Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Саратовский государственный технический университет
имени Гагарина Ю.А.»

Институт прикладных информационных технологий и коммуникаций

Кафедра информационно-коммуникационных систем и программной инженерии

Направление 09.04.01 Информатика и вычислительная техника

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

РАЗРАБОТКА VR-ПРИЛОЖЕНИЯ "RSL FINGERSPELLER" ДЛЯ ОБУЧЕНИЯ РУССКОМУ ДАКТИЛЬНОМУ ЯЗЫКУ

Выполнил студент
Кудряшов Алексей Владимирович
курс 2
группа м-ИВЧТ-21

Руководитель работы:
к.с.н., доцент кафедры ИКСП
Пчелинцева Елена Германовна

Допущен к защите
Протокол № _____ от _____2025 г.

Зав. кафедрой ИКСП д.т.н., профессор _____А.А. Сытник

Саратов 2025

РЕФЕРАТ

Выпускная квалификационная работа «Разработка VR-приложения "RSL Fingerspeller" для обучения русскому дактильному языку» содержит 83 страницы, включая 26 рисунков, 6 таблиц, 29 литературных источников и 3 приложения.

Ключевые слова: ВИРТУАЛЬНАЯ РЕАЛЬНОСТЬ, ДАКТИЛЬНЫЙ АЛФАВИТ, ОБУЧЕНИЕ, HAND-TRACKING, UNITY.

Работа состоит из введения, четырёх разделов, заключения и приложений. Во введении обоснована актуальность разработки VR-приложений для изучения русского дактильного языка, поставлены цель и задачи исследования.

Первый раздел посвящён анализу дактильного алфавита: раскрывается его сущность, область применения, возможности цифровой реализации и особенности моделирования.

Во втором разделе рассматриваются принципы обучения в виртуальной среде, включая иммерсивность, мультимодальную обратную связь, отслеживание рук и рекомендации по проектированию пользовательского взаимодействия.

Третий раздел содержит анализ существующих аналогов, формулировку функциональных и технических требований, а также обоснование архитектурных и технологических решений проекта.

Четвёртый раздел описывает реализацию прототипа: подготовку среды Unity и гарнитуры Oculus Quest 2, структуру приложения, основные сцены и их функционал, взаимодействие через интерфейсы Poke и Ray, а также реализацию ключевых скриптов и работу с данными в формате JSON.

В заключении подведены итоги разработки, выделены основные достижения и обозначены направления для дальнейшего развития проекта.

АННОТАЦИЯ

Кудряшов А. В. Выпускная квалификационная работа на тему: «Разработка VR-приложения "RSL Fingerspeller" для обучения русскому дактильному языку».

Объектом исследования является процесс обучения русскому дактильному алфавиту.

Предметом исследования выступает применение технологий виртуальной реальности и hand-tracking для создания интерактивной обучающей среды.

Целью работы является разработка функционального VR-приложения для Oculus Quest 2, обеспечивающего возможность обучения дактильному алфавиту через практику и распознавание жестов.

Для достижения цели были изучены особенности дактильного языка, проанализированы существующие решения, сформулированы требования к системе, обоснован выбор технологий, а также реализована программная часть с возможностью калибровки и сравнения поз руки в формате JSON.

Практическая ценность работы заключается в создании доступного и адаптируемого VR-инструмента для самостоятельного обучения жестовому языку, полезного как для людей с нарушениями слуха, так и для слышащих пользователей.

THE ABSTRACT

Kudryashov A. V. Final qualification work on the topic: "Development of a VR application 'RSL Fingerspeller' for learning the Russian fingerspelling alphabet".

Object of research: the process of learning the Russian fingerspelling alphabet.

Subject of research: the use of virtual reality and hand-tracking technologies to create an interactive learning environment.

The purpose of the work is to develop a functional VR application for Oculus Quest 2 that enables training and gesture recognition for fingerspelling using real-time hand tracking.

To achieve this, the study explored the features of the fingerspelling alphabet, analyzed existing solutions, formulated system requirements, justified technology choices, and implemented the software part, including calibration and pose comparison using the JSON format.

The practical value of the work lies in providing an accessible and adaptable VR tool for independent learning of sign language, useful for both hearing-impaired and hearing users.

Федеральное государственное бюджетное образовательное учреждение высшего образования
«Саратовский государственный технический университет имени Гагарина Ю.А.»

РЕЦЕНЗИЯ

на выпускную квалификационную работу

Студента учебной группы м-ИВЧТ21 ИнПИТ
(группа) (факультет)

Кудряшова Алексея Владимировича
(фамилия, имя, отчество)

Рецензент Генеральный директор ООО «Спейс Ап»
(должность, ученая степень и звание)

Шух Николай Сергеевич

Выпускная квалификационная работа Кудряшова Алексея Владимировича посвящена разработке VR-приложения “RSL FingerSpeller” для обучения русскому дактильному алфавиту. Актуальность темы обусловлена развитием технологий виртуальной реальности и необходимостью создания доступных образовательных решений для людей с нарушениями слуха.

Работа представляет собой самостоятельное и завершённое исследование, сочетающее теоретическую базу с практической реализацией. Автор изучил особенности дактильного алфавита, провёл анализ существующих решений и сформулировал функциональные и технические требования к системе.

Структура работы логична: в первых главах обоснованы цель, задачи и выбор технологий, далее раскрыт процесс проектирования архитектуры и логики взаимодействия компонентов. Основной упор сделан на четвёртой главе, где подробно описаны этапы разработки: настройка Unity и Oculus Quest, реализация hand tracking, сохранение и сравнение жестов, работа с JSON, а также создание интерфейса и системы заданий.

Решения, реализованные автором, демонстрируют уверенное владение Unity, Meta SDK и OpenXR. Проект включает обучающий режим, проверку жестов, визуализацию ошибок, а также адаптивный интерфейс.

Результатом стало рабочее VR-приложение. Подготовлена заявка на регистрацию ПО, опубликована научная статья, планируется демонстрация в профильных учреждениях. Представляется, что автору удалось реализовать поставленные цели и задачи, а также раскрыть актуальную и социально

значимую тему. Вместе с тем, представленная выпускная квалификационная работа не лишена отдельных недостатков, на которые следовало бы обратить внимание при дальнейшей доработке проекта.

В работе можно было бы уделить больше внимания вопросам пользовательского взаимодействия, особенно с учётом особенностей целевой аудитории — людей с нарушениями слуха. В частности, полезным могло бы стать предварительное тестирование с представителями целевой группы, а также реализация более наглядной визуальной обратной связи при обучении жестам — например, через подсветку неправильно расположенных пальцев. Эти дополнения могли бы повысить удобство и эффективность освоения материала, однако их отсутствие не влияет существенно на общую ценность работы.

Следует подчеркнуть, что указанные замечания не снижают общей положительной оценки работы. Выпускная квалификационная работа Кудряшова Алексея Владимировича является самостоятельным, завершённым исследованием, выполненным на хорошем профессиональном и техническом уровне, демонстрирующим как теоретическую подготовленность автора, так и практические навыки разработки интерактивных VR-систем.

Выпускная квалификационная работа Кудряшова Алексея Владимировича является значимым вкладом в развитие отечественных образовательных VR-приложений. Все поставленные цели и задачи достигнуты. Работа может быть рекомендована к защите. Рекомендована оценка отлично

Генеральный директор ООО «Спейс Ап»_____ Шух Николай Сергеевич

Печать отдела кадров

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Информационные технологии стремительно развиваются, и задачи, которые ранее казались невозможными, становятся реализуемыми. Одним из таких направлений является виртуальная реальность (VR), которая активно развивается и внедряется в различные сферы, включая образование. Применение VR в обучающих системах — это актуальное направление, благодаря которому обучение становится более наглядным, интерактивным и эффективным.

Особенно перспективным использованием VR является обучение жестовому языку. Несмотря на наличие англоязычных обучающих систем виртуальной реальности, русскоязычные аналоги практически отсутствуют. Это создаёт значительный пробел, который может быть восполнен благодаря разработке инновационных решений с использованием VR.

Современные технологии виртуальной реальности предоставляют уникальные возможности для создания иммерсивной среды, позволяя моделировать реальные ситуации общения на жестовом языке. Это особенно важно в условиях ограниченного доступа к профессиональным сурдопереводчикам или специализированным курсам.

Русский жестовый язык (РЖЯ) является важным средством общения для людей с нарушениями слуха. Однако его изучение часто осложняется недостатком интерактивных образовательных инструментов, что ограничивает возможности обучения как для слышащих, так и для глухих пользователей. Преодоление этих барьеров имеет критическое значение для интеграции людей с нарушениями слуха в общество и повышения уровня их взаимодействия с окружающими.

Разработка VR-приложения для обучения русскому жестовому языку сталкивается с рядом вызовов. Среди них ключевыми являются создание системы распознавания жестов, способной интерпретировать сложные движения рук и пальцев в реальном времени, а также обеспечение

интуитивно понятного интерфейса, который сочетает теорию и практику обучения.

В данной научно-исследовательской работе рассматривается разработка VR-приложения для обучения русскому жестовому языку. Приложение будет включать функционал для изучения теории, практики жестов, их распознавания, а также выполнения интерактивных упражнений.

Целью данной работы является разработка полнофункционального VR-приложения RSL FingerSpeller, предназначенного для обучения дактильному русскому алфавиту с использованием hand-tracking и интерактивных упражнений.

Для достижения поставленной цели необходимо решить следующие задачи:

– Изучить современные подходы к реализации VR-приложений, включая Unity, OpenXR и Meta SDK.

– Исследовать особенности применения hand-tracking в образовательных системах.

– Спроектировать архитектуру приложения, включающую модули для калибровки, распознавания и проверки жестов.

– Разработать и реализовать пользовательский интерфейс с возможностью взаимодействия через Poke и Ray Interactions.

– Реализовать систему сохранения и сравнения жестов с использованием формата JSON.

Реализация этих задач позволит создать инновационное средство обучения русскому жестовому языку, полезное как для людей с нарушениями слуха, так и для слышащих пользователей, желающих освоить этот язык.

1 Дактильный алфавит

1.1 Описание дактильного алфавита

Дактильный алфавит представляет собой систему жестов, где каждая буква соответствует определённой позе руки (**Error! Reference source not found.**). Каждая буква имеет уникальное положение пальцев и кисти, что требует высокой точности при распознавании. В отличие от общего жестового языка, где жесты могут быть динамичными и контекстуальными, дактильный алфавит в основном включает статичные позы, что упрощает его использование в цифровых технологиях.

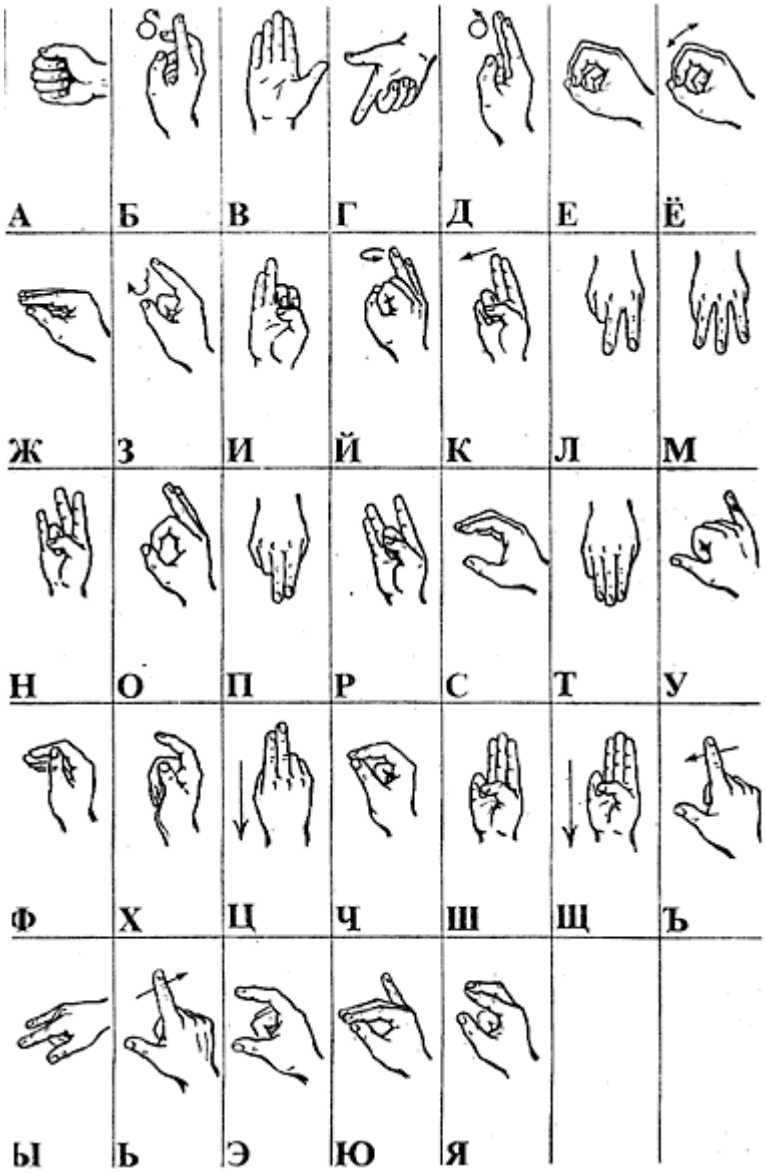


Рисунок 1 – Дактильный алфавит

Дактильный алфавит, как и жестовый язык в целом, развивался в различных культурах и странах, что привело к появлению разнообразных систем. В русском жестовом языке, например, дактильный алфавит был стандартизирован для обеспечения удобства в обучении и общении.

Этот алфавит активно используется среди глухих и слабослышащих людей, помогая преодолевать барьер между устной речью и жестовым языком. Однако для новичков или тех, кто не знаком с системой, распознавание форм может быть сложным. Отсутствие динамики и изменения позы делают его простым для программного распознавания, что становится важным аспектом в контексте разработки приложений для обучения дактильному алфавиту.

1.2 Использование дактильного алфавита

Дактильный язык широко используется среди слабослышащих и глухих людей как средство общения в различных сферах жизни. В образовательной практике дактильный алфавит помогает обучать глухих детей чтению, письму и произношению. Он является важным инструментом для преодоления языкового барьера между жестовым и устным языком.

Кроме того, дактильный алфавит используется в повседневных ситуациях, когда необходимо быстро передать информацию, особенно в шумных местах или в условиях, когда голосовая речь невозможна. Например, он активно используется для заимствования слов из других языков, а также для передачи личных имен, брендов и аббревиатур, для которых нет стандартных жестов.

С развитием технологий дактильный алфавит нашел широкое применение в цифровых системах, таких как приложения для перевода жестового языка, системы распознавания жестов и интерфейсы для слабослышащих пользователей. Статичность поз дактильного языка делает

его удобным для программного распознавания, что открывает новые возможности для улучшения коммуникации между людьми и машинами.

В отличие от общего жестового языка, где используются динамичные и контекстуальные жесты, дактильный язык более упрощен и стандартизирован, что делает его полезным инструментом как для обучения, так и для разработки новых технологий, включая приложения для виртуальной реальности и других интерактивных платформ.

1.3 Дактильный алфавит в цифровых технологиях

С развитием цифровых технологий дактильный алфавит стал важным инструментом для создания инновационных приложений, направленных на улучшение коммуникации для людей с нарушениями слуха. Одним из наиболее значимых аспектов является использование дактильного алфавита в системах распознавания жестов, где статичные формы рук значительно упрощают задачу алгоритмов компьютерного зрения и машинного обучения.

Дактиль активно применяется в приложениях для глухих и слабослышащих людей. Например, переводчики жестового языка и системы автоматического распознавания дактильного алфавита позволяют пользователям общаться с другими людьми или переводить текст в реальном времени. Это открывает новые возможности для улучшения качества жизни людей с нарушениями слуха, делая коммуникацию более доступной и удобной.

Кроме того, дактильный алфавит находит применение в виртуальной (VR) и дополненной реальности (AR). В таких системах дактильный алфавит используется для создания обучающих приложений, где пользователи могут изучать жестовый язык и улучшать свои навыки распознавания жестов в интерактивных виртуальных мирах. Такие приложения полезны как для обучающих целей, так и для расширения возможностей взаимодействия в виртуальных пространствах.

Особую роль дактильный алфавит играет в автоматизации и обучении. Современные виртуальные помощники и системы распознавания жестов, использующие дактильный язык, позволяют интегрировать жестовые команды в пользовательские интерфейсы, создавая более инклюзивные и доступные технологии.

1.4 Технические особенности цифрового моделирования дактильного алфавита

С технической точки зрения дактильный алфавит может быть представлен как набор поз руки, каждая из которых описывается положением и ориентацией суставов. Современные системы виртуальной и дополненной реальности (VR/AR) позволяют отслеживать до 30–35 суставов руки с высокой точностью, что делает возможным создание программ, способных распознавать конкретную букву по конфигурации пальцев.

Каждая поза может быть сохранена как набор координат (Vector3) и кватернионов (Quaternion), описывающих локальное положение и вращение каждого сустава относительно базовой кости (обычно запястья). Это позволяет хранить и сравнивать жесты с высокой точностью, задавая допустимый порог отклонения, при котором жест считается совпадающим с эталоном.

В системе трекинга, реализованной в Meta SDK (Oculus), используется перечисление HandJointId, в котором каждая кость (фаланга, сустав) руки имеет уникальный идентификатор. Таким образом, можно адресно получать и сохранять нужные позы, формируя базу жестов для последующего распознавания или обучения.

Цифровое моделирование дактильного алфавита требует также визуализации руки пользователя, что реализуется с помощью 3D-модели, синхронизированной с положением реальной руки. В VR-приложениях пользователь может не только увидеть свою руку, но и сравнить её с

эталонной визуальной моделью жеста, что делает обучение наглядным и эффективным.

Таким образом, дактильный алфавит в цифровых технологиях становится удобным объектом для анализа, хранения, распознавания и визуального сравнения, что делает его идеальной основой для разработки интерактивных обучающих приложений.

2 Особенности обучения и взаимодействия пользователя в виртуальной реальности

2.1 Обучение в виртуальной реальности

Использование технологий виртуальной реальности (VR) в образовательных целях открывает новые возможности для обучения, особенно в областях, связанных с моторными навыками и визуальной памятью. Одной из таких областей является изучение дактильного языка, где важна точность, концентрация внимания и практика в реальном времени.

Одним из ключевых преимуществ виртуальной реальности является полное погружение в обучающую среду, что позволяет пользователю сосредоточиться на процессе и минимизировать внешние отвлекающие факторы. Такой уровень концентрации способствует более эффективному восприятию и усвоению материала.

В VR-среде можно создавать контролируемые условия для обучения, где каждый пользователь может двигаться в собственном темпе, повторять упражнения неограниченное количество раз и получать индивидуальную обратную связь. Это особенно важно при обучении жестовым языкам, где отработка точных положений пальцев и кисти требует регулярной практики.

Ещё одним значимым преимуществом является возможность визуализации и интерактивного взаимодействия с материалом. Пользователь не просто смотрит на примеры жестов, но и активно участвует в процессе, пробуя воспроизвести позы в трёхмерном пространстве. Такой подход повышает мотивацию и вовлечённость, делая обучение более увлекательным.

Дополнительно, VR позволяет изолировать отдельные элементы обучения — например, сосредоточиться только на определённой группе букв или повторить сложные элементы. Это даёт гибкость в построении образовательной программы и позволяет адаптировать её под конкретные цели и уровень подготовки обучающегося.

2.2 Иммерсивность и мультимодальная обратная связь

Одним из важнейших факторов, определяющих эффективность обучения в виртуальной реальности, является уровень иммерсивности, то есть глубины погружения пользователя в цифровую среду. Чем выше степень погружения, тем сильнее эффект присутствия — ощущение, что пользователь действительно находится внутри обучающего пространства, а не просто наблюдает за ним извне. Это особенно важно при изучении дактильного языка, где необходима высокая концентрация внимания на положении кисти, конфигурации пальцев и точности выполнения жестов.

Иммерсивность способствует формированию мышечной памяти, поскольку пользователь взаимодействует с обучающим контентом не только визуально, но и физически — через движения рук, повороты головы, перемещения в пространстве. Такой подход обеспечивает лучшее запоминание и воспроизведение дактильных поз по сравнению с пассивным изучением на экране.

Немаловажную роль играет мультимодальная обратная связь — система откликов, которая помогает пользователю понять, насколько точно он воспроизвёл заданную позу. В виртуальной реальности обратная связь может быть:

– Визуальной — например, изменение цвета модели руки при правильном жесте или появление текста/символа;

– Аудиальной — звуковые сигналы подтверждения или подсказки;

– Тактильной (при наличии соответствующих устройств) — виброотклик при взаимодействии с объектами или при успешном выполнении задания.

