Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Робототехника и комплексная автоматизация (РК)

КАФЕДРА Системы автоматизированного проектирования (РК6)

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ

НА ТЕМУ:

«Разработка виртуального помощника на Unreal Engine 5»

Студент РК6-82Б _____ **Редколис Р.Р.**
(Подпись, дата) И.О. Фамилия

Руководитель _____ **Витюков Ф.А.**
(Подпись, дата) И.О. Фамилия

2024 г.

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

УТВЕРЖДАЮ
Заведующий кафедрой РК6
А.П. Карпенко

«_____» _____ 2024 г.

З А Д А Н И Е
на выполнение научно-исследовательской работы

по теме: Разработка виртуального помощника на Unreal Engine 5
Студент группы __РК6-82Б___

_____Редколис Руслан Русланович_____
(Фамилия, имя, отчество)

Направленность НИР (учебная, исследовательская, практическая, производственная, др.) _учебная___
Источник тематики (кафедра, предприятие, НИР) _кафедра_____

График выполнения НИР: 25% к 5 нед., 50% к 11 нед., 75% к 14 нед., 100% к 16 нед.

Техническое задание: Изучение видов компьютерного моделирования. Создание модели и её анимации на основе одного из вида компьютерного моделирования. Перенос модели между программными средствами._____

Оформление научно-исследовательской работы:
Расчетно-пояснительная записка на 30 листах формата А4.
Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.):
_____
_____
_____
_____
_____
Дата выдачи задания «7» февраля 2024 г.

Руководитель НИР _____ **Витюков Ф.А.**
(Подпись, дата) И.О. Фамилия

Студент _____ **Редколис Р.Р.**
(Подпись, дата) И.О. Фамилия

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современном мире технологий виртуальная реальность и виртуальные помощники становятся все более распространенными и значимыми. Развитие компьютерных технологий и искусственного интеллекта открывает новые возможности для создания реалистичных и функциональных виртуальных персонажей, которые могут помогать пользователям в самых различных сферах жизни.

Целью данной научно-иследовательской работы является разработка и реализация модели виртуального помощника на платформе Unreal Engine 5 с использованием современных технологий моделирования, анимации и искусственного интеллекта. Этот проект представляет собой исследование процесса создания и функционирования виртуальных персонажей в контексте современных технологий виртуальной реальности.

В работе будет проведен анализ существующих методов и инструментов разработки виртуальных ассистентов, проектирование и реализация модели виртуального помощника, а также анализ результатов и возможных перспектив применения разработанного виртуального помощника. Для достижения поставленной цели мы рассмотрим архитектуру разработки в Unreal Engine 5, методы восприятия речи speech-to-text и её воспроизведения text-to-speech виртуальным персонажем, а также применим современные методы искусственного интеллекта для создания функционального и реалистичного виртуального помощника. Данная работа представляет собой важный вклад в развитие области виртуальной реальности и искусственного интеллекта, а также открывает новые перспективы для применения виртуальных помощников в различных сферах человеческой деятельности.

В данной работе рассматривается создание и разработка элементов пользовательского интерфейса для компьютерной коллекционной карточной игры. Работа делится на три основные части:

1. Создание цифрового аватара.

В данной части работы рассматривается процесс создания и настройки персонажа виртуального помощника с использованием технологий Metahuman, Quixel Bridge и Unreal Engine. Описывается выбор и настройка персонажа, его импорт в игровой движок, исправление ошибки отображение растительности.

2. Разработка системы обработки голоса.

В данной части работы рассматривается процесс обработки голоса человека и воспроизведения речи MetaHuman персонажем с использованием современных технологий распознавания и синтеза речи. Описывается выбор и настройка инструментов для обработки звука, синтеза речи и интеграции с созданным виртуальным персонажем.

1. Создание цифрового аватара

MetaHuman Creator - это облачное приложение, разработанное компанией Epic Games, предназначенное для создания цифровых персонажей в реальном времени с минимальными затратами времени и без потери качества. Позволяет создавать уникальных персонажей менее чем за час и использовать их непосредственно в Unreal Engine.

Особенности MetaHuman Creator:

Создание персонажей в реальном времени: Пользователи могут создавать своих собственных уникальных персонажей MetaHuman и настраивать их внешний вид в реальном времени без необходимости в дополнительных инструментах или навыках моделирования.

Интеграция с Unreal Engine: Созданные персонажи могут быть непосредственно загружены и использованы в Unreal Engine для создания игровых сцен, анимации и взаимодействия с игровым миром.

Возможность редактирования в сторонних приложениях: Разработчики также могут экспортировать созданных персонажей в сторонние программы для дальнейшего редактирования, такие как Autodesk Maya, чтобы дополнительно настроить их внешний вид или добавить новые элементы.

Использование готовых моделей на Quixel Bridge: Пользователи также могут воспользоваться готовыми моделями, доступными на платформе Quixel Bridge. Это позволяет значительно ускорить процесс создания и не тратить время на моделирование с нуля.

Применение в научно-исследовательской работе:

В процессе научно-исследовательской работы MetaHuman Creator будет использован для создания виртуальных персонажей, которые будут дополнительно анимированы и использованы для реализации конкретных задач, связанных с виртуальными помощниками в Unreal Engine.

1.1. Создание проекта

При создании проекта необходимо выбрать соответствующие преднастройки для оптимальной производительности и управления ресурсами.

Отключение Ray Tracing: В настройках проекта следует отключить опцию Ray Tracing для повышения производительности программы. Это можно сделать при создании проекта или в настройках проекта после его создания.

Удаление ненужных материалов из Starter Content: При создании проекта следует убрать галку с Starter Content, чтобы избежать лишнего увеличения размера проекта ненужными материалами.

Установка проекта на C++: Важно выбрать опцию установки проекта на языке программирования C++, так как это позволит в дальнейшем написать и интегрировать необходимый код запроса и получения данных на основе API.

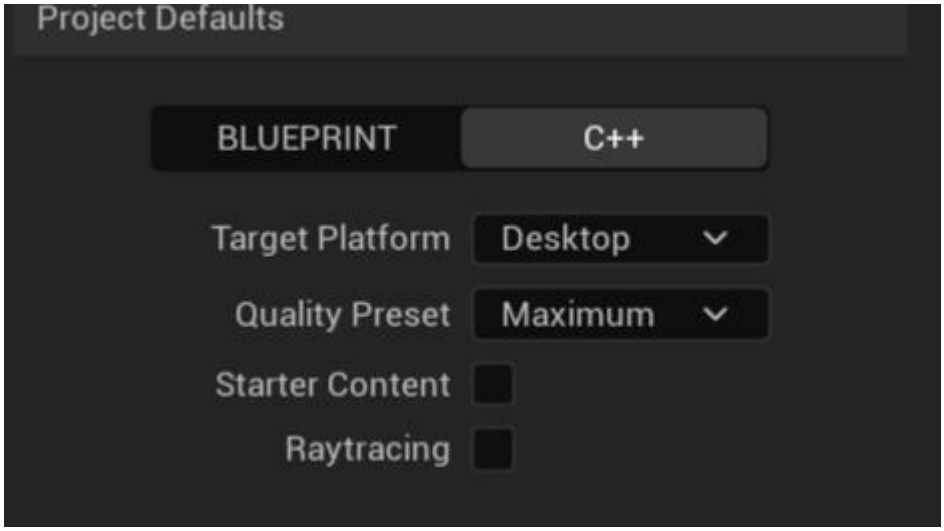


Рисунок 1 Найстроки проекта

1.2. Создание персонажа в облачном сервисе MetaHuman Creator

Для предотвращения проблем, например отсутствием волос, с несоответствием версий между персонажем и Unreal Engine использовано MetaHuman Creator. После авторизации в MetaHuman Creator явно выбираем соответствующую версию Unreal Engine.

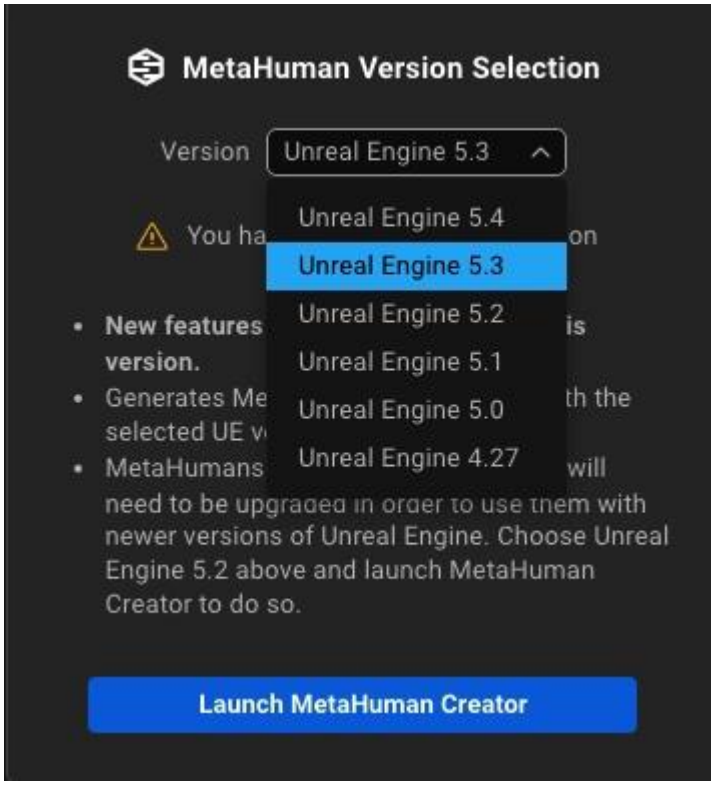


Рисунок 2 выбор версии под проект

Модель созданная в MetaHuman Creator автоматически появится в Quixel Bridge, откуда её можно импортировать в проект.

