переменная *AllDialogs* – переменная представляет собой словарь из всех возможных состояний диалога;

– переменная *CurrentDialogMoves* – переменная, которая хранит массив индексов для перехода к следующей стадии диалога;

– переменная *Selectednumber* – переменная, содержащая ответ пользователя в текущей ситуации;

– переменная *pathToFile* – переменная, содержащая относительный путь к файлу для считывания данных о сценарии;

– переменная *ReadedData* – переменная, содержащая данные после чтения файла со сценарием;

– переменная *CurrentDialogEnd* – переменная, которая хранит массив флагов действий при текущем выборе;

– переменная *CurrentDialogComment* – переменная, которая хранит массив уточнений для определенного варианта выбора;

– переменная *AllChooses* – переменная, содержащая массив всех выборов пользователя;

– переменная *SaveNameFile* – переменная, определяющая путь сохранения данных о прохождении сценария.

Описанные классы являются основными частями *VR* приложения.

Веб-приложение состоит из 10 классов, три из которых являются контроллерами и содержат всю логику приложения: *AllExperimentController*, *AllStatisticController*, *DialogCreatorController*.

Класс *AllExperimentController* содержит методы для управления информацией о всех пройденных сценариях.

Класс *AllStatisticController* содержит методы для получения сводных отчетов о пройденных сценариях.

Класс *DialogCreatorController* содержит методы для создания, удаления, редактирования сценариев.

3.2 Элементы взаимодействия с игровым миром

Для получения информации о состоянии управляющих контроллеров использовалась библиотека функций *SteamVR*. Она имеет множество функций, которые позволяют получить доступ к контроллерам *Vive* (или совместимым аналогам) и базовым станциям [11]. Основными используемыми функциями *SteamVR* являются:

– функция *Get Hand Position and Orientation* – возвращает положение и ориентацию левого или правого контроллера *SteamVR* относительно источника *HMD*;

– функция *Get Tracked Device Position and Orientation* – учитывает идентификатор устройства, возвращает ориентацию и положение устройства относительно источника *HMD*;

– функция *Get valid Tracked Device IDs* – возвращает массив идентификаторов для данного типа устройства.

В качестве элемента управления реализована система телепортаций, позволяющая быстро и эффективно перемещаться пользователь в виртуальном мире на различные дистанции вызывая при этом минимальный дискомфорт и укачивания.

Алгоритм передвижения выглядит следующим образом:

− пользователю необходимо указать область для перемещения, используя специальный указатель;

− с помощью трассировки вычисляется направление взгляда пользователя и просчет нового положения;

− после этого рассчитывается новое положение игрока.

Перемещение в данную точку иногда не может быть выполнено, например оно находится за границей игровой области, или же по направлению встречается непреодолимое препятствие, которое делает передвижение невозможным, тогда пользователь будет перемещен в ближайшую по направлению точку, в которой выполняются все условия передвижения.

Чтобы взаимодействовать с объектами в виртуальной реальности, нужны две вещи: интерфейс, который позволяет говорить объектам, что пользователь взаимодействуете с ними, и компонент для определения и управления этими взаимодействия. Для взаимодействия с объектами используются специальные контроллеры. На рисунке 3.7 изображается три наиболее распространенные версии контроллеров для виртуальной реальности.

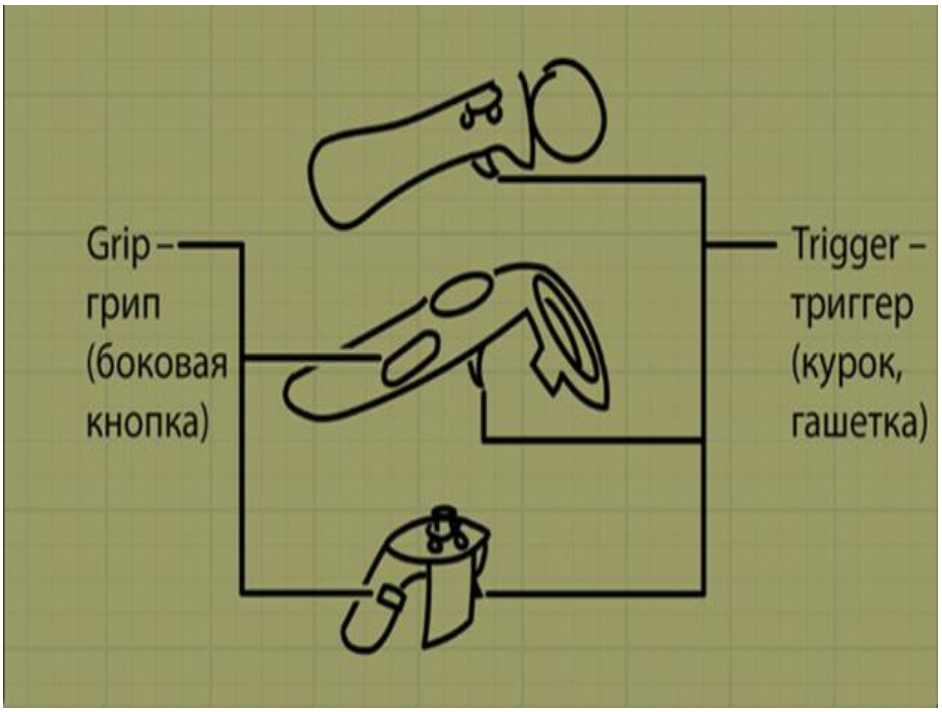

Рисунок 3.7 – Виды контроллеров виртуальной реальности

Приложение осуществляет поддержку ввода от любого из этих трех вариантов строения контроллера. Все три контроллера имеют кнопку под указательный палец («спусковой крючок», или триггер), позволяющую организовать некоторые виды аналогового ввода. Триггер может быть полезен для активации объектов или стрельбы из оружия играх. Кроме того, как *Oculus Touch*, так и

Vive-контроллеры имеют кнопку захвата, которая полезна как специализированная кнопка для подбора объектов.

Для описания положения контроллера в виртуальном пространстве используются различные механики якорей. Самая популярная механика якоря – это симуляция рук на месте расположения контроллеров.

Многие поставщики *VR*-оборудования предлагают точно отслеживаемые контроллеры движения, поэтому теперь есть способ физически представить руки игрока в сюжете и отпадает необходимость в абстрактных системах взаимодействия, которые используют кнопки на геймпаде. Контроллеры движения позволяют игроку физически взаимодействовать с объектами игрового мира. Существует также механика полного отслеживания движения пальцев. Однако на данный момент игры, где можно было бы увидеть такой подход, практически не представлены на рынке. Это обусловлено тем, что только самые последние новинки имеют отслеживаемые контроллеры движения, а это достаточно малый процент в сравнении с уже выпущенными девайсами.

С помощью подходящей функции трассировки *UE*4 можно искать (трассировать) определенные типы объектов в мире, игнорируя те, которые никого не волнуют. Отличие между типами трасс состоит в том, что в то время, как трассировки объекта позволяют определить несколько типов объектов (что может быть полезно, когда выполняется трассировка относительно нескольких типов одновременно, например, игроков и транспортных средств), трассировка относительно канала разрешает выбрать только один канал [12].

Рисунок 3.8 представляет модель руки для обозначения контроллера в пространстве. Синяя область используется для описания нового положения при телепортации, так как эта операция привязана не к игровому аватару, а именно контроллеру.

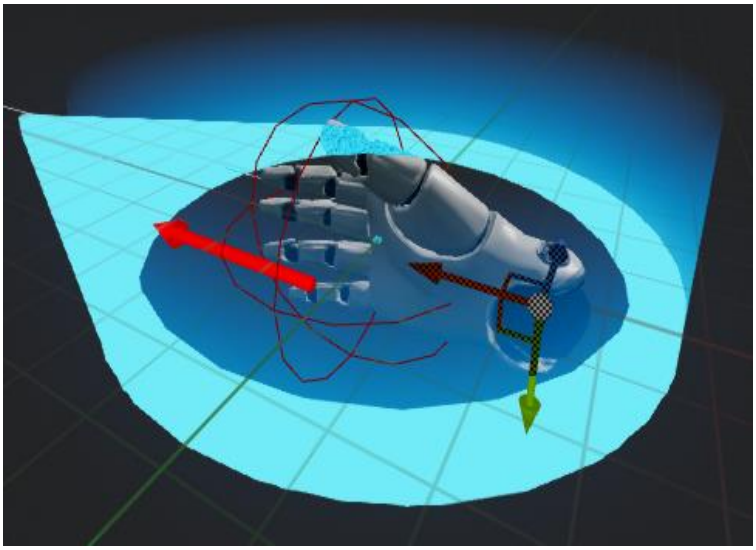

Рисунок 3.8 – Модель якоря контроллера в виртуальной реальности

Также модель руки получила активируемый луч для трассировки направления руки, чтобы было проще понять куда в данный момент производятся наведение. Данная модель позволят пользователю видеть текущее положения руки с контроллером, а также используя луч, выбирать нужный ему объект.

Чтобы настроить контроллеры, в конфигураторе приложения указываются клавиши и триггеры всех доступных устройств и событий, вызываемых ими. Это позволяет проводить отладку функций или игры без наличия контроллеров. В качестве элементов управления диалогами используется виджет на котором размещены несколько кнопок и текстовых полей.

3.3 Объекты виртуального мира

Версия *Unreal Engine* 4 добавила систему *Blueprints*. Данное нововведение являлось быстрым способом создания прототипов игр и элементов игр. Вместо построчного написания кода всё можно делать визуально: перетаскивать ноды (узлы), задавать их свойства в интерфейсе и соединять их «провода». Однако необходимо отметить, что данная возможность обладает рядом ограничений. Поэтому спустя время была введена кросс-поддержка *Blueprints* и *C++* скриптов, что позволило ускорить разработку, создавая скелет с помощью первого инструмента, а затем его улучшение используя скрипты на языке *C++*. Среда упростила механизмы создания и добавления новых материалов, если раньше материалы разрабатывались в сторонних приложениях, то сейчас среда исключает такую необходимость.

Материалы можно наносить на любые объекты, включая меши, частицы и элементы *UI*. Они создаются не с помощью кода, а через сеть визуальных узлов сценариев (материальные выражения) в редакторе материалов. Рисунок 3.9 представляет функции для создания материала.

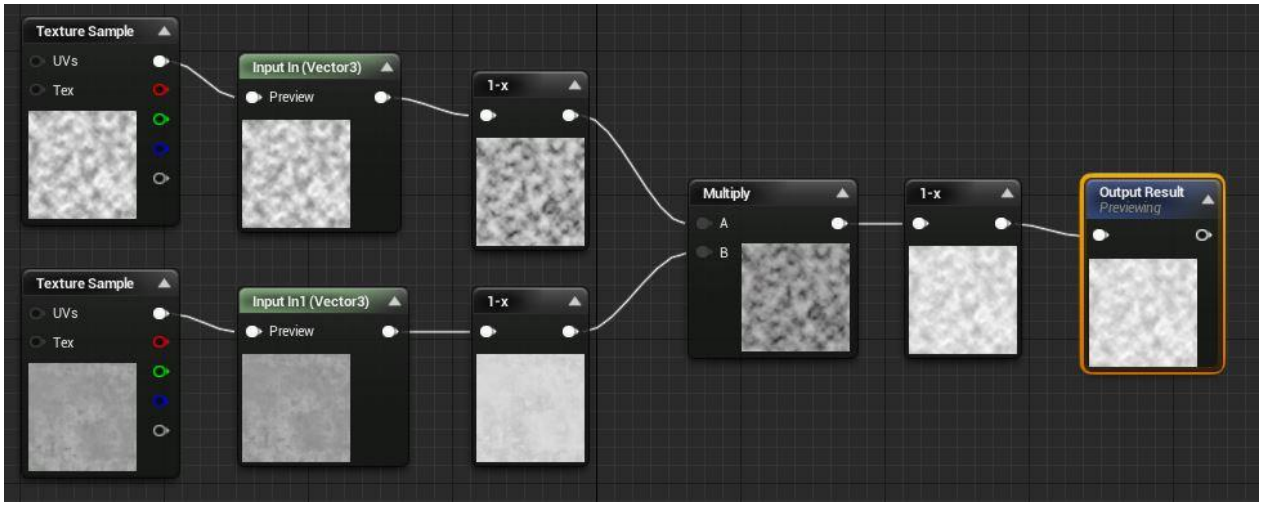

Рисунок 3.9 – Пример формирования материала из нескольких текстур

Каждый узел выражения содержит фрагмент кода *HLSL*, который предназначен для выполнения определенной задачи. Для стен, потолка и пола будут использованы различные вариации кирпичных блоков. Среда позволяет создавать собственные варианты материалов и их физики.

Изначально необходимо найти изображения высокого качества в качестве основы материала. После этого все изображения импортируются и из них создаются различные текстуры с добавлением вариаций шумов и теней. Используя комбинации текстур из них, создаются материалы, которые можно применять к различным, размещенным на уровне, объектам.

В *Unreal Engine* каналы *RGB* имеют интервал значений от 0.0 до 1.0. Управление текстурой осуществляется выполнением операций над каждым пикселем текстуры. Операции могут быть даже простыми, например, добавлением значения к каналам. Обычно в *Unreal Engine* 4 используется текстура типа *TARGA*. Она позволяет работать с каждым каналом отдельно.

После создания всех необходимых материалов, необходимо их применить к каждому объекту на локации. Возможности использования материалов в данной среде является мощным инструментом, с помощью которого можно создавать крайне интересные и сложное виды материалов наподобие камня с растущим в трещинах мхом. Также можно создавать интересные эффекты, например, эффект дезинтеграции объектов.

4 ОПИСАНИЕ ФУНКЦИОНАЛЬНЫХ ПРИЛОЖЕНИЯ «*VRTRAINER*»

4.1 Развертывание разработанного программного комплекса

Для корректной установки и работы приложения ПК должен отвечать некоторому количеству необходимых требований.

Необходимые характеристики ПК для установки и запуска приложения:

– операционная система *Windows* 10 с установленными на ней последними обновлениями;

– процессор *i*5-4590 (аналогичный процессор от *AMD*) или выше;

– наличие внешнего графического ускорителя уровня *NVIDIA GTX* 970 или выше;

– наличие минимум 4 Гб. оперативной памяти или выше;

– наличие одного *USB* порта 2.0 или выше, один порт *HDMI* 1.4, один *DisplayPort* 1.2 или выше.

Данные характеристики в данный момент являются минимальным условием запуска приложений виртуальной реальности на ПК.

В качестве устройства виртуальной реальности использовался шлем *Acer WMR*, рисунок 4.1 описывает данное устройство, который на текущий момент получил широкую популярность из-за поддержки *StemVR*.

Рисунок 4.1 – Устройство *Acer WMR*

Изначально данное устройство поддерживало только технологию *Windows Mixed Reality*, что являлось большим минусом при выборе платформы из-за недостатка приложений и игр, но сейчас, получив доступ к *SteamVR*, платформа стремительно развивается. Сравнительно недорогая система стандарта *plug-and-play* с питанием от ПК стала чрезвычайно привлекательной покупкой, особенно для владельцев устаревших производительных ноутбуков с хорошей встроенной графикой. Шлем виртуальной реальности от *Acer* больше ориентирован как на массовую аудиторию, так и на энтузиастов новых технологий мира *VR*.

Данный продукт является наиболее оптимальным решением по соотношению цены, качества, и предоставляемым им возможностям, сюда так же учитываются параметры самого девайса, шлем достаточно легкий и его можно безболезненно носить в течение часа и больше. Стоит заметить, что основная часть настроек виртуального шлема *Acer* производится автоматически в режиме «включил и пользуйся» (*plug-and-play*).

Для подключения шлема необходим один порт *USB* и один порт *HDMI*, также необходимо наличие встроенного *Bluetooth* или внешний адаптер. После подключения шлема необходимо также подключить контроллеры. Для этого необходимо на них активировать режим подключения и подключить их в настройках операционной системы, рисунке 4.2 описывает пример отображения подключенных контроллеров в настройках.

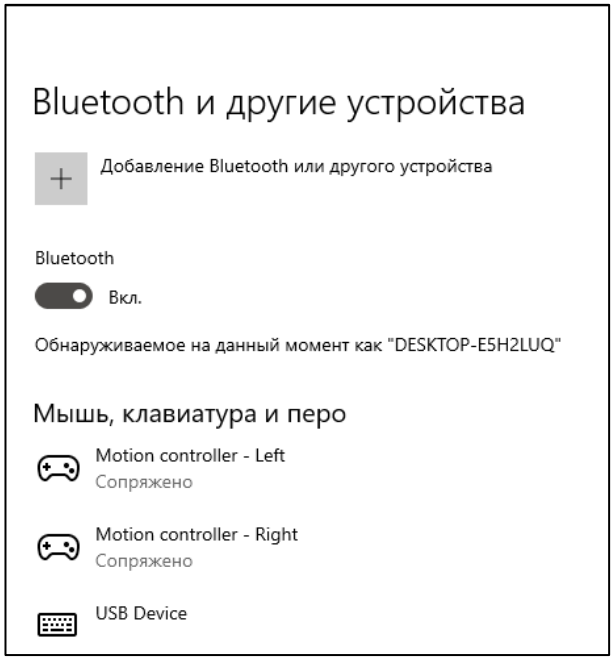

Рисунок 4.2 – Пример подключенных контроллеров

После этого необходимо перейти в приложение *Windows Mixed Reality* для настройки рабочего окружения шлема. Существует два варианта рабочего окружения, сидячий и активный вариант, схема выбора изображается на рисунке 4.3.

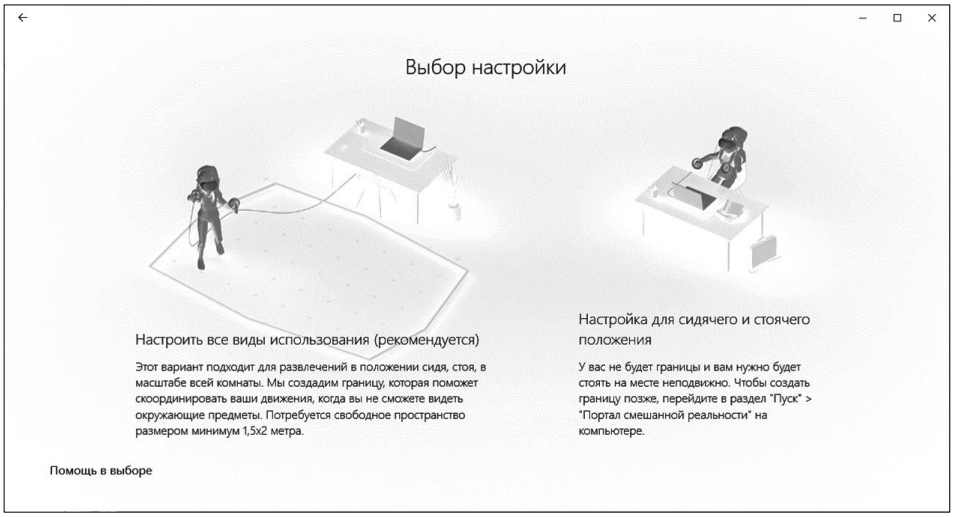

Рисунок 4.3 – Варианты настройки рабочего окружения

Данную процедуру можно производить, как при первом запуске, так и при каждом последующем запуске приложения. Сидячий вариант предполагает использование шлема только в одном положении тела, например, стоя на месте, сидя или лежа, данный вариант используется для работы на месте (например, при разработке виртуального объекта, или просмотре фильма).

Второй вариант предлагает возможность использования шлема в масштабах всей комнаты или другого доступного помещения. Данный вариант используется при любой активной деятельности пользователя.

Далее при выборе активного варианта окружения необходимо создать границу обнаружения устройств, это помогает помнить об окружающем реальном пространстве и объектах, которые могут причинить вред если пользователь покинет данную границу. Рисунок 4.4 описывает сообщение с информацией о границе и пространстве вокруг нее.

Рисунок 4.4 – Размещение границы активного окружения

После того, как площадка будет подготовлена необходимо провести центровку гарнитуры относительно ПК и провести разметку границы, обведя ее шлемом, после этого при выходе контроллеров или шлема из этой зоны пользователь будет получать уведомление об опасности. Рисунок 4.5 описывает пример настройки границы.

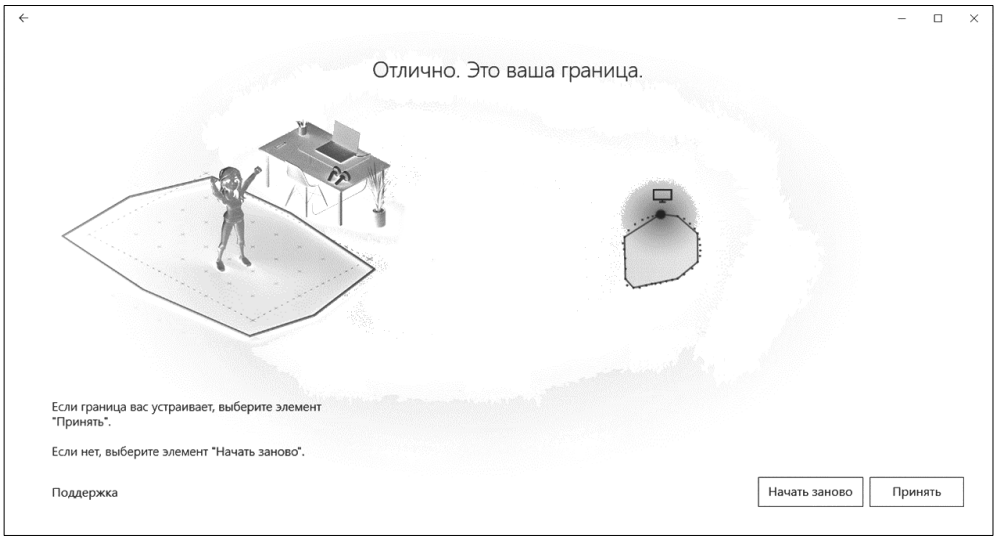

Рисунок 4.5 – Пример создания границы окружения.