Наличие мгновенной обратной связи повышает осознанность действий, помогает быстрее исправлять ошибки и мотивирует к дальнейшему

обучению. Кроме того, она создаёт эффект взаимодействия с живым преподавателем, что особенно ценно при самостоятельной работе.

2.3 Отслеживание и распознавание жестов руки

Точное и надёжное распознавание движений и положений рук — ключевой элемент при разработке приложений для изучения дактильного языка в виртуальной реальности. В отличие от традиционных методов ввода, таких как клавиатура, мышь или контроллеры, трекинг рук позволяет реализовать более естественное и интуитивное взаимодействие, что особенно важно при работе с жестовыми системами общения.

Дактильный язык основан на статичных конфигурациях кисти, и даже небольшое отклонение в положении пальцев может изменить смысл или сделать жест нераспознаваемым. Поэтому системы трекинга должны обеспечивать высокую точность определения положения суставов, ориентации ладони и состояния пальцев в реальном времени. Современные VR-платформы, такие как Oculus Quest или устройства на базе OpenXR, предоставляют встроенные инструменты для отслеживания рук с использованием камер и алгоритмов компьютерного зрения.

Использование трекинга рук позволяет:

– Освободить пользователя от дополнительных устройств, таких как контроллеры, делая обучение более естественным;

– Повысить точность обучения, так как система может сравнивать текущую позу руки с эталонной;

– Предоставлять визуальную обратную связь о правильности выполнения жеста, что облегчает самообучение;

– Собирать данные о типичных ошибках, что важно для анализа и адаптации образовательного процесса.

Однако в реализации трекинга рук существуют и определённые вызовы:

– Сложности с распознаванием жестов при перекрытии пальцев;

– Ошибки при плохом освещении или быстрой смене поз;

– Ограничения в точности отслеживания в зависимости от угла обзора камер.

2.4 Рекомендации к проектированию VR-среды для обучения дактильному языку

На основе анализа особенностей обучения в виртуальной реальности и трекинга рук можно выделить ряд рекомендаций, которые следует учитывать при проектировании обучающего VR-приложения для дактильного языка:

– Минималистичная визуальная среда

Интерфейс не должен перегружать пользователя информацией. Спокойный градиентный фон, простая сцена и ограниченное количество объектов позволяют сосредоточиться на жестах и форме руки.

– Центральная роль руки

В фокусе интерфейса всегда должна находиться виртуальная рука пользователя. Все задания, подсказки и взаимодействия должны располагаться в пределах поля зрения, не отвлекая от основной задачи — выполнения позы.

– Интерактивная обратная связь

Система должна сразу сообщать пользователю, насколько точно выполнен жест. Это может быть:

визуальный сигнал (изменение цвета руки или текста),

текстовая инструкция («попробуйте выпрямить палец»),

звуковой отклик (для подтверждения или коррекции).

– Калибровка под пользователя

Желательно реализовать возможность сохранения собственных эталонных поз. Это особенно важно для индивидуальных особенностей рук, таких как амплитуда движений, длина пальцев, положение запястья.

– Модульность обучения

Обучающий процесс стоит разбить на тематические блоки: буквы, слова, повторение, тестирование. Пользователь должен сам выбирать темп и последовательность.

– Использование анимаций и 3D-моделей эталонных поз

Для повышения наглядности можно выводить эталонную позу руки рядом с текущей или в режиме «призрачной руки», которую пользователь повторяет.

– Поддержка различных уровней сложности

Начинающий может тренироваться с большой погрешностью и подробными подсказками, опытный — с более строгими критериями распознавания.

Эти рекомендации легли в основу архитектурных решений, принятых в процессе разработки приложения, о чём будет подробно рассказано в следующих главах.

3 Анализ аналогов и обоснование архитектуры приложения

3.1 Анализ существующих решений

В последние годы наблюдается активное развитие приложений, направленных на поддержку людей с нарушениями слуха. В первую очередь, это касается систем распознавания жестов, переводчиков жестового языка, а также обучающих программ. Однако большинство подобных решений ориентированы на английский язык жестов (ASL) и не адаптированы под специфику русского дактильного алфавита.

Дактильный алфавит, в отличие от общего жестового языка, требует высокой точности в воспроизведении поз руки, так как каждая буква соответствует конкретной статичной конфигурации пальцев. Это делает дактиль особенно удобным для цифрового представления, но в то же время предъявляет высокие требования к системам отслеживания и обратной связи.

Современные технологии, такие как компьютерное зрение, нейросетевые распознаватели и устройства виртуальной реальности, дают возможность создавать эффективные обучающие среды.

Ниже приведено сравнение (Таблица 2) наиболее известных и технически значимых решений, связанных с обучением жестовому языку и распознаванием жестов.

Таблица 1 – Сравнение аналогов

Название (Платформа)	Тип приложения	Особенности	Недостатки
SignAll (PC + Kinect)	Desktop + камера	Автоматический перевод ASL (англ. язык жестов)	Отсутствие поддержки русского алфавита; высокая стоимость оборудования
SpreadTheSign (веб/моб)	Веб + мобильное	Огромная база видео-жестов в разных языках	Нет VR-погружения; нет проверки выполнения пользователем
Hand Talk (Android/iOS)	Мобильное	Анимированный аватар переводит голос в жесты	Только португальский язык; нет дактиля и обучения жестам

Продолжение таблицы таблицы 1

Название (Платформа)	Тип приложения	Особенности	Недостатки
Leap Motion демонстрации	VR + контроллеры	Высокая точность отслеживания пальцев	Не поддерживает автономные гарнитуры; отсутствует образовательный модуль
VR Sign Language (прототипы)	Исследования в VR	Показывают потенциал иммерсивного обучения	Редко реализованы для обучения дактильному алфавиту; нет готовых решений
ASL Fingerspeller	Обучающее VR приложение	Обучение дактильному языку в виртуальной реальности	Только английский язык

Вывод по анализу аналогов:

– Большинство решений ориентированы на английский язык и не учитывают специфику русского дактильного алфавита.

– Приложения с VR-погружением либо находятся на стадии исследований, либо не имеют обучающих и проверяющих модулей.

– Существующие мобильные решения не предоставляют интерактивную обратную связь и не позволяют пользователю практиковать дактиль в 3D-среде.

– Только отдельные инструменты позволяют отслеживать жесты, но не имеют полноценных систем обучения.

Это подтверждает необходимость разработки нового автономного VR-приложения, адаптированного под русский дактильный алфавит и использующего отслеживание рук и иммерсивные технологии.

3.2 Требования к системе

На основе анализа существующих решений и выявленных недостатков формируются требования к разрабатываемому VR-приложению для обучения дактильному русскому алфавиту. Эти требования делятся на

функциональные, нефункциональные, а также технические и интерфейсные, обеспечивающие стабильную и комфортную работу приложения в условиях виртуальной среды.

3.2.1 Функциональные требования

Большинство функциональных требований уже реализованы в текущей версии прототипа. В частности, сохранение эталонных жестов реализовано в виде системы калибровки, позволяющей пользователю перезаписать позу определённой буквы. Отображение виртуальной руки, переключение между сценами и выдача обратной связи выполнены средствами Unity с использованием Meta SDK и XR Interaction Toolkit. Эти функции формируют основу пользовательского взаимодействия и обеспечивают ключевой обучающий процесс.

– Отображение виртуальной руки пользователя.

Приложение должно корректно отображать трёхмерную модель руки, полностью синхронизированную с реальными движениями пользователя с использованием hand tracking.

– Сохранение эталонных жестов.

Пользователь должен иметь возможность сохранить определённую позу руки как эталонную для дальнейшего сравнения. Реализуется в виде калибровки определённого жеста под пользователя

– Сравнение текущего жеста с эталоном.

Система должна сравнивать текущую позу руки пользователя с сохранённой и оценивать точность совпадения по положению и ориентации суставов.

– Предоставление обратной связи.

Приложение должно выдавать визуальную и/или текстовую обратную связь о том, насколько точно пользователь воспроизвёл жест.

– Интерактивный пользовательский интерфейс.

Интерфейс должен предоставлять функции:

– сохранения и загрузки жестов,

– перехода к следующему заданию,

– управления параметрами отображения.

Вся навигация по сценам и интерфейс обеспечивается логикой Unity и XR Interaction Toolkit.

– Поддержка нескольких жестов (букв).

Пользователь должен иметь возможность переходить от одной буквы к другой и тренироваться на отдельных элементах алфавита.

3.2.2 Нефункциональные требования

Указанные нефункциональные требования были определены с учётом целевой аудитории — пользователей, не имеющих специальной технической подготовки. Простота интерфейса, автономность работы на Oculus Quest 2 и стабильность в условиях ограниченных вычислительных ресурсов были приоритетами при проектировании. Приложение должно оставаться доступным и понятным даже для тех, кто впервые использует VR-гарнитуру.

– Автономная работа на устройстве Oculus Quest 2.

Приложение должно функционировать без необходимости подключения к ПК.

– Интуитивность интерфейса.

Взаимодействие с системой должно быть понятным даже для пользователей без VR-опыта.

– Производительность.

Приложение должно обеспечивать стабильную работу и низкую задержку отображения рук и выдачи обратной связи.

– Возможность масштабирования.

Архитектура должна предусматривать возможность добавления новых модулей: обучение словам, игровые режимы, продвинутая проверка.

– Минималистичный визуальный стиль.

Интерфейс и сцена должны быть выполнены в нейтральных тонах с минимумом отвлекающих объектов, чтобы пользователь концентрировался на позах руки.

3.2.3 Технические и интерфейсные требования

Эти требования связаны с качеством пользовательского опыта и гибкостью интерфейса. Возможность отображать эталонную позу рядом с текущей, работа без контроллеров, сохранение данных и использование понятного языка повышают удобство обучения. Эти элементы также обеспечивают адаптацию интерфейса под разные группы пользователей, включая детей, преподавателей и людей с особенностями восприятия.

– Поддержка hand tracking без использования контроллеров.

– Сохранение данных жестов в файл или внутреннюю память.

– Возможность отображения эталонной позы рядом или в виде “призрачной руки”.

– Поддержка языкового интерфейса (текстовые подсказки на русском языке).

– Поддержка смены моделей руки (например, правая/левая, скин).

Соблюдение этих требований позволит создать эффективную обучающую систему, обеспечивающую иммерсивное взаимодействие, гибкую настройку и точное сравнение жестов пользователя с эталоном.

3.3 Обоснование выбора технологий

Успешная реализация VR-приложения требует выбора подходящего оборудования и программных средств. Эти решения напрямую влияют на производительность, удобство для пользователя, возможности масштабирования и точность распознавания жестов. В данной главе рассматриваются ключевые технологические компоненты, использованные в проекте, а также приводится сравнение возможных альтернатив.

3.3.1 Выбор устройства виртуальной реальности

Разработка приложения для обучения русскому жестовому языку требует инновационных решений, способных обеспечить точное распознавание жестов и удобное обучение. Выбор технологий основывается на необходимости создать интуитивно понятный и иммерсивный интерфейс, а также обеспечить высокую точность распознавания движений рук.

Устройство VR-гарнитуры определяет уровень погружения пользователя, точность отслеживания движений и доступность проекта в целом. Сравнение наиболее распространённых гарнитур позволило выбрать оптимальное решение для образовательного VR-приложения. (Таблица 2 – Сравнение VR гарнитур

Таблица 2 – Сравнение VR гарнитур

Критерий	Oculus Quest 2	Oculus Quest 3	Valve Index
Автономность	Работает без подключения к ПК	Работает без подключения к ПК	Требуется ПК
Hand tracking	Встроенная поддержка	Встроенная поддержка	Только через контроллеры
Поддержка Android	Да	Да	Нет
Качество дисплея	Среднее	Выше среднего	Высокое
Производительность	Достаточная для учебных целей	Улучшенная	Очень высокая, но требует ПК
Стоимость	Средняя	Высокая	Очень высокая
Применимость в обучении	Используется в учебных проектах	Менее распространена	Ограничена из-за привязки к ПК

Oculus Quest 2 был выбран в качестве основной VR-гарнитуры для разработки приложения благодаря сочетанию автономности, доступной стоимости, стабильной поддержки hand tracking и совместимости с платформой Android. В отличие от более дорогих или зависимых от ПК устройств, таких как Oculus Quest 3 и Valve Index, данное решение обеспечивает низкий порог входа для конечного пользователя и простоту в установке и эксплуатации. Устройство активно применяется в образовательных проектах по всему миру, что подтверждает его практическую применимость и актуальность.

3.3.2 Выбор программной платформы

Следующим важным этапом стало определение формата, в котором будет развёрнуто приложение. Рассматривались два варианта: автономный Android APK и ПК-версия. Ключевыми факторами стали доступность, производительность, возможности установки и дальнейшее распространение. (Таблица 3)

Таблица 3 – Сравнение Android APK и ПК-приложений

Критерий	Android APK	ПК-приложение
Автономность	Не требует подключения к компьютеру	Зависит от ПК
Простота установки	Установка через ADB, быстрая	Сложнее, требует установку драйверов и ПО
Производительность	Средняя, ограничена ресурсами гарнитуры	Высокая
Возможности графики	Умеренные, оптимизированные	Широкие, включая фотореализм
Распространение	Простое (через .apk-файл, SideQuest)	Ограниченное, не все пользователи имеют ПК
Масштабируемость	Возможность портирования на другие Android-гарнитуры	Ограничено ПК
Удобство для пользователя	Подходит для самостоятельного обучения	Требуется стационарное рабочее место

Платформа Android в формате APK выбрана как основная для развёртывания VR-приложения, поскольку она обеспечивает автономную

работу на гарнитуре без необходимости подключения к внешнему оборудованию. Это делает приложение доступным для широкой аудитории, упрощает распространение (через SideQuest и подобные платформы), а также снижает требования к пользователю. Несмотря на некоторые ограничения в производительности по сравнению с ПК, формат APK полностью соответствует функциональным и техническим целям проекта, особенно на этапе обучения дактильному алфавиту.

3.3.3 Выбор среды разработки

Для реализации логики приложения, 3D-сцен, пользовательского интерфейса и взаимодействия с hand tracking необходимо было выбрать движок разработки. Рассматривались два популярных варианта: Unity и Unreal Engine 5. (Таблица 4)

Таблица 4 – Сравнение среды разработки

Критерий	Unity Meta SDK	Unreal Engine 5
Простота изучения	Понятный интерфейс, обширная документация	Более сложный порог вхождения
Поддержка Oculus Quest	Официальная поддержка, шаблоны	Требует ручной настройки
Hand Tracking	Через Meta Interaction SDK	Ограниченная поддержка
Оптимизация под APK	Легко адаптируется	Требует дополнительных усилий
Графические возможности	Средние, но достаточные для учебного проекта	Высокие, фотореализм
Язык разработки	C#	C++
Сообщество	Очень большое, множество примеров	Меньше обучающих материалов по VR
Производительность	Достаточная	Высокая, но затратная для автономных устройств

Unity в связке с Meta SDK выбран в качестве основного движка разработки благодаря его простоте интеграции с устройствами Oculus Quest, встроенной поддержке отслеживания рук и широким возможностям для создания интерактивных трёхмерных сцен. По сравнению с Unreal Engine 5, Unity предлагает более низкий порог входа, обширное сообщество и лучше адаптирован для автономных гарнитур. При этом функциональности движка

достаточно для реализации всех запланированных элементов обучающего VR-приложения, включая систему жестов, пользовательский интерфейс и взаимодействие в виртуальной среде.

3.3.4 Дополнительные технологии и перспективы

Для повышения точности распознавания жестов и расширения функциональности VR-приложения в будущем планируется интеграция дополнительных технологий. На данный момент реализация ограничивается сохранением и сравнением жестов в виде статичных поз, однако проект закладывает основу для внедрения более интеллектуальных решений.

В качестве перспективного направления рассматривается использование нейросетевых библиотек, таких как MediaPipe от Google и TensorFlow Lite. Эти инструменты позволяют реализовать анализ видео- или 3D-потока рук в реальном времени и автоматически определять, какая буква дактильного алфавита показана. В частности, такие модели могут обучаться на базе эталонных жестов, хранящихся в структурированном формате.

Хранение поз руки осуществляется в виде JSON-файлов, где для каждого сустава сохраняется положение и ориентация. Это позволяет легко сохранять, загружать и редактировать эталонные жесты, а также использовать их в обучении модели.

Также возможно визуальное отображение эталонной позы в VR-пространстве в виде «призрачной руки» или полупрозрачного силуэта, с которым пользователь может сверяться во время выполнения. Это усилит эффект обучения и обеспечит наглядную обратную связь.

Таким образом, архитектура приложения изначально проектируется так, чтобы легко масштабироваться в будущем — как в сторону добавления новых уровней сложности и обучающих режимов, так и в сторону интеллектуального распознавания.

3.3.5 Актуальность и обоснованность выбранных решений

Выбор технологий, осуществлённый в рамках проекта, направлен на достижение оптимального баланса между функциональностью, доступностью и возможностью дальнейшего развития. Использование Unity в связке с Meta SDK и OpenXR позволяет эффективно реализовать hand tracking и взаимодействие в VR-пространстве, минимизируя сложность разработки.

Oculus Quest 2 выбран как основное аппаратное решение благодаря своей автономности, поддержке hand tracking, доступной цене и широкому распространению в образовательной и исследовательской среде. Применение Android APK в качестве формата развёртывания обеспечило простоту распространения и автономную работу без необходимости в мощном оборудовании.

Всё это делает выбранный стек технологий полностью обоснованным для задач обучения дактильному алфавиту. Проект не только учитывает текущие технические возможности, но и предусматривает задел для расширения — за счёт интеграции нейросетей, визуальных подсказок и игровых элементов в обучении.

Таким образом, архитектурные и технологические решения обеспечивают устойчивую основу для создания эффективного, масштабируемого и доступного образовательного VR-продукта.

4. Разработка

4.1 Подготовка Unity

Для разработки приложения будет использоваться Unity. Скачать Unity можно бесплатно с их официального сайта. После скачивания и установки на рабочем столе появляется Unity Hub (Рисунок 2), который позволяет управлять установленными версиями редактора Unity, создавать новые проекты и использовать уже готовые.

Рисунок 2 – Unity Hub

Для начала требуется установить редактор, для разработки будет использоваться последняя версия редактора – Unity 6(6000.0.41f1). Во вкладке Installs (Рисунок 3) необходимо установить рекомендованную версию.

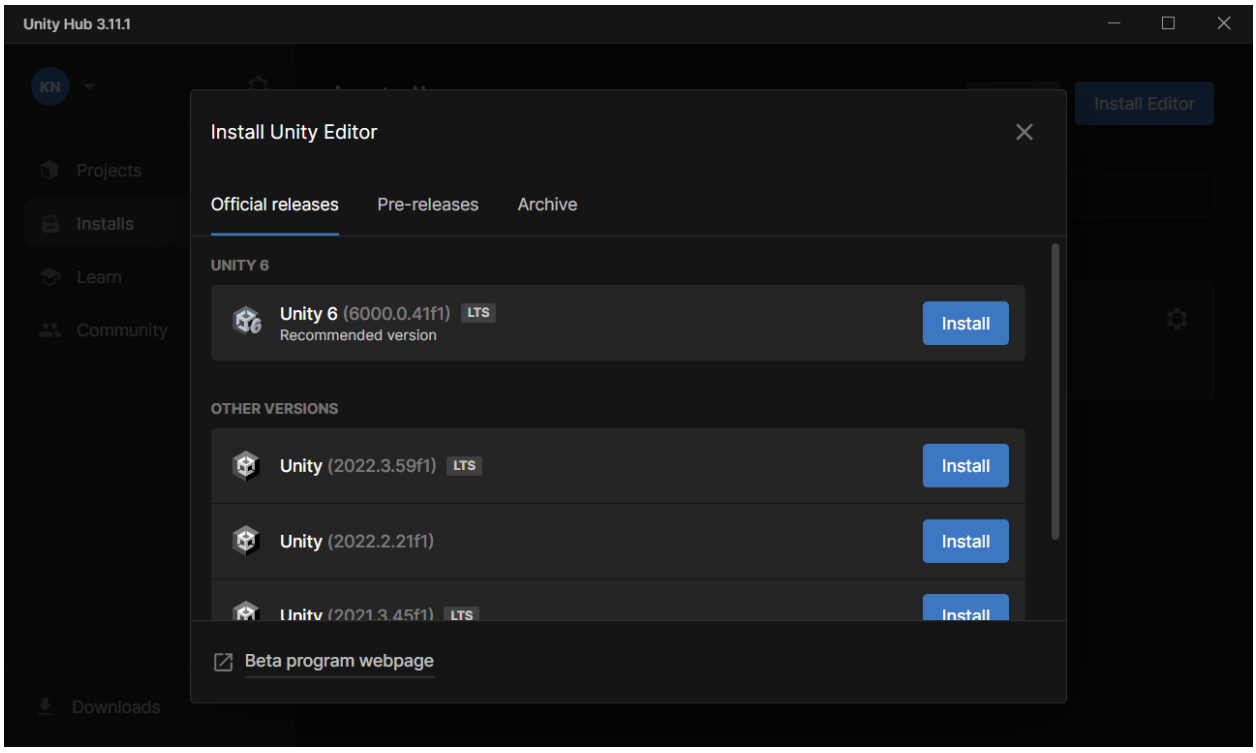


Рисунок 3 – Установка редактора

В установленный редактор следует добавить модули (Рисунок 4), так как разработка ведется для android устройства.