1.3. Импорт персонажа в Unreal Engine через Quixel Bridge

Quixel Bridge - это удобное приложение, которое облегчает работу с разнообразными материалами от Quixel Megascans. Оно предоставляет возможность просматривать, искать, загружать, импортировать и экспортировать ресурсы без необходимости оплаты. Даже в случае использования этих ресурсов в коммерческих проектах, пользователю не требуется оплачивать их, при условии, что они используются в рамках проекта,

созданного на Unreal Engine, для его использования требуется установить плагин Quixel Bridge.

Для переноса уже готового персонажа из Quixel Bridge в Unreal Engine сначала необходимо установить и подключить расширение MetaHumans и сам Quixel Bridge через Epic Games Store.

В самом проекте открыть вкладку Quixel Bridge, выбрать My MetaHumans и выбрать созданного в облачном сервисе персонажа.

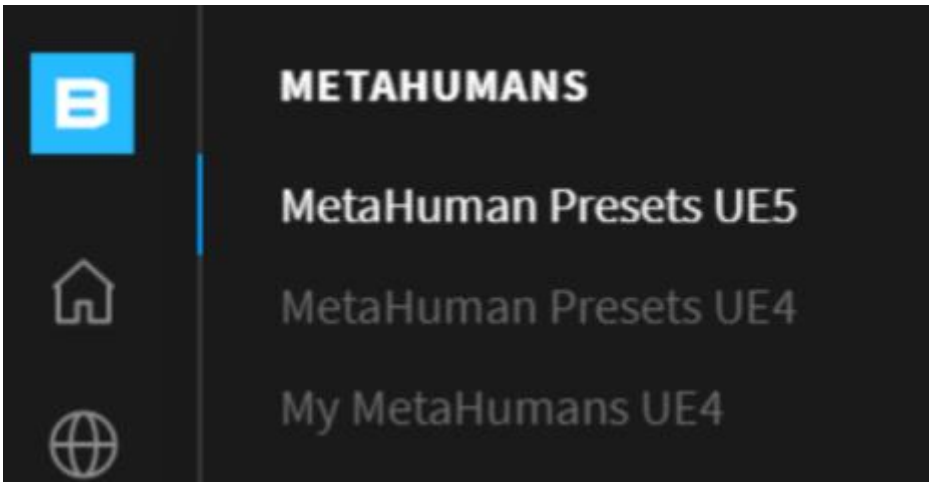


Рисунок 3 Quixel Bridge вкладка MetaHumans

После добавления персонажа у нас появляются в содержании проекта данные об персонаже, текстуры, модель, их привязки и BluePrint.

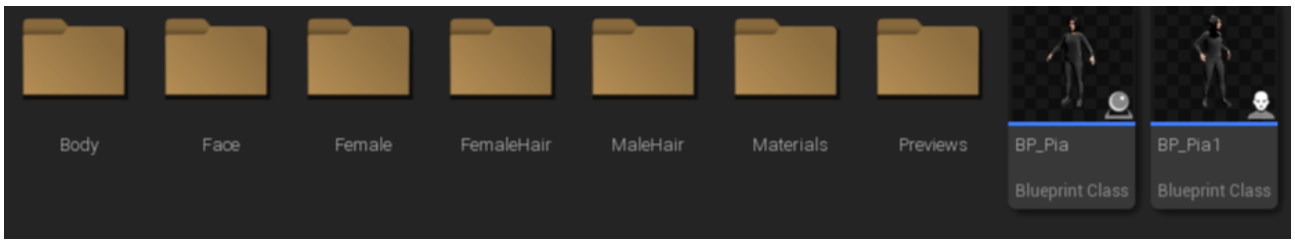


Рисунок 4 среда файлов проекта

В контексте Unreal Engine, "blueprint" — это графическое представление логики и поведения объекта или приложения. Он позволяет создавать функциональность без необходимости писать код с нуля. Вместо этого вы можете использовать визуальный интерфейс для соединения различных элементов (называемых нодами) и создания логики. Это особенно полезно для тех, кто не имеет опыта в программировании, но хочет создавать интерактивные

проекты. Однако при необходимости можно дополнить кодом C++, что и будет реализовано в дальнейших пунктах.

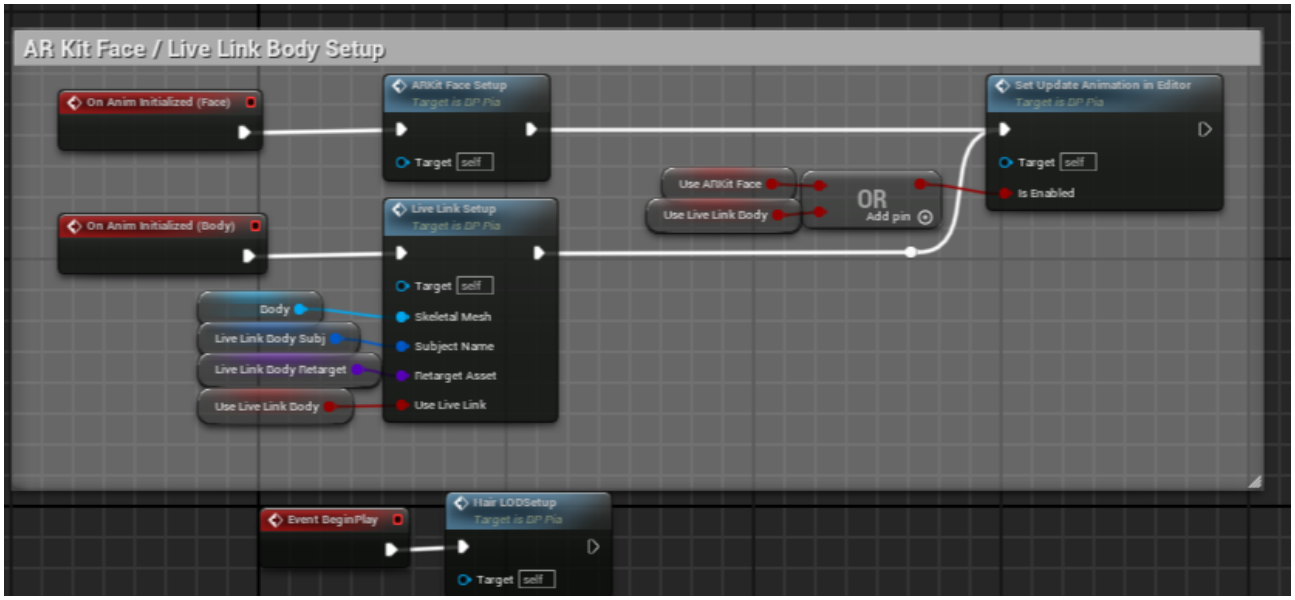


Рисунок 5 blueprint набор привязок тела и головы

Данный Blueprint представляет собой набор настроек для интеграции Metahuman-персонажа с ARKit для распознавания лица и Live Link для анимации тела.

AR Kit Face Setup (Набор ARKit для лица): Этот набор настроек позволяет Metahuman-персонажу взаимодействовать с ARKit для распознавания и отслеживания движений лица пользователя.

Live Link Body Setup (Настройка тела для Live Link): Этот набор настроек позволяет Metahuman-персонажу использовать Live Link для анимации тела в реальном времени.

1.4. Добавление персонажа на сцену

Для переноса персонажа на сцену требуется в среде Unreal Engine открыть Content Drawer, который содержит все доступные ресурсы проекта. Перенести blueprint персонажа на сцену.

Рисунок 6 персонаж в виртуальной среде

Отсутствие волос, возможна из-за нескольких проблем, из-за проблемы версионности или технического ограничения, первое исключено так как версия была указано явно.

Создание реалистичных волос требует значительных вычислительных ресурсов и специализированных технологий, таких как симуляция физики волос и отражение освещения. У таких персонажей выставлено предупреждение на этот случай.

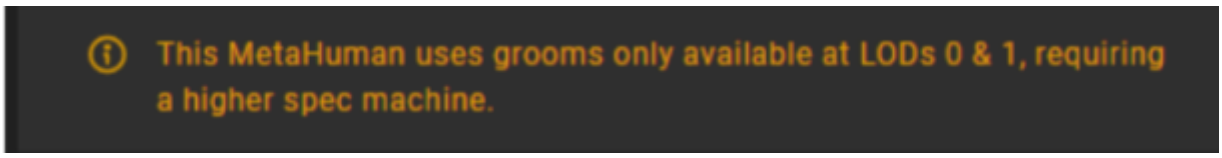


Рисунок 7 Предупрждение к персонажу

Для исправления отображения волос необходимо отредактировать параметр внутри blueprint персонажа, которая находится в среде проекта.