После разметки границы необходимо запустить приложение *SteamVR*, которое выведет на экран минимум информации о текущих подключенных устройствах виртуальной реальности, пример подключения изображается на рисунке 4.6. Здесь также может быть представлена информация о текущей нагрузке на гарнитуру, заряде аккумуляторов контроллеров.

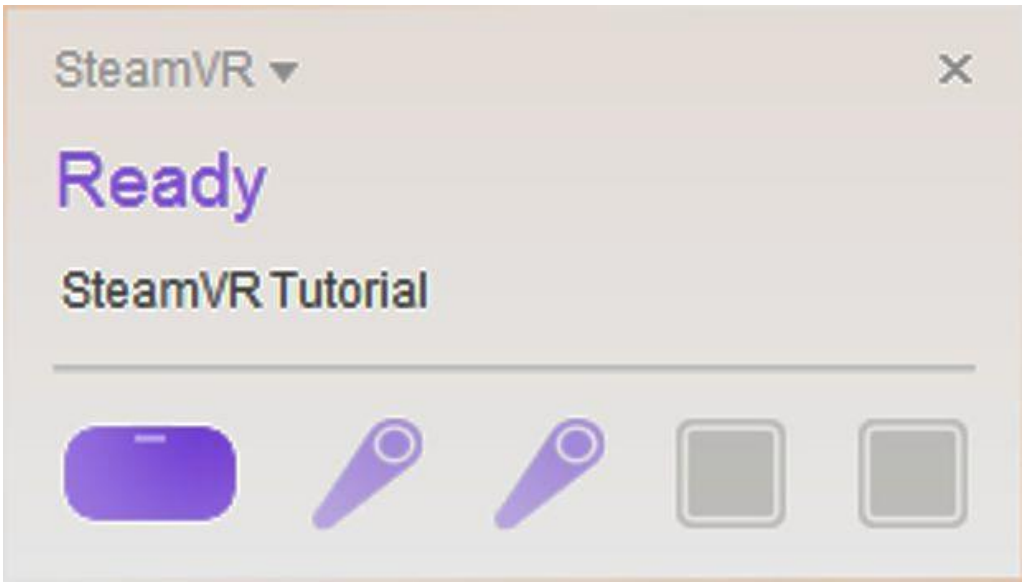

Рисунок 4.6 – Пример подключений *VR* гарнитуры

4.2 Интерфейс пользователя *VR* тренажёра

На рисунке 4.7 приведена диаграмма вариантов использования, которая описывает функциональное назначение разработанного программного средства, взаимоотношение и зависимости между группами вариантов использования и пользователем. Программный комплекс взаимодействует с одним актером – пользователем.

В возможности пользователя входят:

– просмотр всех доступных сценариев;
– выбор и прохождение доступных сценариев;
– загрузка, создание новых сценариев и редактирование существующих;
– просмотр статистики всех ранее пройденных тестов.

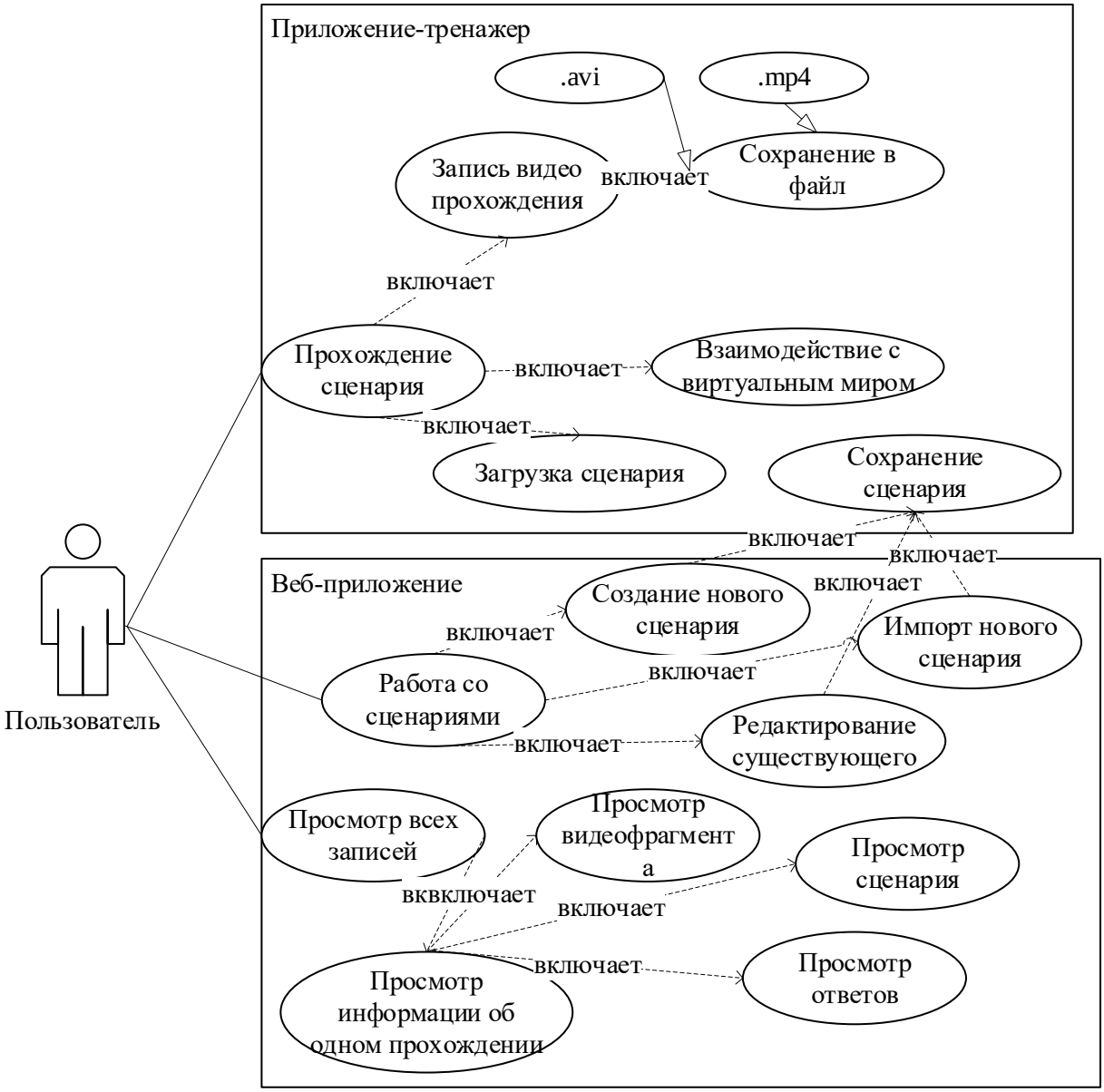

Рисунок 4.7 – Диаграмма вариантов использования приложения

Общий вид программного комплекса сразу после запуска изображен на рисунке 4.8. Здесь пользователь может выбрать для прохождения любой из четырех сценариев.

Рисунок 4.8 – Меню выбора уровня

Каждый отдельный вариант меню может содержать различный сценарий.

После выбора одного из уровней начнется инициализация новой локации, пример данной локации изображается на рисунке 4.9. Данная локация является основной, на которой и проходит непосредственная отработка сценария.

Рисунок 4.9 – Вид разработанного виртуального кабинета

Разработанная сцена содержит:

– элементы пользовательского интерфейса, такие как камера, элементы управления и симуляции рук;

– объекты наполнения рабочего кабинета.

Рисунке 4.10 представляет визуальную структуру разработанной диалоговой системы. Данный объект расположен на основной локации на поверхности мониторов, стоящих на столе.

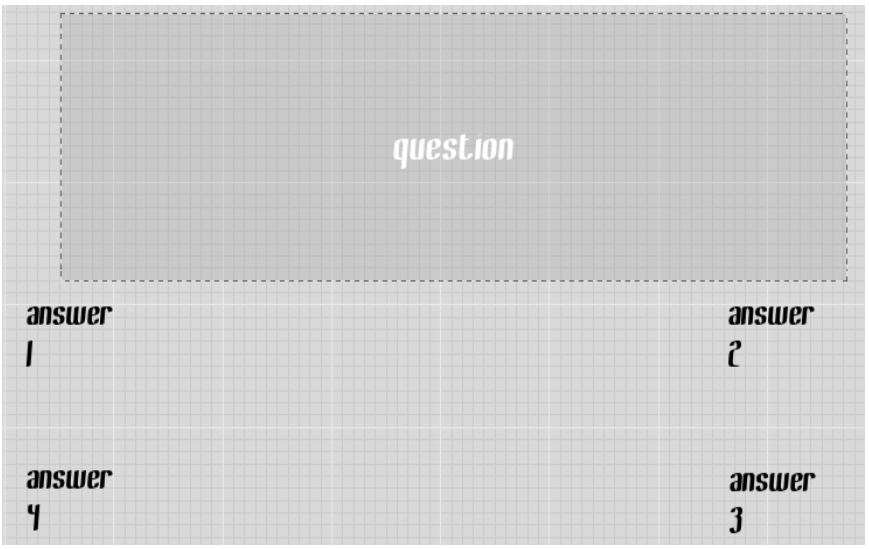

Рисунок 4.10 – Визуальная структура диалоговой системы

После инициализации мира и загрузки диалогов, на данный виджет будет выведена начальное содержание сценария.

4.3 Верификация работы программного комплекса

Основной функцией разработанного приложения является выполнение сценариев для подготовки персонала на предприятии с использованием инструментов виртуальной реальности.

Для тестирования работы приложения необходимо провести проверку следующих функций:

– корректного взаимодействия пользователя с объектами в виртуальной реальности;

– проверка выборов пользователя в диалоговой системе приложения;

– корректность загрузки данных сценариев в приложение;

– проведение записи видео при прохождении теста;

– формирование результатов пройденного теста;

– отображение списка пройденных тестов;

– правильность отображения результатов теста, а также возможность скачивания результатов тестирования;

– общую работоспособность системы при долгой нагрузке.

После запуска определенного теста, пользователю на экран будет выведен вопрос, а также возможные варианты ответа на него. Рисунок 4.11 изображает

пример такого вопроса. Пользователь имеет возможность выбирать ответ из всех представленных вариантов.

Рисунок 4.11 – Пример вопроса в диалоге

В случае неправильного ответа пользователя, есть два варианта результата. Ответ может быть как негативный (полный провал данного сценария), так и нейтральный (сценарий продолжает выполняться). Во втором случае пользователь имеет в запасе варианты пройти сценарий правильно. Если же пользователь дал правильный ответ, то будет совершен переход к следующему вопросу, пример изображается на рисунке 4.12.

Рисунок 4.12 – Пример перехода на следующий вопрос

Перед началом прохождения теста система запускает запись видео, а после окончания прохождения данный видеофайл сохраняется на жесткий диск, пример сохранения файла с прохождением описывается на рисунке 4.13.

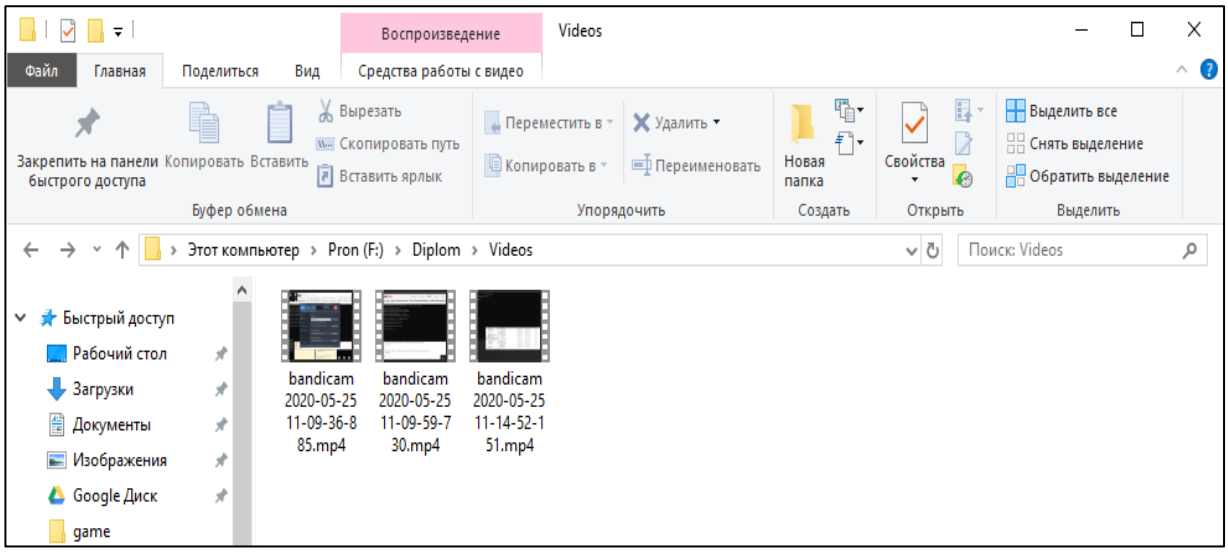

Рисунок 4.13 – Результат работы функции записи видео

Также после прохождения сценария, приложение формирует новый *json* документ, содержащий всю информацию о прохождении, и отправляет его на сервер для последующей обработки. На рисунке 4.14 изображает пример структуры сформированного для отправки на сервер файла.


```
{
    "numofquestion":1,
    "path":"F:\\Diplom\\Videos\\bandicam 2020-05-25 11-09-36-885.mp4",
    "0":
    {
        "pharazes":
        {…
        },
        "comments":
        {…
        },
        "statuses":
        {…
        },
        "moves":
        {…
        },
        "ends":
        {…
        },
        "correct":2,
        "pick":1
    }
}
```


Рисунок 4.14 – Результат функции формирования *json* файла

После того как пользователь пройдет сценарий, информация отправляется на *API* веб-приложения, где пользователь может просмотреть отчет о прохождении, рисунок 4.15 описывает пример отображения информации о выборах пользователя в сценарии.

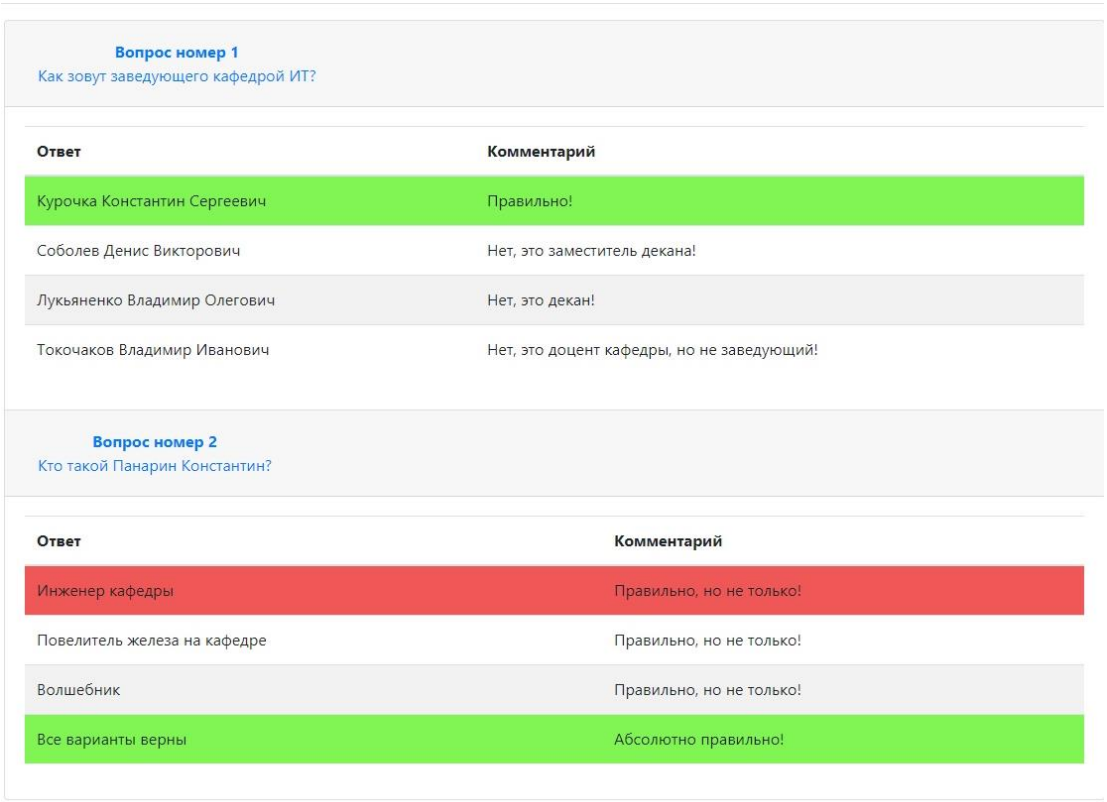

Рисунок 4.15 – Пример отображения выборов пользователя

При просмотре можно свернуть и развернуть каждый вопрос, просмотреть правильные и не правильные ответы, а также получить пояснения и комментарии почему именно так нужно было отвечать на вопрос.

Исходя из полученных результатов тестирования программного комплекса можно сделать вывод о том, что разработанная система работает правильно, так как все основные функции комплекса работают правильно.

5 ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ ДИПЛОМНОЙ РАБОТЫ

5.1 Оценка конкурентоспособности программного обеспечения

Техническая прогрессивность разрабатываемого программного обеспечения определяется коэффициентом эквивалентности ($K_{эк}$) [15, с. 6]:

$$K_{эк} = \frac{K_{т.н}}{K_{т.б}}, \quad (5.1)$$

где $K_{т.н}$, $K_{т.б}$ – коэффициенты технического уровня нового и базисного ПО, которые можно рассчитать по формуле:

$$K_{т} = \sum_{i=1}^{n} \beta \frac{P_i}{P_э}, \quad (5.2)$$

где β – коэффициенты весомости i-го технического параметра;

n – число параметров;

P_i – численное значение i-го технического параметра сравниваемого ПО;

$P_э$ – численное значение i-го технического параметра эталона.

Результат расчета коэффициента эквивалентности приведен в таблице Д.1.

Полученное значение коэффициента эквивалентности больше единицы, следовательно, разрабатываемое программное обеспечение является технически прогрессивным.

Далее рассчитывается коэффициент изменения функциональных возможностей ($K_{ф.в}$) [15, с. 6–7] нового программного обеспечения по формуле:

$$K_{ф.в} = \frac{K_{ф.в.н}}{K_{ф.в.б}}, \quad (5.3)$$

где $K_{ф.в.н}$, $K_{ф.в.б}$ – балльная оценка неизмеримых показателей нового и базового изделия соответственно.

Результат расчета коэффициента изменения функциональных возможностей нового программного обеспечения приведен в таблице Д.2.

Коэффициент функциональных возможностей превышает единицу, т. е. новое программное обеспечение превосходит по своим функциональным возможностям базовый в 1,4 раза.

Конкурентоспособность нового программного обеспечения можно оценить с помощью интегрального коэффициента конкурентоспособности ($K_{и}$) [15, с. 8] по формуле:

$$K_{и} = \frac{K_{ф.в} \cdot K_{н} \cdot K_{эн}}{K_{ц}}, \quad (5.4)$$

где $K_{н}$ – коэффициент соответствия нового ПО нормативам ($K_{н} = 1$);

$K_{ц}$ – коэффициент цены потребления ($K_{ц} = 0{,}95$).

Расчёт интегрального коэффициента конкурентоспособности:

$$K_{и} = \frac{1{,}9 \cdot 1{,}4 \cdot 1}{0{,}95} = 2{,}8.$$

Интегральный коэффициент конкурентоспособности больше единицы, т. е. новое программное обеспечение является более конкурентоспособным, чем базовый.

5.2 Оценка трудоемкости работ по созданию программного обеспечения

Общий объем программного обеспечения ($V_{о}$) [15, с. 9] определяется исходя из объема функций, реализуемых программой, по каталогу функций программного обеспечения по формуле (5.5):

$$V_{о} = \sum_{i=1}^{n} V_{i}, \quad (5.5)$$

где V_{i} – объем отдельной функции ПО;

n – общее число функций.

Уточненный объем программного обеспечения ($V_{у}$) [15, с. 9] определяется по формуле:

$$V_{у} = \sum_{i=1}^{n} V_{уi}, \quad (5.6)$$

где $V_{уi}$ – уточненный объем отдельной функции ПО.

В качестве единицы измерения объема программного обеспечения может быть использована строка исходного кода (*LOC*). Результаты расчетов представлены в таблице Д.3.

Все программные обеспечения в зависимости от их характеристик подразделяются на три категории сложности.

На основании принятого к расчету (уточненного) объема и категории сложности программного обеспечения определяется нормативная трудоемкость на разработку программного обеспечения выполняемых работ.

Дополнительные затраты труда, связанные с повышением сложности разрабатываемого программного обеспечения, учитывается посредством коэффициента повышения сложности программного обеспечения.

Результаты расчетов представлены в таблице Д.4.

Значение коэффициентов удельных весов трудоемкости определяются с учетом установленной категории новизны программного обеспечения приведены таблице Д.5.

Сумма значений коэффициентов удельных весов всех стадий в общей трудоемкости равна единице.

Общая трудоемкость разработки программного обеспечения ($Т_о$) [15, с. 11–12] определяется суммированием нормативной (скорректированной) трудоемкости программного обеспечения по стадиям разработки по формуле:

$$Т_о = \sum_{i=1}^{n} Т_{уi}, \tag{5.7}$$

где $Т_{уi}$ – нормативная (скорректированная) трудоемкость разработки ПО на *i*-й стадии, чел.-дн.;

n – количество стадий разработки.

Нормативная трудоемкость программного обеспечения ($Т_н$) выполняемых работ по стадиям разработки корректируется с учетом следующих коэффициентов:

– повышения сложности ПО ($K_с$);

– учитывающих новизну ПО ($K_н$);

– учитывающих степень использования стандартных модулей ($K_т$);

– средства разработки ПО ($K_{ур}$).