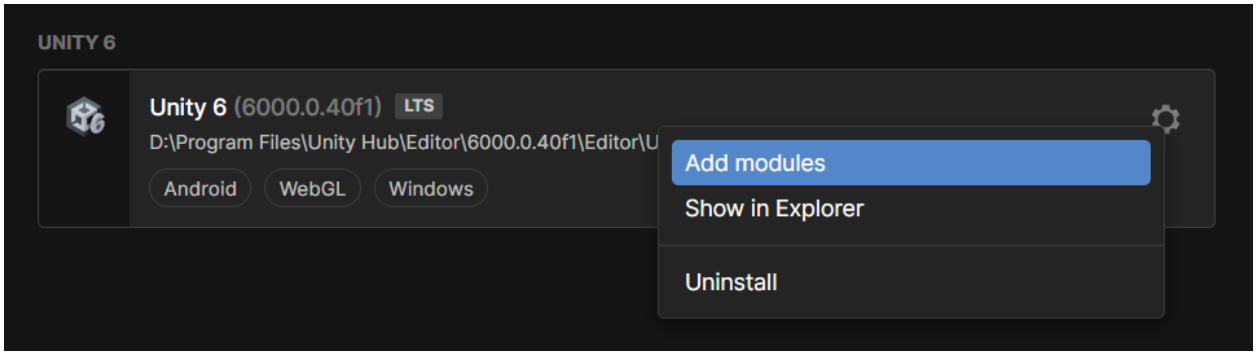


Рисунок 4 – Добавление модулей

Следует найти вкладку Platforms (Платформы) и добавить поддержку android – Android Build Support (Рисунок 5)

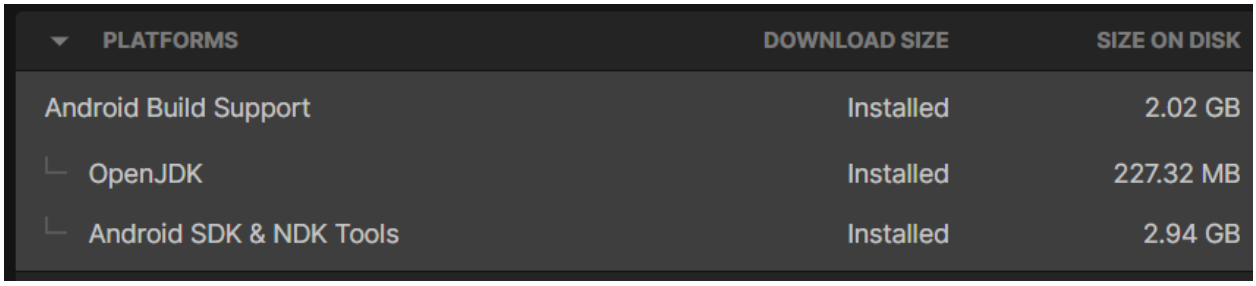


Рисунок 5 – Добавление Android Build Support

Первоначальная настройка Unity редактора для создания приложения для распознавания русского жестового языка в виртуальной реальности завершена. Перейдем к созданию и настройке проекта.

4.1.1 Создание и настройка проекта

Во вкладке Projects (Проекты) создадим новый проект. В ходе выбора ядра приложения был выбор между VR Core, который может использоваться как шаблон для любых VR проектов и Universal 3D, он же URP – пустой проект с возможностью. Для этого была составлена таблица сравнения этих ядер и выбрано наиболее подходящее для моей работы (Таблица 5 – Сравнение VR Core и 3D Universal в Unity)

Таблица 5 – Сравнение VR Core и 3D Universal в Unity

Характеристика	VR Core	3D Universal (URP)
Основное назначение	Готовый шаблон для VR-проектов	Гибкий шаблон для 3D-графики с Universal Render Pipeline (URP)
Поддержка VR	Включает настройки VR по умолчанию	Требуется вручную добавить OpenXR / Meta SDK
Графический рендеринг	Стандартный рендерер (Built-in)	URP – более производительный для мобильных VR-устройств
Готовые сцены и настройки	Включает пресеты камеры, рук, телепортации	Чистый проект без готовых VR-настроек
Производительность	Оптимизирован под VR, но использует Built-in Render Pipeline	URP даёт больше контроля и работает быстрее на слабых устройствах
Гибкость	Подходит для быстрого старта VR-разработки	Лучше для кастомных VR-проектов с высокой оптимизацией

Таким образом был выбрано URP ядро, которая позволяет гибко настроить будущий проект, использовать те модули, которые будут необходимы: нужна разработка только для Oculus Quest разработки, VR Core предоставляет разработку для всех систем, модули, которые могут не понадобиться. Также URP предоставляет высокую производительность для Android систем, таких как Quest 2.

Выбираем Universal 3D Core (Рисунок 6), настраиваем проект: название, расположение проекта, подключаем облако для контроля версий. После того, как всё настроено, можно создать проект.

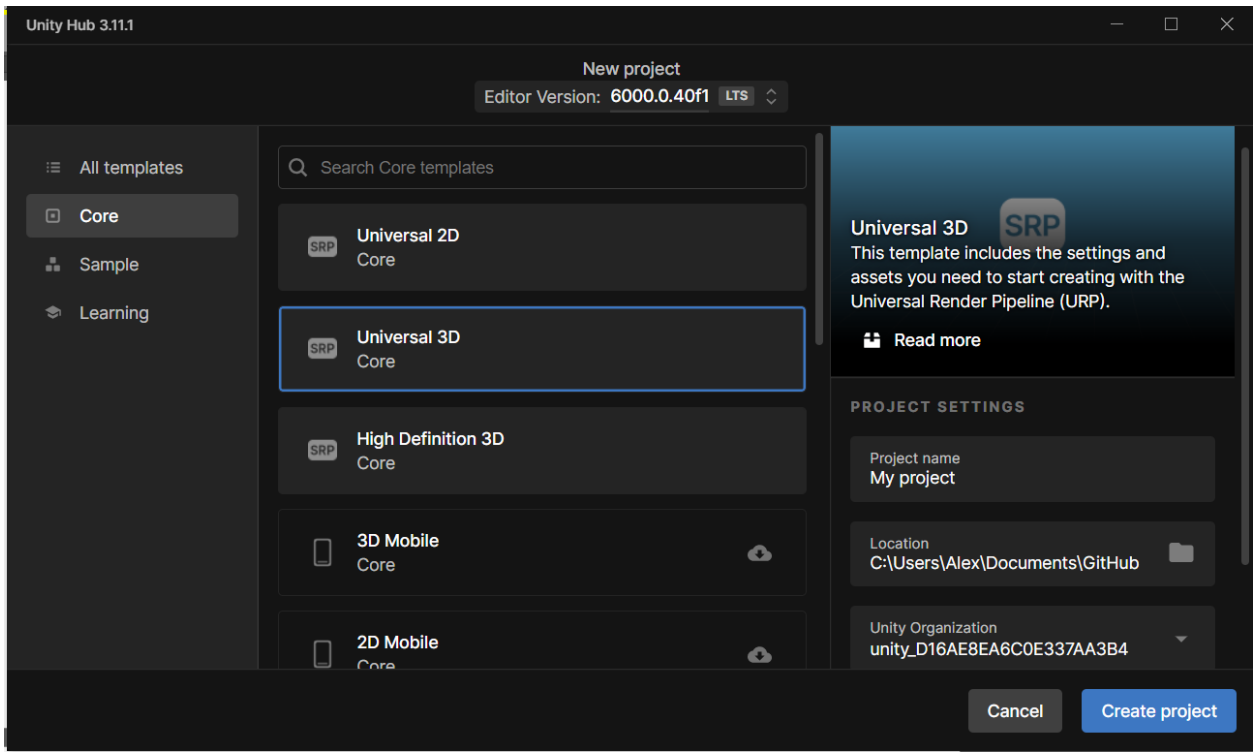


Рисунок 6 – Выбор ядра

Теперь созданный проект появится во вкладке Projects (Рисунок 7).

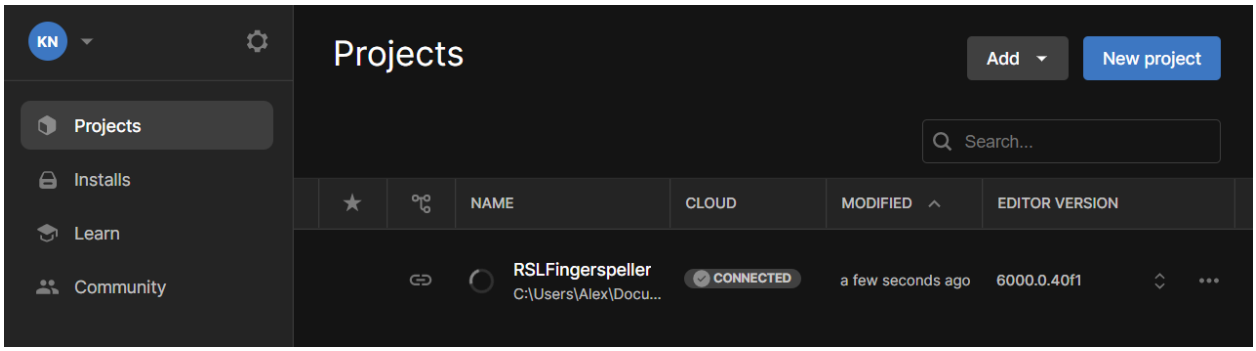


Рисунок 7 – Созданный проект

На данном этапе произведено создание и настройка проекта в Unity Hub. Дальнейшая настройка будет производится в уже запущенном проекте.

4.1.2 Подготовка проекта к виртуальной реальности

Запущенный проект URP представляет собой проект с несколькими окнами, объединенными в один интерфейс (Рисунок 8)

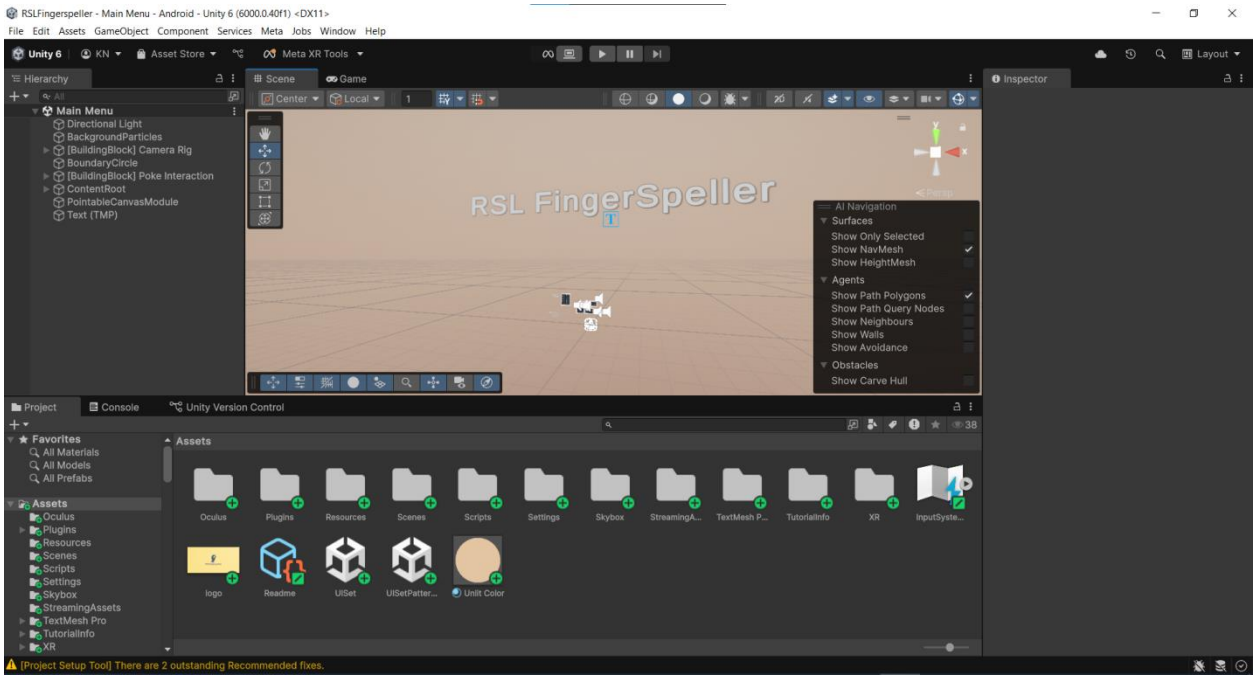


Рисунок 8 – Интерфейс проекта

Слева находится дерево сцены – элементы, которые находятся на текущей сцене в их иерархии. Посередине расположено графическое окно сцены – графический редактор. Снизу расположены ассеты, которые доступны для добавления на сцену. Для удобства их можно располагать в соответствующих папках. Сверху расположена панель инструментов.

Для начала следует подготовить проект для VR, так как при создании URP создаётся пустой проект.

В магазине ассетов для Unity (Рисунок 9) бесплатно приобретаем Meta XR All-in-One SDK, который включает в себя все модули, которая сделала компания Meta для разработчиков приложений.

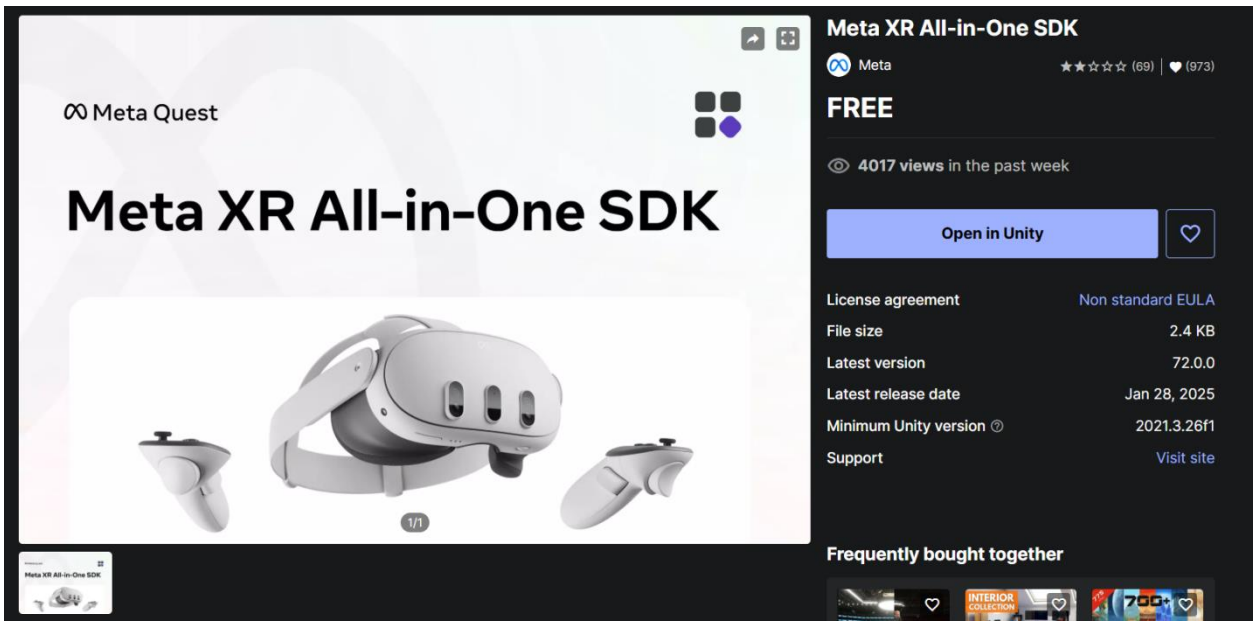


Рисунок 9 – Meta XR SDK

Для установки SDK в проекте открываем менеджер ассетов, и устанавливаем в проекте Meta XR All-in-One SDK (Рисунок 10)

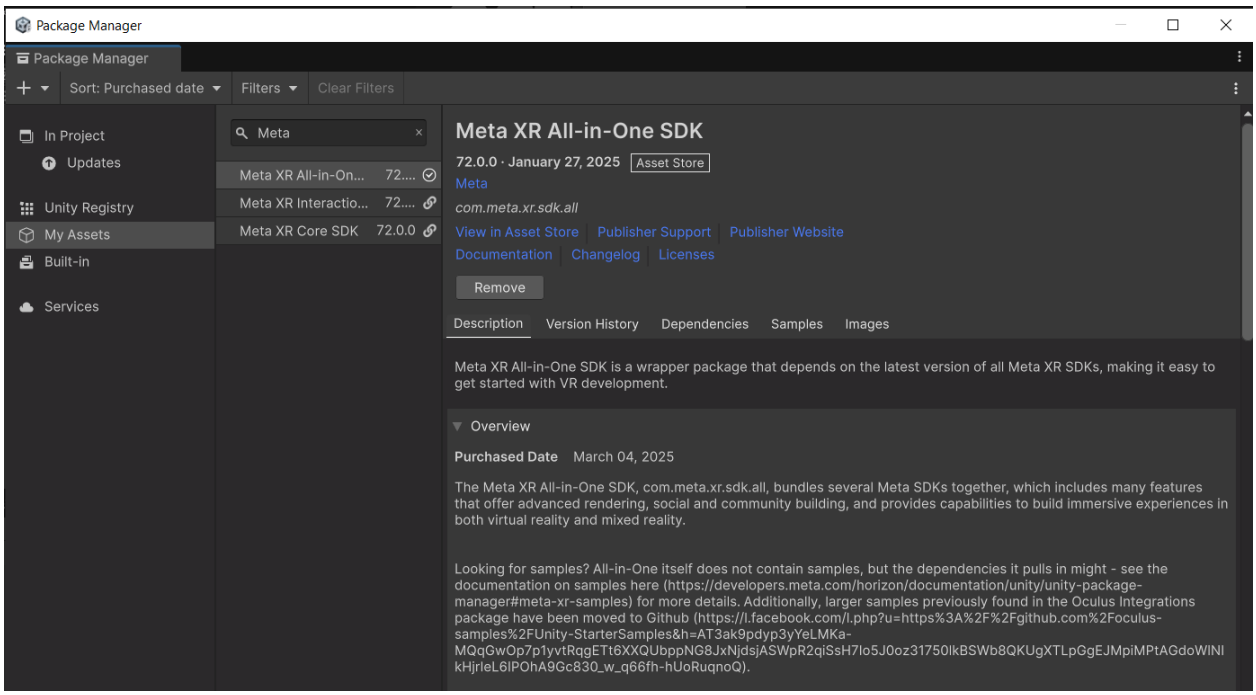


Рисунок 10 – Менеджер пакетов

После установки в настройках проекта (Рисунок 11) станет доступно XR Plug-In Manager, он позволяет настроить, для каких систем будет разрабатываться приложение. В этой вкладке следует переключиться на Android вкладку и включить Oculus провайдера подключения.