Далее требуется перейти в компоненты модели и выбрать LODSync

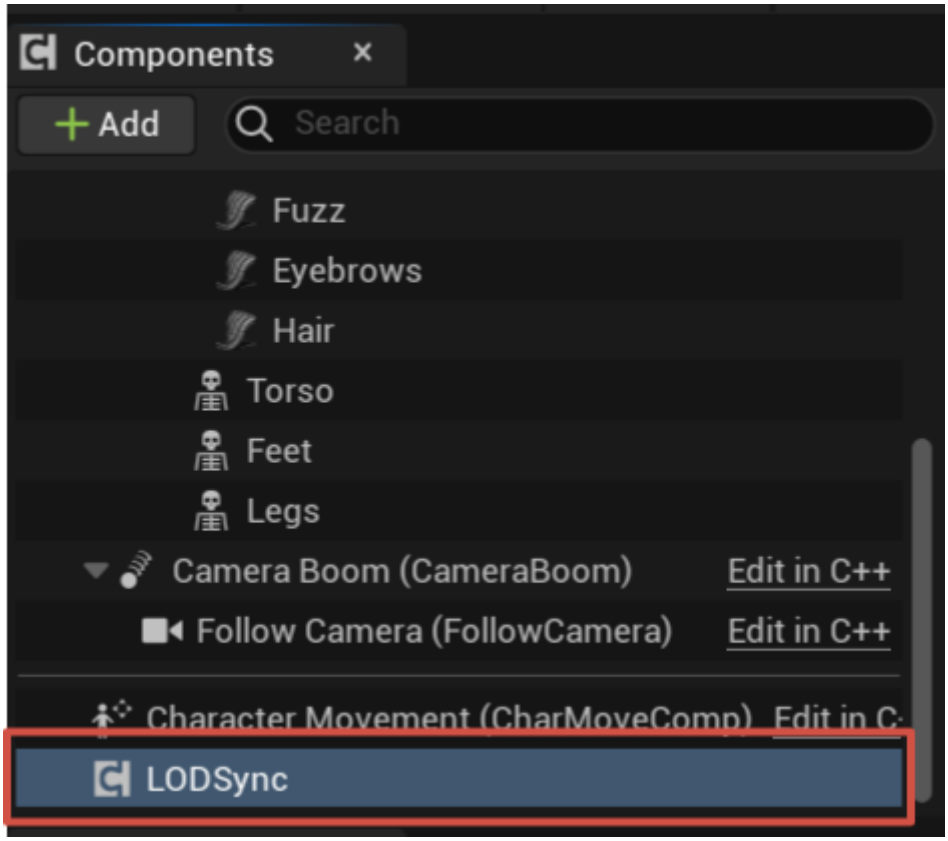


Рисунок 8 компонент детализации

В данной ветке, во вкладке Details (Рисунок 9), необходимо найти раздел, отвечающий за настройку уровней детализации (LOD) персонажа. Обычно этот раздел располагается справа от окна редактирования персонажа. Внутри этого раздела найдите настройку "Forced LOD" или аналогичную, которая контролирует использование уровней детализации. Измените значение этой настройки с "-1" на "0". Это означает, что будет использоваться только уровень детализации 0, без применения других уровней. После внесения изменений

сохраните их, чтобы они вступили в силу. Таким образом, установив значение "Forced LOD" на 0, проблема с пропаданием волос у персонажа должна быть решена.

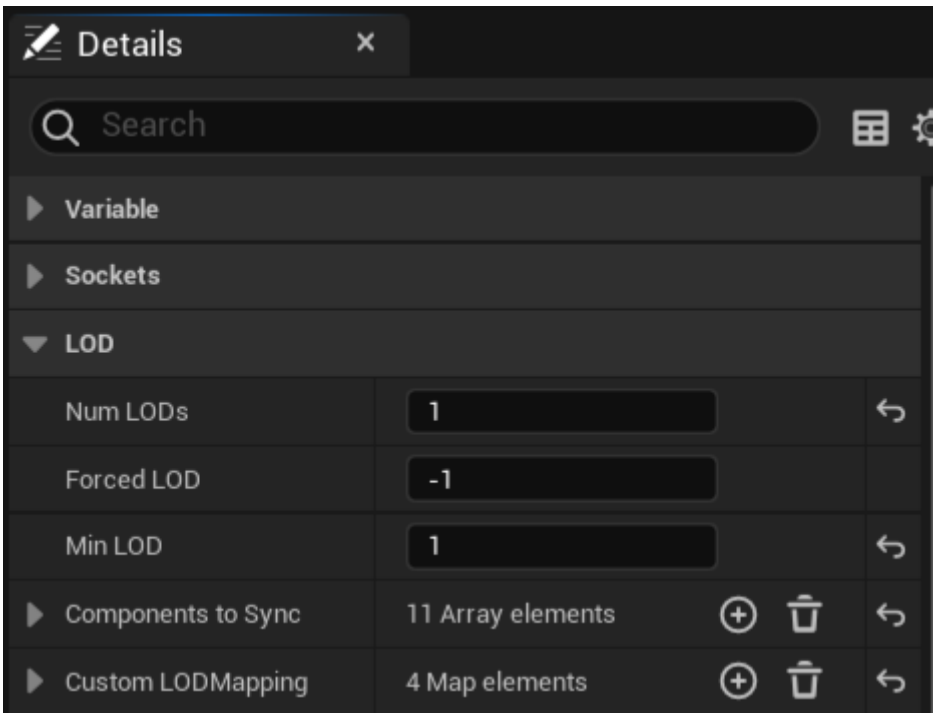


Рисунок 9 Окно настроек детализации

В результате на сцене у персонажа появляется волосы (Рисунок 10).

Рисунок 10 результат изменения параметра

2. Разработка системы обработки голоса пользователя

В современном мире технологии обработки голоса становятся все более значимыми и востребованными в различных областях. Они находят применение не только в сфере развлечений, но и в медицине, образовании, бизнесе и других отраслях. Системы, способные преобразовывать речь в текст и обратно, открывают новые возможности для взаимодействия человека с компьютерными системами и устройствами. Они позволяют управлять устройствами голосом, диктовать текст, создавать голосовых помощников и многое другое.

В данном контексте разработка системы обработки голоса в текст для Unreal Engine 5 представляет собой важную задачу, которая имеет широкий спектр применений. Такая система может использоваться для создания голосовых интерфейсов, автоматизации рутинных задач, анализа аудиоданных, контроля качества речи и других целей. Важным аспектом является выбор подходящих сервисов для обработки голоса, таких как Whisper, Google Cloud Speech-to-Text, Microsoft Azure Speech Services и другие. Эти сервисы предоставляют широкий набор инструментов для распознавания речи с высокой точностью и скоростью, что делает их идеальным выбором для реализации подобных систем.

Для Unreal Engine уже существуют различные плагины и инструменты, которые облегчают интеграцию систем распознавания речи. Например, плагин AzSpeech предоставляет возможность интегрировать Azure Speech Cognitive Services непосредственно в Unreal Engine. Это позволяет использовать возможности распознавания и синтеза речи, предоставляемые Azure, для создания интерактивных голосовых интерфейсов и других приложений.

Кроме того, существует плагин RuntimeSpeechRecognizer, который основан на open-source библиотеке Whisper. Этот плагин также предоставляет функции распознавания речи в реальном времени, что позволяет разработчикам

создавать голосовые управляемые приложения и взаимодействовать с виртуальным миром через голосовые команды.

Плагин RuntimeSpeechRecognizer был выбран в качестве инструмента для системы обработки голоса в Unreal Engine в связи с его рядом преимуществ. В отличие от некоторых других плагинов, RuntimeSpeechRecognizer не требует дополнительного подключения к облачным сервисам, таким как Azure или Google Cloud, и может работать автономно, не завися от доступности интернета. Это обеспечивает большую гибкость и независимость в использовании, особенно в случаях, когда надежный доступ к интернету может быть ограничен или нежелателен.

Кроме того, плагин RuntimeSpeechRecognizer работает offline, что позволяет ему быстро обрабатывать речевые команды на локальном устройстве без необходимости передачи данных в облако и обратно. Это значительно ускоряет процесс обработки голоса и снижает задержки, повышая отзывчивость системы. Благодаря использованию собственных вычислительных мощностей, плагин может эффективно обрабатывать голосовые команды прямо на устройстве пользователя, что делает его более удобным и эффективным в реальном мире. Исходный код плагина RuntimeSpeechRecognizer доступен для просмотра и модификации на платформе GitHub, что позволяет разработчикам адаптировать его под свои потребности и вносить улучшения в соответствии с требованиями своего проекта. Также имеется подробная документация с плагином и возможность напрямую связаться с его создателем для получения поддержки или дополнительной информации.

2.1. Добавление плагина

Данный плагин плагин устанавливается двумя способами.

1. Через Epic Store Marketplace рисунок 11.

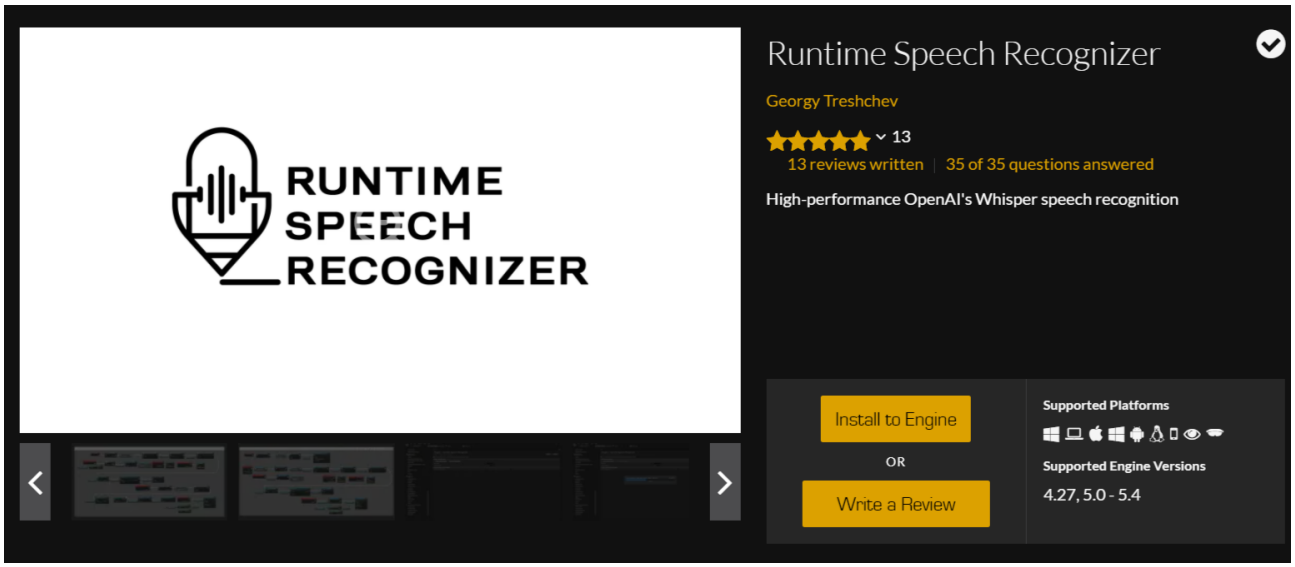


Рисунок 11 установка плагина через сервис

2. Мануально, клонируя репозиторий с плагином через GitHub Desktop рисунок 12.