Данные коэффициенты определяются по формулам:

– для стадии технического задания (ТЗ) $Т_{утз} = Т_н \cdot K_с \cdot K_н \cdot K_{у.р}$;

– для стадии эскизного проекта (ЭП) $Т_{уэп} = Т_н \cdot K_{э.п} \cdot K_с \cdot K_н \cdot K_{у.р.}$;

– для стадии технического проекта (ТП) $Т_{утп} = Т_н \cdot K_{т.п} \cdot K_с \cdot K_н \cdot K_{у.р}$;

– для стадии рабочего проекта (РП) $\mathrm{Т}_{\text{урп}} = \mathrm{Т}_{\text{н}} \cdot K_{\text{р.п}} \cdot K_{\text{с}} \cdot K_{\text{н}} \cdot K_{\text{т}} \cdot K_{\text{у.р}}$;

– для стадии внедрения (ВН) $\mathrm{Т}_{\text{увн}} = \mathrm{Т}_{\text{н}} \cdot K_{\text{вн}} \cdot K_{\text{с}} \cdot K_{\text{н}} \cdot K_{\text{у.р}}$.

Коэффициенты $K_{\text{с}}$, $K_{\text{н}}$, $K_{\text{ур}}$, вводятся на всех стадиях разработки, а коэффициент $K_{\text{т}}$ вводится только на стадии рабочего проекта:

$$\mathrm{Т}_{\text{утз}} = 957 \cdot 0{,}08 \cdot 1{,}06 \cdot 0{,}63 \cdot 0{,}5 = 26 \text{ чел.-дн.},$$

$$\mathrm{Т}_{\text{уэп}} = 957 \cdot 0{,}19 \cdot 1{,}06 \cdot 0{,}63 \cdot 0{,}5 = 61 \text{ чел.-дн.},$$

$$\mathrm{Т}_{\text{утп}} = 957 \cdot 0{,}28 \cdot 1{,}06 \cdot 0{,}63 \cdot 0{,}5 = 89 \text{ чел.-дн.},$$

$$\mathrm{Т}_{\text{утз}} = 957 \cdot 0{,}34 \cdot 1{,}06 \cdot 0{,}63 \cdot 0{,}65 \cdot 0{,}5 = 71 \text{ чел.-дн.},$$

$$\mathrm{Т}_{\text{увн}} = 957 \cdot 0{,}11 \cdot 1{,}06 \cdot 0{,}63 \cdot 0{,}5 = 35 \text{ чел.-дн.}$$

Результаты расчетов по определению нормативной и скорректированной трудоемкости программного обеспечения по стадиям и общую трудоемкость разработки ($\mathrm{Т}_{\text{о}}$) целесообразно оформить в виде таблицы Д.6.

5.3 Расчёт затрат на разработку программного обеспечения

Суммарные затраты на разработку программного обеспечения ($\mathrm{З}_{\text{р}}$) [15, с. 16–25] определяются по формуле:

$$\mathrm{З}_{\text{р}} = \mathrm{З}_{\text{тр}} + \mathrm{З}_{\text{эт}} + \mathrm{З}_{\text{тех}} + \mathrm{З}_{\text{м.в}} + \mathrm{З}_{\text{мат}} + \mathrm{З}_{\text{общ.пр}} + \mathrm{З}_{\text{непр}}. \quad (5.8)$$

Параметры расчета производственных затрат на разработку программного обеспечения приведены в таблице Д.7.

Расходы на оплату труда разработчиков с отчислениями ($\mathrm{З}_{\text{тр}}$) определяются по формуле:

$$\mathrm{З}_{\text{тр}} = \mathrm{ЗП}_{\text{осн}} + \mathrm{ЗП}_{\text{доп}} + \mathrm{ОТЧ}_{\text{зп}}, \quad (5.9)$$

где $\mathrm{ЗП}_{\text{осн}}$ – основная заработная плата разработчиков, руб.;

$\mathrm{ЗП}_{\text{доп}}$ – дополнительная заработная плата разработчиков, руб.;

$ОТЧ_{зп}$ – сумма отчислений от заработной платы (социальные нужды, страхование от несчастных случаев), руб.

Основная заработная плата разработчиков рассчитывается по формуле:

$$ЗП_{осн} = С_{ср.час} \cdot Т_{о} \cdot K_{ув}, \quad (5.10)$$

где $С_{ср.час}$ – средняя часовая тарифная ставка, руб./час;

$Т_{о}$ – общая трудоемкость разработки, чел.-час;

$K_{ув}$ – коэффициент доплаты стимулирующего характера, $K_{ув} = 1,6$.

Средняя часовая тарифная ставка определяется по формуле:

$$С_{ср.час} = \frac{\sum_{i} С_{чi} \cdot m_i}{\sum_{i} n_i}, \quad (5.11)$$

где $С_{чi}$ – часовая тарифная ставка разработчика *i*-й категории, руб./час;

n_i – количество разработчиков *i*-й категории.

Часовая тарифная ставка определяется путем деления месячной тарифной ставки на установленный при восьмичасовом рабочем дне фонд рабочего времени 168 ч. ($F_{мес}$):

$$С_{ч} = \frac{С_{м1} \cdot Т_{ki}}{F_{мес}}, \quad (5.12)$$

где $С_{м1}$ – тарифная ставка 1-го разряда, руб.;

$Т_{ki}$ – тарифный коэффициент.

Расчёт часовой тарифной ставки, средней часовой тарифной ставки и основной заработной платы разработчиков:

$$С_{чi} = \frac{185 \cdot 2,65}{168} = 2,92 \text{ руб.},$$

$$С_{ср.час} = \frac{2,92 \cdot 1}{1} = 2,92 \text{ руб.},$$

$$ЗП_{осн} = 2,92 \cdot 282 \cdot 8 \cdot 1,6 = 10540 \text{ руб.}$$

Дополнительная заработная плата разработчиков включает выплаты, предусмотренные законодательством о труде, и определяется по формуле:

$$ЗП_{доп} = \frac{ЗП_{осн} \cdot Н_{доп}}{100}, \quad (5.13)$$

где $Н_{доп}$ – норматив на дополнительную заработную плату разработчиков, %.

Расчёт дополнительной заработной платы:

$$ЗП_{доп} = \frac{10540 \cdot 12}{100} = 1264,8 \text{ руб.}$$

Отчисления от основной и дополнительной заработной платы (отчисления на социальные нужды и обязательное страхование) рассчитывается по формуле:

$$ОТЧ_{с.н} = \frac{(ЗП_{осн} + ЗП_{доп}) \cdot Н_{з.п}}{100}, \quad (5.14)$$

где $Н_{з.п}$ – процент отчислений на социальные нужды и обязательное страхование от суммы основной и дополнительной заработной платы ($Н_{з.п} = 34\%$).

Расчёт отчислений от основной и дополнительной заработной платы:

$$ОТЧ_{с.н} = \frac{(10540 + 1264,8) \cdot 34}{100} = 4013,63 \text{ руб.}$$

Расчёт затрат на оплату труда разработчиков с отчислениями:

$$З_{тр} = 10540 + 1264,8 + 4013,63 = 15818,43 \text{ руб.}$$

Затраты машинного времени ($З_{м.в}$) определяются по формуле:

$$З_{м.в} = С_{ч} \cdot K_{т} \cdot t_{эвм}, \quad (5.15)$$

где $С_{ч}$ – стоимость 1 часа машинного времени, руб./ч;

$K_{т}$ – коэффициент мультипрограммности, показывающий распределение времени работы ЭВМ в зависимости от количества пользователей ЭВМ, $K_{т} = 1$;

$t_{эвм}$ – машинное время ЭВМ, необходимое для разработки и отладки проекта, ч.

Стоимость 1 машино-часа определяется по формуле:

$$С_{ч} = \frac{ЗП_{об} + З_{ар} + З_{ам} + З_{э.п} + З_{в.м} + З_{т.р} + З_{пр}}{F_{эвм}}, \quad (5.16)$$

где $ЗП_{об}$ – затраты на заработную плату обслуживающего персонала с учетом всех отчислений, руб./год;

$З_{ар}$ – стоимость аренды помещения под размещение вычислительной техники, руб./год;

$З_{ам}$ – амортизационные отчисления за год, руб./год;

$З_{э.п}$ – затраты на электроэнергию, руб./год;

$З_{в.м}$ – затраты на материалы, необходимые для обеспечения нормальной работы ПЭВМ (вспомогательные), руб./год;

$З_{т.р}$ – затраты на текущий и профилактический ремонт ЭВМ, руб./год;

$З_{пр}$ – прочие затраты, связанные с эксплуатацией ПЭВМ, руб./год;

$F_{эвм}$ – действительный фонд времени работы ЭВМ, ч./год.

Затраты на заработную плату обслуживающего персонала ($ЗП_{об}$) определяются по формулам:

$$ЗП_{об} = \frac{ЗП_{осн.об} + ЗП_{доп.об} + ОТЧ_{эп.об}}{Q_{эвм}}, \quad (5.17)$$

$$ЗП_{осн.об} = 12 \cdot \sum_{i} \left(С_{м.об i} \cdot n_i \right), \quad (5.18)$$

$$ЗП_{доп.об} = \frac{ЗП_{осн.об} \cdot Н_{доп}}{100}, \quad (5.19)$$

$$ОТЧ_{зп.об} = \frac{(ЗП_{осн.об} + ЗП_{доп.об}) \cdot Н_{з.п}}{100}, \quad (5.20)$$

где $ЗП_{осн.об}$ – основная заработная плата обслуживающего персонала, руб.;

$ЗП_{доп.об}$ – дополнительная заработная плата обслуживающего персонала, руб.;

$ОТЧ_{эп.об}$ – сумма отчислений от заработной платы (социальные нужды, страхование от несчастных случаев), руб.;

$Q_{эвм}$ – количество обслуживаемых ПЭВМ, шт.;

$С_{м.об i}$ – месячная тарифная ставка *i*-го работника, руб.;

n – численность обслуживающего персонала, чел.;

$Н_{доп}$ – процент дополнительной заработной платы обслуживающего персонала от основной;

$Н_{з.п}$ – процент отчислений на социальные нужды и обязательное страхование от суммы основной и дополнительной заработной платы.

Расчёт затрат на заработную плату обслуживающего персонала:

$$ЗП_{осн.об} = 12 \cdot 185 = 2220 \text{ руб.},$$

$$ЗП_{доп.об} = \frac{2220 \cdot 12}{100} = 266,4 \text{ руб.},$$

$$ОТЧ_{зп.об} = \frac{(2220 + 266,4) \cdot 34}{100} = 845,38 \text{ руб.},$$

$$ЗП_{об} = \frac{2220 + 266,4 + 845,38}{1} = 3331,78 \text{ руб.}$$

Годовые затраты на аренду помещения ($З_{ар}$) определяются по формуле:

$$З_{ар} = \frac{С_{ар} \cdot S}{Q_{эвм}}, \quad (5.21)$$

где $С_{ар}$ – средняя годовая ставка арендных платежей, руб./м^2;

S – площадь помещения, м^2.

Расчёт годовых затрат на аренду помещения:

$$З_{ар} = \frac{15,2 \cdot 12}{1} = 182,4 \text{ руб.}$$

Сумма годовых амортизационных отчислений ($З_{ам}$) определяется по формуле:

$$З_{ам} = \frac{\sum_i З_{прi} \cdot \left(1 + K_{доп}\right) \cdot m_i \cdot Н_{амi}}{Q_{эвм}}, \quad (5.22)$$

где $З_{прi}$ – затраты на приобретение *i*-го вида основных фондов, руб.;

$K_{доп}$ – коэффициент, дополнительных затраты, связанные с доставкой, монтажом и наладкой оборудования, $K_{доп} = 13\%$ от $З_{пр}$;

$З_{прi} \cdot \left(1 + K_{доп}\right)$ – балансовая стоимость ЭВМ, руб.;

m_i – количество оборудования *i*-го вида;

$Н_{амi}$ – норма амортизации, %.

Расчёт суммы годовых амортизационных отчислений:

$$З_{ам} = \frac{2500 \cdot (1 + 0{,}13) \cdot 1 \cdot 0{,}125}{1} = 353{,}13 \text{ руб.}$$

Стоимость электроэнергии, потребляемой за год ($З_{э.п}$), определяется по формуле:

$$З_{э.п} = \frac{М_{сум} \cdot F_{эвм} \cdot С_{эл} \cdot A}{Q_{эвм}}, \quad (5.23)$$

где $М_{сум}$ – паспортная мощность ПЭВМ, $М_{сум} = 0{,}41$ кВт;

$F_{эвм}$ – действительный годовой фонд времени работы ПЭВМ, $F_{эвм} = 1619$ ч.;

$С_{эл}$ – стоимость одного кВт-часа электроэнергии, руб.;

A – коэффициент интенсивного использования мощности, $A = 0{,}98...0{,}9$.

Расчёт стоимости электроэнергии, потребляемой за год:

$$З_{э.п} = \frac{0{,}5 \cdot 1619 \cdot 0{,}33048 \cdot 0{,}98}{1} = 262{,}17 \text{ руб.}$$

Затраты на материалы ($З_{в.м}$), необходимые для обеспечения нормальной работы ПЭВМ составляют около одного процента от балансовой стоимости ЭВМ и определяются по формуле:

$$З_{в.м} = \frac{\sum_i З_{прi} \cdot \left(1 + K_{доп}\right) \cdot m_i}{Q_{эвм}} \cdot K_{м.з}, \quad (5.24)$$

где $З_{пр}$ – затраты на приобретение (стоимость) ЭВМ, руб.;

$K_{доп}$ – коэффициент, характеризующий дополнительные затраты, связанные с доставкой, монтажом и наладкой оборудования, $K_{доп} = 12 - 13\%$ от $З_{пр}$;

$K_{\text{м.з}}$ – коэффициент, характеризующий затраты на вспомогательные материалы ($K_{\text{м.з}} = 0,01$).

Расчёт затрат на материалы:

$$З_{\text{в.м}} = \frac{2500 \cdot (1 + 0,13)}{1} \cdot 0,01 = 28,25 \text{ руб.}$$

Затраты на текущий и профилактический ремонт ($З_{\text{т.р}}$) принимаются равными в промежутке от пяти до девяти процентов от балансовой стоимости ЭВМ и определяются по формуле:

$$З_{\text{т.р}} = \frac{\sum_i З_{\text{пр}i} \cdot (1 + K_{\text{доп}}) \cdot m_i}{Q_{\text{эвм}}} \cdot K_{\text{т.р}}, \quad (5.25)$$

где $K_{\text{т.р}}$ – коэффициент, характеризующий затраты на текущий и профилактический ремонт, $K_{\text{т.р}} = 0,05$.

Расчёт затрат на текущий и профилактический ремонт:

$$З_{\text{т.р}} = \frac{2500 \cdot (1 + 0,13)}{1} \cdot 0,05 = 141,25 \text{ руб.}$$

Прочие затраты, связанные с эксплуатацией ЭВМ ($З_{\text{пр}}$), состоят из амортизационных отчислений на здания, стоимости услуг сторонних организаций и составляют пять процентов от балансовой стоимости и определяются по формуле:

$$З_{\text{пр}} = \frac{\sum_i З_{\text{пр}i} \cdot (1 + K_{\text{доп}}) \cdot m_i}{Q_{\text{эвм}}} \cdot K_{\text{пр}}, \quad (5.26)$$

где $K_{\text{пр}}$ – коэффициент, характеризующий размер прочих затрат, связанных с эксплуатацией ЭВМ ($K_{\text{пр}} = 0,05$).

Расчёт прочих затрат, связанных с эксплуатацией ЭВМ:

$$З_{\text{пр}} = \frac{2500 \cdot (1 + 0,13)}{1} \cdot 0,05 = 141,25 \text{ руб.}$$

Для расчета машинного времени ЭВМ ($t_{эвм}$ в часах), необходимого для разработки и отладки проекта, следует использовать формулу:

$$t_{эвм} = (t_{р.п} + t_{вн}) \cdot F_{см} \cdot K_{см}, \quad (5.27)$$

где $t_{р.п}$ – срок реализации стадии «Рабочий проект» (РП);

$t_{вн}$ – срок реализации стадии «Ввод в действие» (ВП);

$F_{см}$ – продолжительность рабочей смены, $F_{см} = 8$ ч.;

$K_{см}$ – количество рабочих смен, $K_{см} = 1$.

Расчёт машинного времени ЭВМ в часах:

$$t_{эвм} = 26 \cdot 8 \cdot 1 = 208 \text{ ч.}$$

Расчёт стоимости 1 машино-часа:

$$С_{ч} = \frac{3331,78 + 182,4 + 353,13 + 262,17 + 28,25 + 141,25 + 141,25}{1619} = 2,7 \text{ руб/ч.}$$

Расчёт затрат машинного времени:

$$З_{мв} = 2,7 \cdot 1 \cdot 208 = 561,6 \text{ руб.}$$

Затраты на изготовление эталонного экземпляра ($З_{эт}$) осуществляется по формуле:

$$З_{эт} = (З_{т.р} + З_{тех} + З_{мв}) \cdot К_{эт}, \quad (5.28)$$

где $К_{эт}$ – коэффициент, учитывающий размер затрат на изготовление эталонного экземпляра, $К_{эт} = 0,05$.

Расчёт затрат на изготовление эталонного экземпляра:

$$З_{эт} = (141,25 + 0 + 561,6) \cdot 0,05 = 35,14 \text{ руб.}$$

Затраты на материалы (носители информации и прочее) ($З_{мат}$), необходимые для обеспечения нормальной работы ПЭВМ, рассчитываются по формуле:

$$З_{мат} = \sum_{i=1}^{n} (Ц_i \cdot N_i \cdot (1 + K_{т.з}) - Ц_{оi} \cdot N_{оi}), \quad (5.29)$$

где $Ц_i$ – цена *i*-го наименования материала полуфабриката, комплектующего, руб.;

N_i – потребность в *i*-м материале, полуфабрикате, комплектующем, натур. ед.;

$K_{т.з}$ – коэффициент, учитывающий сложившийся процент транспортно-заготовительных расходов в зависимости от способа доставки товаров, $K_{т.з} = 0,1 - 1,3$;

$Ц_{оi}$ – цена возвратных отходов *i*-го наименования материала, руб.;

$N_{оi}$ – количество возвратных отходов *i*-го наименования, натур. ед.;

n – количество наименований материалов, полуфабрикатов и т. д.

Расчёт затрат на материалы:

$$З_{мат} = 2500 \cdot (1 + 0,13) \cdot 0,01 = 28,25 \text{ руб.}$$

Общепроизводственные затраты ($З_{общ.пр}$) рассчитываются по формуле:

$$З_{общ.пр} = \frac{ЗП_{осн} \cdot Н_{доп}}{100}, \quad (5.30)$$

где $Н_{доп}$ – норматив общепроизводственных затрат, %.

Расчёт общепроизводственных затрат:

$$З_{общ.пр} = \frac{10540 \cdot 10}{100} = 1054 \text{ руб.}$$

Непроизводственные (коммерческие) затраты ($З_{непр}$) рассчитываются по формуле:

$$З_{непр} = \frac{ЗП_{осн} \cdot Н_{непр}}{100}, \quad (5.31)$$

где $Н_{непр}$ – норматив непроизводственных затрат, %.

Расчёт непроизводственных (коммерческих) затрат:

$$З_{непр} = \frac{10540 \cdot 5}{100} = 527 \text{ руб.}$$

Итого получаем суммарные затраты на разработку:

$$З_р = 141,25 + 35,14 + 0 + 561,6 + 28,25 + 1054 + 527 = 2347,24 \text{ руб.}$$

Итог расчёта суммарных затрат на разработку программного обеспечения приведён в таблице Д.8.

5.4 Расчёт договорной цены разрабатываемого программного обеспечения

Оптовая цена программного обеспечения ($Ц_{опт}$) [15, с. 26–27] определяется по следующим формулам:

$$Ц_{опт} = С(З_р) + П_р, \quad (5.32)$$

$$П_р = \frac{С(З_р) \cdot У_р}{100}, \quad (5.33)$$

где $С(З_р)$ – себестоимость ПО, руб.;

$П_р$ – прибыль от реализации ПО, руб.;

$У_р$ – уровень рентабельности ПО, ($У_р = 40\%$).

Расчёт прибыли и оптовой цены программного обеспечения:

$$П_р = \frac{2347,24 \cdot 40}{100} = 938,90 \text{ руб.},$$

$$Ц_{опт} = 2347,24 + 938,90 = 3286,14 \text{ руб.}$$

Прогнозируемая отпускная цена программного обеспечения ($Ц_{отп}$) рассчитывается по формуле:

$$Ц_{отп} = З_р + П_р + Р_{ндс}. \quad (5.34)$$

Налог на добавленную стоимость ($Р_{ндс}$) рассчитывается по формуле:

$$Р_{ндс} = \frac{\left(З_р + П_р\right) \cdot Н_{ндс}}{100}, \quad (5.35)$$

где $Н_{ндс}$ – ставка налога на добавленную стоимость, $Н_{ндс} = 20\%$.

Расчёт налога на добавленную стоимость и прогнозируемую отпускную цену программного обеспечения:

$$Р_{ндс} = \frac{\left(2347,24 + 938,9\right) \cdot 20}{100} = 657,23 \text{ руб.},$$

$$Ц_{отп} = 2347,24 + 938,9 + 657,23 = 3943,37 \text{ руб.}$$

Срок окупаемости затрат (инвестиций) ($T_{пр}$) [15, с. 37] служит для определения степени рисков реализации проекта и ликвидации инвестиций:

$$T_{пр} = \frac{З(И)}{Э(П)}, \quad (5.36)$$

где З(И) – сумма затрат или инвестиций, руб.;

Э(П) – эффект или прибыль от проекта, руб.

Расчёт срока окупаемости затрат (инвестиций) программного обеспечения

$$T_{пр} = \frac{2500}{938,9} = 2,6 \text{ лет.}$$

В таблице Д.9 представлены значения прибыли, рентабельности и срока окупаемости программного обеспечения.