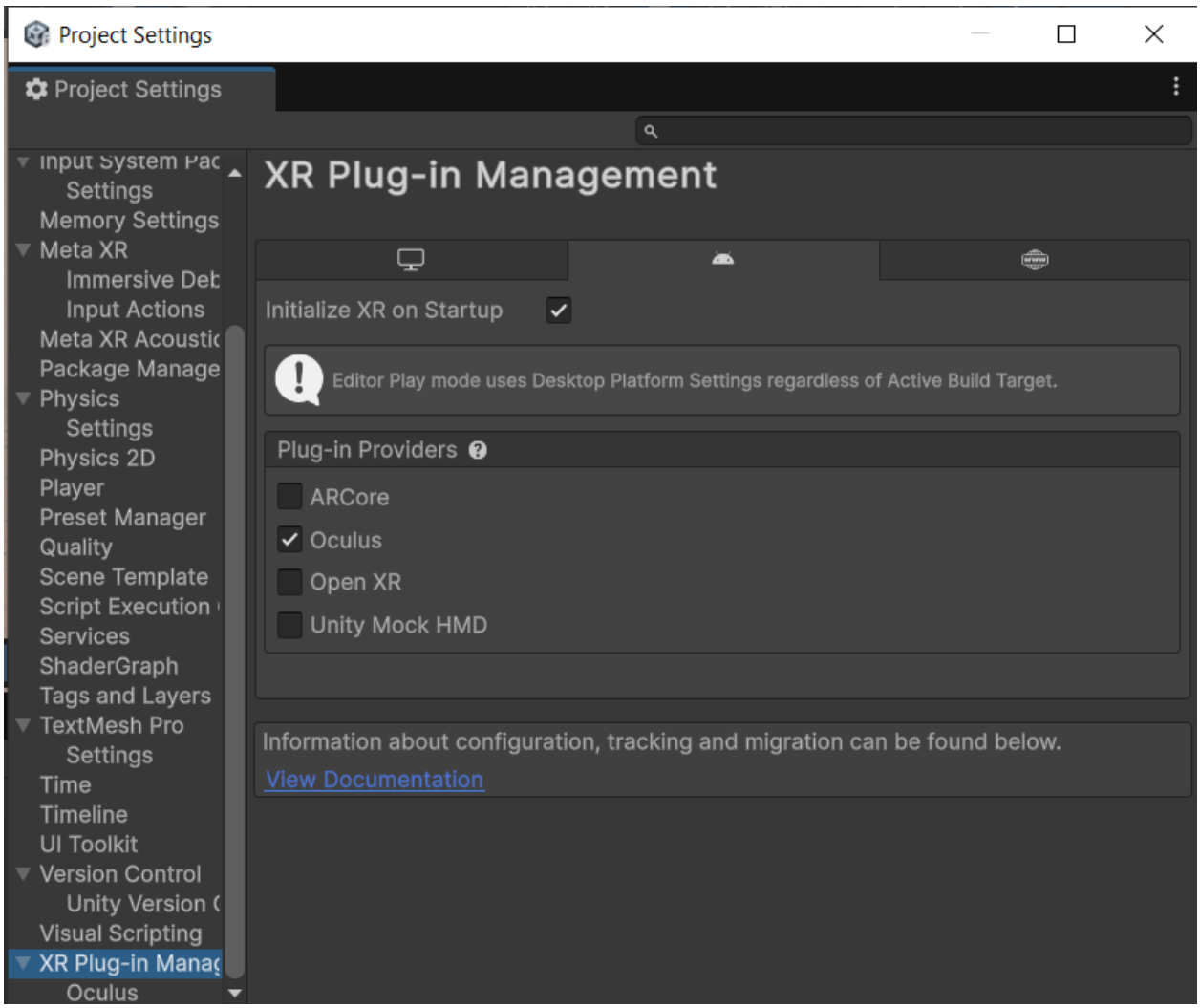


Рисунок 11 – Настройки проекта

Для оптимизации проекта и установки обязательных и рекомендуемых исправлений в этом же окне нужно выбрать Meta XR (Рисунок 12). Инструмент настройки проекта Unity оптимизирует настройки проекта Android для приложений Meta Quest Unity, в том числе настройки текстур и графики. Он применяет необходимые настройки для создания приложений Meta Quest XR, в том числе настройку минимальной версии API и использования ARM64. Для применения оптимизации и подготовки проекта для VR нажать на Fix All и Apply All.

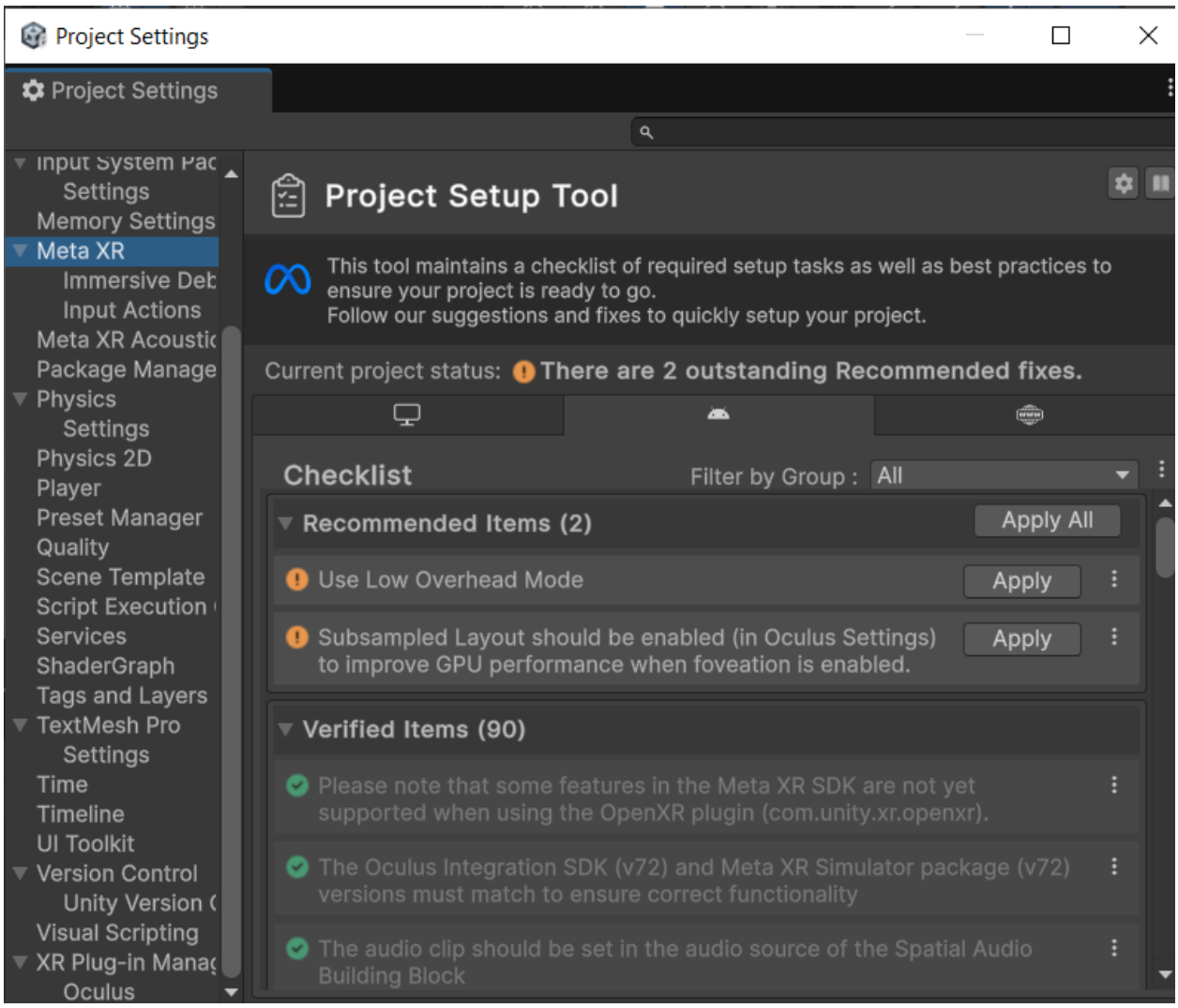


Рисунок 12 – Meta XR оптимизация

На текущем этапе проект полностью готов для разработки приложения для разработки приложения для изучения русского жестового языка в виртуальной реальности.

4.2 Подготовка Oculus Quest 2

Для разработки и тестирования VR-приложений на устройствах Meta Quest необходимо перевести гарнитуру в режим разработчика. Это позволит устанавливать собственные приложения, использовать инструменты отладки и подключать гарнитуру к Unity для тестирования.

– Включение режима разработчика

Переключение гарнитуры Meta Quest в режим разработчика выполняется через официальное мобильное приложение Meta Horizon:

– Открыть приложение Meta Horizon на смартфоне.

– Перейти в Menu (Меню) > Devices (Устройства) и выбрать свою гарнитуру (Рисунок 13).

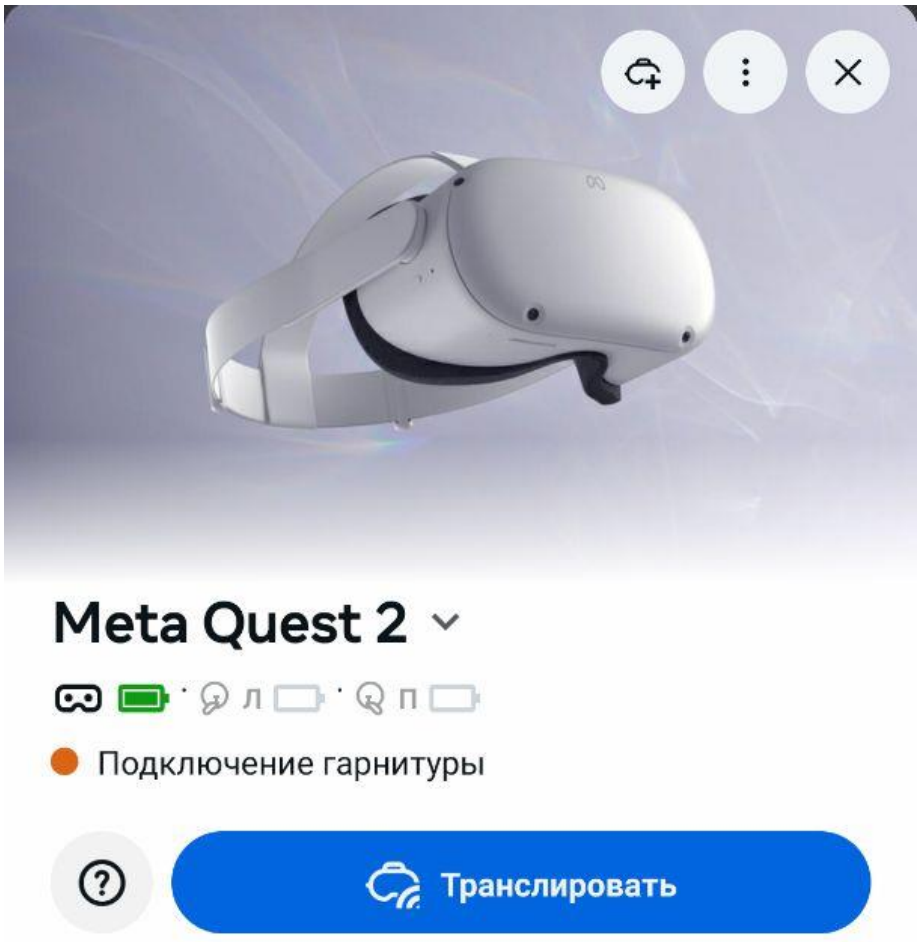


Рисунок 13 – Настройка гарнитуры

– Открыть Headset Settings (Настройки гарнитуры) > Developer Mode (Режим разработчика).

– Включить Developer Mode (Режим разработчика) (Рисунок 14).

Рисунок 14 – Режим разработчика

– Подключение гарнитуры к компьютеру

После активации режима разработчика необходимо подключить устройство к ПК:

– Подключить гарнитуру к компьютеру с помощью USB-C кабеля.

– Надеть гарнитуру и убедиться, что система обнаружила подключение.

– Перейти в Settings (Настройки) > Advanced (Расширенные) > Developer (Разработчик).

– Включить параметры:

– Enable custom settings (Включить пользовательские настройки)

– MTP Notification (Уведомление MTP)

– Когда появится запрос на разрешение отладки по USB, выбрать Always allow from this computer (Всегда разрешать с этого компьютера).

После этих шагов гарнитура готова к разработке и тестированию VR-приложений.

Для проверки того, что всё действительно работает, в настройках сборки в Unity в настройке устройства для записи станет доступен шлем (Рисунок 15).

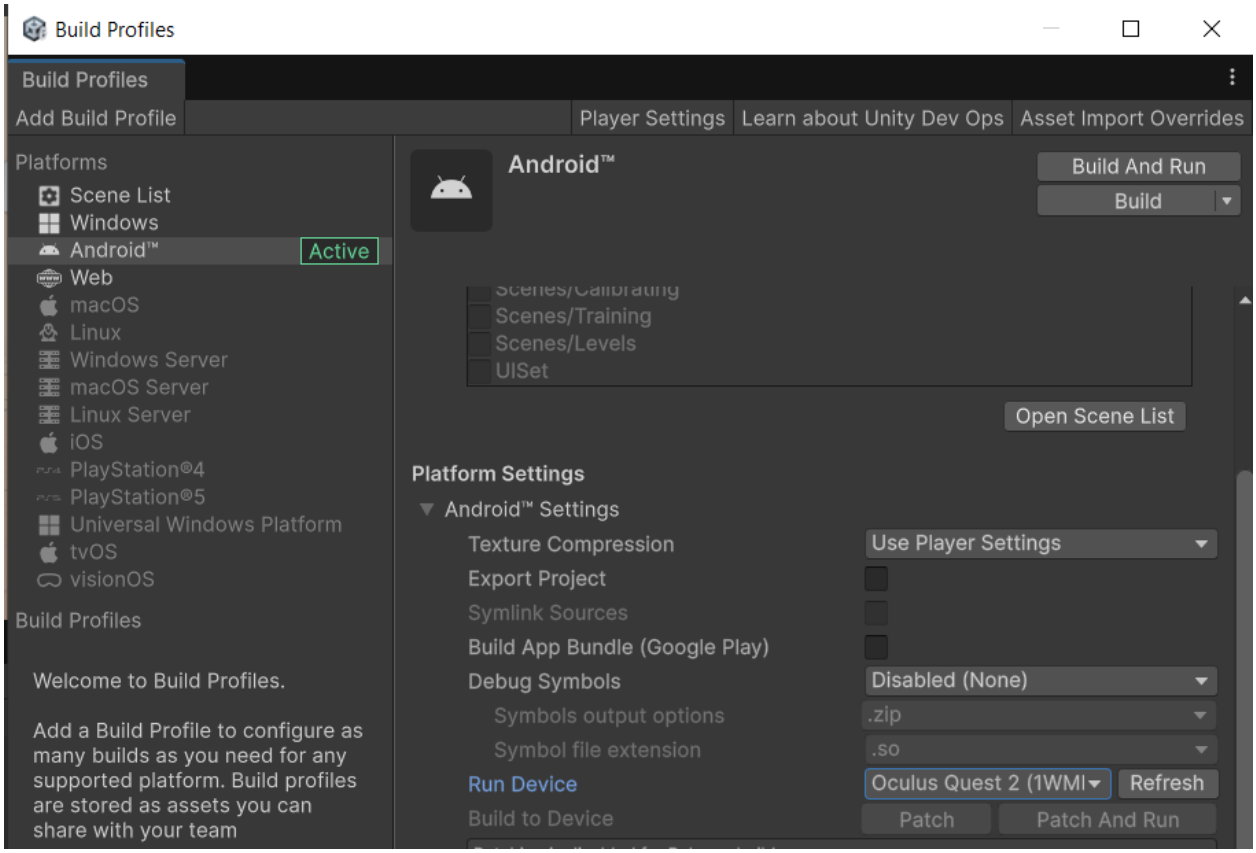


Рисунок 15 - Настройка сборки

На текущем этапе шлем полностью готов для тестирования приложений, разработанных в Unity.

4.3 Структура приложения RSL Fingerspeller

Приложение RSL Fingerspeller реализовано в виде модульной архитектуры, где каждый основной этап взаимодействия пользователя вынесен в отдельную сцену. Это позволяет не только упростить разработку и отладку, но и масштабировать проект в будущем: добавлять новые уровни сложности, режимы или расширенные обучающие модули.

Все сцены имеют общее визуальное оформление: градиентный фон, минималистичный интерфейс и частички, создающие эффект живого пространства. В каждой сцене (Рисунок 16) настроено управление с использованием камеры гарнитуры и отслеживания рук (hand tracking), реализованного через Meta SDK с поддержкой OpenXR.

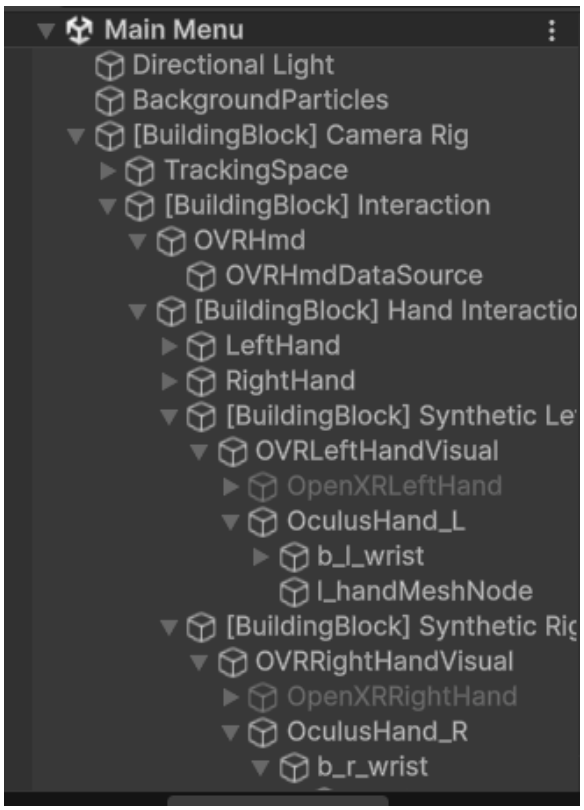


Рисунок 16 – Настроенное управление в иерархии сцены

Для визуализации рабочей области в сцене скриптом отрисован круг, обозначающий зону, в которой пользователь должен показывать жесты. Это упрощает распознавание и повышает точность анализа поз руки.

4.3.1 Общая структура и подход к реализации

Приложение организовано по следующим принципам:

— Разделение функциональности по сценам: каждая сцена отвечает за строго определённый блок — калибровка, обучение, меню и т. д.

— Модульность логики: каждый крупный функционал реализован отдельным скриптом. Общие методы вынесены в вспомогательные классы.

— Работа с данными в JSON: сохранение эталонных жестов происходит в отдельный JSON-файл, который копируется при запуске и используется как временная база. Это позволяет редактировать его, не повреждая исходные эталоны.

— Гибкость взаимодействия с пользователем: во всех сценах реализована возможность управления интерфейсом как с помощью Poke, так и с помощью Ray Interactions.

4.3.2 Состав сцен и их назначение

Каждая сцена в проекте (Таблица 6). выполняет конкретную задачу и содержит свой собственный скрипт, который управляет её логикой и поведением

Взаимодействие пользователя с интерфейсом, включая кнопки и другие интерактивные элементы, происходит двумя основными способами:

– Физические касания (Poke) – когда игрок напрямую касается объекта пальцем или контроллером (Рисунок 17).

Рисунок 17 – Нажатие на кнопки

– Наведение луча (Ray) – когда пользователь направляет виртуальный луч (например, с помощью лазерного указателя от контроллера) на элемент интерфейса (Рисунок 18).

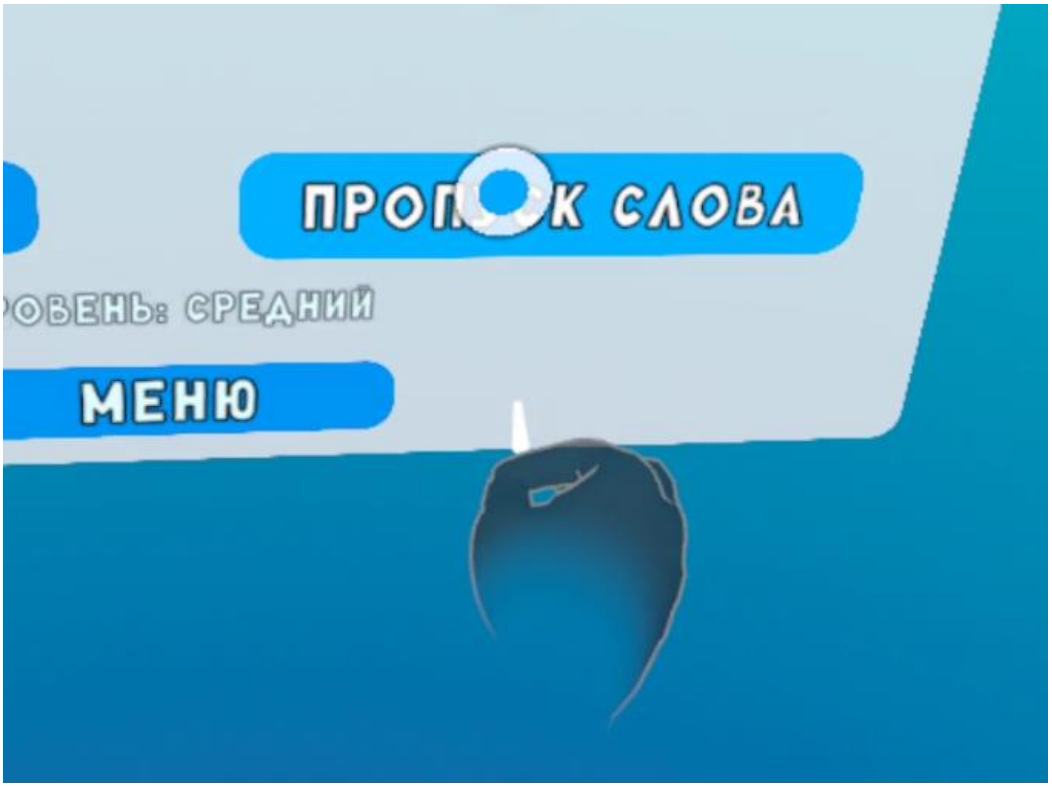


Рисунок 18 – Возможность управления лучом

Оба метода обеспечивают интуитивно понятное управление, адаптированное под разные сценарии использования в VR/AR-среде.