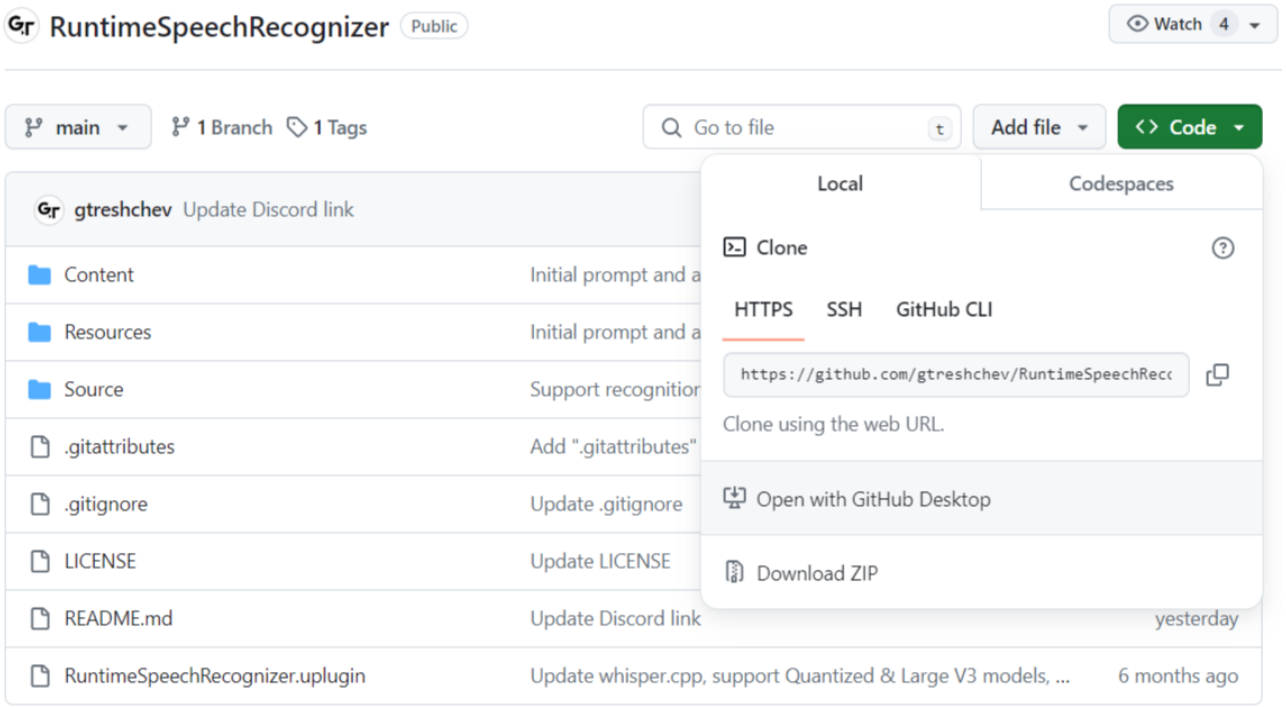


Рисунок 12 Репозиторий автора плагина

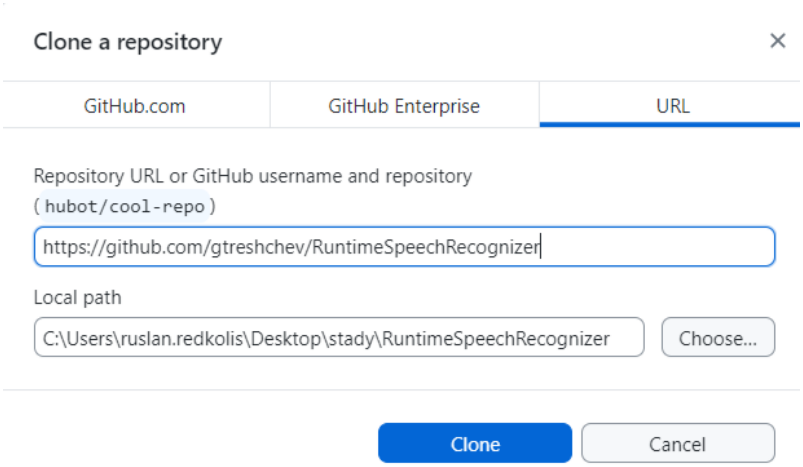


Рисунок 13 клонирование репозитория

Далее требуется импортировать репозиторий в папку проекта.

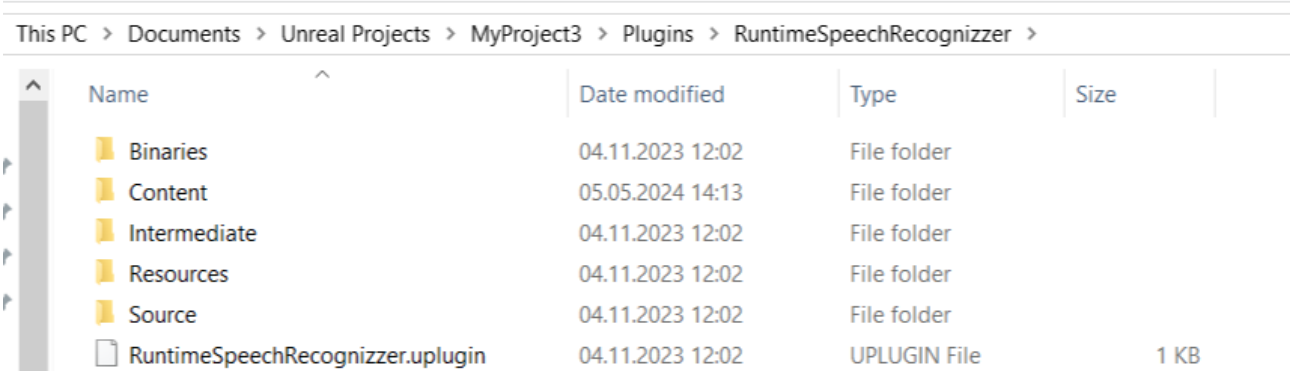


Рисунок 14 импорт плагина в папку проекта

В версии плагина Marketplace могут возникнуть проблемы с размещением ресурса языковой модели из-за проблемы с поиском ресурса, указанного в DirectoriesToAlwaysCook (Дополнительных каталогах ресурсов для редактирования в редакторе) для модулей, расположенных в папке engine, что может привести к невозможности использования плагина в пакетной сборке. Эта проблема, связанная с Marketplace. Поэтому настоятельно рекомендуется использовать способ установки из GitHub.

2.2. Установка параметров

Плагин считывает звуковой сигнал с микрофона или другого источника аудио в реальном времени и передает его на обработку. С помощью алгоритмов машинного обучения и нейронных сетей, встроенных в библиотеку Whisper, происходит анализ звукового сигнала и определение содержащейся в нем речи. Полученный текст затем передается в приложение Unreal Engine для дальнейшей обработки и выполнения необходимых действий на основе распознанной команды.

В контексте библиотеки Whisper, размер модели (model size) играет важную роль в процессе распознавания речи. Больший размер модели обычно связан с увеличением точности распознавания, так как большие модели содержат больше параметров и слоев, что позволяет им лучше улавливать нюансы в аудиосигнале, такие как различные акценты, диалекты и фоновые шумы. Однако, увеличение размера модели может повлечь за собой и увеличение вычислительной нагрузки и времени обработки аудиосигнала, что может быть неприемлемым в ситуациях, где требуется высокая скорость обработки или на устройствах с ограниченными ресурсами. Поэтому при выборе модели для системы распознавания речи важно учитывать баланс между точностью, скоростью и ресурсоемкостью, чтобы обеспечить оптимальную производительность и качество работы системы.