Подводя итоги организационно-экономической части дипломной работы, составляем таблицу Д.10.

По результатам проведенной оценки выявлено, что реализация проекта позволит уменьшить экономические расходы компании на обучение и повышение квалификации сотрудников на 50%. Следовательно, проект является экономически целесообразным. Об этом свидетельствует годовой экономический эффект от использования нового программного обеспечения.

6 ОХРАНА ТРУДА И ТЕХНИКА БЕЗОПАСНОСТИ

6.1 Электромагнитное излучение компьютерной техники

Как и все приборы, потребляющие электроэнергию, компьютер испускает электромагнитное излучение, причем из бытовых приборов с компьютерной техникой по силе этого излучения могут сравниться разве что микроволновая печь или телевизор, однако в непосредственной близости с ним мы не проводим очень много времени, а электромагнитное излучение имеет меньшее воздействие с увеличением расстояния от источника до объекта [16].

Многие люди, постоянно работающие с компьютером, отмечают, что часто через короткое время после начала работы появляются головная боль, болезненные ощущения в области мышц лица и шеи, ноющие боли в позвоночнике, резь в глазах, слезоточивость, нарушение четкого видения, боли при движении рук.

В каждом случае степень риска оказывается в прямой зависимости от времени, которое отводится работе за компьютером ил же нахождением вблизи.

Основными источниками неблагоприятного воздействия компьютера на здоровье пользователя являются мониторы. Однако не стоит недооценивать и излучения, связанные с работой системного блока (в первую очередь – процессора), источников бесперебойного питания и прочих устройств. Все эти элементы формируют сложную электромагнитную обстановку на рабочем месте.

6.1.1 Электромагнитное излучение монитора. Наиболее сильным источником электромагнитного излучения является монитор с электронно-лучевой трубкой, особенно его боковые и задние стенки, т. к. они не имеют специального защитного покрытия, которое есть у лицевой части экрана. Однако даже в самых плохих трубках интенсивность рентгеновского, светового и сверхвысокочастотного излучений намного ниже уровней, могущих оказать какое-либо биологическое действие.

Монитор обладает пониженным уровнем электромагнитного излучения. Для любой электронно-лучевой трубки кинескопа – и телевизионной, и компьютерной – характерно рентгеновское излучение, возникающее при торможении электронов. Однако в современных кинескопах применяются настолько эффективные меры по снижению рентгеновского излучения, что оно практически не обнаруживается на естественном радиационном фоне Земли [16].

Кроме того, мониторы создают электростатическое поле. Во время работы экран монитора заряжается до потенциала в десятки тысяч вольт. Сильное электрическое поле небезопасно для человеческого организма. Сверхнизкочастотные электрические переменные поля повышают выброс ионов кальция из костной ткани. При удалении от экрана влияние электростатического поля значительно

убывает, причем применение специальных экранных защитных фильтров позволяет практически свести его к нулю.

При работе монитора электризуется не только его экран, но и воздух в помещении. Он приобретает положительный заряд. Положительные ионы воздуха опасны для человеческого организма. В помещении, где работает монитор, отрицательных ионов почти нет, а положительные почти в избытке. В помещении может быть сколько угодно свежего воздуха, но, если он имеет положительный заряд – это все равно, что его нет.

Стекло монитора практически непрозрачно для рентгеновского излучения, поэтому дозы его малы. Фон, создаваемый излучением естественных радионуклидов и космических лучей, многократно превышает излучение монитора в рентгеновском диапазоне. Тем не менее, доза рентгеновского излучения нормируется. Источником ультрафиолетового излучения является плазменный разряд на внутреннюю поверхность экрана.

При работе, компьютер образует вокруг себя электростатическое поле, которое демонизирует окружающую среду, а при нагревании платы и корпус монитора испускают в воздух вредные вещества. Все это делает воздух сухим, слабо ионизированным, со специфическим запахом в тяжелым для дыхания.

Мельчайшие частички пыли, пролетая в непосредственной близости от поверхности дисплея, заряжаются статическим электричеством и устремляются к лицу оператора. Через дыхательные пути они проникают в легкие. Попадая на кожу, эти частички забивают поры, препятствуют дыханию кожи, вызывают аллергическую реакцию и способствуют развитию рака кожи.

Небольшую угрозу для здоровья человека представляют электромагнитные поля. Электромагнитные поля влияют на электрическое напряжение между клетками тела. Это приводит к необратимым последствиям.

Зрительная деятельность пользователя компьютера сопровождается наблюдением самосветящегося экранного изображения, отличающегося дискретностью, мерцанием, пониженным контрастом и другими неоптимальными с физиологической точки зрения светотехническими особенностями.

При этом работа с экраном происходит зачастую в неоптимальных условиях внешней световой среды.

6.1.2 Излучение системного блока. Человеческий организм наиболее чувствителен к электромагнитному полю, находящемуся на частотах 40–70 ГГц, так как длины волн на этих частотах соизмеримы с размерами клеток и достаточно незначительного уровня электромагнитного поля, чтоб нанести существенный урон здоровью человека.

Отличительной же особенностью современных компьютеров является увеличение рабочих частот центрального процессора и периферийных устройств, а также повышение потребляемой мощности до 400–500 Вт. В результате этого

уровень излучения системного блока на частотах 40–70 ГГц за последние 2–3 года увеличился в тысячи раз и стал намного более серьезной проблемой, чем излучение монитора.

6.1.3 Излучение ноутбука (портативного компьютера) и жидкокристаллических мониторов. В ноутбуках используются экраны на основе жидких кристаллов, которые не генерируют всего букета вредных электромагнитных излучений, присущих обычным мониторам с электронно-лучевой трубкой. Электромагнитное излучение портативных компьютеров типа *Notebook* значительно превышает экологические нормативы.

Ноутбук обычно располагается ближе к пользователю, и, следовательно, источники излучения будут с большей вероятностью воздействовать на области жизненного важных органов человека, тем более что некоторые пользователи ноутбуков и вовсе имеют обыкновение расположить компьютер на коленях.

Электронно-лучевая трубка не единственный источник излучения электромагнитных полей. Генерировать поля может преобразователь напряжения питания (при работе от электросети), схемы управления и формирования информации на дискретных жидкокристаллических экранах и другие элементы аппаратуры [16].

Кроме того, статическая поза во время работы, повторяющиеся движения и нерациональная организация рабочего места могут приводить к развитию общего утомления, возникновению расстройств скелетно-мышечной системы пользователя видеодисплейных терминалов, которые сопровождаются многочисленными офтальмологическими симптомами.

6.2 Методы снижения уровня излучения

Сегодня мы не можем отказаться от вычислительной техники, зато можем не забывать проветривать помещение, проводить как можно больше времени на свежем воздухе и не включать аппаратуру без необходимости.

Владельцем ноутбуков и всех производных от них портативных устройств стоит отказаться от привычки ставить ноутбук на колени.

Чтобы свести к минимуму негативное влияние электромагнитного излучения от монитора, достаточно придерживаться простых правил:

– выбирая монитор, лучше отдать предпочтение жидкокристаллическому варианту. Излучение мониторов с электронно-лучевой трубкой намного сильнее, чем у жидкокристаллических аналогов;

– расположить монитор в углу. Стены будут поглощать электромагнитное излучение, которое испускают боковые и задние стенки;

– выключать монитор при необходимости отойти от рабочего стола;

– использование специальных защитных экранов по-прежнему актуально, особенно если в семье есть дети;

– монитор должен стоять от кресла не ближе, чем на расстоянии вытянутой руки.

Существует также ряд универсальных правил для работы за компьютером, которые помогут сохранить собственное здоровье и продлить жизнь техники:

– системный блок должен располагаться как можно дальше от пользователя. Нельзя ставить компьютер рядом со спальным местом, а лучше вообще не располагать компьютерный стол в спальне;

– нельзя оставлять компьютер включенным, если он не используется, также не включать его без необходимости. Помимо возможного вреда, наносимого вашему здоровью, стоит помнить об износе аппаратной части компьютерной техники;

– не стоит, как это довольно часто делается, отключать использование «спящего режима» для монитора;

– сократить время, которое проводится за компьютером. Если профессиональная деятельность проходит перед экраном монитора, как можно чаще необходимо прерывать работу, чтобы немного пройтись или просто выпить чаю. В свободное время стараться не сидеть перед монитором.

Нахождение в поле положительно заряженных ионов (а именно такое поле и представляет собой зона перед экраном дисплея на электронно-лучевой трубке), несомненно, связано с вредом для здоровья человека. Положительное электростатическое поле от высокого напряжения на трубке как бы «высасывает» из пространства между пользователем компьютера и экраном компьютерного монитора все отрицательные ионы, что приводит к насыщению данной области положительными ионами воздуха, пыли и дыма. Выходом из такой ситуации может послужить установка качественного заземленного защитного фильтра на экран электронно-лучевой трубки, который позволит снять указанное поле.

Хороший и качественный фильтр, установленный на экран электронно-лучевой трубки, приводит к значительному снижению интенсивности переменного электрического поля [17].

6.2.1 Организационно-технические методы. Комплексы данной части оказывают благотворное влияние и способствуют восстановлению нормальной работоспособности глаз и мышц тела, а также помогут снять симптомы синдрома компьютерного стресса.

6.2.2 К организационным методам по защите от действия электромагнитных полей относятся: выбор режимов работы излучающего оборудования, обеспечивающих уровень излучения, не превышающий предельно допустимый; обозначение и ограничение зон с повышенным уровнем излучения.

Защита временем применяется, когда нет возможности снизить интенсивность излучения в данной точке до предельно допустимого уровня. Путем обозначения, оповещения и т.п. ограничивается время нахождения людей в зоне выраженного воздействия электромагнитного поля.

Защита расстоянием применяется, если невозможно ослабить воздействие другими мерами, в том числе и защитой временем. Метод основан на падении интенсивности излучения, пропорциональному квадрату расстояния до источника.

6.2.3 Технические защитные мероприятия строятся на использовании явления экранирования электромагнитных полей, либо на ограничении эмиссионных параметров источника поля (снижении интенсивности излучения). Второй метод применяется в основном на этапе проектирования излучающего объекта. Для защиты от электромагнитного воздействия населения чаще всего применяется стекло, металлизированное напылением. Для защиты населения от воздействия электромагнитных излучений могут применяться специальные строительные конструкции: металлическая сетка, металлический лист или любое другое проводящее покрытие, а также специально разработанные строительные материалы. В ряде случаев достаточно использования заземленной металлической сетки, помещаемой под облицовку стен помещения [17].

6.2.4 Организация рабочего места и нормирование. Причиной отклонений здоровья пользователей являются не столько сами компьютеры, сколько недостаточно строгое соблюдение принципов эргономики. Для этого необходимо, чтобы рабочее место отвечало бы гигиеническим требованиям безопасности.

Гигиенические требования к ПЭВМ (персональная электронная вычислительная машина) и организация работы (СанПиН (санитарные правила и нормы) 9-131 РБ, 2000):

– площадь на одно рабочее место пользователей ПЭВМ с ВДТ (видеодисплейный терминал) на базе плоских дискретных экранов (жидкокристаллические, плазменные) должна быть не менее 4,5 кв.м.;

– площадь на одного работающего с копировально-множительной техникой должна быть не менее 6 кв.м.;

– расстояние до монитора должно быть достаточно большим (от 50 до 80 сантиметров). По высоте монитор необходимо располагать так, чтобы центр экрана был чуть ниже уровня глаз;

– монитор должен находить прямо впереди посредине стола. Абсолютно неприемлемо расположение монитора на углу стола, когда пользователь сидит к нему вполоборота. Экран монитора должен быть абсолютно чистым;

– материал кресла и одежды, как ни странно, тоже имеет значение при работе за компьютером: необходимо избегать синтетических тканей, которые накапливают статическое электричество;

– стол должен быть как можно большим. Большой стол удобен и позволяет располагать без напряжения документы, периферическое оборудование, компакт-диски;

– должен быть правильно подобран рабочий стул и кресло. Это необходимо для правильной осанки.

Другая, не менее серьезная проблема – обеспечение электромагнитной безопасности работающих за компьютером с дополнительными периферийными устройствами. При одновременном их включении вокруг пользователя создается поле с широким частотным спектром. В этом случае немаловажную роль играет оборудование рабочего места в помещении. Однако на практике обеспечить нормальную электромагнитную обстановку удается далеко не всегда. Предлагается принять во внимание, следующее:

– помещение, где эксплуатируются компьютеры и периферия к ним, должно быть удалено от посторонних источников электромагнитных излучений (электрощиты, трансформаторы и т.д.);

– если на окнах помещения имеются металлические решетки, то они должны быть заземлены, т.к. несоблюдение этого правила может привести к резкому локальному повышению уровня полей в какой-либо точке помещения и сбоям в работе компьютера;

– групповые рабочие места желательно размещать на нижних этажах здания, так как вследствие минимального значения сопротивления заземления именно на нижних этажах здания существенно снижается общий электромагнитный фон.

Для поддержания нормальной температуры и относительной влажности в помещении необходимо регулярное проветривание, а также наличие систем ионизирования и кондиционирования воздуха. Для улучшения микроклимата так же важна грамотная организация освещения. Рекомендуется применять люминесцентные лампы.

Особое внимание следует уделять организации групповых рабочих мест, так как в этом случае пользователь подвержен излучению не только своего компьютера, но и тех, которые расположены рядом с ним. Каждое рабочее место создает своеобразное магнитное поле, радиус которого может быть 1,5 м и более, причем излучение исходит не только от экрана, но и от задней и боковых стенок монитора. Специалисты советуют размещать рабочие места с компьютерами так, чтобы расстояние между боковыми стенками дисплея соседних мониторов было не менее 1,2 м, а расстояние между передней поверхностью монитора в направлении тыла соседнего монитора – не менее 2-х метров [17].

7 РЕСУРСО- И ЭНЕРГОСБЕРЕЖЕНИЕ ПРИ ВНЕДРЕНИИ ПРОГРАММНО-АППАРАТНОГО КОМПЛЕКСА

7.1 Вопросы ресурсосбережения, связанные с внедрением программного обеспечения

Ресурсосбережение – совокупность мер по бережливому и эффективному использованию фактов производства (капитала, земли, труда).

Ресурсы – ценности, запасы, возможности, источники дохода в государственном бюджете. В общем виде ресурсы делятся на природные и экономические (материальные, трудовые, финансовые) [18].

Ресурсосбережение обеспечивается посредством:

- ресурсосберегающих и энергосберегающих технологий;
- снижения фондоёмкости и материалоемкости продукции;
- повышения производительности труда;
- сокращения затрат живого и овеществленного труда;
- повышения качества продукции;
- рационального применения труда менеджеров и маркетологов;
- использования выгод международного разделения труда.

Ресурсосбережение и энергосбережение способствует росту эффективности экономики, повышению её конкурентоспособности.

ГОСТ 30166-95 «Ресурсосбережение. Основные положения» является действующим стандарт. Является основополагающим и устанавливает цель, задачи, объекты, основные принципы, термины и классификацию групп требований рационального использования и экономного расходования материальных ресурсов на всех стадиях жизненного цикла веществ, материалов, изделий, продукции при проведении работ и оказании услуг юридическим и физическим лицам [19]. Настоящий стандарт распространяется на все виды деятельности, связанные с добычей, переработкой, транспортированием, хранением, распределением и потреблением материальных ресурсов.

Научно-технический прогресс – это непрерывный процесс открытия новых знаний и применения их в общественном производстве, позволяющий по-новому соединять и комбинировать имеющиеся ресурсы в интересах увеличения выпуска высококачественных конечных продуктов при наименьших затратах. В широком смысле на любом уровне – от фирмы до национальной экономики – под научно-техническим прогрессом подразумевается создание и внедрение новой техники, технологии, материалов, использование новых видов энергии, а также появление ранее неизвестных методов организации и управления производством.

Внедрение программно-аппаратного комплекса – способ оптимизировать скорость выполнения рутинных процессов, облегчить работу людей и снизить трудозатраты, долю труда в стоимости единицы продукции. Конкурентоспособность фирмы или предприятия, их способность удержаться на рынке товаров и услуг зависит, в первую очередь, от восприимчивости производителей товаров к новинкам техники и технологии, позволяющим обеспечить выпуск и реализацию высококачественных товаров при наиболее эффективном использовании материальных ресурсов.

Основной задачей ресурсосбережения, как науки, является экономия материальных ресурсов. Экономия материальных ресурсов – это экономическая категория, которая характеризуется снижением удельного расхода материальных ресурсов на единицу продукции по сравнению с базисным или текущим периодом, но без снижения качества и технического уровня продукции.

7.2 Экономия энергоресурсов в результате внедрения программного обеспечения

Энергосбережение является важнейшей отраслью каждой страны, решающей задачу по сохранению природных ресурсов. Также и в Республике Беларусь.

Энергосбережение (экономия электроэнергии) – комплекс мероприятий, направленных на сохранение и рациональное использование электричества и тепла. Реализация правовых, организационных, научных, производственных, технических и экономических мер, направленных на рациональное использование (и экономное расходование) энергетических ресурсов и на вовлечение в хозяйственный оборот возобновляемых источников энергии.

Энергосбережение – одна из важнейших задач по сохранению природных ресурсов [19]. Исходя из того, что внедряемый программно-аппаратный комплекс предназначено для работы на персональном компьютере и существующий расчет вручную также выполняется с использованием компьютера, то можно рассчитать стоимость сэкономленной электроэнергии, потребляемой персональным компьютером.

Экономия электрической энергии при работе с программным средством рассчитывается по следующей формуле:

$$Э = (Т_р - Т_а) \cdot P \cdot С_{эл} \cdot K_и, \quad (7.1)$$

где $Т_р$ – трудоемкость работы вручную, ч;

$Т_а$ – трудоемкость работы с помощью программно-аппаратного комплекса, ч;

P – паспортная мощность персонального компьютера, кВт;

$С_{эл}$ – стоимость одного кВт·ч электроэнергии, руб;

$K_{и}$ – коэффициент использования ($K_{и} = 0,5$).

Для проведения всего цикла работы от реализации всех входных данных до расчетов тратится около двух человеко-часов рабочего времени с использованием одного персонального компьютера, для той же операции с применением разработанного программного продукта требуется порядка одного человека-часа на том же компьютере. По состоянию на май 2020 года стоимость одного кВт·ч. электроэнергии в Республике Беларусь равна 0,33048 руб. Приняв паспортную мощность персонального компьютера в 0,5 кВт, расчёт стоимости сэкономленной электроэнергии за один месяц при пятидневной рабочей неделе и восьмичасовом рабочем дне:

$$Э = (5-2) \cdot 0,5 \cdot 0,33048 \cdot 0,5 = 0,25 \text{ руб.}$$

Таким образом, рассчитывается экономия электроэнергии за нужный период времени по формуле:

$$Э_{пер} = Э \cdot n, \tag{7.2}$$

где $Э$ – сэкономленная электроэнергия за рабочий день, руб;

n – период использования ПО.

Расчёт экономии электроэнергии за 2020 год при 5-дневной рабочей неделе:

$$Э_{пер} = 0,25 \cdot 256 \cdot 8 = 512 \text{ руб.}$$

Данный расчет показывает, что при внедрении разработанного программного обеспечения, можно сэкономить на электроэнергии 512 руб. за год.

ЗАКЛЮЧЕНИЕ

В результате проделанной работы было создано приложение для обучения персонала с использованием инструментов *VR*. Для этого проведен анализ существующих сред и технологий разработки игр, предназначенных для создания приложений виртуальной реальности, рассмотрены типы существующих тренажерных систем [13, 14].

Разработаны алгоритмы для работы приложения, спроектирован и создан интерфейс программы, разработано приложение виртуальной реальности, обладающее всеми основными функциями, и выполняющее все поставленные задачи по проведению обучения персонала.

Программный комплекс, позволяет выполнять следующие функции:

- проводить обучение персонала по различным сценариям;
- создавать, редактировать и загружать новые сценарии в приложение;
- записывать видео, с текущим прохождением сценария;
- просматривать всю информацию о ранее проведенных тестированиях.

Для проверки корректности работы и удобства использования проведено тестирование, которое подтвердило, что проверяемые функции работают исправно.

Программное обеспечение может быть востребовано в образовательных учрежденьях, предприятиях, занимающихся подготовкой и повышением квалификации персонала в различных сферах труда.

Программное обеспечение было протестировано на нескольких устройствах (различные конфигурации компьютеров на базе ОС *Windows*), что подтвердило стабильную работу приложения. результаты работы функций приложения соответствуют ожидаемым результатам.

В качестве возможного улучшения разработанного программного обеспечения может выступать переработка системы диалогов с добавлением искусственного интеллекта и персонажей.