Таблица 6 – Сцены и их назначение

Сцена	Назначение	Основные действия	Интерфейс и взаимодействие
Меню	Главная сцена приложения	Выбор режима: тренировка, калибровка, уровень сложности	Кнопи выбора сцены
Калибровка	Сохранение и сравнение эталонных жестов	Переключение букв, сохранение текущей позы, сравнение с эталоном	Текст текущей буквы, кнопки
Выбор уровня	Выбор сложности для тренировки слов	Нажатие кнопки «лёгкий», «средний», «тяжёлый»	Три кнопки выбора уровня
Уровень	Режим практики: показать слово жестами по буквам	Ввод слов жестами, переход между словами, отслеживание времени	Таймер, текст с прогрессом слова
Тренировка	Свободное распознавание дактильных жестов в реальном времени	Демонстрация произвольных жестов, визуальное подтверждение	Вывод распознанной буквы

4.3.3 Работа с позами и сценами

Каждая сцена имеет доступ к общей структуре хранения и обработки жестов. Система позволяет:

- переключаться между буквами алфавита (в калибровке);
- сохранять текущую позу руки в файл;
- сравнивать текущую позу с сохранённой;

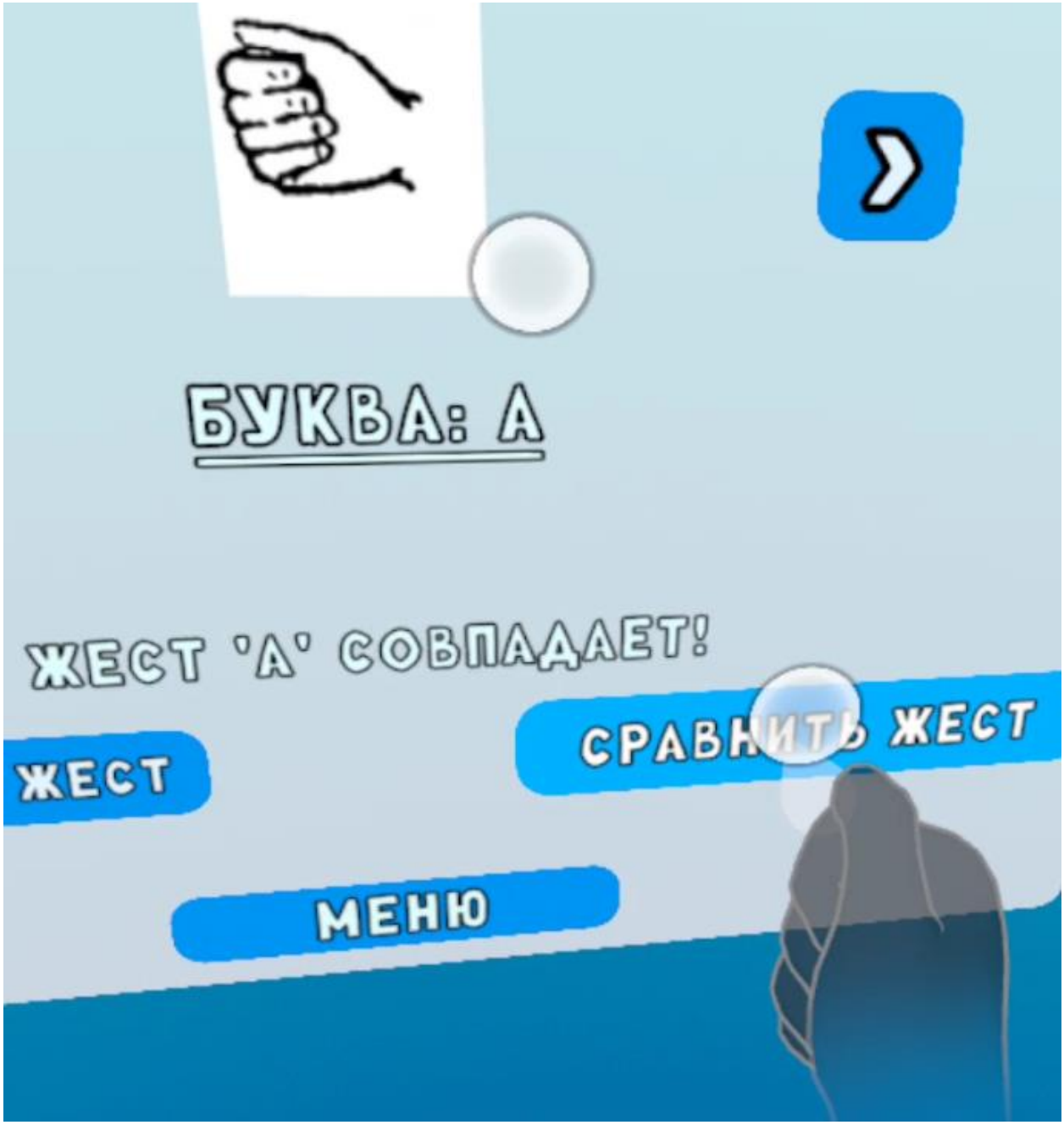


Рисунок 19 – Сравнение жеста с сохранённым

- отображать распознанные буквы (в сцене тренировки и уровнях);

– загружать JSON-файл с сохранёнными позами при запуске;

– удалять временный файл при выходе, чтобы не повредить базу эталонов.

Всю работу с позами обеспечивает отдельный менеджер (скрипт VRHandPoseManager).

4.4 Диаграмма переходов между сценами

Приложение RSL Fingerspeller построено по сценарию пошагового взаимодействия с пользователем, при котором каждый режим работы (главное меню, калибровка, тренировка, ввод слов) реализован в виде отдельной сцены Unity. Такой подход позволяет логически разделить функциональные блоки, упростить отладку, повторное использование компонентов и последующее масштабирование проекта.

Каждая сцена отвечает за конкретный этап обучения или настройки, обеспечивая последовательность и структурированность пользовательского опыта. Переход между сценами осуществляется с помощью встроенного пользовательского интерфейса (UI), в котором применяются методы взаимодействия Poke и Ray Interactions, предоставляемые Meta SDK. Кнопки реализованы средствами Unity UI и обрабатываются через специализированные скрипты, привязанные к соответствующим объектам на сцене.

Переходы между режимами происходят при нажатии на соответствующие кнопки и позволяют пользователю гибко перемещаться между этапами обучения. Все основные переходы и логика навигации представлены в виде схемы (Рисунок 20), которая наглядно демонстрирует структуру взаимодействия пользователя с приложением и взаимосвязь между ключевыми сценами.

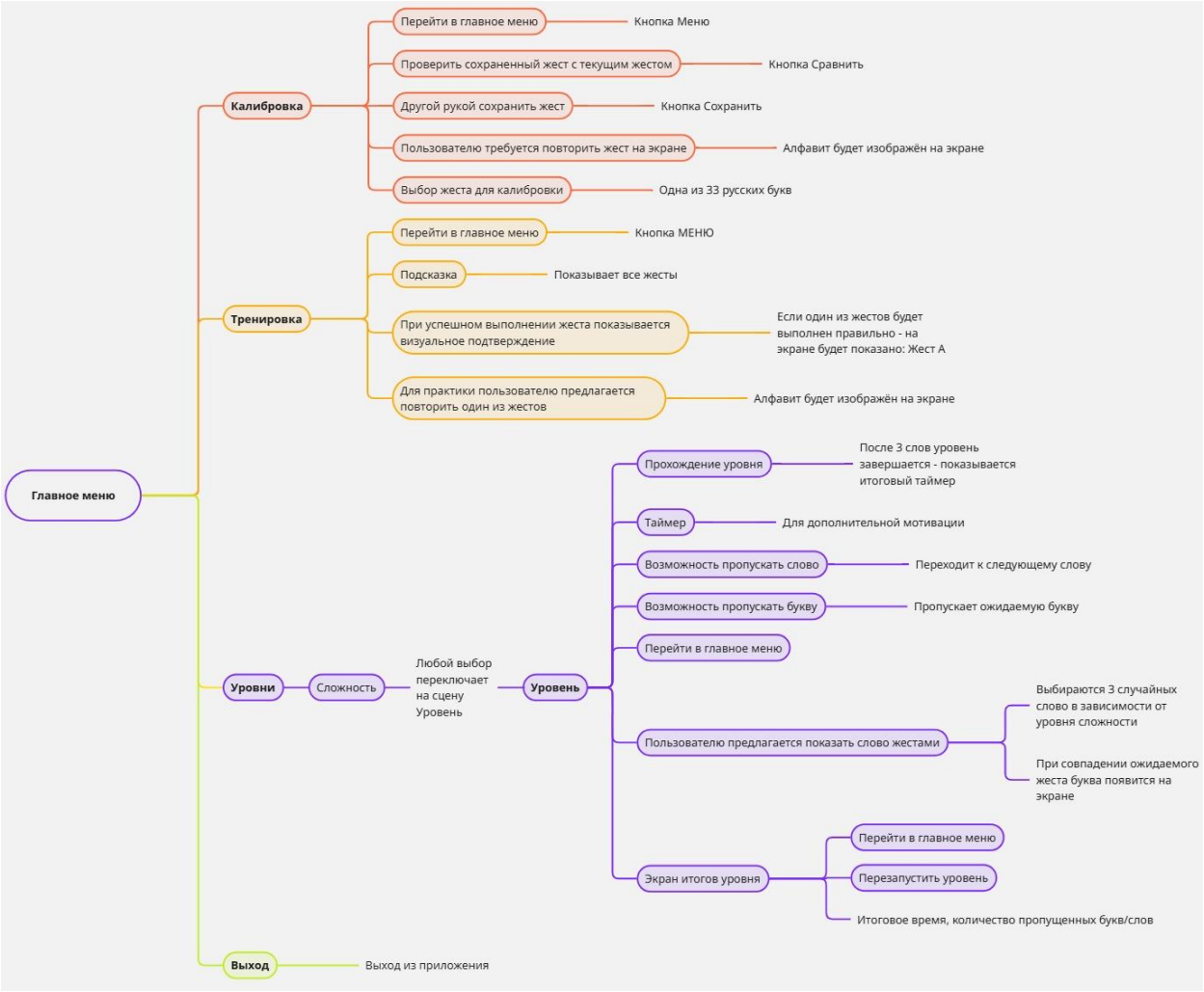


Рисунок 20 – Диаграмма переходов между сценами

Основные сценарии переходов:

– Главное меню (начальная сцена):

– Кнопка «Калибровка» — открывает сцену калибровки.

– Кнопка «Тренировка» — открывает свободный режим распознавания жестов.

– Кнопка «Выбор уровня» — открывает сцену выбора сложности.

– Кнопка «Выход» — завершает приложение.

– Сцена выбора уровня:

Кнопки «Лёгкий», «Средний», «Тяжёлый» — задают уровень сложности и переходят к сцене «Уровень».

– Сцена уровня:

После завершения всех слов появляется окно с результатами, и кнопка «Вернуться в меню» — возвращает в главное меню.

– Сцена калибровки:

Кнопка «Назад» — возвращает в главное меню после завершения сохранения жестов.

– Сцена тренировки:

По окончании работы пользователь может выйти в меню через кнопку «Назад».

Диаграмма переходов показывает, что пользователь всегда может вернуться в главное меню из любого режима. Это делает навигацию интуитивной и обеспечивает понятную структуру взаимодействия. Такое разделение по сценам также упрощает реализацию логики, повышает читаемость кода и улучшает стабильность проекта.

4.5 Функционал сцен и интерфейсов

Приложение RSL Fingerspeller состоит из пяти основных сцен, каждая из которых реализует определённую часть функциональности: от сохранения жестов до полноценной тренировки. Пользователь взаимодействует с интерфейсами с помощью отслеживания рук: как через Poke (касание), так и через Ray (луч). Во всех сценах используется единый визуальный стиль — градиентный фон, минималистичный UI и рабочая область, выделенная кругом, отрисованным через скрипт.

Для управления каждой сценой разработаны отдельные скрипты, отвечающие за загрузку и сохранение данных, переключение между режимами и взаимодействие с жестами.

4.5.1 Сцена «Главное меню»

Это стартовая сцена (Рисунок 21), с которой пользователь начинает работу в приложении. Интерфейс состоит из следующих элементов:

Рисунок 21 – Сцена главного меню

Логотип приложения (временно реализован в виде текстовой надписи);

Кнопки:

- «Калибровка» — переход к сцене настройки эталонных жестов;
- «Тренировка» — переход к сцене свободной практики;
- «Выбор уровня» — переход к выбору сложности;
- «Выход» — завершение приложения.

Все кнопки работают как на касание (Poke), так и на наведение луча (Ray). Их взаимодействие реализовано с использованием XR Interaction Toolkit.

4.5.2 Сцена «Калибровка»

В этой сцене пользователь может:

- Переключать текущую букву дактильного алфавита;
- Сохранять показанную позу руки как эталонную для текущей буквы;
- Сравнивать текущую позу с сохранённой;
- Видеть название выбранной буквы;
- Вернуться в главное меню.

За логику сохранения и сравнения поз отвечает скрипт HandPoseManager.cs, в котором реализованы методы SavePose() и ComparePose(). Сохранение происходит в JSON-файл, который копируется из базового шаблона.

Интерфейс содержит текстовое поле для отображения текущей буквы и текстовую обратную связь после сохранения или сравнения (Рисунок 22).

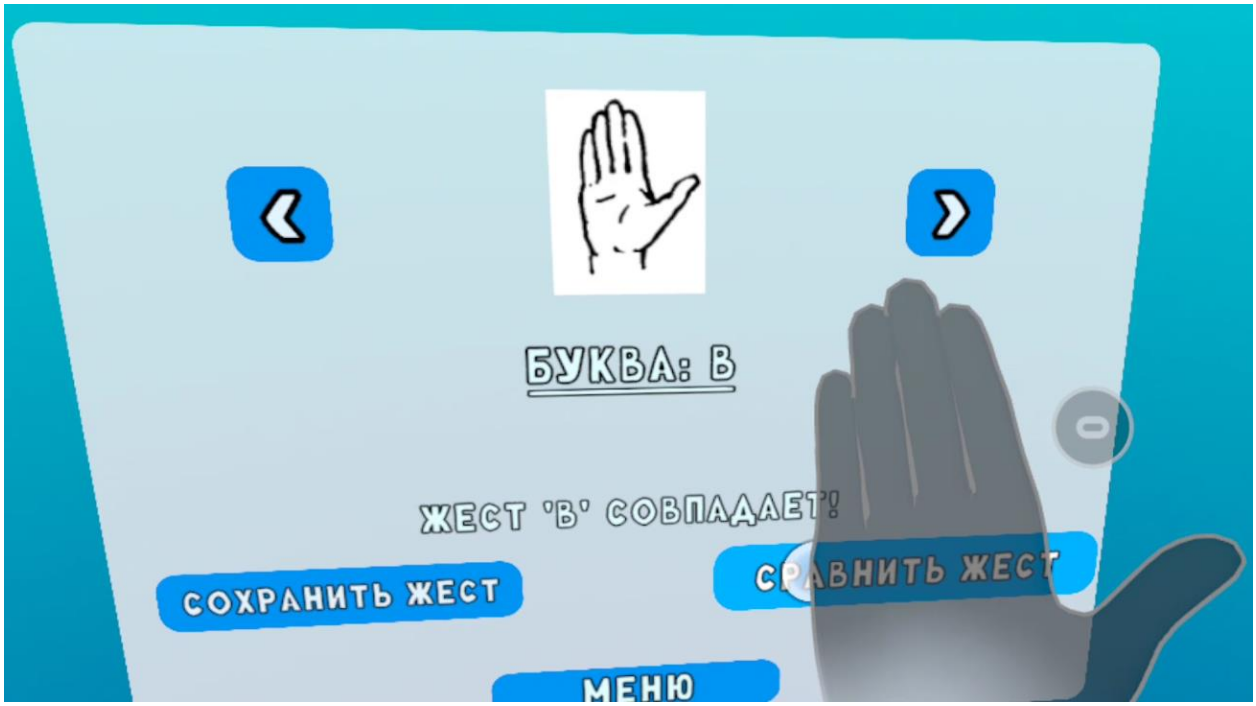


Рисунок 22 - Сцена калибровки

4.5.3 Сцена «Выбор уровня»

Интерфейс состоит из заголовка и трёх кнопок (Рисунок 23):

- «Лёгкий»;
- «Средний»;
- «Тяжёлый».

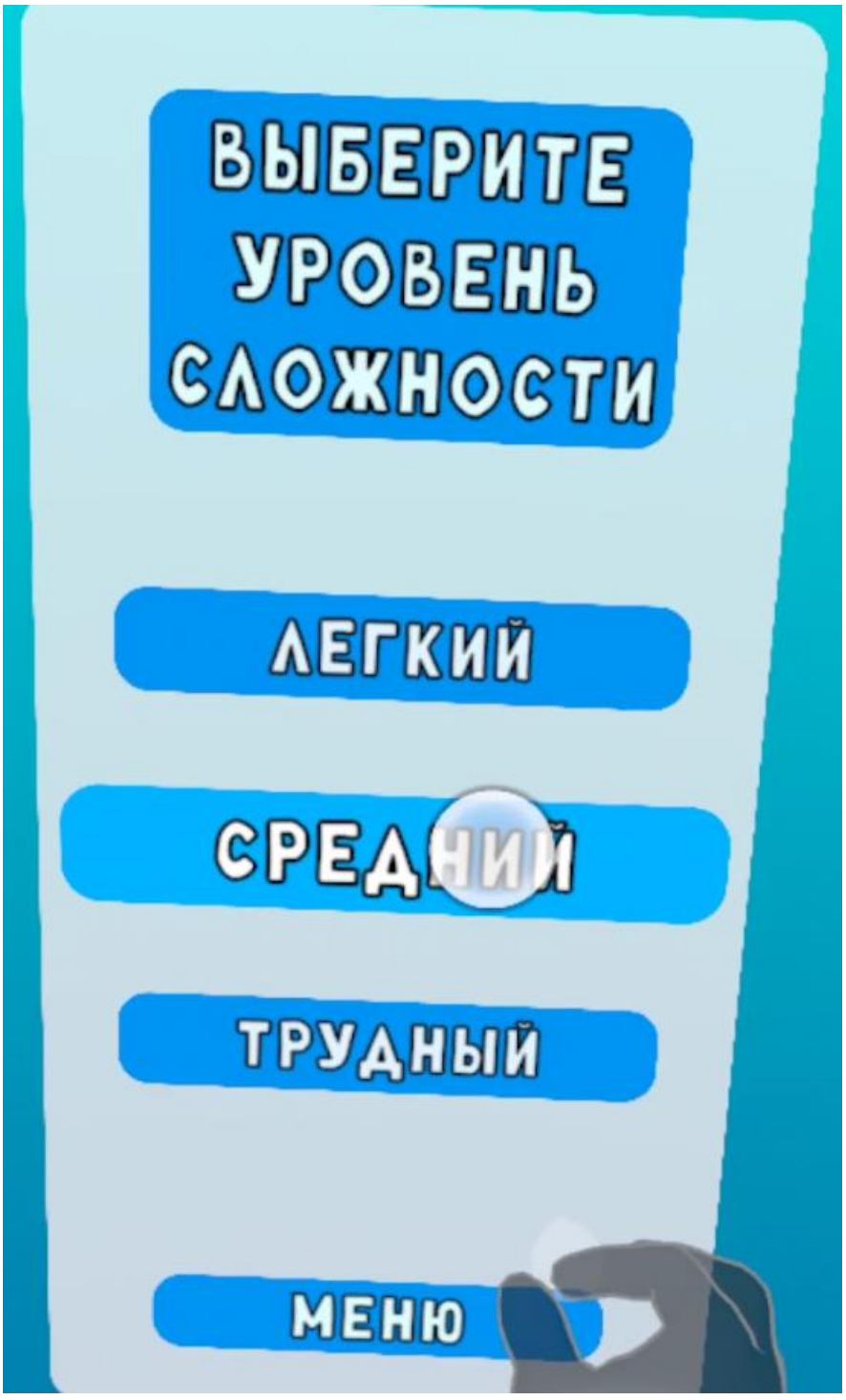


Рисунок 23 – Сцена выбора уровня сложности

Выбор сложности записывается во временную переменную и передаётся на сцену «Уровень». Уровень сложности влияет на то, из какого набора слов будет выбираться задание (лёгкие, средние или сложные слова).