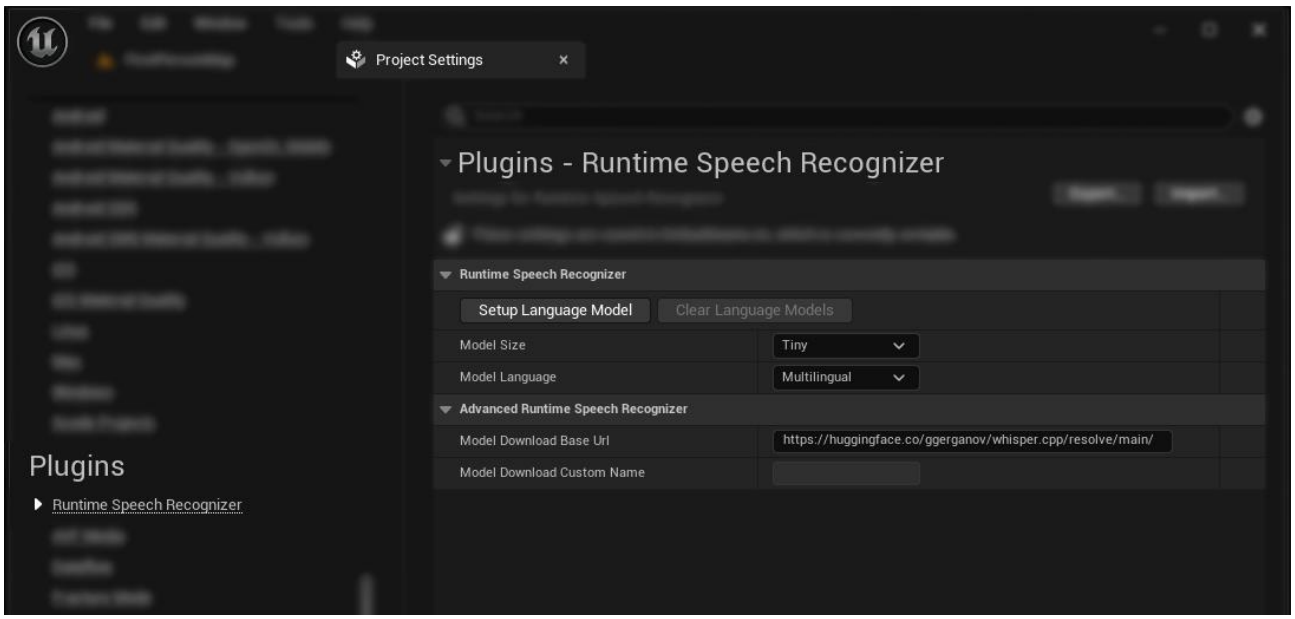
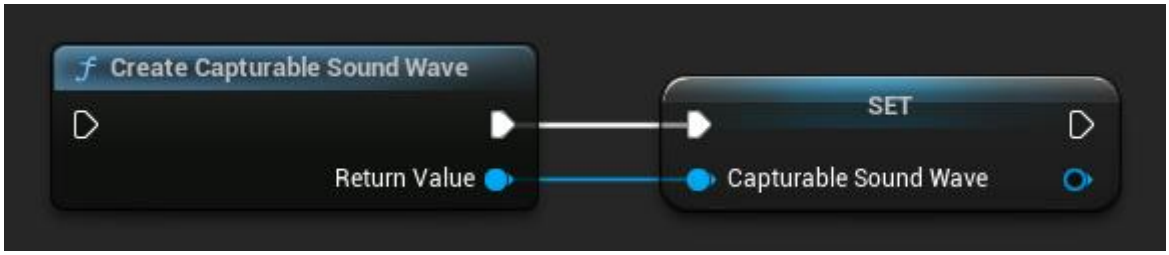


Рисунок 15 выбор размера модели

В дальнейшем исследовании была выбрана модель "tiny multilingual" (Рисунок 15) для распознавания речи на различных языках, среди которых будет явно указан русский язык. Модель "tiny" была выбрана из-за своей эффективности и компактного размера, который позволяет обеспечить достаточную точность и скорость распознавания речи, несмотря на ограниченные ресурсы вычислительной мощности. Это обусловлено тем, что модель "tiny" оптимизирована для работы на устройствах с ограниченными ресурсами и обеспечивает приемлемое качество распознавания речи даже при небольшом объеме модели. Таким образом, модель "tiny multilingual" позволяет эффективно и точно распознавать речь на русском языке, а также на других языках, что делает ее подходящим выбором для данного исследования. Дополнительно было указано из какого источника будет установлена модель.

В плагине присутствуют два основных класса. Первый - CapturableSoundWave, предназначен для захвата аудиоданных с устройств ввода, таких как микрофон, и последующего воспроизведения. Этот класс обладает возможностью захвата аудиоданных с различных источников и их воспроизведения с использованием тех же функций, что и импортированные

звуковые волны. Он обеспечивает такие функциональности, как перемотка, использование в звуковых репликах и другие. На рисунке 15



Второй класс - SpeechRecognition, предназначен для обработки аудиоданных с целью распознавания речи. Этот класс позволяет распознавать и интерпретировать аудиоданные, преобразуя их в текстовый формат. Он обеспечивает функциональность распознавания речи с использованием различных алгоритмов и моделей, что позволяет понимать и обрабатывать речевые команды или ввод пользователя.

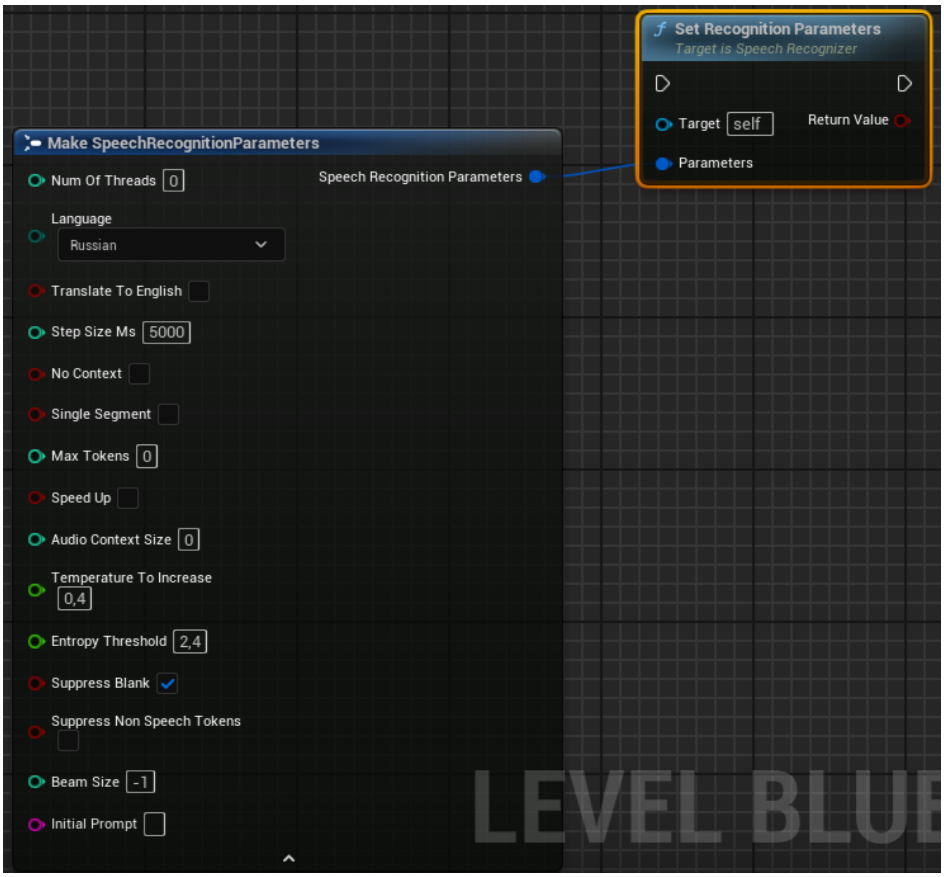


Рисунок 16 параметры класса speechrecognizer

num of threads: Этот параметр указывает количество потоков, которые будут использоваться для обработки аудиоданных и выполнения алгоритмов распознавания речи. Большее количество потоков может повысить скорость обработки, особенно на многоядерных процессорах.

step size ms: Этот параметр определяет размер шага (в миллисекундах) при разбиении аудиофайла на небольшие фрагменты для обработки. Меньший размер шага обычно повышает точность распознавания, но требует больше вычислительных ресурсов.

max tokens: Этот параметр указывает максимальное количество токенов (слов или фраз), которые могут быть распознаны в рамках одного запроса. Увеличение этого значения может улучшить распознавание длинных фраз или текстов, но может также увеличить объем потребляемых ресурсов.

audio context size: Этот параметр определяет размер контекста аудиоданных, используемого для распознавания речи. Больший размер контекста может помочь улучшить точность распознавания, особенно в случаях, когда контекст играет важную роль в понимании речи.

temperature to increase: Этот параметр используется для управления "температурой" в алгоритме генерации токенов. Увеличение этого значения может увеличить вероятность генерации менее вероятных токенов, что может быть полезно для улучшения разнообразия распознаваемых фраз.

entropy threshold: Этот параметр задает порог энтропии для фильтрации распознаваемых токенов. Токены с энтропией ниже этого порога будут отфильтрованы. Это может помочь улучшить качество распознавания, исключая менее вероятные варианты.

suppress blank: Этот параметр указывает, должны ли пустые (бланковые) токены быть подавлены в результате распознавания. Подавление бланковых токенов может быть полезно для улучшения читаемости и точности результата.

beam size: Этот параметр определяет размер луча в алгоритме поиска наилучшего пути в распознавании речи. Больший размер луча позволяет

рассматривать больше альтернативных вариантов при распознавании, что может улучшить качество распознавания, но требует больше ресурсов.

Значения 0 в некоторых параметрах означает неограниченное или максимальное количество из возможных.

2.3. Оптимизация производительности.