Список использованных источников

1. Nooriafshar, M. The use of virtual reality in education / M. Nooriafshar // Journal of Virtual Worlds Research. – 2004. – №8. – P. 9–15.
2. Воронцов, Н.С. Виртуальная реальность упростит обучение и программирование роботов / Н.С. Воронцов // N+1. – 2016. – №8. – С. 24–27.
3. Захарова, Г. Компьютерные тренажеры как средство эффективного обучения: классификация и пример разработки / Г.Б. Захарова, Д.Н. Первухин, Д.В. Байгозин // Новые образовательные технологии в вузе: Шестая международная научно-методическая конференция. – 2009. – №2. – С. 124–127.
4. CryEngine Documentation – Электронные данные. – Режим доступа: https://docs.cryengine.com/. – Дата доступа: 03.06.2020.
5. Unreal Engine 4 Documentation. – Электрон. данные. – Режим доступа: https://docs.unrealengine.com/en-US/index.html/. – Дата доступа: 03.06.2020.
6. EpicGames Twinmotion. – Электрон. данные. – Режим доступа: https://www.unrealengine.com/en-US/twinmotion. – Дата доступа: 03.06.2020.
7. Ingeno, J. N-tier architecture - Software Architect's Handbook / J. Ingeno. – Publisher Packt Publishing, 2018. – 594 p.
8. Laguna, M. Business Process Modeling, Simulation and Design / M. Laguna, J. Marklund. – Publisher Prentice Hall, 2004. – 525 p.
9. NVIDIA NVENC. – Электрон. данные. – Режим доступа: https://www.nvidia.com/en-us/geforce/guides/broadcasting-guide. – Дата доступа: 03.06.2020.
10. Маккефри, М. Unreal Engine VR для разработчиков / М. Маккефри. – издательство Эксмо, 2019. – 256 с.
11. Visual C++ Documentation. – Электрон. данные. – Режим доступа: https://docs.microsoft.com/en-us/cpp/?view=vs-2019/. – Дата доступа: 03.06.2020.
12. SteamVR Documentation. – Электрон. данные. – Режим доступа: https://support.steampowered.com/kb_article.php?ref=1131-WSFG-3320&l=en/. – Дата доступа: 03.06.2020.
13. Kurochka, K. Classifier for Confocal Thyroid Imaging on Single Board Computers / K. Kurochka, A. Demidov // Open Semantic Technologies for Intelligent Systems. – 2020. – № 4. – P. 287–290.
14. Курочка, К. Низкоресурсоемкий алгоритм подсчёта монет на изображении / К. Курочка, А. Демидов, К. Панарин // Материалы, оборудование и ресурсосберегающие технологии: материалы Междунар. науч.-техн. конф., Могилев, 23–24 апр. 2020 г. / М-во образования Респ. Беларусь, М-во науки и высшего образования Рос. Федерации, Белорус.-Рос. ун-т; редкол.: гл. ред. М. Е. Лустенков. – Могилев: БРУ, 2020. – С. 340–342.

15. Кожевников, Е.А. Расчет экономической эффективности разработки программных продуктов: метод. указания по подготовке организационно-экономического раздела дипломных работ для студентов специальности 1-40 01 02 «Информационные системы и технологии (по направлениям)» дневной формы обучения / Е.А. Кожевников, Н.В. Ермалинская. – Гомель: ГГТУ им. П. О. Сухого, 2012. – 68 с.

16. Излучение от компьютера: вред, способы защиты. [Электронный ресурс]. – 2014. – Режим доступа: http://otravleniya.net/izluchenie/izluchenie-ot-kompyutera-vred. Дата доступа: 05.06.2020 г.

17. Защита от электромагнитного излучения. [Электронный ресурс]. – 2014. – Режим доступа: http://gamma7.m-l-m.info/zashhita-ot-elektromagnitnogo-izlucheniya. Дата доступа: 05.06.2020 г.

18. Самойлов, М.В. Основы энергосбережения / М.В. Самойлов, В.В. Паневчик, А.Н. Ковалев. – Мн.: БГЭУ, 2002. – 198 с.

19. Ресурсосбережение. Основные положения: ГОСТ 30166-95. – Введ. 01.01.2002. – Минск: Межгос. совет по стандартизации, метрологии и сертификации: Беларус. гос. ин-т стандартизации и сертификации, 2000. – 5 с.

ПРИЛОЖЕНИЕ А
(обязательное)
Листинг программы

```
// Copyright 1998-2019 Epic Games, Inc. All Rights Reserved.
#pragma once
#include "CoreMinimal.h"
#include "HAL/ThreadSafeCounter.h"
#include "UObject/ObjectMacros.h"
#include "UObject/UObjectGlobals.h"
#include "UObject/Object.h"
#include "Misc/Guid.h"
#include "UObject/Class.h"
#include "Engine/EngineTypes.h"
#include "Engine/EngineBaseTypes.h"
#include "CollisionQueryParams.h"
#include "WorldCollision.h"
#include "GameFramework/Pawn.h"
#include "EngineDefines.h"
#include "Engine/Blueprint.h"
#include "Engine/PendingNetGame.h"
#include "Engine/LatentActionManager.h"
#include "Engine/GameInstance.h"
#include "Physics/PhysicsInterfaceDeclares.h"
#include "Particles/WorldPSCPool.h"
#include "Containers/SortedMap.h"
#include "Subsystems/WorldSubsystem.h"
#include "Subsystems/SubsystemCollection.h"
#include "World.generated.h"
class ABrush;
class ACameraActor;
class AController;
class AGameModeBase;
class AGameStateBase;
class AMatineeActor;
class APhysicsVolume;
class APlayerController;
class AWorldSettings;
class Error;
class FTimerManager;
class FUniqueNetId;
class FWorldInGamePerformanceTrackers;
class IInterface_PostProcessVolume;
class UAISystemBase;
class UCanvas;
class UDemoNetDriver;
class UGameViewportClient;
class ULevel;
class ULevelStreaming;
class ULocalPlayer;
class UMaterialParameterCollection;
class UMaterialParameterCollectionInstance;
class UModel;
class UNavigationSystemBase;
class UNetConnection;
class UNetDriver;
class UPrimitiveComponent;
class UTexture2D;
class FPhysScene_Chaos;
class FSceneView;
struct FUniqueNetIdRepl;
```

```
struct FEncryptionKeyResponse;
template<typename,typename> class TOctree;
template <typename ActorType> class TActorIterator;
typedef TArray<TWeakObjectPtr<AController> >::TConstIterator FConstControllerIterator;
typedef TArray<TWeakObjectPtr<APlayerController> >::TConstIterator FConstPlayerControllerIterator;
typedef TArray<TWeakObjectPtr<ACameraActor> >::TConstIterator FConstCameraActorIterator;
typedef TArray<ULevel*>::TConstIterator FConstLevelIterator;
typedef TArray<TWeakObjectPtr<APhysicsVolume> >::TConstIterator FConstPhysicsVolumeIterator;
struct ENGINE_API FPawnIteratorObject
{
	APawn* operator->() const { return Pawn; }
	APawn& operator*() const { return *Pawn; }
	APawn* Get() const { return Pawn; }
	bool operator==(const UObject* Other) const { return Pawn == Other; }
	bool operator!=(const UObject* Other) const { return Pawn != Other; }
private:
	FPawnIteratorObject()
		: Pawn(nullptr)
	{
	}
	FPawnIteratorObject(APawn* InPawn)
		: Pawn(InPawn)
	{
	}
	APawn* Pawn;
	friend class FConstPawnIterator;
};
template< class T > FORCEINLINE T* Cast(const FPawnIteratorObject& Src) { return Cast<T>(Src.Get()); }
class ENGINE_API FConstPawnIterator
{
private:
	FConstPawnIterator(UWorld* World);
public:
	~FConstPawnIterator();
	FConstPawnIterator(FConstPawnIterator&&);
	FConstPawnIterator& operator=(FConstPawnIterator&&);
	explicit operator bool() const;
	FPawnIteratorObject operator*() const;
	TUniquePtr<FPawnIteratorObject> operator->() const;
	FConstPawnIterator& operator++();
	FConstPawnIterator& operator++(int);
	UE_DEPRECATED(4.23, "Decrement operator no longer means anything on a pawn iterator")
	FConstPawnIterator& operator--() { return *this; }
	UE_DEPRECATED(4.23, "Decrement operator no longer means anything on a pawn iterator")
	FConstPawnIterator& operator--(int) { return *this; }
private:
	TUniquePtr<TActorIterator<APawn>> Iterator;
	friend UWorld;
}
DECLARE_LOG_CATEGORY_EXTERN(LogSpawn, Warning, All);
DECLARE_MULTICAST_DELEGATE_OneParam(FOnActorSpawned, AActor*);
DECLARE_MULTICAST_DELEGATE_OneParam(FOnFeatureLevelChanged, ERHIFeatureLevel::Type);
class UWorldProxy
{
public:
	UWorldProxy() :
		World(NULL)
	{}
	inline UWorld* operator->()
	{
		checkSlow(IsInGameThread());
		return World;
	}
```

```
	inline const UWorld* operator->() const
	{
		checkSlow(IsInGameThread());
		return World;
	}
	inline UWorld& operator*()
	{
		checkSlow(IsInGameThread());
		return *World;
	}
	inline const UWorld& operator*() const
	{
		checkSlow(IsInGameThread());
		return *World;
	}
	inline UWorldProxy& operator=(UWorld* InWorld)
	{
		World = InWorld;
		return *this;
	}
	inline UWorldProxy& operator=(const UWorldProxy& InProxy)
	{
		World = InProxy.World;
		return *this;
	}
	inline bool operator==(const UWorldProxy& Other) const
	{
		return World == Other.World;
	}
	inline operator UWorld*() const
	{
		checkSlow(IsInGameThread());
		return World;
	}
	inline UWorld* GetReference()
	{
		checkSlow(IsInGameThread());
		return World;
	}
private:
	UWorld* World;
};

/** class that encapsulates seamless world traveling */
class FSeamlessTravelHandler
{
private:
	FURL PendingTravelURL;
	FGuid PendingTravelGuid;
	UObject* LoadedPackage;
	UWorld* CurrentWorld;
	UWorld* LoadedWorld;
	bool bTransitionInProgress;
	bool bSwitchedToDefaultMap;
	bool bPauseAtMidpoint;
	bool bNeedCancelCleanUp;
	FName WorldContextHandle;
	double SeamlessTravelStartTime = 0.f;
	void CopyWorldData();
	void SeamlessTravelLoadCallback(const FName& PackageName, UPackage* LevelPackage, EAsyncLoadingResult::Type
Result);
	void SetHandlerLoadedData(UObject* InLevelPackage, UWorld* InLoadedWorld);
	void StartLoadingDestination();
```

```
public:
	FSeamlessTravelHandler()
		: PendingTravelURL(NoInit)
		, PendingTravelGuid(0, 0, 0, 0)
		, LoadedPackage(NULL)
		, CurrentWorld(NULL)
		, LoadedWorld(NULL)
		, bTransitionInProgress(false)
		, bSwitchedToDefaultMap(false)
		, bPauseAtMidpoint(false)
		, bNeedCancelCleanUp(false)
	{}
	bool StartTravel(UWorld* InCurrentWorld, const FURL& InURL, const FGuid& InGuid);
	FORCEINLINE bool IsInTransition() const
	{
		return bTransitionInProgress;
	}
	FORCEINLINE bool HasSwitchedToDefaultMap() const
	{
		return IsInTransition() && bSwitchedToDefaultMap;
	}
	inline FString GetDestinationMapName() const
	{
		return (IsInTransition() ? PendingTravelURL.Map : TEXT(""));
	inline const UWorld* GetLoadedWorld() const
	{
		return LoadedWorld;
	}
	void CancelTravel();
	void SetPauseAtMidpoint(bool bNowPaused);
	ENGINE_API UWorld* Tick();
};
struct ENGINE_API FLevelStreamingGCHelper
{
	/** Called when streamed out levels are going to be garbage collected  */
	DECLARE_MULTICAST_DELEGATE(FOnGCStreamedOutLevelsEvent);
	static FOnGCStreamedOutLevelsEvent OnGCStreamedOutLevels;

	/**
	 * Register with the garbage collector to receive callbacks pre and post garbage collection
	 */
	static void AddGarbageCollectorCallback();

	/**
	 * Request to be unloaded.
	 *
	 * @param InLevel Level that should be unloaded
	 */
	static void RequestUnload( ULevel* InLevel );

	/**
	 * Cancel any pending unload requests for passed in Level.
	 */
	static void CancelUnloadRequest( ULevel* InLevel );

	/**
	 * Prepares levels that are marked for unload for the GC call by marking their actors and components as
	 * pending kill.
	 */
	static void PrepareStreamedOutLevelsForGC();

	/**
```

```
		 * Verify that the level packages are no longer around.
		 */
		static void VerifyLevelsGotRemovedByGC();

		/**
		 * @return		The number of levels pending a purge by the garbage collector
		 */
		static int32 GetNumLevelsPendingPurge();

private:
		/** Static array of levels that should be unloaded */
		static TArray<TWeakObjectPtr<ULevel> > LevelsPendingUnload;
		/** Static array of level packages that have been marked by PrepareStreamedOutLevelsForGC */
		static TArray<FName> LevelPackageNames;
};
USTRUCT()
struct ENGINE_API FLevelViewportInfo
{
		GENERATED_BODY()

		/** Where the camera is positioned within the viewport. */
		UPROPERTY()
		FVector CamPosition;

		/** The camera's position within the viewport. */
		UPROPERTY()
		FRotator CamRotation;
		/** The zoom value  for orthographic mode. */
		UPROPERTY()
		float CamOrthoZoom;

		/** Whether camera settings have been systematically changed since the last level viewport update. */
		UPROPERTY()
		bool CamUpdated;

		FLevelViewportInfo()
			: CamPosition(FVector::ZeroVector)
			, CamRotation(FRotator::ZeroRotator)
			, CamOrthoZoom(DEFAULT_ORTHOZOOM)
			, CamUpdated(false)
		{
		}

		FLevelViewportInfo(const FVector& InCamPosition, const FRotator& InCamRotation, float InCamOrthoZoom)
			: CamPosition(InCamPosition)
			, CamRotation(InCamRotation)
			, CamOrthoZoom(InCamOrthoZoom)
			, CamUpdated(false)
		{
		}

		friend FArchive& operator<<( FArchive& Ar, FLevelViewportInfo& I )
		{
			Ar << I.CamPosition;
			Ar << I.CamRotation;
			Ar << I.CamOrthoZoom;

			if ( Ar.IsLoading() )
			{
				I.CamUpdated = true;

				if ( I.CamOrthoZoom == 0.f )
				{
```

```
                                        I.CamOrthoZoom = DEFAULT_ORTHOZOOM;
                                }
                        }

                        return Ar;
                }
};
USTRUCT()
struct FStartPhysicsTickFunction : public FTickFunction
{
        GENERATED_USTRUCT_BODY()

        /** World this tick function belongs to **/
        class UWorld*       Target;


        virtual void ExecuteTick(float DeltaTime, enum ELevelTick TickType, ENamedThreads::Type CurrentThread, const
FGraphEventRef& MyCompletionGraphEvent) override;
        /** Abstract function to describe this tick. Used to print messages about illegal cycles in the dependency graph **/
        virtual FString DiagnosticMessage() override;
        /** Function used to describe this tick for active tick reporting. **/
        virtual FName DiagnosticContext(bool bDetailed) override;
};

template<>
struct TStructOpsTypeTraits<FStartPhysicsTickFunction> : public TStructOpsTypeTraitsBase2<FStartPhysicsTickFunction>
{
        enum
        {
                WithCopy = false
        };
};

/**
* Tick function that ends the physics tick
**/
USTRUCT()
struct FEndPhysicsTickFunction : public FTickFunction
{
        GENERATED_USTRUCT_BODY()

        /** World this tick function belongs to **/
        class UWorld*       Target;


        virtual void ExecuteTick(float DeltaTime, enum ELevelTick TickType, ENamedThreads::Type CurrentThread, const
FGraphEventRef& MyCompletionGraphEvent) override;

        virtual FString DiagnosticMessage() override;
        virtual FName DiagnosticContext(bool bDetailed) override;};

template<>
struct TStructOpsTypeTraits<FEndPhysicsTickFunction> : public TStructOpsTypeTraitsBase2<FEndPhysicsTickFunction>
{
        enum
        {
                WithCopy = false
        };
};
/* Struct of optional parameters passed to SpawnActor function(s). */
struct ENGINE_API FActorSpawnParameters
{
        FActorSpawnParameters();
```

```
		FName Name;
		AActor* Template;
		AActor* Owner;
APawn*	Instigator;

		/* The ULevel to spawn the Actor in, i.e. the Outer of the Actor. If left as NULL the Outer of the Owner is used. If the Owner is NULL the persistent level is used. */
		class	ULevel* OverrideLevel;

		/** Method for resolving collisions at the spawn point. Undefined means no override, use the actor's setting. */
		ESpawnActorCollisionHandlingMethod SpawnCollisionHandlingOverride;

private:

		friend class UPackageMapClient;

		/* Is the actor remotely owned. This should only be set true by the package map when it is creating an actor on a client that was replicated from the server. */
		uint8	bRemoteOwned:1;

public:

		bool IsRemoteOwned() const { return bRemoteOwned; }

		/* Determines whether spawning will not fail if certain conditions are not met. If true, spawning will not fail because the class being spawned is `bStatic=true` or because the class of the template Actor is not the same as the class of the Actor being spawned. */
		uint8	bNoFail:1;
struct ENGINE_API FWorldAsyncTraceState
{		FWorldAsncTraceState();
		/** Get the Buffer for input Frame */
		FORCEINLINE AsyncTraceData& GetBufferForFrame	(int32 Frame) { return DataBuffer[ Frame		% 2]; }
		/** Get the Buffer for Current Frame */
		FORCEINLINE AsyncTraceData& GetBufferForCurrentFrame ()		{ return DataBuffer[ CurrentFrame	% 2]; }
		/** Get the Buffer for Previous Frame */
		FORCEINLINE AsyncTraceData& GetBufferForPreviousFrame()		{ return DataBuffer[(CurrentFrame + 1) % 2]; }
		/** Async Trace Data Buffer Array. For now we only saves 2 frames. **/
		AsyncTraceData DataBuffer[2];
		/** Used as counter for Buffer swap for DataBuffer. Right now it's only 2, but it can change. */
		int32 CurrentFrame;};
#endif
USTRUCT()
struct ENGINE_API FLevelCollection
{
		GENERATED_BODY(
};

template<>
struct TStructOpsTypeTraits<FLevelCollection> : public TStructOpsTypeTraitsBase2<FLevelCollection>
{
		enum
		{
			WithCopy = false
		};
};
class ENGINE_API FScopedLevelCollectionContextSwitch
{
public:
		~FScopedLevelCollectionContextSwitch();

private:
		class UWorld* World;
		int32 SavedTickingCollectionIndex;
```

```
};

USTRUCT()
struct FLevelStreamingWrapper
{
	GENERATED_BODY(
	FLevelStreamingWrapper()
		: StreamingLevel(nullptr)
	{}
	FLevelStreamingWrapper(ULevelStreaming* InStreamingLevel)
		: StreamingLevel(InStreamingLevel)
	{}
	ULevelStreaming*& Get() { return StreamingLevel; }
	ULevelStreaming* Get() const { return StreamingLevel; }
	bool operator<(const FLevelStreamingWrapper& Other) const;
	bool operator==(const FLevelStreamingWrapper& Other) const { return StreamingLevel == Other.StreamingLevel; }
private:
	UPROPERTY()
	ULevelStreaming* StreamingLevel;
};

USTRUCT()
struct FStreamingLevelsToConsider
{
	GENERATED_BODY()
	FStreamingLevelsToConsider()
		: bStreamingLevelsBeingConsidered(false)
	{}
	UPROPERTY()
	TArray<FLevelStreamingWrapper> StreamingLevels;

private:
	enum class EProcessReason
	{
		Add,
		Reevaluate
	};
	TSortedMap<FLevelStreamingWrapper, EProcessReason> LevelsToProcess
	bool bStreamingLevelsBeingConsidered;
	void Add_Internal(ULevelStreaming* StreamingLevel, bool bGuaranteedNotInContainer);

public:
	void AddReferencedObjects(UObject* InThis, FReferenceCollector& Collector);
	void BeginConsideration();
	void EndConsideration();
	void Add(ULevelStreaming* StreamingLevel) { Add_Internal(StreamingLevel, false); }
	bool Remove(ULevelStreaming* StreamingLevel);
	bool Contains(ULevelStreaming* StreamingLevel) const;
	void Reset();
	void Reevaluate(ULevelStreaming* StreamingLevel);
};


UCLASS(customConstructor, config=Engine)
class ENGINE_API UWorld final : public UObject, public FNetworkNotify
{
	GENERATED_UCLASS_BODY()

	~UWorld();

#if WITH_EDITORONLY_DATA
	/** List of all the layers referenced by the world's actors */
	UPROPERTY()
```

```
	TArray< class ULayer* > Layers;

	// Group actors currently "active"
	UPROPERTY(transient)
	TArray<AActor*> ActiveGroupActors;

	/** Information for thumbnail rendering */
	UPROPERTY(VisibleAnywhere, Instanced, Category=Thumbnail)
	class UThumbnailInfo* ThumbnailInfo;
#endif // WITH_EDITORONLY_DATA
	class AGameNetworkManager*                          NetworkManager;   UPROPERTY(Transient)
	class UPhysicsCollisionHandler*                     PhysicsCollisionHandler;
	UPROPERTY(Transient)
	TArray<UObject*>                                    ExtraReferencedObjects;
	UPROPERTY(Transient)
	TArray<UObject*>                                    PerModuleDataObjects;
	UPROPERTY(transient)
	TArray<AActor*>                                     LevelSequenceActors;

private:
	UPROPERTY(Transient)
	TArray<ULevelStreaming*> StreamingLevels;
	UPROPERTY(Transient, DuplicateTransient)
	FStreamingLevelsToConsider StreamingLevelsToConsider;

public:
	const TArray<ULevelStreaming*>& GetStreamingLevels() const { return StreamingLevels; }

	bool IsStreamingLevelBeingConsidered(ULevelStreaming* StreamingLevel) const { return StreamingLevelsToConsider.Contains(StreamingLevel); }
	ULevel* GetCurrentLevelPendingVisibility() const { return CurrentLevelPendingVisibility; }
	ULevel* GetCurrentLevelPendingInvisibility() const { return CurrentLevelPendingInvisibility; }
	void AddStreamingLevel(ULevelStreaming* StreamingLevelToAdd);
	void AddStreamingLevels(TArrayView<ULevelStreaming* const> StreamingLevelsToAdd);
	void AddUniqueStreamingLevel(ULevelStreaming* StreamingLevelToAdd);
	void AddUniqueStreamingLevels(TArrayView<ULevelStreaming* const> StreamingLevelsToAdd);
	void SetStreamingLevels(TArray<ULevelStreaming*>&& StreamingLevels);
	bool RemoveStreamingLevel(ULevelStreaming* StreamingLevelToRemove);
	bool RemoveStreamingLevelAt(int32 IndexToRemove);
	int32 RemoveStreamingLevels(TArrayView<ULevelStreaming* const> StreamingLevelsToRemove);
	void ClearStreamingLevels();
	void UpdateStreamingLevelShouldBeConsidered(ULevelStreaming* StreamingLevelToConsider);
	void UpdateStreamingLevelPriority(ULevelStreaming* StreamingLevel);
	/** Examine all streaming levels and determine which ones should be considered. */
	void PopulateStreamingLevelsToConsider();
	UPROPERTY(Transient)
	UCanvas* CanvasForRenderingToTarget;
	UPROPERTY(Transient)
	UCanvas* CanvasForDrawMaterialToRenderTarget;
public:
	/** Set the pointer to the Navigation System instance. */
	void SetNavigationSystem(UNavigationSystemBase* InNavigationSystem);
	/** The interface to the scene manager for this world. */
	class FSceneInterface*                              Scene;
	class ULevel* GetCurrentLevel() const;
	static TMap<FName, EWorldType::Type> WorldTypePreLoadMap;
	typedef TMap<TWeakObjectPtr<class UBlueprint>, TWeakObjectPtr<UObject> > FBlueprintToDebuggedObjectMap;

	/** Return the array of objects currently bieng debugged. */
	const FBlueprintToDebuggedObjectMap& GetBlueprintObjectsBeingDebugged() const{ return BlueprintObjectsBeingDebugged; };
#endif
```

```
		/** Creates a new FX system for this world */
		void CreateFXSystem();
		/** Initialize all world subsystems */
		void InitializeSubsystems();
		/** Change the feature level that this world is current rendering with */
		void ChangeFeatureLevel(ERHIFeatureLevel::Type InFeatureLevel, bool bShowSlowProgressDialog = true);
		void RecreateScene(ERHIFeatureLevel::Type InFeatureLevel);
#endif // WITH_EDITOR
		DECLARE_EVENT( UWorld, FOnSelectedLevelsChangedEvent);

		/** Broadcasts whenever selected level list changes. */
		FOnSelectedLevelsChangedEvent SelectedLevelsChangedEvent;
		UPROPERTY(Transient)
		TArray<class ULevel*> SelectedLevels;
		uint32 bBroadcastSelectionChange:1;
		FOnFeatureLevelChanged OnFeatureLevelChanged;

#endif //WITH_EDITORONLY_DATA
public:

		bool GetShouldForceUnloadStreamingLevels() const { return bShouldForceUnloadStreamingLevels; }
		void SetShouldForceUnloadStreamingLevels(bool bInShouldForceUnloadStreamingLevels);
		bool GetShouldForceVisibleStreamingLevels() const { return bShouldForceVisibleStreamingLevels; }
		void SetShouldForceVisibleStreamingLevels(bool bInShouldForceVisibleStreamingLevels);
#if !(UE_BUILD_SHIPPING || UE_BUILD_TEST)
		/** When non-'None', all line traces where the TraceTag match this will be drawn */
		FName	DebugDrawTraceTag;
		/** When set to true, all scene queries will be drawn */
		bool bDebugDrawAllTraceTags;
		bool DebugDrawSceneQueries(const FName& UsedTraceTag) const
		{
			return (bDebugDrawAllTraceTags || ((DebugDrawTraceTag != NAME_None) && (DebugDrawTraceTag ==
UsedTraceTag))) && IsInGameThread();
		}
#endif
		/** Called when the world computes how post process volumes contribute to the scene. */
		DECLARE_EVENT_OneParam(UWorld, FOnBeginPostProcessSettings, FVector);
		FOnBeginPostProcessSettings OnBeginPostProcessSettings;
		/** Name of persistent level if we've loaded levels via CommitMapChange() that aren't normally in the StreamingLevels
array (to inform newly joining clients) */		FName CommittedPersistentLevelName;
#if !UE_BUILD_SHIPPING
		/**
		 * This is a int on the level which is set when a light that needs to have lighting rebuilt
		 * is moved.  This is then checked in CheckMap for errors to let you know that this level should
		 * have lighting rebuilt.
		 **/
		uint32 NumLightingUnbuiltObjects;

		uint32 NumUnbuiltReflectionCaptures;
		/** Num of components missing valid texture streaming data. Updated in map check. */
		int32 NumTextureStreamingUnbuiltComponents;
		/** Num of resources that have changed since the last texture streaming build. Updated in map check. */
		int32 NumTextureStreamingDirtyResources;
#endif
		/** Indicates that the world has marked contained objects as pending kill */
		bool HasMarkedObjectsPendingKill() const { return bMarkedObjectsPendingKill; }
private:
		uint32 bMarkedObjectsPendingKill:1;
		uint32 CleanupWorldTag;
		static uint32 CleanupWorldGlobalTag;
public:
#if WITH_EDITORONLY_DATA
		/** List of DDC async requests we need to wait on before we register components. Game thread only. */
```

```
	TArray<TSharedPtr<FAsyncPreRegisterDDCRequest>> AsyncPreRegisterDDCRequests;
#endif
	//Experimental: In game performance tracking.
	FWorldInGamePerformanceTrackers* PerfTrackers;
	/**
	 * UWorld default constructor
	 */
	UWorld(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());
	bool LineTraceTestByChannel(const FVector& Start,const FVector& End,ECollisionChannel TraceChannel, const FCollisionQueryParams& Params = FCollisionQueryParams::DefaultQueryParam, const FCollisionResponseParams& ResponseParam = FCollisionResponseParams::DefaultResponseParam) const;
	bool LineTraceTestByObjectType(const FVector& Start,const FVector& End,const FCollisionObjectQueryParams& ObjectQueryParams, const FCollisionQueryParams& Params = FCollisionQueryParams::DefaultQueryParam) const;
	bool LineTraceTestByProfile(const FVector& Start, const FVector& End, FName ProfileName, const FCollisionQueryParams& Params = FCollisionQueryParams::DefaultQueryParam) const;
	bool LineTraceSingleByChannel(struct FHitResult& OutHit,const FVector& Start,const FVector& End,ECollisionChannel TraceChannel,const FCollisionQueryParams& Params = FCollisionQueryParams::DefaultQueryParam, const FCollisionResponseParams& ResponseParam = FCollisionResponseParams::DefaultResponseParam) const;
	bool LineTraceSingleByObjectType(struct FHitResult& OutHit,const FVector& Start,const FVector& End,const FCollisionObjectQueryParams& ObjectQueryParams, const FCollisionQueryParams& Params = FCollisionQueryParams::DefaultQueryParam) const;
	bool LineTraceSingleByProfile(struct FHitResult& OutHit, const FVector& Start, const FVector& End, FName ProfileName, const FCollisionQueryParams& Params = FCollisionQueryParams::DefaultQueryParam) const;
	bool LineTraceMultiByChannel(TArray<struct FHitResult>& OutHits,const FVector& Start,const FVector& End,ECollisionChannel TraceChannel,const FCollisionQueryParams& Params = FCollisionQueryParams::DefaultQueryParam, const FCollisionResponseParams& ResponseParam = FCollisionResponseParams::DefaultResponseParam) const;
	bool LineTraceMultiByObjectType(TArray<struct FHitResult>& OutHits,const FVector& Start,const FVector& End,const FCollisionObjectQueryParams& ObjectQueryParams, const FCollisionQueryParams& Params = FCollisionQueryParams::DefaultQueryParam) const;
	bool LineTraceMultiByProfile(TArray<struct FHitResult>& OutHits, const FVector& Start, const FVector& End, FName ProfileName, const FCollisionQueryParams& Params = FCollisionQueryParams::DefaultQueryParam) const;
	bool SweepTestByChannel(const FVector& Start, const FVector& End, const FQuat& Rot, ECollisionChannel TraceChannel, const FCollisionShape& CollisionShape, const FCollisionQueryParams& Params = FCollisionQueryParams::DefaultQueryParam, const FCollisionResponseParams& ResponseParam = FCollisionResponseParams::DefaultResponseParam) const;
	bool SweepTestByObjectType(const FVector& Start, const FVector& End, const FQuat& Rot, const FCollisionObjectQueryParams& ObjectQueryParams, const FCollisionShape& CollisionShape, const FCollisionQueryParams& Params = FCollisionQueryParams::DefaultQueryParam) const;
	bool SweepTestByProfile(const FVector& Start, const FVector& End, const FQuat& Rot, FName ProfileName, const FCollisionShape& CollisionShape, const FCollisionQueryParams& Params) const;
	bool SweepSingleByChannel(struct FHitResult& OutHit, const FVector& Start, const FVector& End, const FQuat& Rot, ECollisionChannel TraceChannel, const FCollisionShape& CollisionShape, const FCollisionQueryParams& Params = FCollisionQueryParams::DefaultQueryParam, const FCollisionResponseParams& ResponseParam = FCollisionResponseParams::DefaultResponseParam) const;
	bool SweepSingleByObjectType(struct FHitResult& OutHit, const FVector& Start, const FVector& End, const FQuat& Rot, const FCollisionObjectQueryParams& ObjectQueryParams, const FCollisionShape& CollisionShape, const FCollisionQueryParams& Params = FCollisionQueryParams::DefaultQueryParam) const;
	bool SweepSingleByProfile(struct FHitResult& OutHit, const FVector& Start, const FVector& End, const FQuat& Rot, FName ProfileName, const FCollisionShape& CollisionShape, const FCollisionQueryParams& Params = FCollisionQueryParams::DefaultQueryParam) const;
	bool SweepMultiByChannel(TArray<struct FHitResult>& OutHits, const FVector& Start, const FVector& End, const FQuat& Rot, ECollisionChannel TraceChannel, const FCollisionShape& CollisionShape, const FCollisionQueryParams& Params = FCollisionQueryParams::DefaultQueryParam, const FCollisionResponseParams& ResponseParam = FCollisionResponseParams::DefaultResponseParam) const;
	bool SweepMultiByObjectType(TArray<struct FHitResult>& OutHits, const FVector& Start, const FVector& End, const FQuat& Rot, const FCollisionObjectQueryParams& ObjectQueryParams, const FCollisionShape& CollisionShape, const FCollisionQueryParams& Params = FCollisionQueryParams::DefaultQueryParam) const;
	bool SweepMultiByProfile(TArray<FHitResult>& OutHits, const FVector& Start, const FVector& End, const FQuat& Rot, FName ProfileName, const FCollisionShape& CollisionShape, const FCollisionQueryParams& Params = FCollisionQueryParams::DefaultQueryParam) const;
	bool OverlapBlockingTestByChannel(const FVector& Pos, const FQuat& Rot, ECollisionChannel TraceChannel, const FCollisionShape& CollisionShape, const FCollisionQueryParams& Params = FCollisionQueryParams::DefaultQueryParam, const FCollisionResponseParams& ResponseParam = FCollisionResponseParams::DefaultResponseParam) const;
```