4.5.4 Сцена «Уровень»

Основной режим обучения представляет собой ключевой этап взаимодействия пользователя с приложением. В этом режиме пользователю последовательно предлагается продемонстрировать жестами три слова, автоматически подобранные в зависимости от ранее выбранного уровня сложности (лёгкий, средний или сложный).

Интерфейс содержит:

- отображение текущего слова в верхней части экрана;
- строку с уже введёнными буквами, расположенную по центру интерфейса;
- набор управляющих кнопок, включая:
  - «Пропустить слово» — для перехода к следующему слову;
  - «Пропустить букву» — для пропуска текущей буквы;
- Таймер (отображает общее время выполнения задания);
- Текущий выбор сложности.

В процессе обучения, если пользователь корректно воспроизводит жест, соответствующий определённой букве, эта буква автоматически добавляется в строку набранных символов. Таким образом осуществляется обратная связь, позволяющая отслеживать прогресс выполнения задания (Рисунок 24)

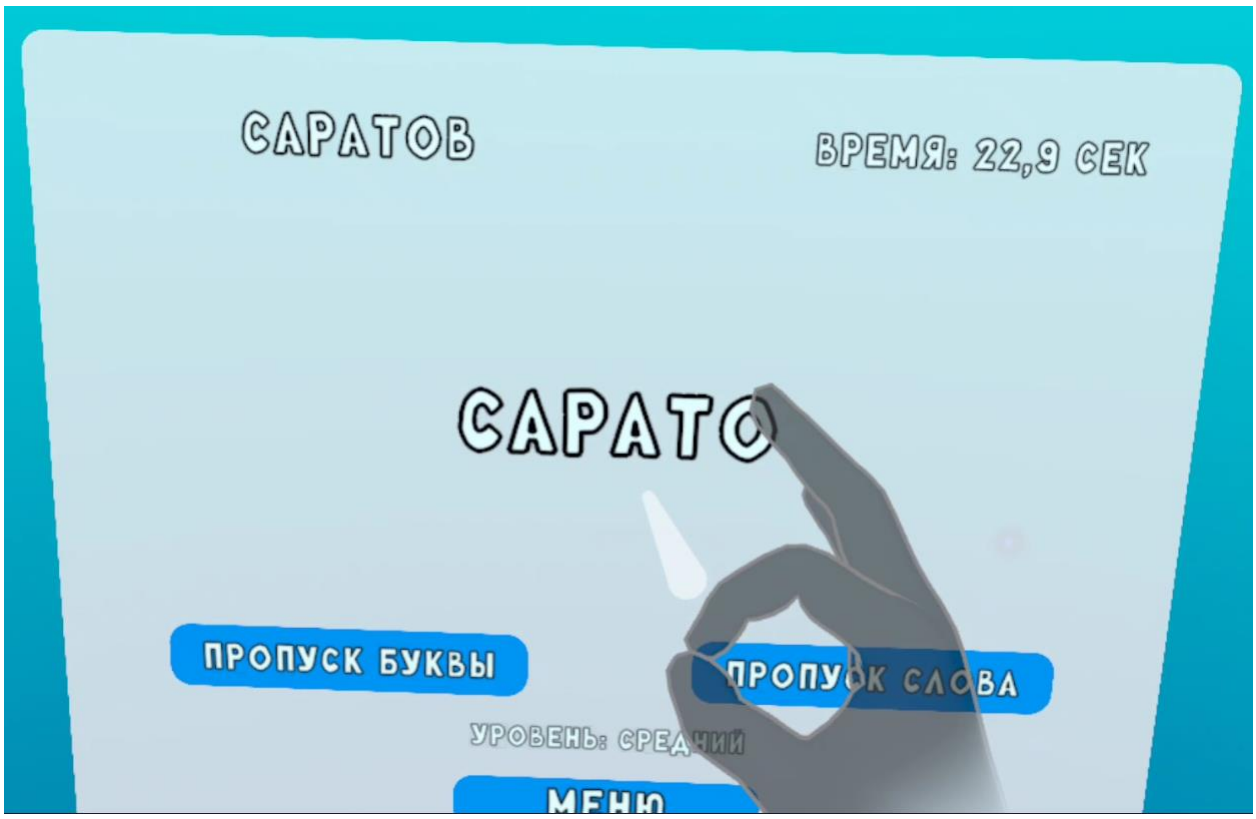


Рисунок 24 – Прохождение уровня

Когда слово собрано — начинается следующее. После 3 слов появляется окно с результатом и кнопка «В меню» (Рисунок 25).

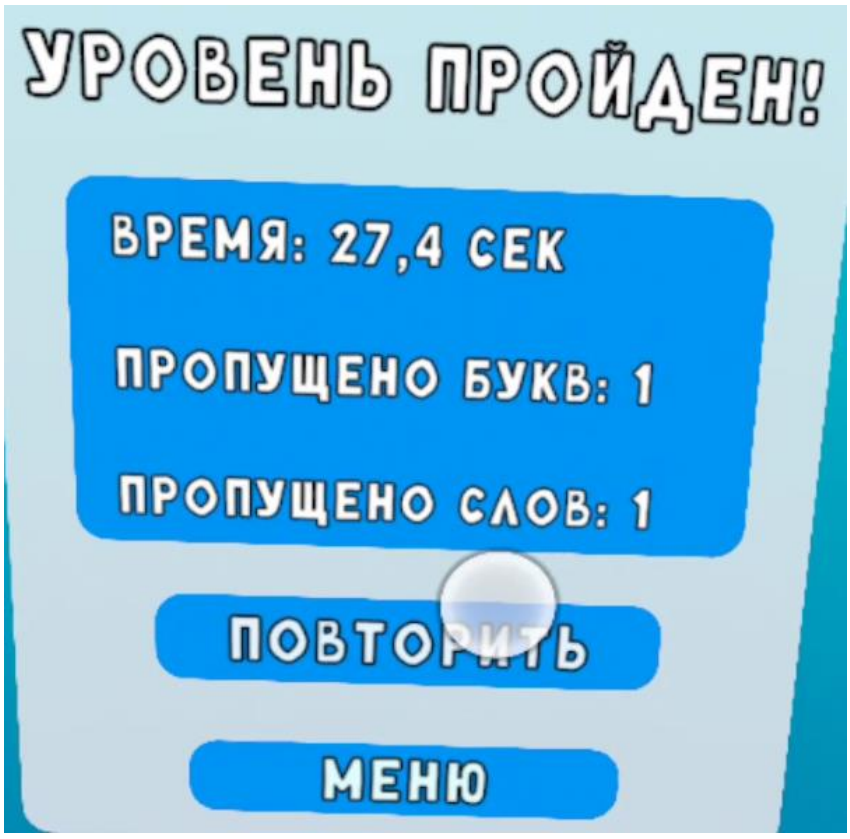


Рисунок 25 – Результаты прохождения уровня

За работу режима отвечает отдельный скрипт, в котором:

- Выбираются случайные слова;
- Сравниваются показанные жесты;
- Реализуется ввод и проверка.

4.5.5 Сцена «Тренировка»

Свободный режим, в котором система каждую секунду сравнивает текущую позу руки с базой эталонных жестов.

Интерфейс минимален (Рисунок 26):

- Текстовая метка с результатом (распознанная буква);
- Кнопка возврата в меню;
- Кнопка подсказки – показывает картинку со всеми жестами.

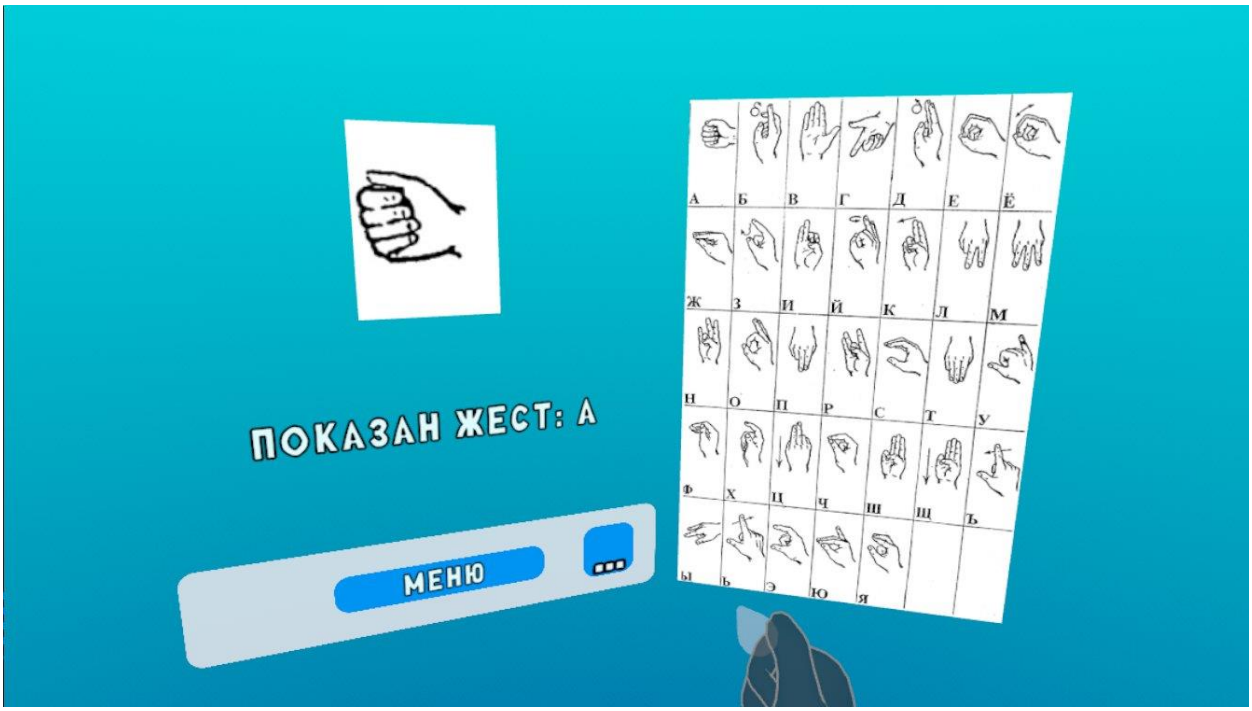


Рисунок 26 – Тренировка жестов

В планах — добавить вокруг пользователя модели руки (по одной на каждую букву), и при совпадении с одной из них будет срабатывать индикатор.

Скрипт:

- Загружает все эталонные позы из JSON;
- Ежесекундно сравнивает текущую руку с каждой сохранённой;
- Выводит результат.

Все сцены приложения реализованы с учётом удобства пользователя и соответствуют логике обучающего процесса. Использование отдельных сцен и скриптов позволило организовать чистую архитектуру и облегчить сопровождение проекта. Использование двух типов взаимодействия (Poke и Ray) расширяет доступность управления в VR-среде, а обратная связь помогает пользователю эффективно обучаться.

4.6 Реализация функционала с помощью скриптов

4.6.1 Скрипт HandPoseManager

Ключевую роль в системе калибровки и распознавания жестов играет скрипт HandPoseManager.cs (приложение А). Он отвечает за сохранение эталонных жестов, их загрузку из файла и последующее сравнение с текущими положениями руки пользователя. Ниже представлены основные блоки кода, реализующие эту функциональность.

- Сохранение жеста

Метод SaveCurrentPose() сохраняет текущую конфигурацию суставов в локальные координаты относительно ладони и записывает результат в словарь savedPoses. Затем вызывается функция сохранения в JSON-файл.

```
Vector3 localPos = Quaternion.Inverse(palmPose.rotation) * (jointPose.position - palmPose.position);
Quaternion localRot = Quaternion.Inverse(palmPose.rotation) * jointPose.rotation;
pose[joint] = new Pose(localPos, localRot);
```

Чтобы жесты можно было сравнивать независимо от положения руки в пространстве, все координаты суставов преобразуются в локальную систему координат, где опорной точкой считается HandPalm.

После сбора всех поз вызвается метод SavePosesToFile().

– Сохранение в файл

Метод SavePosesToFile() сериализует все сохранённые позы в формате JSON и записывает их во временный файл default_poses_temp.json.

```
string newJson = JsonUtility.ToJson(poseDataSet, true);
File.WriteAllText(path, newJson);
```

Жесты сохраняются в файл во временной директории Application.temporaryCachePath. Это позволяет изменять жесты во время работы, не повреждая оригинальные эталоны. При перезапуске приложения файл удаляется и создаётся заново.

– Сравнение с сохранённой позой

Метод CompareCurrentPose() сравнивает текущую конфигурацию руки с сохранённой для выбранной буквы. Сначала также используется HandPalm как опорная точка, а затем сравниваются позиции и ориентации всех отслеживаемых суставов.

```
float posDiff = Vector3.Distance(localPos, savedPose.position);
float rotDiff = Quaternion.Angle(localRot, savedPose.rotation);

if (posDiff > posThreshold || rotDiff > rotThreshold)
{
   feedbackText.text = $"Жест '{poseName}' не совпадает (кость {joint})!";
   return;
}
```

Если отклонения по позиции или ориентации превышают заданные пороги (0.04f и 25° соответственно), то пользователю выводится сообщение о несовпадении. В противном случае — подтверждение правильного жеста.

– Загрузка поз при запуске

Метод LoadPosesFromFile() вызывается при старте скрипта. Он читает файл default_poses_temp.json и восстанавливает словарь savedPoses.

– Навигация по алфавиту

Методы NextPose() и PreviousPose() позволяют переключаться между буквами алфавита. Это реализует простой механизм листания.

```
currentPoseIndex = (currentPoseIndex + 1) % poseNames.Count;
UpdatePoseNameDisplay();
```

Индекс текущей буквы изменяется циклически, и обновляется текстовая подсказка на экране.

Скрипт HandPoseManager является центральной частью приложения, обеспечивая все основные функции по работе с жестами: сохранение, сравнение и загрузку. Он использует локальные координаты, сериализацию в JSON и систему обратной связи для пользователя, обеспечивая точность и удобство взаимодействия. Разделение логики по функциям делает код модульным и расширяемым для будущих доработок.

4.6.2 Скрипт HandPoseRecognizer

В отличие от HandPoseManager, где пользователь вручную сохраняет и сравнивает жесты, скрипт HandPoseRecognizer (Приложение 2) отвечает за автоматическое распознавание текущей позы руки на основе загруженных эталонных данных. Он используется, например, в тренировочной сцене, где каждую секунду система проверяет, не совпадает ли текущий жест с одним из сохранённых.

- Загрузка эталонных поз

Метод LoadAllPoses() вызывается при старте и загружает все сохранённые жесты из временного файла JSON. Он формирует словарь allPoses, где ключом выступает имя буквы, а значением — набор поз суставов.

```
string json = File.ReadAllText(path);
PoseDataSet poseDataSet = JsonUtility.FromJson<PoseDataSet>(json);
...
jointDict[jointId] = new Pose(position, rotation);
```

Вся информация загружается из default_poses_temp.json, ранее сформированного в процессе калибровки. Это позволяет использовать одни и те же данные как для тренировки, так и для обучения.

– Автоматический запуск проверки

Метод InvokeRepeating в Start() задаёт вызов функции распознавания каждую секунду:

```
InvokeRepeating(nameof(TryRecognizeGesture), 1f, 1f);
```

Распознавание происходит автоматически, без нажатий. Это удобно в режиме тренировки, когда пользователь может просто повторять жесты перед камерой.

– Сравнение с базой жестов

Затем каждый сохранённый жест сравнивается с текущим. Если все суставы находятся в пределах допустимых отклонений, жест считается распознанным.

```
if (posDiff > posThreshold || rotDiff > rotThreshold)
{
   match = false;
   break;
}
```

Как и в калибровке, здесь заданы пороговые значения — 4 см по позиции и 25° по повороту. Эти значения можно адаптировать для более чувствительной или более гибкой системы распознавания.

Скрипт HandPoseRecognizer реализует фоновое распознавание текущего жеста и подходит для тренировочных режимов. Он использует тот же формат данных, что и модуль калибровки, что позволяет переиспользовать базу жестов без дублирования кода. Такой подход обеспечивает расширяемость проекта и удобство для пользователя.

4.6.3 Скрипт GameManager

Скрипт GameManager (Приложение В) реализует игровую механику уровня, где пользователь должен ввести слово, показывая дактильные жесты по буквам. Программа сравнивает текущий жест с сохранёнными эталонами и при совпадении продвигается к следующей букве. Скрипт также отслеживает время прохождения, управляет интерфейсом и завершает уровень после ввода всех слов.

– Загрузка и отображение слов

В методе LoadWordsByDifficulty() загружается три случайных слова из выбранной категории сложности. Используется PlayerPrefs для получения сохранённого значения, установленного ранее в сцене выбора.

```
string difficulty = PlayerPrefs.GetString("Difficulty", "Лёгкий");
List<string> source = difficulty switch { ... };
```

Позволяет адаптировать уровень сложности, изменяя длину и сложность слов. Это одна из основ системы обучения.

– Отображение текущего слова и буквы

Методы LoadNextWord() и UpdateInputDisplay() управляют выводом текста на экран: текущее слово и то, что уже введено пользователем с помощью жестов.

```
currentWord = currentWords[currentWordIndex];
inputText.text = currentInput.ToUpper();
```

Обеспечивает понятный для пользователя визуальный интерфейс процесса прохождения уровня.

– Распознавание жестов

В методе TryRecognizeGesture() раз в 0.5 секунды осуществляется попытка распознать жест и сопоставить его с нужной буквой текущего слова.

```
if (recognized == expected)
{
    currentInput += recognized;
    currentLetterIndex++;
}
```

Если пользователь правильно показал жест, буква добавляется, а интерфейс обновляется. После ввода всего слова система автоматически переходит к следующему.

- Сравнение поз

Метод IsMatchingPose() проверяет, насколько текущая поза соответствует эталонной. Используются локальные координаты (относительно ладони) и пороги по положению и вращению.

```
Vector3 localCurrentPos = Quaternion.Inverse(palmPose.rotation) * (currentPose.position - palmPose.position);
float posDiff = Vector3.Distance(localCurrentPos, savedPose.position);
```

Позволяет игнорировать поворот руки в пространстве и фокусироваться на форме жеста, что особенно важно в VR-среде.

- Управление таймером и завершением уровня

Скрипт также отвечает за отображение времени (timerText) и экран завершения (endPanel), который появляется после успешного ввода всех слов.

```
finalTimeText.text = "Время: " + timer.ToString("F1") + " сек";
endPanel.SetActive(true);
```

Создаёт элемент соревнования и оценки, что важно в игровой и обучающей механике.

- Дополнительные функции

Также реализованы функции:

- SkipLetter() — пропуск текущей буквы.
- SkipWord() — пропуск текущего слова.
- ReturnToMenu() — возвращение в главное меню.

Скрипт GameManager — основной управляющий модуль сцены прохождения. Он объединяет распознавание жестов, управление интерфейсом, выбор слов, таймер и переходы между этапами. Благодаря этому создаётся целостный игровой опыт, где пользователь в интерактивной форме закрепляет знание дактильного алфавита.

4.6.4 Дополнительные скрипты

Для корректной работы приложения и управления данными жестов реализован ряд вспомогательных скриптов. Они не участвуют напрямую в распознавании или обучении, но обеспечивают стабильность, конфигурацию и удобную структуру хранения.

– Скрипт PoseFileManager

Отвечает за копирование эталонного файла с жестами из папки StreamingAssets во временную директорию temporaryCachePath при запуске приложения.