При запуске захватываемой звуковой волны с использованием функции StartCapture возможно возникновение краткой задержки, которая является специфичной для движка и в настоящее время не может быть устранена без модификации кода, специфичного для движка. Продолжительность этой задержки различается в зависимости от платформы, так как она связана с выполнением платформо-специфичного кода для извлечения аудиоданных с устройства ввода (микрофона).

Для минимизации этой задержки запуск захвата звука, путем вызова функции StartCapture, начинается по зажатию клавиши ‘1’ с выдаваемым сообщением о старте записи (Рисунок 17), а по отпуску клавиши об завершении записи.

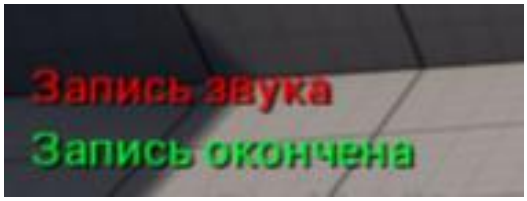


Рисунок 17 результат нажатия клавиши

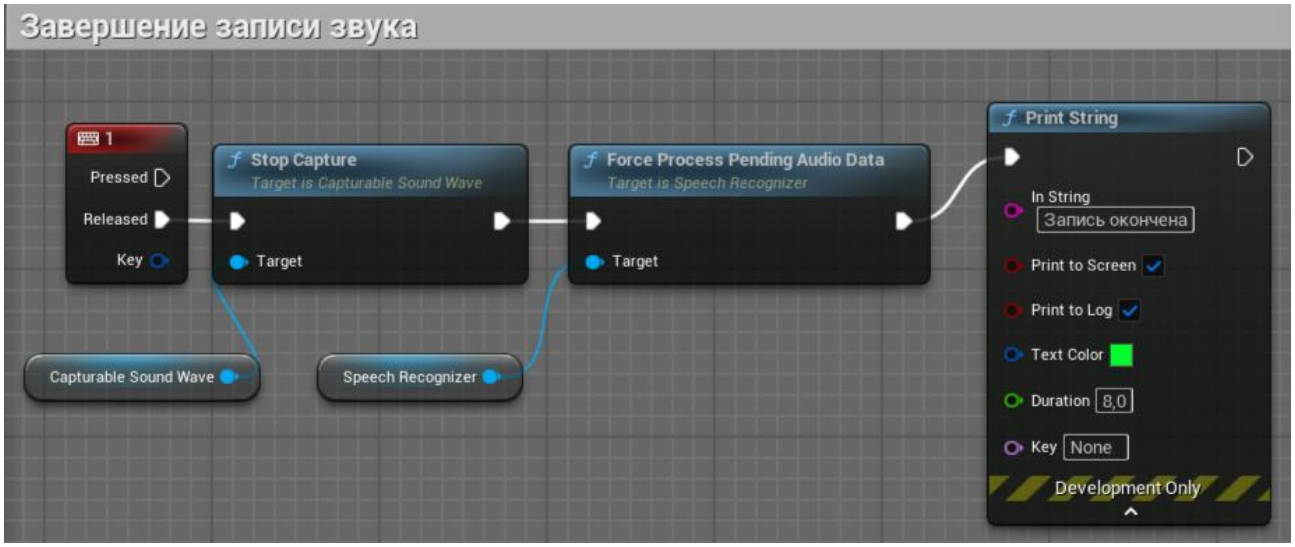


Рисунок 18 логика остановки записи звука

Вызывая StartSpeechRecognition, происходит небольшая задержка из-за загрузки языковой модели. Несмотря на то, что этот процесс предназначен для выполнения асинхронно и запускается в отдельном потоке, движок все еще внутренне выполняет определенные операции на игровом потоке, что приводит к заметному небольшому промежутку, особенно при использовании крупных активов, таких как языковые модели.

Для устранения этой задержки придерживайтесь того же принципа, описанного выше: вызывайтся StartSpeechRecognition в момент окончания записи звука (рисунок 19).

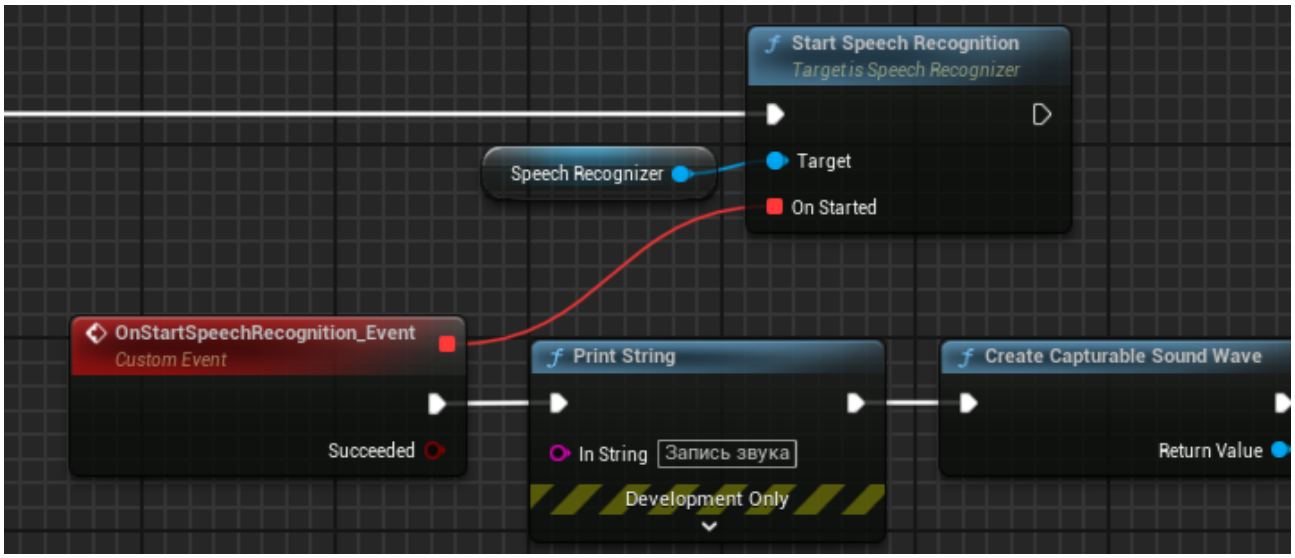


Рисунок 19 Логика запуска функции распознания речи

2.4. Запись и вывод переменной SpeechToText.

В данном разделе рассматривается процесс записи и вывода переменной в среде разработки Unreal Engine с использованием языка программирования C++. Для создания интерактивных и динамических приложений часто требуется возможность взаимодействия с переменными, а также их отображение для пользователя. Unreal Engine предоставляет разработчикам удобные инструменты для работы с переменными и их вывода на экран. Для этого требуется создать новый класс GameInstance, рисунок 20.

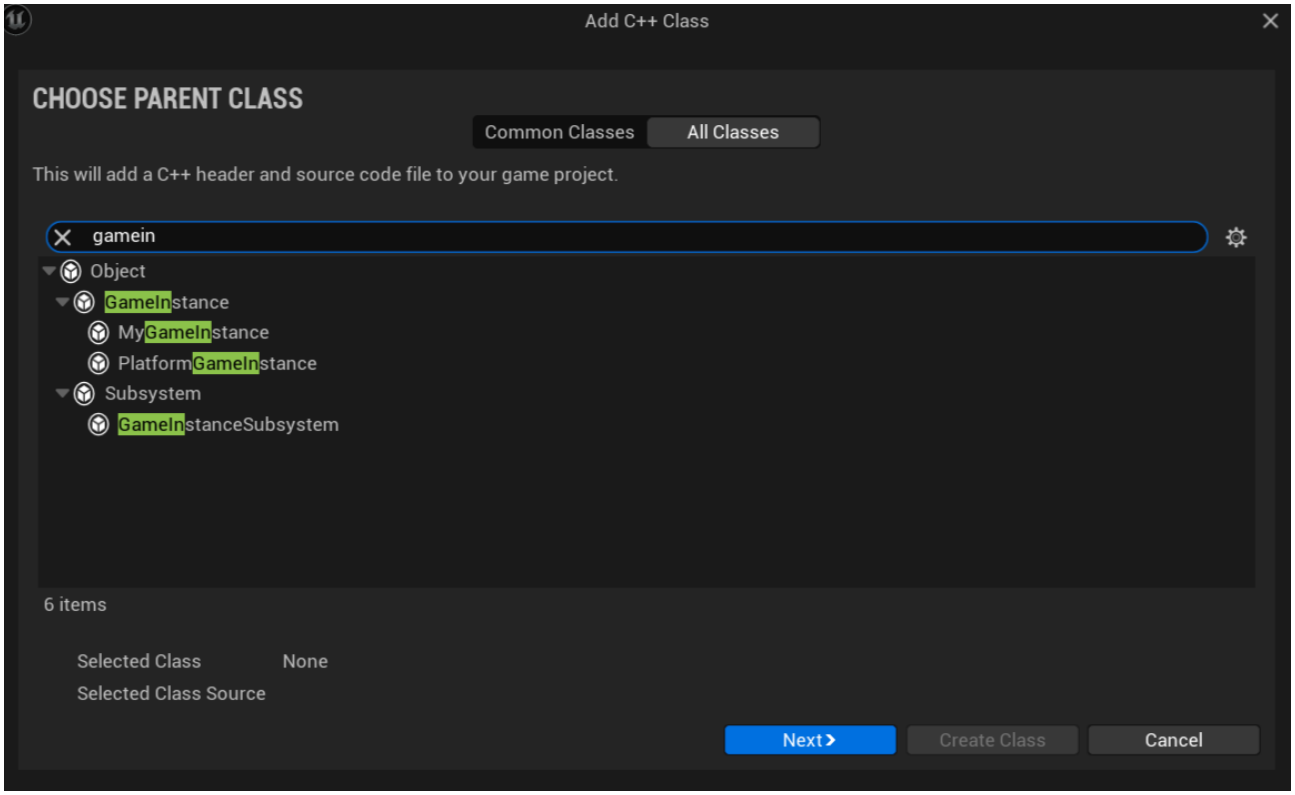


Рисунок 20 Класс GameInstance

GameInstance в Unreal Engine является ключевым компонентом, обеспечивающим хранение глобальных данных и функционала, доступных в течение всего времени жизни игры. Его роль заключается в создании единого хранилища данных и функций, которые могут быть доступны для всех уровней и объектов в игре без необходимости повторной инициализации, его нужно установить в настройках проекта.