bool OverlapAnyTestByChannel(const FVector& Pos, const FQuat& Rot, ECollisionChannel TraceChannel, const FCollisionShape& CollisionShape, const FCollisionQueryParams& Params = FCollisionQueryParams::DefaultQueryParam, const FCollisionResponseParams& ResponseParam = FCollisionResponseParams::DefaultResponseParam) const; bool OverlapAnyTestByObjectType(const FVector& Pos, const FQuat& Rot, const FCollisionObjectQueryParams& ObjectQueryParams, const FCollisionShape& CollisionShape, const FCollisionQueryParams& Params = FCollisionQueryParams::DefaultQueryParam) const;

bool OverlapBlockingTestByProfile(const FVector& Pos, const FQuat& Rot, FName ProfileName, const FCollisionShape& CollisionShape, const FCollisionQueryParams& Params = FCollisionQueryParams::DefaultQueryParam) const;

bool OverlapAnyTestByProfile(const FVector& Pos, const FQuat& Rot, FName ProfileName, const FCollisionShape& CollisionShape, const FCollisionQueryParams& Params = FCollisionQueryParams::DefaultQueryParam) const;

bool OverlapMultiByChannel(TArray<struct FOverlapResult>& OutOverlaps, const FVector& Pos, const FQuat& Rot, ECollisionChannel TraceChannel, const FCollisionShape& CollisionShape, const FCollisionQueryParams& Params = FCollisionQueryParams::DefaultQueryParam, const FCollisionResponseParams& ResponseParam = FCollisionResponseParams::DefaultResponseParam) const;

bool OverlapMultiByObjectType(TArray<struct FOverlapResult>& OutOverlaps, const FVector& Pos, const FQuat& Rot, const FCollisionObjectQueryParams& ObjectQueryParams, const FCollisionShape& CollisionShape, const FCollisionQueryParams& Params = FCollisionQueryParams::DefaultQueryParam) const;

bool OverlapMultiByProfile(TArray<struct FOverlapResult>& OutOverlaps, const FVector& Pos, const FQuat& Rot, FName ProfileName, const FCollisionShape& CollisionShape, const FCollisionQueryParams& Params = FCollisionQueryParams::DefaultQueryParam) const;

bool ComponentSweepMulti(TArray<struct FHitResult>& OutHits, class UPrimitiveComponent* PrimComp, const FVector& Start, const FVector& End, const FQuat& Rot, const FComponentQueryParams& Params) const;

bool ComponentSweepMulti(TArray<struct FHitResult>& OutHits, class UPrimitiveComponent* PrimComp, const FVector& Start, const FVector& End, const FRotator& Rot, const FComponentQueryParams& Params) const;

bool ComponentOverlapMulti(TArray<struct FOverlapResult>& OutOverlaps, const class UPrimitiveComponent* PrimComp, const FVector& Pos, const FQuat& Rot, const FComponentQueryParams& Params = FComponentQueryParams::DefaultComponentQueryParams, const FCollisionObjectQueryParams& ObjectQueryParams=FCollisionObjectQueryParams::DefaultObjectQueryParam) const;

bool ComponentOverlapMulti(TArray<struct FOverlapResult>& OutOverlaps, const class UPrimitiveComponent* PrimComp, const FVector& Pos, const FRotator& Rot, const FComponentQueryParams& Params = FComponentQueryParams::DefaultComponentQueryParams, const FCollisionObjectQueryParams& ObjectQueryParams=FCollisionObjectQueryParams::

bool ComponentOverlapMultiByChannel(TArray<struct FOverlapResult>& OutOverlaps, const class UPrimitiveComponent* PrimComp, const FVector& Pos, const FQuat& Rot, ECollisionChannel TraceChannel, const FComponentQueryParams& Params = FComponentQueryParams::DefaultComponentQueryParams, const FCollisionObjectQueryParams& ObjectQueryParams=FCollisionObjectQueryParams::DefaultObjectQueryParam) const;

bool ComponentOverlapMultiByChannel(TArray<struct FOverlapResult>& OutOverlaps, const class UPrimitiveComponent* PrimComp, const FVector& Pos, const FRotator& Rot, ECollisionChannel TraceChannel, const FComponentQueryParams& Params = FComponentQueryParams::DefaultComponentQueryParams, const FCollisionObjectQueryParams& ObjectQueryParams=FCollisionObjectQueryParams::DefaultObjectQueryParam) const;

FTraceHandle AsyncLineTraceByChannel(EAsyncTraceType InTraceType, const FVector& Start,const FVector& End, ECollisionChannel TraceChannel, const FCollisionQueryParams& Params = FCollisionQueryParams::DefaultQueryParam, const FCollisionResponseParams& ResponseParam = FCollisionResponseParams::DefaultResponseParam, FTraceDelegate * InDelegate=NULL, uint32 UserData = 0);

FTraceHandle AsyncLineTraceByObjectType(EAsyncTraceType InTraceType, const FVector& Start,const FVector& End, const FCollisionObjectQueryParams& ObjectQueryParams, const FCollisionQueryParams& Params = FCollisionQueryParams::DefaultQueryParam, FTraceDelegate * InDelegate=NULL, uint32 UserData = 0);

FTraceHandle AsyncSweepByChannel(EAsyncTraceType InTraceType, const FVector& Start, const FVector& End, ECollisionChannel TraceChannel, const FCollisionShape& CollisionShape, const FCollisionQueryParams& Params = FCollisionQueryParams::DefaultQueryParam, const FCollisionResponseParams& ResponseParam = FCollisionResponseParams::DefaultResponseParam, FTraceDelegate * InDelegate = NULL, uint32 UserData = 0);

FTraceHandle AsyncSweepByObjectType(EAsyncTraceType InTraceType, const FVector& Start, const FVector& End, const FCollisionObjectQueryParams& ObjectQueryParams, const FCollisionShape& CollisionShape, const FCollisionQueryParams& Params = FCollisionQueryParams::DefaultQueryParam, FTraceDelegate * InDelegate = NULL, uint32 UserData = 0);

FTraceHandle AsyncOverlapByChannel(const FVector& Pos, const FQuat& Rot, ECollisionChannel TraceChannel, const FCollisionShape& CollisionShape, const FCollisionQueryParams& Params = FCollisionQueryParams::DefaultQueryParam, const FCollisionResponseParams& ResponseParam = FCollisionResponseParams::DefaultResponseParam, FOverlapDelegate * InDelegate = NULL, uint32 UserData = 0);

FTraceHandle AsyncOverlapByObjectType(const FVector& Pos, const FQuat& Rot, const FCollisionObjectQueryParams& ObjectQueryParams, const FCollisionShape& CollisionShape, const FCollisionQueryParams& Params = FCollisionQueryParams::DefaultQueryParam, FOverlapDelegate * InDelegate = NULL, uint32 UserData = 0);