- При первом запуске за сессию файл default_poses_temp.json копируется или перезаписывается.
- На Android используется UnityWebRequest, так как прямой доступ к StreamingAssets невозможен.
- В редакторе/на ПК файл копируется напрямую через File.Copy().

```
if (File.Exists(destPath))
   File.Delete(destPath); // Очистка устаревшего файла

File.Copy(sourcePath, destPath, true); // Копирование
```

Гарантирует, что приложение всегда работает с "чистой" версией базы жестов, которую можно изменить во время сессии, не повреждая оригинал.

– Скрипт DifficultySelector

Обрабатывает выбор уровня сложности пользователем на сцене выбора и передаёт его между сценами с помощью PlayerPrefs.

```
PlayerPrefs.SetString("Difficulty", difficulty);
SceneManager.LoadScene("Level");
```

Обеспечивает персонализацию уровня — разные списки слов в зависимости от выбранной сложности (используется в GameManager).

– Скрипт DataTypes

Содержит сериализуемые структуры (PoseDataSet, NamedPose, JointData, PoseData) для удобного хранения и чтения поз из JSON-файла.

- Используется Unity JsonUtility.
- Структура PoseData содержит массивы для позиции и кватерниона, а также метод ToPose() для преобразования обратно в UnityEngine.Pose.

```
public Pose ToPose()
{
   return new Pose(
      new Vector3(position[0], position[1], position[2]),
      new Quaternion(rotation[0], rotation[1], rotation[2], rotation[3])
   );
}
```

Позволяет гибко и надёжно сериализовать данные о позах руки в JSON-формате и обратно, поддерживая кроссплатформенность и расширяемость.

Скрипты PoseFileManager, DifficultySelector и DataTypes создают фундамент для стабильной работы приложения, обеспечивают передачу данных между сценами и удобное хранение жестов. Хотя они не связаны напрямую с интерфейсом или игровым процессом, их использование значительно повышает удобство и надёжность архитектуры системы.

4.7 Работа с данными: JSON-файлы, загрузка, удаление и перезапись

Одним из ключевых аспектов работы VR-приложения RSL Fingerspeller является сохранение и загрузка пользовательских и эталонных поз рук. Для этих целей используется формат JSON, позволяющий сериализовать данные в виде удобной и читаемой структуры. Работа с файлами реализована через временную директорию temporaryCachePath, что обеспечивает гибкость и защиту исходного файла с эталонами от повреждений.

– Структура JSON-файла

JSON-файл содержит список жестов, каждый из которых представлен названием (poseName) и списком суставов (joints). Каждый сустав содержит своё имя (jointId), позицию и ориентацию:

```
{
  "poses": [
    {
      "poseName": "A",
      "joints": [
        {
          "jointId": "HandThumb1",
          "pose": {
            "position": [
              -0.023384613916277887,
              -0.016178488731384279,
              -0.028133884072303773
            ],
            "rotation": [
              0.09661506861448288,
              -0.3838643729686737,
              0.6781459450721741,
              0.6192181706428528
            ]
          }
        }
```

– Загрузка и десериализация поз

Загрузка происходит из временного файла default_poses_temp.json. Это реализовано в таких скриптах, как HandPoseManager, HandPoseRecognizer, GameManager. Все они используют один и тот же путь и структуру:

```
string path = Path.Combine(Application.temporaryCachePath, "default_poses_temp.json");
string json = File.ReadAllText(path);
PoseDataSet poseDataSet = JsonUtility.FromJson<PoseDataSet>(json);
```

После этого каждая поза преобразуется в словарь: ключ — HandJointId, значение — Pose. Такая структура обеспечивает быстрый доступ при сравнении поз.

– Сохранение и перезапись

Сохранение нового жеста (например, при калибровке) происходит с перезаписью текущей буквы в JSON-файле, если она уже существует. Это реализовано методом SavePosesToFile() в HandPoseManager

```
// Проверка на существующую позу:
if (poseDataSet.poses[i].poseName == namedPose.poseName)
{
   poseDataSet.poses[i] = namedPose;
   isPoseFound = true;
}
```

После формирования обновлённого списка поз JSON сохраняется с форматированием:

```
string newJson = JsonUtility.ToJson(poseDataSet, true);
File.WriteAllText(path, newJson);
```

Такой подход позволяет пользователю самостоятельно настраивать или повторно сохранять жесты, не нарушая структуру файла.

– Копирование и сброс файла

При запуске приложения выполняется копирование оригинального файла default_poses.json в default_poses_temp.json, которое реализовано в скрипте PoseFileManager. Таким образом, оригинальные эталонные данные всегда остаются неизменными. Темповая копия удаляется и перезаписывается:

```
if (File.Exists(destPath))
{
   File.Delete(destPath);
}
File.Copy(sourcePath, destPath, true);
```

На Android применяется UnityWebRequest для загрузки файла из StreamingAssets, что обусловлено ограничениями платформы.

Работа с JSON-файлами — важная часть архитектуры приложения. Она обеспечивает:

- Гибкость: пользователь может повторно калибровать любые жесты.
- Надёжность: оригинальные данные не повреждаются.
- Простоту: сериализация через встроенный JsonUtility упрощает реализацию.

Такая система делает возможной адаптацию приложения под конкретного пользователя и облегчает дальнейшее масштабирование, например, при добавлении сохранения профилей или обучающих прогрессов.

ЗАКЛЮЧЕНИЕ

В ходе выпускной квалификационной работы было разработано VR-приложение RSL Fingerspeller, предназначенное для обучения дактильному русскому алфавиту с использованием технологий виртуальной реальности и трекинга рук. Приложение реализовано для автономной VR-гарнитуры Oculus Quest 2 и позволяет пользователю проходить этапы калибровки, тренировки и выполнения заданий по распознаванию букв.

Основной целью проекта было создание удобной и интуитивной системы, в которой пользователь может сохранять эталонные жесты, сравнивать их с текущими показанными позами, а также тренировать распознавание букв в интерактивной форме. Были реализованы ключевые функциональные модули: обработка и сохранение данных о позах руки, визуальная и текстовая обратная связь, пользовательский интерфейс с поддержкой Poke и Ray взаимодействий.

Приложение построено на модульной архитектуре, что обеспечивает его расширяемость и простоту поддержки. Все сцены оформлены в едином стиле, а логика взаимодействия пользователя и программы реализована через отдельные скрипты, объединённые в общую систему управления. Хранение и сравнение поз происходит с использованием формата JSON, что делает систему гибкой и легко модифицируемой.

Общий объём программной реализации составил около 1500 строк кода, охватывающих все основные скрипты, включая обработку жестов, пользовательский интерфейс, систему управления сценами и сериализацию данных.

- Разработка и тестирование проводились на реальном устройстве Oculus Quest 2, что позволило учесть особенности взаимодействия в условиях автономной VR-среды. Приложение прошло апробацию в рамках Ежегодной международной научно-практической конференции «Проблемы управления в социально-экономических и технических

системах» на тему «Разработка VR-приложения для обучения дактильному русскому языку»;(в печати)

С учётом продемонстрированной функциональности и потенциала для дальнейшего развития, проект рекомендован к внедрению ООО Всероссийское общество глухих».

Разработка проекта позволила не только овладеть технологиями создания VR-приложений, но и на практике реализовать функциональный инструмент, который в дальнейшем может быть использован для образовательных целей. Полученные результаты демонстрируют потенциал виртуальной реальности как эффективной среды для изучения дактильного алфавита и других форм жестового общения.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Unity: Getting Started with the Meta XR Interaction SDK [Электронный ресурс]. URL: https://developers.meta.com/horizon/documentation/unity/unity-isdk-getting-started (дата обращения: 01.02.2025).
2. Unity: Getting Started with UnityXR [Электронный ресурс]. URL: https://developers.meta.com/horizon/documentation/unity/unity-isdk-getting-started-unityxr (дата обращения: 03.02.2025).
3. Unity Development Overview – Обзор среды разработки [Электронный ресурс]. URL: https://developers.meta.com/horizon/documentation/unity/unity-development-overview (дата обращения: 05.02.2025).
4. Unity Tutorial – Вводные уроки по разработке приложений [Электронный ресурс]. URL: https://developers.meta.com/horizon/documentation/unity/unity-tutorial (дата обращения: 07.02.2025).
5. Meta XR SDK Example Scenes – Примеры сцен с использованием SDK [Электронный ресурс]. URL: https://developers.meta.com/horizon/documentation/unity/unity-isdk-example-scenes (дата обращения: 09.02.2025).
6. Unity ISDK: UI Creation – Создание пользовательского интерфейса в VR [Электронный ресурс]. URL: https://developers.meta.com/horizon/documentation/unity/unity-isdk-create-ui (дата обращения: 10.02.2025).
7. Unity ISDK: Canvas Integration – Интеграция Canvas в VR [Электронный ресурс]. URL: https://developers.meta.com/horizon/documentation/unity/unity-isdk-canvas-integration/ (дата обращения: 11.02.2025).
8. Meta Design: Buttons – Дизайн и поведение кнопок [Электронный ресурс]. URL: https://developers.meta.com/horizon/design/buttons (дата обращения: 12.02.2025).
9. Unity ISDK: Poke Interaction – Взаимодействие касанием [Электронный ресурс]. URL: https://developers.meta.com/horizon/documentation/unity/unity-isdk-poke-interaction/ (дата обращения: 13.02.2025).

10. Unity ISDK: Ray Interaction – Взаимодействие через луч [Электронный ресурс]. URL: https://developers.meta.com/horizon/documentation/unity/unity-isdk-ray-interaction/ (дата обращения: 14.02.2025).

11. Unity ISDK: Pointable – Настройка объектов для наведения [Электронный ресурс]. URL: https://developers.meta.com/horizon/documentation/unity/unity-isdk-pointable (дата обращения: 15.02.2025).

12. Unity ISDK: Hand Tracking Setup – Настройка отслеживания рук [Электронный ресурс]. URL: https://developers.meta.com/horizon/documentation/unity/unity-handtracking-hands-setup (дата обращения: 16.02.2025).

13. Unity ISDK: OpenXR Hand Support – Поддержка OpenXR для рук [Электронный ресурс]. URL: https://developers.meta.com/horizon/documentation/unity/unity-isdk-openxr-hand (дата обращения: 17.02.2025).

14. Unity ISDK: Customize Hand Model – Кастомизация моделей рук [Электронный ресурс]. URL: https://developers.meta.com/horizon/documentation/unity/unity-isdk-customize-hand-model (дата обращения: 18.02.2025).

15. Unity ISDK: Hand Pose Detection – Обнаружение поз рук [Электронный ресурс]. URL: https://developers.meta.com/horizon/documentation/unity/unity-isdk-hand-pose-detection/ (дата обращения: 20.02.2025).

16. Unity ISDK: Building a Hand Pose Recognizer – Построение системы распознавания поз [Электронный ресурс]. URL: https://developers.meta.com/horizon/documentation/unity/unity-isdk-building-hand-pose-recognizer/ (дата обращения: 21.02.2025).

17. Unity ISDK: Hand Pose Selector and Recorder – Инструмент записи и выбора поз [Электронный ресурс]. URL: https://developers.meta.com/horizon/documentation/unity/unity-isdk-hand-pose-selector-recorder/ (дата обращения: 22.02.2025).

18. Unity ISDK: Gesture Examples Scene – Примеры распознавания жестов [Электронный ресурс]. URL: https://developers.meta.com/horizon/documentation/unity/unity-isdk-gesture-examples-scene (дата обращения: 23.02.2025).

19. Unity ISDK: Pose Examples Scene – Сцена с примерами поз [Электронный ресурс]. URL: https://developers.meta.com/horizon/documentation/unity/unity-isdk-pose-examples-scene (дата обращения: 24.02.2025).

20. Unity XR Hands: Hand Poses – Работа с позами рук [Электронный ресурс]. URL: https://docs.unity3d.com/Packages/com.unity.xr.hands@1.4/manual/gestures/hand-poses.html (дата обращения: 26.02.2025).

21. Unity XR Hands: Static Hand Gestures – Статические жесты [Электронный ресурс]. URL: https://docs.unity3d.com/Packages/com.unity.xr.hands@1.4/manual/gestures/static-hand-gesture.html (дата обращения: 27.02.2025).

22. Oculus Developer Code Samples – Примеры кода от разработчиков Oculus [Электронный ресурс]. URL: https://developer.oculus.com/code-samples/ (дата обращения: 01.03.2025).

23. Oculus Integration в Unity Asset Store [Электронный ресурс]. URL: https://assetstore.unity.com/packages/tools/integration/oculus-integration-82022 (дата обращения: 02.03.2025).

24. Unity Discussions – Форум сообщества Unity [Электронный ресурс]. URL: https://discussions.unity.com/ (дата обращения: 05.03.2025).

25. Stack Overflow – Вопросы и решения по разработке Unity и VR [Электронный ресурс]. URL: https://stackoverflow.com/questions (дата обращения: 07.03.2025).

26. Unity Documentation: UI Toolkit – Документация по интерфейсам Unity [Электронный ресурс]. URL: https://docs.unity3d.com/6000.1/Documentation/Manual/UIToolkits.html (дата обращения: 08.03.2025).

27. Обучающие видеоматериалы по жестовому языку [Электронный ресурс] // Всероссийское общество глухих. URL: https://voginfo.ru/zhestovyj-jazyk/obuchajushhie-videomaterialy/ (дата обращения: 10.03.2025).

28. Дактилология – термин и описание [Электронный ресурс] // Википедия. URL: https://ru.wikipedia.org/wiki/Дактилология (дата обращения: 12.03.2025).

29. Дактильный алфавит – материалы для изучения [Электронный ресурс] // SignLang. URL: https://signlang.ru/studyrsl/daktil/ (дата обращения: 14.03.2025).

ПРИЛОЖЕНИЕ А

Листинг скрипта HandPoseManager

```
using System.Collections.Generic;
using UnityEngine;
using TMPro;
using Oculus.Interaction.Input;
using System.IO;
using System.Collections;
using UnityEngine.Networking;
using System;

public class VRHandPoseManager : MonoBehaviour
{
    [SerializeField] private Hand hand;
    [SerializeField] private TextMeshProUGUI feedbackText;
    [SerializeField] private TextMeshProUGUI poseNameText;
    [SerializeField] private SpriteRenderer recognizedSpriteRenderer;
    [SerializeField] private string spriteSheetName = "Dactil"; // имя PNG без
расширения
    private Dictionary<int, Sprite> gestureSpritesDict;

    private readonly Dictionary<string, Dictionary<HandJointId, Pose>> savedPoses =
new();

    // 33 буквы алфавита
    private readonly List<string> poseNames = new()
    {
        "А", "Б", "В", "Г", "Д", "Е", "Ё", "Ж", "З", "И", "Й", "К", "Л", "М",
        "Н", "О", "П", "Р", "С", "Т", "У", "Ф", "Х", "Ц", "Ч", "Ш", "Щ", "Ъ", "Ы", "Ь", "Э",
"Ю", "Я"
    };

    private int currentPoseIndex = 0;

    private readonly HandJointId[] jointsToTrack = new HandJointId[] {
        HandJointId.HandThumb1, HandJointId.HandThumb2,
HandJointId.HandThumb3,
        HandJointId.HandIndex0, HandJointId.HandIndex1, HandJointId.HandIndex2,
HandJointId.HandIndex3,
        HandJointId.HandMiddle0, HandJointId.HandMiddle1,
HandJointId.HandMiddle2, HandJointId.HandMiddle3,
        HandJointId.HandRing0, HandJointId.HandRing1, HandJointId.HandRing2,
HandJointId.HandRing3,
        HandJointId.HandPinky0, HandJointId.HandPinky1, HandJointId.HandPinky2,
HandJointId.HandPinky3,
        HandJointId.HandWristRoot, HandJointId.HandPalm
    };

    // Сохранение текущего жеста
    public void SaveCurrentPose()
```

```
    {
        string poseName = GetCurrentPoseName();
        var pose = new Dictionary<HandJointId, Pose>();

        if (!hand.GetJointPose(HandJointId.HandPalm, out Pose palmPose))
        {
            feedbackText.text = "Не удалось получить позу ладони!";
            return;
        }

        foreach (var joint in jointsToTrack)
        {
            if (hand.GetJointPose(joint, out Pose jointPose))
            {
                Vector3 localPos = Quaternion.Inverse(palmPose.rotation) *
(jointPose.position - palmPose.position);
                Quaternion localRot = Quaternion.Inverse(palmPose.rotation) *
jointPose.rotation;

                pose[joint] = new Pose(localPos, localRot);
            }
        }

        savedPoses[poseName] = pose;
        feedbackText.text = $"Жест '{poseName}' сохранён!";

        // Сразу записываем жесты в файл
        SavePosesToFile();
    }

    // Сохранение всех жестов в файл
    public void SavePosesToFile()
    {
        string path = Path.Combine(Application.temporaryCachePath,
"default_poses_temp.json");

        PoseDataSet poseDataSet = new PoseDataSet { poses = new
List<NamedPose>() };

        // Проверка на существующие жесты и перезапись при необходимости
        if (File.Exists(path))
        {
            string json = File.ReadAllText(path);
            poseDataSet = JsonUtility.FromJson<PoseDataSet>(json);
        }

        foreach (var poseEntry in savedPoses)
        {
            NamedPose namedPose = new NamedPose { poseName = poseEntry.Key,
joints = new List<JointData>() };
```

```
            foreach (var jointEntry in poseEntry.Value)
            {
                JointData jointData = new JointData
                {
                    jointId = jointEntry.Key.ToString(),
                    pose = new PoseData
                    {
                        position = new float[] { jointEntry.Value.position.x,
jointEntry.Value.position.y, jointEntry.Value.position.z },
                        rotation = new float[] { jointEntry.Value.rotation.x,
jointEntry.Value.rotation.y, jointEntry.Value.rotation.z, jointEntry.Value.rotation.w }
                    }
                };
                namedPose.joints.Add(jointData);
            }

            // Добавить новый жест или перезаписать существующий
            bool isPoseFound = false;
            for (int i = 0; i < poseDataSet.poses.Count; i++)
            {
                if (poseDataSet.poses[i].poseName == namedPose.poseName)
                {
                    poseDataSet.poses[i] = namedPose;
                    isPoseFound = true;
                    break;
                }
            }

            if (!isPoseFound)
            {
                poseDataSet.poses.Add(namedPose);
            }
        }

        // Запись в файл
        string newJson = JsonUtility.ToJson(poseDataSet, true); // true -
форматирование
        File.WriteAllText(path, newJson);

        feedbackText.text = "Жесты сохранены в файл!";
    }

    // Получение текущего названия жеста
    public string GetCurrentPoseName()
    {
        return poseNames[currentPoseIndex];
    }

    // Переход к следующему жесту
    public void NextPose()
    {
```

```
        currentPoseIndex = (currentPoseIndex + 1) % poseNames.Count;
        UpdatePoseNameDisplay();
    }

    // Переход к предыдущему жесту
    public void PreviousPose()
    {
        currentPoseIndex = (currentPoseIndex - 1 + poseNames.Count) %
poseNames.Count;
        UpdatePoseNameDisplay();
    }

    // Сравнение текущего жеста с сохраненным
    public void CompareCurrentPose()
    {
        string poseName = GetCurrentPoseName();

        if (!savedPoses.ContainsKey(poseName))
        {
            feedbackText.text = $"Жест '{poseName}' не сохранён!";
            return;
        }

        if (!hand.GetJointPose(HandJointId.HandPalm, out Pose palmPose))
        {
            feedbackText.text = "Не удалось получить позу ладони!";
            return;
        }

        var saved = savedPoses[poseName];
        float posThreshold = 0.04f;
        float rotThreshold = 25f;

        foreach (var joint in jointsToTrack)
        {
            if (!hand.GetJointPose(joint, out Pose currentPose)) continue;

            Vector3 localPos = Quaternion.Inverse(palmPose.rotation) *
(currentPose.position - palmPose.position);
            Quaternion localRot = Quaternion.Inverse(palmPose.rotation) *
currentPose.rotation;

            if (!saved.TryGetValue(joint, out Pose savedPose)) continue;

            float posDiff = Vector3.Distance(localPos, savedPose.position);
            float rotDiff = Quaternion.Angle(localRot, savedPose.rotation);

            if (posDiff > posThreshold || rotDiff > rotThreshold)
            {
                feedbackText.text = $"Жест '{poseName}' не совпадает!";
                return;
```

```
            }
        }

        feedbackText.text = $"Жест '{poseName}' совпадает!";
    }

    private void LoadPosesFromFile()
    {
        string path = Path.Combine(Application.temporaryCachePath,
"default_poses_temp.json");

        if (!File.Exists(path))
        {
            Debug.LogWarning($"Файл не найден: {path}");
            return;
        }

        try
        {
            string json = File.ReadAllText(path);
            PoseDataSet poseDataSet = JsonUtility.FromJson<PoseDataSet>(json);

            if (poseDataSet == null || poseDataSet.poses == null)
            {
                Debug.LogError("Ошибка при разборе JSON — данные пустые или
некорректные.");
                return;
            }

            savedPoses.Clear();

            foreach (var namedPose in poseDataSet.poses)
            {
                var jointDict = new Dictionary<HandJointId, Pose>();

                foreach (var jointData in namedPose.joints)
                {
                    if (System.Enum.TryParse(jointData.jointId, out HandJointId jointId))
                    {
                        Vector3 position = new Vector3(jointData.pose.position[0],
jointData.pose.position[1], jointData.pose.position[2]);
                        Quaternion rotation = new Quaternion(jointData.pose.rotation[0],
jointData.pose.rotation[1], jointData.pose.rotation[2], jointData.pose.rotation[3]);
                        jointDict[jointId] = new Pose(position, rotation);
                    }
                    else
                    {
                        Debug.LogWarning($"Неизвестный сустав: {jointData.jointId}");
                    }
                }
```

```
            savedPoses[namedPose.poseName] = jointDict;
        }
        Debug.Log($"Загружено жестов: {savedPoses.Count}");
    }
    catch (Exception ex)
    {
        Debug.LogError($"Ошибка при загрузке файла жестов: {ex.Message}");
    }
}


private void Start()
{
    LoadPosesFromFile();
    UpdatePoseNameDisplay();
}

private void Awake()
{
    LoadSpriteSheet();
}

public void SetCurrentPoseIndex(int index)
{
    currentPoseIndex = index;
    UpdatePoseNameDisplay();
}

private void LoadSpriteSheet()
{
    // Загружаем все спрайты с листа (Multiple) по имени
    gestureSpritesDict = new Dictionary<int, Sprite>();
    var allSprites = Resources.LoadAll<Sprite>(spriteSheetName);

    foreach (var sprite in allSprites)
    {
        // Имена спрайтов вида: Dactil_1, Dactil_2 и т.д.
        string[] parts = sprite.name.Split('_');
        if (parts.Length == 2 && int.TryParse(parts[1], out int index))
        {
            gestureSpritesDict[index] = sprite;
        }
    }
}

private void UpdateSpriteByPoseIndex()
{
    if (recognizedSpriteRenderer == null || gestureSpritesDict == null)
        return;

    int spriteIndex = currentPoseIndex + 1; // т.к. Dactil_1 — это "A",
```

```
currentPoseIndex = 0

        if (gestureSpritesDict.TryGetValue(spriteIndex, out Sprite sprite))
        {
            recognizedSpriteRenderer.sprite = sprite;
        }
        else
        {
            recognizedSpriteRenderer.sprite = null;
        }
    }


    private void UpdatePoseNameDisplay()
    {
        if (poseNameText != null)
        {
            poseNameText.text = $"Буква: {GetCurrentPoseName()}";
        }

        UpdateSpriteByPoseIndex();
    }

    public Dictionary<HandJointId, Pose> GetCurrentPoseData()
    {
        string poseName = GetCurrentPoseName();
        return savedPoses.ContainsKey(poseName) ? savedPoses[poseName] : null;
    }

}
```