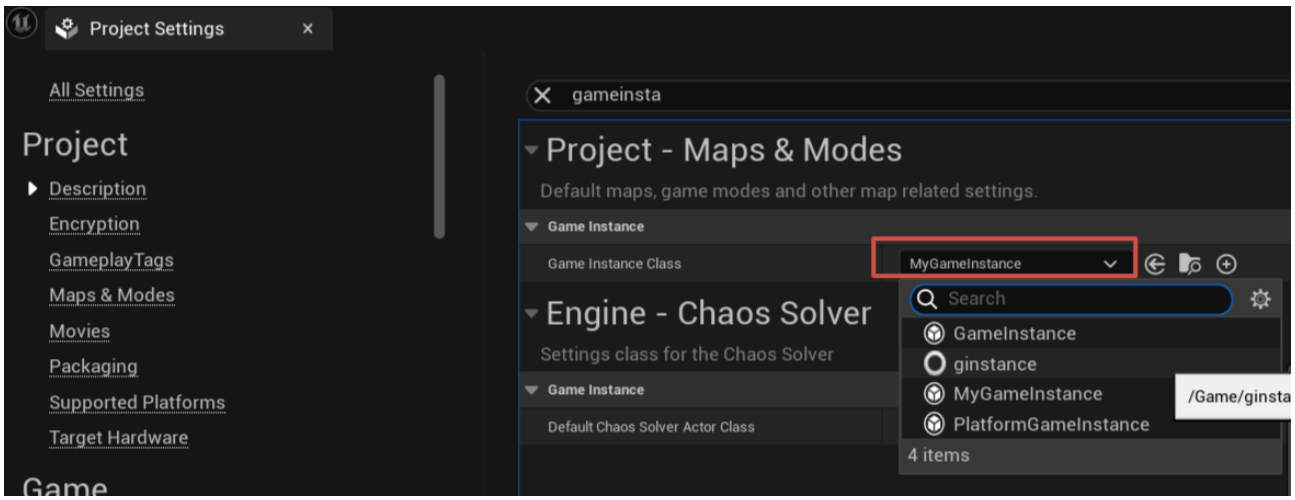


Рисунок 21 Найстрока GameInstance

В заголовочном файле нашего проекта мы объявили класс UMyGameInstance, который содержит приватное поле типа FString, предназначенное для хранения строкового значения, в классе используются макросы UFUNCTION, которые позволяют экспортировать функции в Blueprint для использования в Unreal Engine. В области публичных членов класса мы определили три функции на рисунке 22:

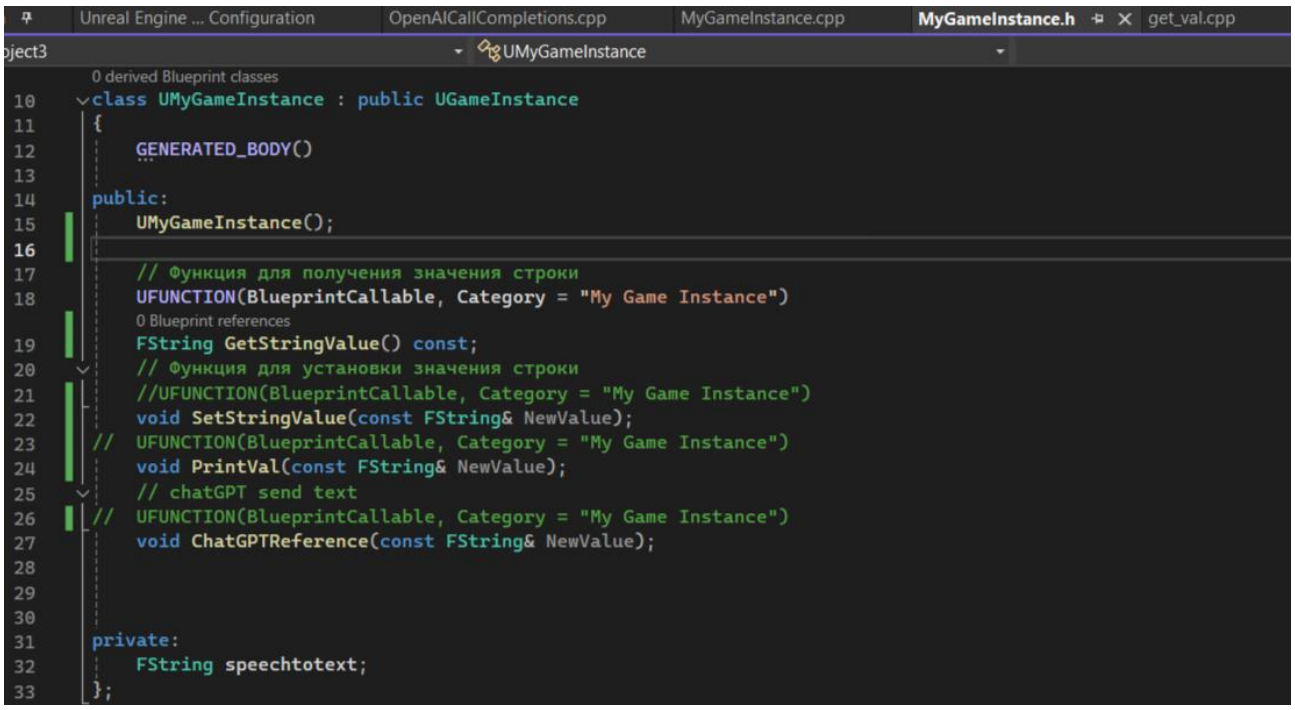


Рисунок 22 Заголовочный файл

GetStringValue(): эта функция, помеченная макросом UFUNCTION(BlueprintCallable), позволяет получить текущее значение строки из

приватного поля класса. Она возвращает строковое значение типа FString и доступна для вызова из Blueprint в Unreal Engine.

SetStringValue(): данная функция, также помеченная макросом UFUNCTION(BlueprintCallable), предназначена для установки нового значения строки в приватное поле класса. Она принимает в качестве аргумента новое значение типа FString.

PrintVal(): эта функция представленная в коде ниже (рисунок 23), она используется для вывода значения строки в консоль . Она не изменяет состояние объекта и не возвращает значений. Эта функция была помечена макросом UFUNCTION, что означает, что она экспортируется в Blueprint.

Таким образом, класс UMyGameInstance предоставляет интерфейс для управления строковым значением и его вывода без необходимости напрямую обращаться к приватному полю класса.

```
void UMyGameInstance::PrintVal(const FString& NewValue)
{
    UMyGameInstance* MyGameInstance = Cast<UMyGameInstance>(UGameplayStatics::GetGameInstance(this));
    FString SpeechToText = MyGameInstance->GetStringValue();
    UE_LOG(LogTemp, Warning, TEXT("Recognized text: %s"), *SpeechToText);

}
```

Рисунок 23 Реализация вывода в консоль

В данном коде используется функция UGameplayStatics::GetGameInstance(), чтобы получить экземпляр игрового объекта, а затем с помощью приведения типа (Cast<UMyGameInstance>) получить доступ к методу GetStringValue(), который возвращает текущее значение строки. Далее результат значения строки выводится в консоль (Рисунок 24) с помощью функции UE_LOG с указанием уровня предупреждения (Warning).

Визуализации логики на Blueprint представлена на рисунке 24, здесь можно увидеть следующую последовательность действий:

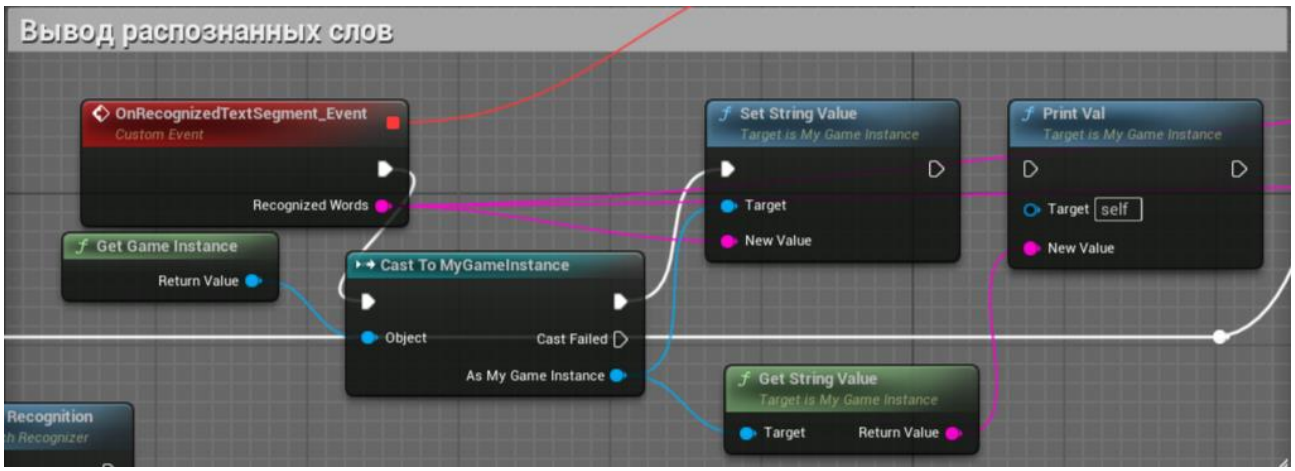


Рисунок 24 Логика вывода текста

- Вызов события: на этапе инициализации происходит вызов события, которое запускает процесс распознавания речи.
- Обращение к функции: затем происходит обращение к функции PrintVal, которая реализована в Blueprint и отображена на рисунке 24. Эта функция выполняет вызов функции GetStringValue для получения значения строки.
- Вывод на консоль: после получения значения строки функция PrintVal выводит его на консоль при помощи функции UE_LOG.