bool QueryTraceData(const FTraceHandle& Handle, FTraceDatum& OutData);

```
	bool QueryOverlapData(const FTraceHandle& Handle, FOverlapDatum& OutData);
	bool IsTraceHandleValid(const FTraceHandle& Handle, bool bOverlapTrace);

	FORCEINLINE UNavigationSystemBase* GetNavigationSystem() { return NavigationSystem; }
	FORCEINLINE const UNavigationSystemBase* GetNavigationSystem() const { return NavigationSystem; }
	FORCEINLINE class UAvoidanceManager* GetAvoidanceManager() { return AvoidanceManager; }
	/** Avoidance manager getter */
	FORCEINLINE const class UAvoidanceManager* GetAvoidanceManager() const { return AvoidanceManager; }
	struct FActorsInitializedParams
	{
		FActorsInitializedParams(UWorld* InWorld, bool InResetTime) : World(InWorld), ResetTime(InResetTime) {}
		UWorld* World;
		bool ResetTime;
	};
	DECLARE_MULTICAST_DELEGATE_OneParam(FOnWorldInitializedActors, const FActorsInitializedParams&);
	FOnWorldInitializedActors OnActorsInitialized;
AWorldSettings actor associated with this world
	 */
	UFUNCTION(BlueprintCallable, Category="Utilities|World", meta=(DisplayName="GetWorldSettings", Script-
Name="GetWorldSettings"))
	AWorldSettings* K2_GetWorldSettings();
	AWorldSettings* GetWorldSettings( bool bCheckStreamingPersistent = false, bool bChecked = true ) const;
	bool AllowAudioPlayback() const;

	//~ Begin UObject Interface
	virtual void Serialize( FArchive& Ar ) override;
	virtual void BeginDestroy() override;
	virtual void FinishDestroy() override;
	virtual void PostLoad() override;
	virtual bool PreSaveRoot(const TCHAR* Filename) override;
	virtual void PostSaveRoot( bool bCleanupIsRequired ) override;
	virtual UWorld* GetWorld() const override;
	virtual FPrimaryAssetId GetPrimaryAssetId() const override;
	static void AddReferencedObjects(UObject* InThis, FReferenceCollector& Collector);
#if WITH_EDITOR
	virtual bool Rename(const TCHAR* NewName = NULL, UObject* NewOuter = NULL, ERenameFlags Flags =
REN_None) override;
	virtual void GetAssetRegistryTags(TArray<FAssetRegistryTag>& OutTags) const override;
#endif
	virtual void PostDuplicate(bool bDuplicateForPIE) override;
	void ClearWorldComponents();
	void UpdateWorldComponents(bool bRerunConstructionScripts, bool bCurrentLevelOnly);
	bool UpdateCullDistanceVolumes(AActor* ActorToUpdate = nullptr, UPrimitiveComponent* ComponentToUpdate =
nullptr);
	void PropagateLightingScenarioChange();	void AddToWorld( ULevel* Level, const FTransform& LevelTransform
= FTransform::Identity, bool bConsiderTimeLimit = true );
Level				Level object we should remove
	 */
	void RemoveFromWorld( ULevel* Level, bool bAllowIncrementalRemoval = false );
	void UpdateLevelStreaming();
	void ReleasePhysicsScene();
public:
	void AsyncLoadAlwaysLoadedLevelsForSeamlessTravel();
	bool AllowLevelLoadRequests();
	/** Creates instances for each parameter collection in memory.  Called when a world is created. */
	void SetupParameterCollectionInstances();
	/** Adds a new instance of the given collection, or overwrites an existing instance if there is one. */
	void AddParameterCollectionInstance(class UMaterialParameterCollection* Collection, bool bUpdateScene);
	/** Gets this world's instance for a given collection. */
	UMaterialParameterCollectionInstance* GetParameterCollectionInstance(const UMaterialParameterCollection* Collec-
tion);
	/** Updates this world's scene with the list of instances, and optionally updates each instance's uniform buffer. */
	void UpdateParameterCollectionInstances(bool bUpdateInstanceUniformBuffers, bool bRecreateUniformBuffer);
```

```
	/** Gets the canvas object for rendering to a render target.  Will allocate one if needed. */
	UCanvas* GetCanvasForRenderingToTarget();
	UCanvas* GetCanvasForDrawMaterialToRenderTarget();
	/** Struct containing a collection of optional parameters for initialization of a World. */
	struct InitializationValues
	{
		InitializationValues()
			: bInitializeScenes(true)
			, bAllowAudioPlayback(true)
			, bRequiresHitProxies(true)
			, bCreatePhysicsScene(true)
			, bCreateNavigation(true)
			, bCreateAISystem(true)
			, bShouldSimulatePhysics(true)
			, bEnableTraceCollision(false)
			, bTransactional(true)
			, bCreateFXSystem(true)
		{
		}
		TSubclassOf<class AGameModeBase> DefaultGameMode;

		InitializationValues& InitializeScenes(const bool bInitialize) { bInitializeScenes = bInitialize; return *this; }
		InitializationValues& AllowAudioPlayback(const bool bAllow) { bAllowAudioPlayback = bAllow; return *this;
}
		InitializationValues& RequiresHitProxies(const bool bRequires) { bRequiresHitProxies = bRequires; return
*this; }
		InitializationValues& CreatePhysicsScene(const bool bCreate) { bCreatePhysicsScene = bCreate; return *this; }
		InitializationValues& CreateNavigation(const bool bCreate) { bCreateNavigation = bCreate; return *this; }
		InitializationValues& CreateAISystem(const bool bCreate) { bCreateAISystem = bCreate; return *this; }
		InitializationValues& ShouldSimulatePhysics(const bool bInShouldSimulatePhysics) { bShouldSimulatePhysics
= bInShouldSimulatePhysics; return *this; }
		InitializationValues& EnableTraceCollision(const bool bInEnableTraceCollision) { bEnableTraceCollision =
bInEnableTraceCollision; return *this; }
		InitializationValues& SetTransactional(const bool bInTransactional) { bTransactional = bInTransactional; return
*this; }
		InitializationValues& CreateFXSystem(const bool bCreate) { bCreateFXSystem = bCreate; return *this; }
		InitializationValues& SetDefaultGameMode(TSubclassOf<class AGameModeBase> GameMode) { DefaultGa-
meMode = GameMode; return *this; }
	/** Network Tick events */
	FOnNetTickEvent& OnTickDispatch() { return TickDispatchEvent; }
	FOnTickFlushEvent& OnPostTickDispatch() { return PostTickDispatchEvent; }
	FOnNetTickEvent& OnTickFlush() { return TickFlushEvent; }
	FOnTickFlushEvent& OnPostTickFlush() { return PostTickFlushEvent; }
	FLevelCollection& FindOrAddCollectionByType(const ELevelCollectionType InType);

	/** Returns the index of the first FLevelCollection of the given InType. If one does not exist, it is created and its index
returned. */
	int32 FindOrAddCollectionByType_Index(const ELevelCollectionType InType);

	/** Returns the FLevelCollection for the given InType, or null if a collection of that type hasn't been created yet. */
	FLevelCollection* FindCollectionByType(const ELevelCollectionType InType);

	/** Returns the FLevelCollection for the given InType, or null if a collection of that type hasn't been created yet. */
	const FLevelCollection* FindCollectionByType(const ELevelCollectionType InType) const;

	/** Returns the index of the FLevelCollection with the given InType, or INDEX_NONE if a collection of that type hasn't
been created yet. */
	int32 FindCollectionIndexByType(const ELevelCollectionType InType) const;

	/**
	 * Returns the level collection which currently has its context set on this world. May be null.
	 * If non-null, this implies that execution is currently within the scope of an FScopedLevelCollectionContextSwitch for this
world.
```

```
		 */
		const FLevelCollection* GetActiveLevelCollection() const;

		/**
		 * Returns the index of the level collection which currently has its context set on this world. May be INDEX_NONE.
		 * If not INDEX_NONE, this implies that execution is currently within the scope of an FScopedLevelCollectionContextSwitch for this world.
		 */
		int32 GetActiveLevelCollectionIndex() const { return ActiveLevelCollectionIndex; }

		/** Sets the level collection and its context on this world. Should only be called by FScopedLevelCollectionContextSwitch. */
		void SetActiveLevelCollection(int32 LevelCollectionIndex);

		/** Returns a read-only reference to the list of level collections in this world. */
		const TArray<FLevelCollection>& GetLevelCollections() const { return LevelCollections; }

		/**
		 * Creates a new level collection of type DynamicDuplicatedLevels by duplicating the levels in DynamicSourceLevels.
		 * Should only be called by engine.
		 *
		 * @param MapName The name of the soure map, used as a parameter to UEngine::Experimental_ShouldPreDuplicateMap
		 */
		void DuplicateRequestedLevels(const FName MapName);

		/** Handle Exec/Console Commands related to the World */
		bool Exec( UWorld* InWorld, const TCHAR* Cmd, FOutputDevice& Ar=*GLog );

		/** Mark the world as being torn down */
		void BeginTearingDown();
		bool HandleDemoSpeedCommand(const TCHAR* Cmd, FOutputDevice& Ar, UWorld* InWorld);
		bool IsCameraMoveable() const;
		bool EditorDestroyActor( AActor* Actor, bool bShouldModifyLevel );
		bool DestroyActor( AActor* Actor, bool bNetForce=false, bool bShouldModifyLevel=true );
		void RemoveActor( AActor* Actor, bool bShouldModifyLevel ) const;
		AActor* SpawnActor( UClass* InClass, FVector const* Location=NULL, FRotator const* Rotation=NULL, const FActorSpawnParameters& SpawnParameters = FActorSpawnParameters() );
		AActor* SpawnActor( UClass* Class, FTransform const* Transform, const FActorSpawnParameters& SpawnParameters = FActorSpawnParameters());
		AActor* SpawnActorAbsolute( UClass* Class, FTransform const& AbsoluteTransform, const FActorSpawnParameters& SpawnParameters = FActorSpawnParameters());
		template< class T >
		T* SpawnActor( const FActorSpawnParameters& SpawnParameters = FActorSpawnParameters() )
		{
			return CastChecked<T>(SpawnActor(T::StaticClass(), NULL, NULL, SpawnParameters),ECastCheckedType::NullAllowed);
		}
		template< class T >
		T* SpawnActor( FVector const& Location, FRotator const& Rotation, const FActorSpawnParameters& SpawnParameters = FActorSpawnParameters() )
		{
			return CastChecked<T>(SpawnActor(T::StaticClass(), &Location, &Rotation, SpawnParameters),ECastCheckedType::NullAllowed);
		}
		template< class T >
		T* SpawnActor( UClass* Class, const FActorSpawnParameters& SpawnParameters = FActorSpawnParameters() )
		{
			return CastChecked<T>(SpawnActor(Class, NULL, NULL, SpawnParameters),ECastCheckedType::NullAllowed);
		}
		template< class T >
```

```
	T* SpawnActor( UClass* Class, FVector const& Location, FRotator const& Rotation, const FActorSpawnParameters& SpawnParameters = FActorSpawnParameters() )
	{
		return CastChecked<T>(SpawnActor(Class, &Location, &Rotation, SpawnParameters),ECastCheckedType::NullAllowed);
	}
	template< class T >
	T* SpawnActor(UClass* Class, FTransform const& Transform,const FActorSpawnParameters& SpawnParameters = FActorSpawnParameters())
	{
		return CastChecked<T>(SpawnActor(Class, &Transform, SpawnParameters), ECastCheckedType::NullAllowed);
	}
	template< class T >
	T* SpawnActorAbsolute(FVector const& AbsoluteLocation, FRotator const& AbsoluteRotation, const FActorSpawnParameters& SpawnParameters = FActorSpawnParameters())
	{
		return CastChecked<T>(SpawnActorAbsolute(T::StaticClass(), FTransform(AbsoluteRotation, AbsoluteLocation), SpawnParameters), ECastCheckedType::NullAllowed);
	}
	template< class T >
	T* SpawnActorAbsolute(UClass* Class, FTransform const& Transform,const FActorSpawnParameters& SpawnParameters = FActorSpawnParameters())
	{
		return CastChecked<T>(SpawnActorAbsolute(Class, Transform, SpawnParameters), ECastCheckedType::NullAllowed);
	}
	template< class T >
	T* SpawnActorDeferred(
		UClass* Class,
		FTransform const& Transform,
		AActor* Owner = nullptr,
		APawn* Instigator = nullptr,
		ESpawnActorCollisionHandlingMethod CollisionHandlingOverride = ESpawnActorCollisionHandlingMethod::Undefined
		)
	{
		if( Owner ){check(this==Owner->GetWorld());}
		FActorSpawnParameters SpawnInfo;
		SpawnInfo.SpawnCollisionHandlingOverride = CollisionHandlingOverride;
		SpawnInfo.Owner = Owner;
		SpawnInfo.Instigator = Instigator;
		SpawnInfo.bDeferConstruction = true;
		return (Class != nullptr) ? Cast<T>(SpawnActor(Class, &Transform, SpawnInfo)) : nullptr;
	}
	template< class T > T* GetAuthGameMode() const
	{	return Cast<T>(AuthorityGameMode); }
	AGameModeBase* GetAuthGameMode() const { return AuthorityGameMode; }

	template< class T >
	T* GetGameState() const	{		return Cast<T>(GameState);  }
	void SetGameState(AGameStateBase* NewGameState);
	void CopyGameState(AGameModeBase* FromGameMode, AGameStateBase* FromGameState);
	DECLARE_EVENT_OneParam(UWorld, FOnGameStateSetEvent, AGameStateBase*);
	FOnGameStateSetEvent GameStateSetEvent;
	ABrush*  SpawnBrush();
	APlayerController* SpawnPlayActor(class UPlayer* Player, ENetRole RemoteRole, const FURL& InURL, const TSharedPtr<const FUniqueNetId>& UniqueId, FString& Error, uint8 InNetPlayerIndex = 0);
	APlayerController* SpawnPlayActor(class UPlayer* Player, ENetRole RemoteRole, const FURL& InURL, const FUniqueNetIdRepl& UniqueId, FString& Error, uint8 InNetPlayerIndex = 0);
	bool FindTeleportSpot( const AActor* TestActor, FVector& PlaceLocation, FRotator PlaceRotation );
	bool EncroachingBlockingGeometry( const AActor* TestActor, FVector TestLocation, FRotator TestRotation, FVector* ProposedAdjustment = NULL );
```

```
		void StartPhysicsSim();
		void FinishPhysicsSim();	bool SetGameMode(const FURL& InURL);
		void InitializeActorsForPlay(const FURL& InURL, bool bResetTime = true);
		void BeginPlay();
		bool DestroySwappedPC(UNetConnection* Connection);
		virtual EAcceptConnection::Type NotifyAcceptingConnection() override;
		virtual void NotifyAcceptedConnection( class UNetConnection* Connection ) override;
		virtual bool NotifyAcceptingChannel( class UChannel* Channel ) override;
		virtual void NotifyControlMessage(UNetConnection* Connection, uint8 MessageType, class FInBunch& Bunch) override;
		FORCEINLINE_DEBUGGABLE UNetDriver* GetNetDriver() const{		return NetDriver;	}
		ENetMode GetNetMode() const;
		bool IsNetMode(ENetMode Mode) const;

private:
		ENetMode InternalGetNetMode() const;
		ENetMode AttemptDeriveFromPlayInSettings() const;
#endif

		/** Attempts to derive the net mode from URL */
		ENetMode AttemptDeriveFromURL() const;

		APhysicsVolume* InternalGetDefaultPhysicsVolume() const;

public:
		void DelayStreamingVolumeUpdates(int32 InFrameDelay) {StreamingVolumeUpdateDelay = InFrameDelay;}
		void TransferBlueprintDebugReferences(UWorld* NewWorld);
		FOnBeginTearingDownEvent& OnBeginTearingDown() { return BeginTearingDownEvent; }
		int32 GetProgressDenominator();
		int32 GetActorCount();

public:
		void SetAudioDeviceHandle(const uint32 InAudioDeviceHandle);
		class FAudioDevice* GetAudioDevice();
		bool UsesGameHiddenFlags() const;
		virtual FString GetAddressURL() const;
		void LoadSecondaryLevels(bool bForce = false, TSet<FName>* FilenamesToSkip = NULL);
		ULevelStreaming* GetLevelStreamingForPackageName(FName PackageName);
		void RefreshStreamingLevels();
		void RefreshStreamingLevels( const TArray<class ULevelStreaming*>& InLevelsToRefresh );
private:
		bool bIsRefreshingStreamingLevels;
public:
		bool IsRefreshingStreamingLevels() const { return bIsRefreshingStreamingLevels; }
		void IssueEditorLoadWarnings();
#endif
		virtual bool ServerTravel(const FString& InURL, bool bAbsolute = false, bool bShouldSkipGameNotify = false)
		void SeamlessTravel(const FString& InURL, bool bAbsolute = false, FGuid MapPackageGuid = FGuid());
		bool IsInSeamlessTravel();
		void SetSeamlessTravelMidpointPause(bool bNowPaused);
		int32 GetDetailMode();
		UE_DEPRECATED(4.18, "Call GEngine->ForceGarbageCollection instead")
		void ForceGarbageCollection( bool bFullPurge = false );
		void PrepareMapChange(const TArray<FName>& LevelNames);
		bool IsPreparingMapChange();
		bool IsMapChangeReady();
		void CancelPendingMapChange();
		void CommitMapChange();
		void SetMapNeedsLightingFullyRebuilt(int32 InNumLightingUnbuiltObjects, int32 InNumUnbuiltReflectionCaptures);
		inline FTimerManager& GetTimerManager() const
		{
			return (OwningGameInstance ? OwningGameInstance->GetTimerManager() : *TimerManager);
		}
		inline FLatentActionManager& GetLatentActionManager()
```

```
	{
		return (OwningGameInstance ? OwningGameInstance->GetLatentActionManager() : LatentActionManager);
	}
	UWorldSubsystem* GetSubsystemBase(TSubclassOf<UWorldSubsystem> SubsystemClass) const
	{
		return SubsystemCollection.GetSubsystem<UWorldSubsystem>(SubsystemClass);
	}
	template <typename TSubsystemClass>
	TSubsystemClass* GetSubsystem() const
	{
		return SubsystemCollection.GetSubsystem<TSubsystemClass>(TSubsystemClass::StaticClass());
	}
	template <typename TSubsystemClass>
	static FORCEINLINE TSubsystemClass* GetSubsystem(const UWorld* World)
	{
		if (World)
		{
			return World->GetSubsystem<TSubsystemClass>();
		}
		return nullptr;
	}
	template <typename TSubsystemClass>
	const TArray<TSubsystemClass*>& GetSubsystemArray() const
	{
		return SubsystemCollection.GetSubsystemArray<TSubsystemClass>(TSubsystemClass::StaticClass());
	}
	inline void SetGameInstance(UGameInstance* NewGI)
	{
		OwningGameInstance = NewGI;
	}
	inline UGameInstance* GetGameInstance() const
	{
		return OwningGameInstance;
	}
	template<class T>
	T* GetGameInstance() const
	{
		return Cast<T>(OwningGameInstance);
	}
	/** Returns the OwningGameInstance cast to the template type, asserting that it is of the correct type. */
	template<class T>
	T* GetGameInstanceChecked() const
	{
		return CastChecked<T>(OwningGameInstance);
	}
	/** Retrieves information whether all navigation with this world has been rebuilt */
	bool IsNavigationRebuilt() const;
	/** Request to translate world origin to specified position on next tick */
	void RequestNewWorldOrigin(FIntVector InNewOriginLocation);

	/** Translate world origin to specified position  */
	bool SetNewWorldOrigin(FIntVector InNewOriginLocation);
	/** Sets world origin at specified position and stream-in all relevant levels */
	void NavigateTo(FIntVector InLocation);
	/** Gets all matinee actors for the current level */
	void GetMatineeActors( TArray<AMatineeActor*>& OutMatineeActors );
	/** Updates all physics constraint actor joint locations.  */
	virtual void UpdateConstraintActors();
	/** Gets all LightMaps and ShadowMaps associated with this world. Specify the level or leave null for persistent */
	void GetLightMapsAndShadowMaps(ULevel* Level, TArray<UTexture2D*>& OutLightMapsAndShadowMaps);
public:
	/** Rename this world such that it has the prefix on names for the given PIE Instance ID */
	void RenameToPIEWorld(int32 PIEInstanceID);
```

```
		/** Given a level script actor, modify the string such that it points to the correct instance of the object. For replays. */
		bool RemapCompiledScriptActor(FString& Str) const;
		/** Given a PackageName and a PIE Instance ID return the name of that Package when being run as a PIE world */
		static FString ConvertToPIEPackageName(const FString& PackageName, int32 PIEInstanceID);
		/** Given a PackageName and a prefix type, get back to the original package name (i.e. the saved map name) */
		static FString StripPIEPrefixFromPackageName(const FString& PackageName, const FString& Prefix);
		/** Return the prefix for PIE packages given a PIE Instance ID */
		static FString BuildPIEPackagePrefix(int32 PIEInstanceID);
		/** Given a loaded editor UWorld, duplicate it for play in editor purposes with OwningWorld as the world with the persistent level. */
		static UWorld* DuplicateWorldForPIE(const FString& PackageName, UWorld* OwningWorld);
		/** Given a string, return that string with any PIE prefix removed */
		static FString RemovePIEPrefix(const FString &Source);
		/** Given a package, locate the UWorld contained within if one exists */
		static UWorld* FindWorldInPackage(UPackage* Package);
		/** If the specified package contains a redirector to a UWorld, that UWorld is returned. Otherwise, nullptr is returned. */
		static UWorld* FollowWorldRedirectorInPackage(UPackage* Package, UObjectRedirector** OptionalOutRedirector = nullptr);
		FORCEINLINE FWorldPSCPool& GetPSCPool() { return PSCPool; }
		private:
		UPROPERTY()
		FWorldPSCPool PSCPool;
		//PSC Pooling END
		FSubsystemCollection<UWorldSubsystem> SubsystemCollection;
};
/** Global UWorld pointer. Use of this pointer should be avoided whenever possible. */
extern ENGINE_API class UWorldProxy GWorld;

/** World delegates */
class ENGINE_API FWorldDelegates
{
public:
		DECLARE_MULTICAST_DELEGATE_TwoParams(FWorldInitializationEvent, UWorld* /*World*/, const UWorld::InitializationValues /*IVS*/);
		DECLARE_MULTICAST_DELEGATE_ThreeParams(FWorldCleanupEvent, UWorld* /*World*/, bool /*bSessionEnded*/, bool /*bCleanupResources*/);
		DECLARE_MULTICAST_DELEGATE_OneParam(FWorldEvent, UWorld* /*World*/);
		/**
		 * Post UWorld duplicate event.
		 *
		 * Sometimes there is a need to duplicate additional element after
		 * duplicating UWorld. If you do this using this event you need also fill
		 * ReplacementMap and ObjectsToFixReferences in order to properly fix
		 * duplicated objects references.
		 */
		typedef TMap<UObject*, UObject*> FReplacementMap; // Typedef needed so the macro below can properly digest comma in template parameters.
		DECLARE_MULTICAST_DELEGATE_FourParams(FWorldPostDuplicateEvent, UWorld* /*World*/, bool /*bDuplicateForPIE*/, FReplacementMap& /*ReplacementMap*/, TArray<UObject*>& /*ObjectsToFixReferences*/);
#if WITH_EDITOR
		DECLARE_MULTICAST_DELEGATE_FiveParams(FWorldRenameEvent, UWorld* /*World*/, const TCHAR* /*InName*/, UObject* /*NewOuter*/, ERenameFlags /*Flags*/, bool& /*bShouldFailRename*/);
#endif // WITH_EDITOR
		DECLARE_MULTICAST_DELEGATE_TwoParams(FOnLevelChanged, ULevel*, UWorld*);
		DECLARE_MULTICAST_DELEGATE_TwoParams(FWorldGetAssetTags, const UWorld*, TArray<UObject::FAssetRegistryTag>&);
		DECLARE_MULTICAST_DELEGATE_ThreeParams(FOnWorldTickStart, UWorld*, ELevelTick, float);
		static FOnWorldTickStart OnWorldTickStart;
		DECLARE_MULTICAST_DELEGATE_ThreeParams(FOnWorldPreActorTick, UWorld* /*World*/, ELevelTick/**Tick Type*/, float/**Delta Seconds*/);
		static FOnWorldPreActorTick OnWorldPreActorTick;
		DECLARE_MULTICAST_DELEGATE_ThreeParams(FOnWorldPostActorTick, UWorld* /*World*/, ELevelTick/**Tick Type*/, float/**Delta Seconds*/);
```

```
		static FOnWorldPostActorTick OnWorldPostActorTick;
		static FWorldEvent OnPostWorldCreation;

		static FWorldInitializationEvent OnPreWorldInitialization;

		static FWorldInitializationEvent OnPostWorldInitialization;
#if WITH_EDITOR
		static FWorldRenameEvent OnPreWorldRename;
#endif // WITH_EDITOR
		static FWorldPostDuplicateEvent OnPostDuplicate;
		static FWorldCleanupEvent OnWorldCleanup;
		static FWorldCleanupEvent OnPostWorldCleanup;
		static FWorldEvent OnPreWorldFinishDestroy;
		// Sent when a ULevel is added to the world via UWorld::AddToWorld
		static FOnLevelChanged 				LevelAddedToWorld;
		static FOnLevelChanged 				LevelRemovedFromWorld;
		DECLARE_MULTICAST_DELEGATE_FourParams(FLevelOffsetEvent, ULevel*,  UWorld*, const FVector&, bool);
		static FLevelOffsetEvent 		PostApplyLevelOffset;
		static FWorldGetAssetTags GetAssetTags;
		DECLARE_MULTICAST_DELEGATE_OneParam(FRefreshLevelScriptActionsEvent, UWorld*);
		static FRefreshLevelScriptActionsEvent RefreshLevelScriptActions;
#endif

		static UWorld::FOnWorldInitializedActors OnWorldInitializedActors;
		static FWorldEvent OnWorldBeginTearDown;
private:
		FWorldDelegates() {}
};
struct FWorldNotifyStreamingLevelLoading
{
private:
		static void Started(UWorld* World)
		{
			++World->NumStreamingLevelsBeingLoaded;
		}
		static void Finished(UWorld* World)
		{
			if (ensure(World->NumStreamingLevelsBeingLoaded > 0))
			{
				--World->NumStreamingLevelsBeingLoaded;
			}
		}

		friend ULevelStreaming;};

FORCEINLINE_DEBUGGABLE float UWorld::GetTimeSeconds() const
{		return TimeSeconds;}

FORCEINLINE_DEBUGGABLE float UWorld::GetUnpausedTimeSeconds() const
{		return UnpausedTimeSeconds;}

FORCEINLINE_DEBUGGABLE float UWorld::GetRealTimeSeconds() const
{		checkSlow(!IsInActualRenderingThread());		return RealTimeSeconds;
}

FORCEINLINE_DEBUGGABLE float UWorld::GetAudioTimeSeconds() const
{		return AudioTimeSeconds;}
FORCEINLINE_DEBUGGABLE float UWorld::GetDeltaSeconds() const
{		return DeltaTimeSeconds;}
FORCEINLINE_DEBUGGABLE float UWorld::TimeSince(float Time) const
{		return GetTimeSeconds() - Time;  }
FORCEINLINE_DEBUGGABLE FConstPhysicsVolumeIterator UWorld::GetNonDefaultPhysicsVolumeIterator() const
{		auto Result = NonDefaultPhysicsVolumeList.CreateConstIterator();
```

```
	return (const FConstPhysicsVolumeIterator&)Result;}
FORCEINLINE_DEBUGGABLE int32 UWorld::GetNonDefaultPhysicsVolumeCount() const
{	return NonDefaultPhysicsVolumeList.Num();}
FORCEINLINE_DEBUGGABLE bool UWorld::ComponentOverlapMulti(TArray<struct FOverlapResult>& OutOverlaps, const class UPrimitiveComponent* PrimComp, const FVector& Pos, const FRotator& Rot, const FComponentQueryParams& Params, const FCollisionObjectQueryParams& ObjectQueryParams) const
{	return ComponentOverlapMulti(OutOverlaps, PrimComp, Pos, Rot.Quaternion(), Params, ObjectQueryParams);}
FORCEINLINE_DEBUGGABLE bool UWorld::ComponentOverlapMultiByChannel(TArray<struct FOverlapResult>& OutOverlaps, const class UPrimitiveComponent* PrimComp, const FVector& Pos, const FRotator& Rot, ECollisionChannel TraceChannel, const FComponentQueryParams& Params /* = FComponentQueryParams::DefaultComponentQueryParams */, const FCollisionObjectQueryParams& ObjectQueryParams/* =FCollisionObjectQueryParams::DefaultObjectQueryParam */) const
{	return ComponentOverlapMultiByChannel(OutOverlaps, PrimComp, Pos, Rot.Quaternion(), TraceChannel, Params);}
FORCEINLINE_DEBUGGABLE bool UWorld::ComponentSweepMulti(TArray<struct FHitResult>& OutHits, class UPrimitiveComponent* PrimComp, const FVector& Start, const FVector& End, const FRotator& Rot, const FComponentQueryParams& Params) const
{	return ComponentSweepMulti(OutHits, PrimComp, Start, End, Rot.Quaternion(), Params);}
FORCEINLINE_DEBUGGABLE ENetMode UWorld::GetNetMode() const
{
	if (IsRunningDedicatedServer())
	{		return NM_DedicatedServer; }
	return InternalGetNetMode();
}
FORCEINLINE_DEBUGGABLE bool UWorld::IsNetMode(ENetMode Mode) const
{
#if UE_EDITOR
	return GetNetMode() == Mode;
#else
	if (Mode == NM_DedicatedServer)
	{		return IsRunningDedicatedServer();	}
	else
	{		return !IsRunningDedicatedServer() && (InternalGetNetMode() == Mode);	}
#endif
}
```