ПРИЛОЖЕНИЕ Б

Листинг скрипта HandPoseRecognizer

```
using System.Collections.Generic;
using UnityEngine;
using TMPro;
using System.IO;
using Oculus.Interaction.Input;

public class AutoGestureRecognizer : MonoBehaviour
{
    [SerializeField] private Oculus.Interaction.Input.Hand hand;
    [SerializeField] private TextMeshProUGUI feedbackText;
    [SerializeField] private SpriteRenderer recognizedSpriteRenderer;
    [SerializeField] private string spriteSheetName = "Dactil"; // имя без .png

    private Dictionary<string, Dictionary<HandJointId, Pose>> allPoses;

    private readonly HandJointId[] jointsToTrack = new HandJointId[] {
        HandJointId.HandThumb1, HandJointId.HandThumb2,
HandJointId.HandThumb3,
        HandJointId.HandIndex0, HandJointId.HandIndex1, HandJointId.HandIndex2,
HandJointId.HandIndex3,
        HandJointId.HandMiddle0, HandJointId.HandMiddle1,
HandJointId.HandMiddle2, HandJointId.HandMiddle3,
        HandJointId.HandRing0, HandJointId.HandRing1, HandJointId.HandRing2,
HandJointId.HandRing3,
        HandJointId.HandPinky0, HandJointId.HandPinky1, HandJointId.HandPinky2,
HandJointId.HandPinky3,
        HandJointId.HandWristRoot, HandJointId.HandPalm
    };

    private void Start()
    {
        LoadAllPoses();
        InvokeRepeating(nameof(TryRecognizeGesture), 1f, 1f);
    }

    private void LoadAllPoses()
    {
        allPoses = new Dictionary<string, Dictionary<HandJointId, Pose>>();
        string path = Path.Combine(Application.temporaryCachePath,
"default_poses_temp.json");

        if (!File.Exists(path))
        {
            feedbackText.text = "Файл жестов не найден!";
            return;
        }

        string json = File.ReadAllText(path);
```

```
        PoseDataSet poseDataSet = JsonUtility.FromJson<PoseDataSet>(json);

        foreach (var namedPose in poseDataSet.poses)
        {
            var jointDict = new Dictionary<HandJointId, Pose>();

            foreach (var jointData in namedPose.joints)
            {
                if (System.Enum.TryParse(jointData.jointId, out HandJointId jointId))
                {
                    Vector3 position = new Vector3(
                        jointData.pose.position[0],
                        jointData.pose.position[1],
                        jointData.pose.position[2]);

                    Quaternion rotation = new Quaternion(
                        jointData.pose.rotation[0],
                        jointData.pose.rotation[1],
                        jointData.pose.rotation[2],
                        jointData.pose.rotation[3]);

                    jointDict[jointId] = new Pose(position, rotation);
                }
            }

            allPoses[namedPose.poseName] = jointDict;
        }
    }

    private int GetPoseIndex(string poseName)
    {
        string[] russianAlphabet = new string[] {
            "А", "Б", "В", "Г", "Д", "Е", "Ё", "Ж", "З", "И", "Й", "К", "Л", "М",
            "Н", "О", "П", "Р", "С", "Т", "У", "Ф", "Х", "Ц", "Ч", "Ш", "Щ", "Ъ", "Ы", "Ь", "Э",
"Ю", "Я"
        };

        for (int i = 0; i < russianAlphabet.Length; i++)
        {
            if (russianAlphabet[i] == poseName)
                return i;
        }

        return -1;
    }

    public void TryRecognizeGesture()
    {
        if (allPoses == null || allPoses.Count == 0)
        {
            feedbackText.text = "Жесты не загружены!";
```

```
            recognizedSpriteRenderer.sprite = null;
            return;
        }

        if (!hand.GetJointPose(HandJointId.HandPalm, out Pose palmPose))
        {
            feedbackText.text = "Рука не отслеживается!";
            recognizedSpriteRenderer.sprite = null;
            return;
        }

        Dictionary<HandJointId, Pose> currentHand = new();

        foreach (var joint in jointsToTrack)
        {
            if (hand.GetJointPose(joint, out Pose worldPose))
            {
                Vector3 localPos = Quaternion.Inverse(palmPose.rotation) *
(worldPose.position - palmPose.position);
                Quaternion localRot = Quaternion.Inverse(palmPose.rotation) *
worldPose.rotation;
                currentHand[joint] = new Pose(localPos, localRot);
            }
        }

        float posThreshold = 0.04f;
        float rotThreshold = 25f;

        foreach (var poseEntry in allPoses)
        {
            string poseName = poseEntry.Key;
            var savedPose = poseEntry.Value;
            bool match = true;

            foreach (var joint in jointsToTrack)
            {
                if (!currentHand.TryGetValue(joint, out Pose current)) continue;
                if (!savedPose.TryGetValue(joint, out Pose target)) continue;

                float posDiff = Vector3.Distance(current.position, target.position);
                float rotDiff = Quaternion.Angle(current.rotation, target.rotation);

                if (posDiff > posThreshold || rotDiff > rotThreshold)
                {
                    match = false;
                    break;
                }
            }

            if (match)
            {
```

```
            feedbackText.text = $"Показан жест: {poseName}";

            int index = GetPoseIndex(poseName);
            if (index != -1)
            {
                string targetSpriteName = $"Dactil_{index + 1}";

                Sprite[] sprites = Resources.LoadAll<Sprite>(spriteSheetName);

                foreach (var sprite in sprites)
                {
                    if (sprite.name == targetSpriteName)
                    {
                        recognizedSpriteRenderer.sprite = sprite;
                        return;
                    }
                }
            }
            recognizedSpriteRenderer.sprite = null;
            return;
        }
    }
    recognizedSpriteRenderer.sprite = null;
    feedbackText.text = "Жест не распознан";
  }
}
```

ПРИЛОЖЕНИЕ В

Листинг скрипта GameManager

```
using System.Collections.Generic;
using UnityEngine;
using TMPro;
using System.IO;
using Oculus.Interaction.Input;
using UnityEngine.SceneManagement;

public class GameManager : MonoBehaviour
{
    // ==== UI элементы ====
    [Header("UI")]
    public TMP_Text difficultyText; // Текст для отображения уровня сложности
    public TMP_Text wordText;       // Текущий выводимый текст слова
    public TMP_Text inputText;      // Текущий набранный жестами текст
    public TMP_Text timerText;      // Таймер прохождения

    [SerializeField] private GameObject endPanel;         // Панель завершения уровня
    [SerializeField] private TMP_Text finalTimeText;      // Отображение итогового
времени

    // ==== Рука ====
    [Header("Hand")]
    public Hand userHand; // Ссылка на трекаемую руку

    // ==== Параметры распознавания ====
    [Header("Распознавание")]
    public float posThreshold = 0.04f;    // Порог по позиции
    public float rotThreshold = 25f;      // Порог по повороту

    private bool timerRunning = true;

    // // Словарь, с загруженными позами (название → {сустав → поза})
}
    private Dictionary<string, Dictionary<HandJointId, Pose>> loadedPoses = new();

    // Список отслеживаемых суставов
    private readonly HandJointId[] trackedJoints = new HandJointId[]
    {
        HandJointId.HandThumb1, HandJointId.HandThumb2,
HandJointId.HandThumb3,
        HandJointId.HandIndex0, HandJointId.HandIndex1, HandJointId.HandIndex2,
HandJointId.HandIndex3,
        HandJointId.HandMiddle0, HandJointId.HandMiddle1,
HandJointId.HandMiddle2, HandJointId.HandMiddle3,
        HandJointId.HandRing0, HandJointId.HandRing1, HandJointId.HandRing2,
HandJointId.HandRing3,
        HandJointId.HandPinky0, HandJointId.HandPinky1, HandJointId.HandPinky2,
HandJointId.HandPinky3,
```

```
        HandJointId.HandWristRoot, HandJointId.HandPalm
    };

    // Данные для игрового процесса
    private List<string> currentWords = new(); // Список слов текущей сессии
    private string currentWord = "";           // Текущее слово
    private string currentInput = "";          // Текущий ввод пользователя
    private int currentLetterIndex = 0;        // Индекс вводимой буквы
    private int currentWordIndex = 0;          // Индекс слова

    private float timer = 0f;                  // Время прохождения уровня
    private float gestureTimer = 0f;           // Таймер для задержки между
распознаваниями
    private float gestureCheckCooldown = 0.5f; // Пауза между проверками

    void Start()
    {
        // Инициализация состояния и загрузка данных
        currentWords.Clear();
        currentWordIndex = 0;
        currentLetterIndex = 0;
        timer = 0f;

        LoadPosesFromJson();         // Загрузка поз из файла
        LoadWordsByDifficulty();     // Загрузка слов по сложности
        LoadNextWord();              // Начало с первого слова
    }

    void Update()
    {
        if (timerRunning)
        {
            timer += Time.deltaTime;
            timerText.text = "Время: " + timer.ToString("F1") + " сек";
        }

        // Распознавание жестов через промежутки времени
        gestureTimer += Time.deltaTime;
        if (gestureTimer >= gestureCheckCooldown)
        {
            TryRecognizeGesture();
            gestureTimer = 0f;
        }
    }

    // === Загрузка сохранённых жестов из JSON ===
    void LoadPosesFromJson()
    {
        string path = Path.Combine(Application.temporaryCachePath,
"default_poses_temp.json");
```

```
        if (!File.Exists(path))
        {
            Debug.LogError("Файл с позами не найден: " + path);
            return;
        }

        string json = File.ReadAllText(path);
        PoseDataSet dataset = JsonUtility.FromJson<PoseDataSet>(json);

        foreach (var namedPose in dataset.poses)
        {
            var jointDict = new Dictionary<HandJointId, Pose>();
            foreach (var jointData in namedPose.joints)
            {
                if (System.Enum.TryParse(jointData.jointId, out HandJointId id))
                {
                    jointDict[id] = jointData.pose.ToPose();
                }
            }
            loadedPoses[namedPose.poseName.ToUpper()] = jointDict;
        }

        Debug.Log("Загружено поз: " + loadedPoses.Count);
    }

    // === Загрузка слов в зависимости от уровня сложности ===
    void LoadWordsByDifficulty()
    {
        string difficulty = PlayerPrefs.GetString("Difficulty", "Лёгкий");
        difficultyText.text = "Уровень: " + difficulty;
        currentWords.Clear();

        List<string> easy = new() { "дом", "кот", "мир", "нос", "луна" };
        List<string> medium = new() { "окно", "река", "игра", "лист", "вода" };
        List<string> hard = new() { "архив", "платформа", "комплекс", "студент",
"словарь" };

        List<string> source = difficulty switch
        {
            "Средний" => medium,
            "Тяжёлый" => hard,
            _ => easy
        };

        // Выбираем 3 случайных слова
        while (currentWords.Count < 3)
        {
            string word = source[Random.Range(0, source.Count)];
            if (!currentWords.Contains(word))
                currentWords.Add(word);
        }
```

```
    }

    // === Загрузка следующего слова ===
    void LoadNextWord()
    {
        if (currentWordIndex >= currentWords.Count)
        {
            wordText.text = "";
            inputText.text = "";
            ShowEndScreen();
            return;
        }

        currentWord = currentWords[currentWordIndex];
        currentInput = "";
        currentLetterIndex = 0;
        wordText.text = "Слово: " + currentWord.ToUpper();
        UpdateInputDisplay();
    }

    // === Обновление текста ввода на экране ===
    void UpdateInputDisplay()
    {
        inputText.text = currentInput.ToUpper();
    }

    // === Пропустить текущую букву ===
    public void SkipLetter()
    {
        if (currentLetterIndex < currentWord.Length)
        {
            currentInput += currentWord[currentLetterIndex];
            currentLetterIndex++;
            UpdateInputDisplay();

            if (currentLetterIndex >= currentWord.Length)
            {
                currentWordIndex++;
                Invoke(nameof(LoadNextWord), 1f);
            }
        }
    }

    // === Пропустить текущее слово ===
    public void SkipWord()
    {
        currentWordIndex++;
        Invoke(nameof(LoadNextWord), 0.5f);
    }

    // === Распознавание текущего жеста и сравнение с буквой ===
```

```
    void TryRecognizeGesture()
    {
        if (currentWordIndex >= currentWords.Count || currentLetterIndex >=
currentWord.Length)
            return;

        if (!userHand.GetJointPose(HandJointId.HandPalm, out Pose palmPose)) return;

        foreach (var pose in loadedPoses)
        {
            if (IsMatchingPose(pose.Value, palmPose))
            {
                string recognized = pose.Key;
                string expected = currentWord[currentLetterIndex].ToString().ToUpper();

                if (recognized == expected)
                {
                    currentInput += recognized;
                    currentLetterIndex++;
                    UpdateInputDisplay();

                    if (currentLetterIndex >= currentWord.Length)
                    {
                        currentWordIndex++;
                        Invoke(nameof(LoadNextWord), 1f);
                    }
                }

                break;
            }
        }
    }

    // === Отображение экрана завершения ===
    void ShowEndScreen()
    {
        if (endPanel != null)
        {
            endPanel.SetActive(true);
        }

        finalTimeText.text = "Время: " + timer.ToString("F1") + " сек";
        timerRunning = false;
    }

    // === Возврат в главное меню ===
    public void ReturnToMenu()
    {
        SceneManager.LoadScene("Scenes/Main Menu");
    }
```

```
    // === Сравнение текущей позы с эталонной ===
    bool IsMatchingPose(Dictionary<HandJointId, Pose> saved, Pose palmPose)
    {
        foreach (var joint in trackedJoints)
        {
            if (!userHand.GetJointPose(joint, out Pose currentPose)) continue;
            if (!saved.TryGetValue(joint, out Pose savedPose)) continue;

            // Переводим в локальные координаты относительно ладони
            Vector3 localCurrentPos = Quaternion.Inverse(palmPose.rotation) *
(currentPose.position - palmPose.position);
            Quaternion localCurrentRot = Quaternion.Inverse(palmPose.rotation) *
currentPose.rotation;

            float posDiff = Vector3.Distance(localCurrentPos, savedPose.position);
            float rotDiff = Quaternion.Angle(localCurrentRot, savedPose.rotation);

            if (posDiff > posThreshold || rotDiff > rotThreshold)
                return false;
        }

        return true;
    }
}
```

АНТИПЛАГИАТ
ОБНАРУЖЕНИЕ ЗАИМСТВОВАНИЙ

СПРАВКА

о результатах проверки текстового документа
на наличие заимствований

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ образовательное учреждение высшего образования "Саратовский государственный технический университет имени Гагарина Ю.А."

ПРОВЕРКА ВЫПОЛНЕНА В СИСТЕМЕ АНТИПЛАГИАТ.ВУЗ

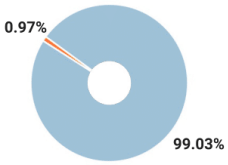
Автор работы: Пчелинцева Елена Германовна
Самоцитирование рассчитано для:
Название работы: 230781_м-ИВЧТ_2025_1
Тип работы: Выпускная квалификационная работа
Подразделение: ИнПИТ

РЕЗУЛЬТАТЫ

■ **ВНИМАНИЕ, ДОКУМЕНТ ПОДОЗРИТЕЛЬНЫЙ:** ОБНАРУЖЕНЫ ПОПЫТКИ МАСКИРОВКИ ЗАИМСТВОВАНИЙ. РЕКОМЕНДУЕТСЯ ПРОВЕРИТЬ ПОЛНЫЙ ОТЧЕТ

СОВПАДЕНИЯ	0.97%
ОРИГИНАЛЬНОСТЬ	99.03%
ЦИТИРОВАНИЯ	0%
САМОЦИТИРОВАНИЯ	0%

ДАТА ПОСЛЕДНЕЙ ПРОВЕРКИ: 02.06.2025



Структура документа: Проверенные разделы: основная часть с.13-15, 17-61, содержание с.1-2, приложение с.66-82, введение с.3-13, выводы с.16-17
Модули поиска: Переводные заимствования; Цитирование; СПС ГАРАНТ: нормативно-правовая документация; Рувики; Коллекция НБУ; Перефразированные заимствования по коллекции Интернет в русском сегменте; Переводные заимствования IEEE; Кольцо вузов (переводы и перефразирования); Диссертации НББ; СПС ГАРАНТ: аналитика; СМИ России и СНГ; Медицина; Патенты СССР, РФ, СНГ; IEEE; Публикации eLIBRARY; Публикации РГБ (переводы и перефразирования); Публикации eLIBRARY (переводы и перефразирования); Шаблонные фразы; Сводная коллекция ЭБС; ИПС Адилет; Публикации РГБ; Перефразирования по СПС ГАРАНТ: аналитика; Кольцо вузов; Переводные заимствования по коллекции Гарант: аналитика; Перефразирования по коллекции IEEE; Переводные заимствования по коллекции Интернет в русском сегменте; Перефраз...

Работу проверил: Пчелинцева Елена Германовна

ФИО проверяющего

Дата подписи:

Подпись проверяющего

Чтобы убедиться
в подлинности справки, используйте QR-код,
который содержит ссылку на отчет.

Ответ на вопрос, является ли обнаруженное заимствование корректным, система оставляет на усмотрение проверяющего. Предоставленная информация не подлежит использованию в коммерческих целях.