Результат выполнения этой последовательности действий отображен на рисунке 25. На нем можно увидеть вывод строки "Recognized text: [значение строки]" в консоль Unreal Engine, что подтверждает успешное распознавание речи и корректную работу логики на Blueprint.

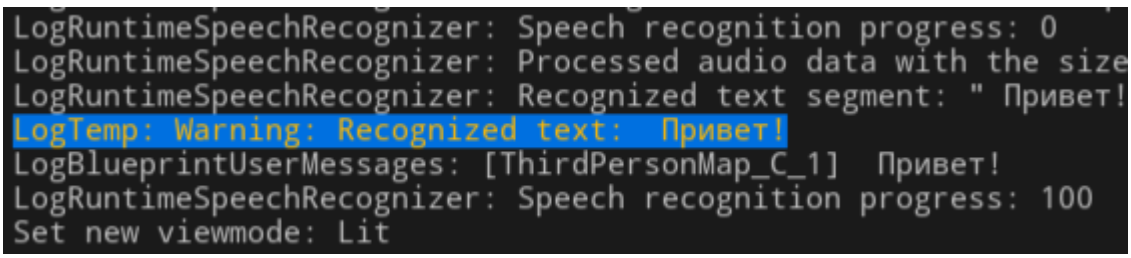


Рисунок 25 Результат вывода

2.5. Результаты разработки системы распознавания речи в Unreal Engine.

В процессе разработки системы распознавания речи в Unreal Engine мы активно использовали результаты предыдущих этапов и интегрировали их в создание полного Blueprint, изображенного на рисунке N. Предыдущие этапы включали в себя выбор и настройку необходимых плагинов, оптимизацию производительности и настройку параметров.

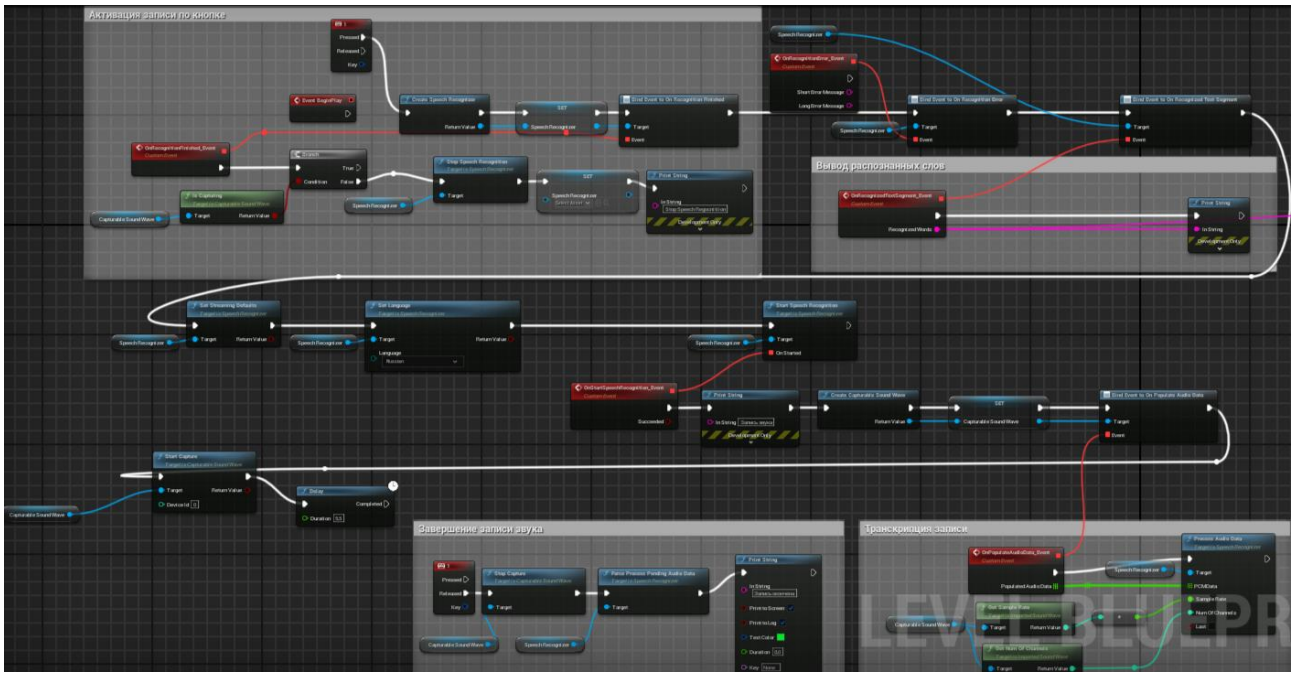


Рисунок 26 общая логика системы

Выбранные на предыдущих этапах плагины были интегрированы в Blueprint для реализации функциональности распознавания речи. Настройки производительности, проведенные ранее, позволили минимизировать задержки при вызове распознавания. Каждый узел в Blueprint был внимательно спроектирован с учетом полученных результатов и целей проекта.

Итоговый Blueprint представляет собой комплексное решение, которое демонстрирует эффективное и плавное функционирование системы распознавания речи в рамках разрабатываемого проекта. Этот Blueprint является визуальным представлением результатов нашей работы и показывает, каким образом все предшествующие этапы взаимодействуют и дополняют друг друга

для достижения поставленных целей, результат которого можно увидеть на рисунке 27.

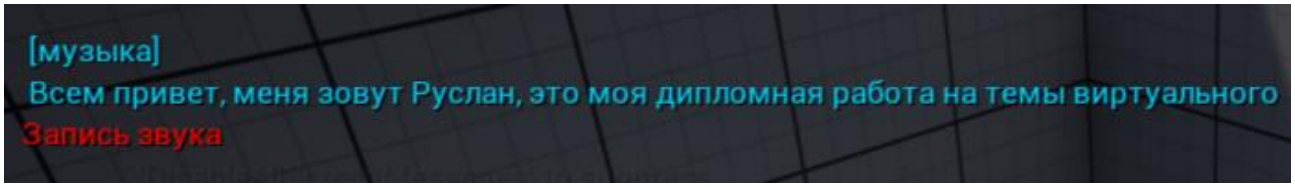


Рисунок 27 Вывод обработки голоса

В дальнейшем переведённая речь в текст, будет использована в качестве запроса для голосового помощника.

Заключение

В результате научно-иследовательской работы удалось достичь значительного прогресса в разработке виртуального помощника на платформе Unreal Engine 5. Были решены проблемы с отображением волос персонажа, что повысило реалистичность его внешнего вида. Кроме того, была успешно разработана и интегрирована система обработки голоса, оптимизированная для повышения эффективности и производительности. Необходимые параметры были настроены для достижения требуемой точности распознавания речи и экономии вычислительных мощностей. Кроме того, был написан и реализован код на C++ в среде Unreal Engine, обеспечивающий вывод распознанной речи в удобном формате. Эти результаты являются важным шагом в развитии функциональности виртуального помощника и открывают перспективы для его дальнейшего совершенствования и применения. Успешная интеграция этих элементов открывает перспективы для создания более эффективного и полезного виртуального помощника, который будет активно использоваться в рамках дальнейших этапов моей исследовательской работы и в дипломной работе.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. MetaHuman – Unreal Engine // MetaHuman URL: https://www.unrealengine/com/en-US/metahuman/ Дата обращения: 18.03.2024;
2. GitHub – Runtime Speech Recognizer // Wiki. URL: https://github.com/gtreshchev/RuntimeSpeechRecognizer/wiki/ Дата обращения: 21.03.2024
3. Youtube – MetaHuman SDK // MetaHuman SDK URL: https://www.youtube.com/@metahumansdk Дата обращения: 19.03.2024.
4. Level of Detail (computer graphics) // Wikipedia, the free encyclopedia URL: https://en.wikipedia.org/wiki/Level_of_detail_(computer_graphics). Дата обращения: 22.03.2024;
5. Creating and Using LODs // Unreal Engine Documentation URL: https://docs.unrealengine.com/5.3/en-US/WorkingWithContent/Types/StaticMeshes /HowTo/LODs/. Дата обращения: 22.03.2024;
6. Божко А.Н., Жук Д.М., Маничев В.Б. Компьютерная графика. [Электронный ресурс] // Учебное пособие для вузов. − М.: Изд-во МГТУ им. Н. Э. Баумана, 2007. - 389 с., - ISBN 978-5-7038-3015-4, Режим доступа: http://ebooks.bmstu.ru/catalog/55/book1141.html. Дата обращения: 10.02.2024;