ПРИЛОЖЕНИЕ Б
(обязательное)
Руководство пользователя

Б.1 Введение

В данном программном документе приведено руководство пользователя по настройке и использованию программы-тренажера, предназначенной для проведения обучения в различных ситуациях.

Может использоваться в образовательных учреждениях, предприятиях, занимающихся подготовкой и повышением квалификации персонала в различных сферах труда.

Пользователь должен обладать общими навыками по работе с персональным компьютером, устройствами виртуальной реальности, знать и соблюдать правила техники безопасности и охраны труда.

Б.2 Назначение и условия применения

На основании полученной информации специалист сможет делать выводы об уровне сотрудника в определенной сфере труда. Это позволит улучшить работоспособность коллектива, выявить у пользователя недостаточную базу знаний в той или иной области для дальнейшей работы по улучшению качества знаний.

Основной функцией приложения-тренажера является выполнение различных сценариев.

Программа реализует следующие функции:

- проводить обучение персонала по различным сценариям;
- создавать, редактировать и загружать новые сценарии в приложение;
- записывать видео, с текущим прохождением сценария;
- просматривать всю информацию о ранее проведенных тестированиях.

Для функционирования программы необходимы персональный компьютер, отвечающий минимальным требованиям, и устройство виртуальной реальности.

Б.3 Подготовка к работе

Для использования программы требуется установка приложений *Windows Mixed Reality, SteamVR, Bandicam.*

При запуске исполняемого файла программного средства вид главного окна приложения представлен на рисунке Б.1.

На данном экране представлен стартовый уровень приложения, где пользователь выбирает один из четырех доступных для выполнения сценариев, при необходимости сценарии можно загружать, используя веб приложение.

Рисунок Б.1 – Главное окно программы-тренажера

На рисунке Б.2 представлено главное окно веб-приложения для создания, загрузки и редактирования сценариев, а также для просмотра сводных отчетов о всех прохождениях.

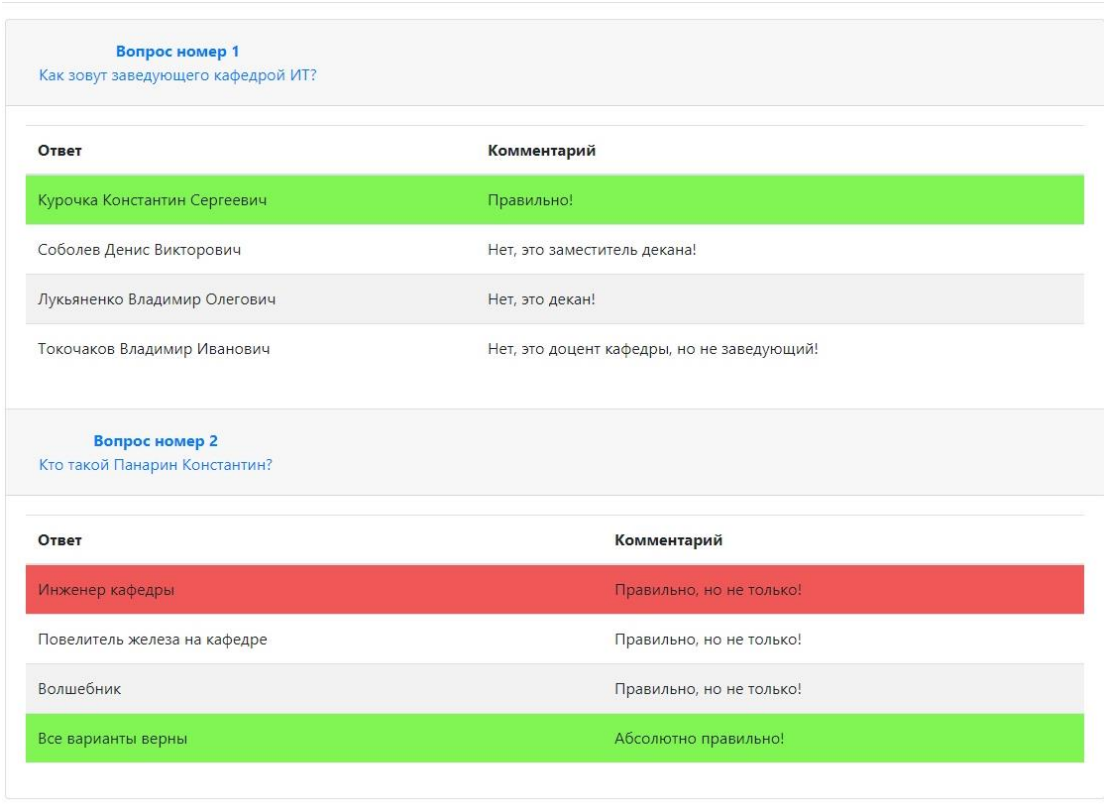

Рисунок Б.2 – Главное окно веб-приложения

Б.4 Описание операций

Основные операции, доступные в программном комплексе:

− загрузка создание и редактирование сценариев;

− прохождение сценариев;

− просмотр сводной статистики прохождения теста;

− возможность получения видеофайла прохождения.

Б.5 Технические требования к ПО

Минимальные технические требования к компьютеру, на котором может использоваться разработанное программное обеспечение следующие:

– *Windows* 8, 8.1, 10 (Рекомендуемым и оптимальным решением является *Windows* 10);

– процессор *i*5-4590 (аналогичный процессор от *AMD*) или выше;

– наличие внешнего графического ускорителя уровня *NVIDIA GTX* 970 или выше.

– наличие минимум 4 гигабайт оперативной памяти или выше;

– наличие одного *USB* порта 2.0 или выше, один порт *HDMI* 1.4, один *DisplayPort* 1.2 или выше.

Минимальные технические требования устанавливают порог, при котором достигается номинальная производительность разработанных алгоритмов.

Б.6 Аварийные ситуации

Программа не выдает никаких сообщений пользователю об ошибке. В случае возникновения аварийной ситуации рекомендуется выполнить перезапуск приложения. Проверить работу других приложений виртуальной реальности, переподключить устройства ввода. В случае если проблемы не могут быть устранены следует обратиться к системному администратору.

ПРИЛОЖЕНИЕ В

(обязательное)

Руководство системного программиста

В.1 Общие сведения о программном комплексе

В данном программном документе приведено руководство системного программиста по настройке и использованию программы-тренажера, предназначенной для проведения обучения в различных ситуациях.

Может использоваться в образовательных учреждениях, предприятиях, занимающихся подготовкой и повышением квалификации персонала в различных сферах труда.

Основной функцией приложения-тренажера является выполнение различных сценариев.

Программа реализует следующие функции:

- проводить обучение персонала по различным сценариям;
- создавать, редактировать и загружать новые сценарии в приложение;
- записывать видео, с текущим прохождением сценария;
- просматривать всю информацию о ранее проведенных тестированиях.

Для функционирования программы необходимы персональный компьютер, отвечающий минимальным требованиям, и устройство виртуальной реальности.

В.2 Структура программного комплекса

Программное средство состоит из двух независимых приложений: настольного приложения виртуальной реальности, веб-приложения для просмотра информации.

В.3 Настройка программного комплекса

Для использования программы требуется установка приложений *Windows Mixed Reality, SteamVR, Bandicam*. А также проверка установки последних обновлений операционной системы.

В.4 Проверка программного комплекса

После установки и настройки программного комплекса требуется осуществить его запуск. Если главное окно загрузилось успешно, то программное средство установлено и настроено корректно.

В.5 Сообщения системному программисту

Программа-тренажер не выдает никаких сообщений системному программисту.

ПРИЛОЖЕНИЕ Г
(обязательное)
Руководство программиста

Г.1 Назначение и условия применения программного комплекса

В данном программном документе приведено руководство системного программиста по настройке и использованию программы-тренажера, предназначенной для проведения обучения в различных ситуациях.

Может использоваться в образовательных учреждениях, предприятиях, занимающихся подготовкой и повышением квалификации персонала в различных сферах труда.

Основной функцией приложения-тренажера является выполнение различных сценариев.

Программа реализует следующие функции:

- проводить обучение персонала по различным сценариям;
- создавать, редактировать и загружать новые сценарии в приложение;
- записывать видео, с текущим прохождением сценария;
- просматривать всю информацию о ранее проведенных тестированиях.

Для функционирования программы необходимы персональный компьютер, отвечающий минимальным требованиям, и устройство виртуальной реальности.

Для использования программы требуется установка приложений *Windows Mixed Reality, SteamVR, Bandicam*. А также проверка установки последних обновлений операционной системы.

Г.2 Характеристики программного комплекса

Приложение тренажер разработано на языке *C++* в среде *Unreal Engine* 4, веб-приложение для просмотра статистики разработано на языке *C#* в среде *Microsoft Visual Studio* 2019, с платформой для разработки *.NET Core* 3.1.

Приложение может функционировать в ОС *Windows*.

Весь программный код отформатирован в соответствии с правилами оформления данных языков и любые вносимые изменения обязательно должны полностью соответствовать данному стилю.

Г.3 Обращение к программному комплексу

Для внесения изменений в автоматизированную систему необходимо использовать среду *Microsoft Visual Studio* и *Unreal Engine* 4.

Г.4 Входные и выходные данные

Для корректной работы программы конечный пользователь должен загрузить необходимые входные данные. Обязательными данными является загрузка в приложение четырех файлов, содержащих сценарий тестирования.

Выходными данными в программе является информация о прохождениях сценариев с возможностью сохранения в файл (путь скачивания указывается пользователем);

Г.5 Сообщения

Программа-тренажер не выдает никаких сообщений программисту.

ПРИЛОЖЕНИЕ Д

(справочное)

Результаты расчёта экономических показателей

Таблица Д.1 – Расчет коэффициента эквивалентности

Наименование параметра	Вес параметра, β	Значения параметра			$\frac{P_б}{P_э}$	$\frac{P_н}{P_э}$	$\beta\frac{P_б}{P_э}$	$\beta\frac{P_н}{P_э}$
		$P_б$	$P_н$	$P_э$				
Объем памяти	0,1	9	8	6	0,7	0,75	0,07	0,075
Время обработки данных	0,4	0,8	0,4	0,2	0,13	0,25	0,1	0,2
Отказы	0,5	2	1	1	0,5	1	0,3	0,5
Итого							0,407	0,775
Коэффициент эквивалентности							$\frac{0,775}{0,407}=1,9$	

Таблица Д.2 – Расчет коэффициента изменения функциональных возможностей

Наименование показателя	Балльная оценка базового ПО	Балльная оценка нового ПО
Объем памяти	3	5
Функциональные возможности	2	5
Быстродействие	4	4
Удобство интерфейса	3	4
Степень утомляемости	3	3
Итого	15	21
Коэффициент функциональных возможностей	$\frac{21}{15}=1,4$	

Таблица Д.3 – Исходный и уточнённый объём строк исходного кода

Код функции	Наименование (содержание) функции	Объем функции строк исходного кода (*LOC*)	
		по каталогу ($V_о$)	уточненный ($V_у$)
1	2	3	4
Ввод, анализ входной информации, генерация кодов и процессор входного языка			

Продолжение таблицы Д.3

1	2	3	4
101	Инициализация форм и элементов управления	130	1002
102	Контроль, предварительная обработка и ввод информации	490	403
109	Управление вводом-выводом	1970	891
Формирование и обработка файлов			
301	Формирование последовательного файла	590	381
303	Обработка файлов	1050	346
304	Управление файлами	5240	4249
305	Формирование файлов	2130	1507
Генерация программ и ПО, а также настройка ПО			
402	Генерация рабочих программ	3120	2099
405	Система настройки ПО	340	69
Управление ПО, компонентами ПО и внешними устройствами			
505	Управление внешней памятью	180	34
506	Обработка ошибочных сбойных ситуаций	1540	734
507	Обеспечение интерфейса между компонентами	1680	1109
Расчетные задачи, формирование и вывод на внешние носители документов сложной формы и файлов			
702	Расчетные задачи (расчет режимов обработки)	11700	8037
707	Графический вывод результатов	420	339
709	Изменение состояния ресурсов в интерактивном режиме	570	354
Итого		31150	21554

Таблица Д.4 – Нормативная трудоемкость на разработку ПО ($T_н$)

Уточнённый объем, $V_у$	Категория сложности ПО	Номер нормы
	2-я	
21554	957	76

Таблица Д.5 – Значения коэффициентов удельных весов трудоемкости стадий разработки ПО в общей трудоемкости ПО

Категория новизны ПО	Без применения *CASE*-технологии				
	Стадии разработки ПО				
	ТЗ	ЭП	ТП	РП	ВН
	Значения коэффициентов				
	$K_{ТЗ}$	$K_{ЭП}$	$K_{ТП}$	$K_{РП}$	$K_{ВН}$
В	0,08	0,19	0,28	0,34	0,11

Таблица Д.6 – Расчет общей трудоемкости разработки ПО

Показатели	Стадии разработки					Итого
	ТЗ	ЭП	ТП	РП	ВН	
1	2	3	4	5	6	7
Общий объем ПО ($V_о$), кол-во строк *LOC*	–	–	–	–	–	31150
Общий уточненный объем ПО ($V_у$), кол-во строк *LOC*	–	–	–	–	–	21554
Категория сложности разрабатываемого ПО	–	–	–	–	–	2
Нормативная трудоемкость разработки ПО ($Т_н$), чел.-дн.	–	–	–	–	–	957
Коэффициент повышения сложности ПО ($K_с$)	1,06	1,06	1,06	1,06	1,06	–
Коэффициент, учитывающий новизну ПО ($K_н$)	0,63	0,63	0,63	0,63	0,63	–
Коэффициент, учитывающий степень использования стандартных модулей ($K_т$)	–	–	–	0,65	–	–
Коэффициент, учитывающий средства разработки ПО ($K_{ур}$)	0,5	0,5	0,5	0,5	0,5	–

Продолжение таблицы Д.6

1	2	3	4	5	6	7
Коэффициенты удельных весов трудоемкости стадий разработки ПО ($K_{ТЗ}$, $K_{ЭП}$, $K_{ТП}$, $K_{РП}$, $K_{ВН}$)	0,08	0,19	0,28	0,34	0,11	1,0
Распределение скорректированной (с учетом $K_{с}$, $K_{н}$, $K_{т}$, $K_{ур}$) трудоемкости ПО по стадиям, чел.-дн.	26	61	89	71	35	282
Общая трудоемкость разработки ПО ($Т_{о}$), чел.-дн.	–	–	–	–	–	282

Таблица Д.7 – Параметры для расчета производственных затрат на разработку ПО

Параметр	Единица измерения	Значение
Коэффициент ($K_{ув}$)	–	1,6
Тарифная ставка 1-го разряда ($С_{м1}$)	руб.	185
Разряд разработчика	–	11
Тарифный коэффициент ($Т_{ki}$)	–	2,65
Норматив отчислений на доп. Зарплату разработчиков ($Н_{доп}$)	%	12
Численность обслуживающего персонала	чел.	1
Средняя годовая ставка арендных платежей ($С_{ар}$)	руб./м^2	15,2
Площадь помещения (S)	м^2	12
Количество ПЭВМ ($Q_{эвм}$)	шт.	1
Капитальные вложения в проект	руб.	2500
Стоимость одного кВт-часа электроэнергии ($С_{эл}$)	руб.	0,33048
Коэффициент потерь рабочего времени ($К_{пот}$)	–	0,15
Затраты на технологию ($З_{тех}$)	руб.	–
Норматив общепроизводственных затрат ($Н_{доп}$)	%	10
Норматив непроизводственных затрат ($Н_{непр}$)	%	5

Таблица Д.8 – Результаты расчета суммарных затрат на разработку ПО

Статья затрат	Итого
Затраты на оплату труда разработчиков ($З_{тр}$), руб.	15818,43
Основная заработная плата разработчиков ($ЗП_{осн}$), руб.	10540
Дополнительная заработная плата разработчиков ($ЗП_{доп}$), руб.	1264,8
Отчисления от основной и дополнительной заработной платы ($ОТЧ_{с.н}$), руб.	4013,63
Затраты на заработную плату обслуживающего персонала ($ЗП_{об}$), руб.	3331,78
Годовые затраты на аренду помещения ($З_{ар}$), руб.	182,4
Сумма годовых амортизационных отчислений ($З_{ам}$), руб.	353,13
Стоимость электроэнергии, потребляемой за год ($З_{э.п}$), руб.	262,17
Затраты на материалы ($З_{в.м}$), руб.	28,25
Затраты на текущий и профилактический ремонт ($З_{т.р}$), руб.	141,25
Прочие затраты, связанные с эксплуатацией ЭВМ ($З_{пр}$), руб.	141,25
Стоимость машино-часа ($С_{ч}$), руб./ч.	2,7
Действительный годовой фонд времени работы ПЭВМ ($F_{эвм}$), ч.	1619
Машинное время ЭВМ ($t_{эвм}$), ч.	208
Затраты машинного времени ($З_{м.в}$), руб.	561,6
Затраты на изготовление эталонного экземпляра ($З_{эт}$), руб.	35,14
Затраты на технологию ($З_{тех}$), руб.	–
Общепроизводственные затраты ($З_{общ.пр}$), руб.	1054
Непроизводственные (коммерческие) затраты ($З_{непр}$), руб.	527
Суммарные затраты на разработку ПО ($З_{р}$), руб.	2347,24

Таблица Д.9 – Показатели экономической эффективности проекта

Показатель	Единица измерения	Значение
Прибыль	руб.	938,9
Уровень рентабельности	%	40
Срок окупаемости	лет	2,6

Таблица Д.10 – Технико-экономические показатели проекта

Наименование показателя	Единица измерения	Проектный вариант
Интегральный коэффициент конкурентоспособности ($K_{и}$)	–	2,8
Коэффициент эквивалентности ($K_{э}$)	–	1,9
Коэффициент изменения функциональных возможностей ($K_{ф.в}$)	–	1,4
Коэффициент соответствия нормативам ($K_{н}$)	–	1
Коэффициент цены потребления ($K_{ц}$)	–	0,95
Общая трудоемкость разработки ПО ($Т_{о}$)	чел.-дн.	282
Капитальные вложения в проект	руб.	2500
Затраты на оплату труда разработчиков ($З_{тр}$)	руб.	15818,43
Затраты машинного времени ($З_{мв}$)	руб.	561,6
Прибыль от реализации программного продукта $П_{р}$	руб.	938,90
Отпускная цена без НДС ($Ц_{опт}$)	руб.	3286,14
НДС	%	20
Отпускная цена с НДС ($Ц_{опт}$)	руб.	3943,35
Срок окупаемости проекта ($T_{пр}$)	лет	2,6