|  |  |
| --- | --- |
| Герб МГТУ | **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» |
| КАФЕДРА | «Системы автоматизированного проектирования (РК-6)» |
|  | |
| **РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА** | |
| ***К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ*** | |
| ***НА ТЕМУ:*** | |
| ***«Разработка виртуального помощника на Unreal Engine 5»*** | |
|  | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | РК6-82Б |  | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | **Редколис Р.Р.** |
| (Группа) | (Подпись, дата) | (Фамилия И.О.) |
| Руководитель ВКР | | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | **Витюков Ф.А.** |
| (Подпись, дата) | (Фамилия И.О.) |
| Нормоконтролёр | | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | **Грошев С.В.** |
| (Подпись, дата) | (Фамилия И.О.) |

|  |
| --- |
|  |
| *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 |
| (Индекс) |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | Карпенко А.П. | |
| (Фамилия И.О.) | |
| «15» февраля 2024 г. | | | |

|  |  |
| --- | --- |
| **ЗАДАНИЕ** | |
| **на выполнение выпускной квалификационной работы бакалавра** | |
| Студент группы | РК6-82Б |
| Редколис Руслан Русланович | |
| (фамилия, имя, отчество) | |
| Тема квалификационной работы: «Разработка виртуального помощника на Unreal Engine 5» | |
| Источник тематики (НИР кафедры, заказ организаций и т.п.): кафедра. | |
| Тема квалификационной работы утверждена распоряжением по факультету РК № от « »  2024 г. | |
| **Техническое задание** | |
| **Часть 1.** *Аналитическая часть* | |
| *Разобрать полный цикл создания статических и анимированных 3d-моделей и сцен. Изучить материалы и средства разработки, предоставляемые трехмерным движком Unreal Engine 5. Исследовать и освоить методы 3d моделирования и принципы наложения ландшафтных материалов. Провести анализ экспорта объектов и проблем, возникающих при интеграции в среде Unreal Engine 5.* | |
| **Часть 2.** *Практическая часть 1. 3d-моделирование в программных средах Blender 3D и Marvelous Designer.* | |
| *Создание основной концепции модели. Определение внешнего вида, анатомических особенностей и стиля модели. Моделирование 3D объекта в программе Blender 3D в соответствии с предварительно разработанной концепцией. Создание одежды и дополнительных элементов модели в программе Marvelous Designer. Создание рига костей для анимации модели.* | |
| **Часть 3.** *Практическая часть 2. Разработка в Unreal Enigne: восприятия и воспроизведения речи .* | |
| *Интеграция средств распознавания речи или текстового ввода в проект Unreal Engine 5.*  *Разработка системы и обработки и интерпретации голосовых команд или текстовых сообщений. Создание механизмов взаимодействия персонажа с окружающей средой на основе распознанных команд.* | |
| **Оформление выпускной квалификационной работы** | |
| Расчетно-пояснительная записка на ?? листах формата A4. | |
| Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.): | |
| |  | | --- | | *Работа содержит 8 графических листов формата A4.* | |  | |  | |  | | |
| Дата выдачи задания: «10» февраля 2024 г. | |
| В соответствии с учебным планом выпускную квалификационную работу выполнить в полном объеме в срок до «11» июня 2024 г. | |

|  |  |  |
| --- | --- | --- |
| **Руководитель квалификационной работы** | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | **Витюков Ф.А.** |
|  | (Подпись, дата) | (Фамилия И.О.) |
| **Студент** | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | **Редколис Р.Р.** |
|  | (Подпись, дата) | (Фамилия И.О.) |

|  |
| --- |
| Примечание: Задание оформляется в двух экземплярах: один выдается студенту, второй хранится на кафедре. |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **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 | РК |  | УТВЕРЖДАЮ | | |
| КАФЕДРА | РК6 | Заведующий кафедрой | | | РК6 |
| (Индекс) |
| ГРУППА | РК6-82Б | \_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | Карпенко А.П. | |
| (Фамилия И.О.) | |
| «15» февраля 2024 г. | | | | | |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **КАЛЕНДАРНЫЙ ПЛАН** | | | | | | | | |
| **выполнения выпускной квалификационной работы** | | | | | | | | |
| Студент группы | | | РК6-82Б | | | | | |
| Редколис Руслан Русланович | | | | | | | | |
| (фамилия, имя, отчество) | | | | | | | | |
| Тема квалификационной работы: «Разработка виртуального помощника на Unreal Engine 5» | | | | | | | | |
| **№ п/п** | **Наименование этапов выпускной квалификационной работы** | | **Сроки выполнения этапов** | | **Отметка о выполнении** | |
| **план** | **факт** | **Должность** | **ФИО, подпись** |
|  | Задание на выполнение работы. Формулирование проблемы, цели и задач работы | | 10.02.2024 | \_\_\_\_\_\_\_\_\_ | Руководитель ВКР | Витюков Ф.А. |
|  | 1 часть. Аналитическая часть | | 18.02.2024 | \_\_\_\_\_\_\_\_\_ | Руководитель ВКР | Витюков Ф.А. |
|  | Утверждение окончательных формулировок решаемой проблемы, цели работы и перечня задач | | 28.02.2024 | \_\_\_\_\_\_\_\_\_ | Заведующий кафедрой | Карпенко А.П. |
|  | 2 часть. Практическая часть 1 | | 21.04.2024 | \_\_\_\_\_\_\_\_\_ | Руководитель ВКР | Витюков Ф.А. |
|  | 3 часть. Практическая часть 2 | | 23.05.2024 | \_\_\_\_\_\_\_\_\_ | Руководитель ВКР | Витюков Ф.А. |
|  | 1-я редакция работы | | 28.05.2024 | \_\_\_\_\_\_\_\_\_ | Руководитель ВКР | Витюков Ф.А. |
|  | Подготовка доклада и презентации | | 04.06.2024 | \_\_\_\_\_\_\_\_\_ | Студент | Редколис Р.Р. |
|  | Заключение руководителя | | 10.06.2024 | \_\_\_\_\_\_\_\_\_ | Руководитель ВКР | Витюков Ф.А. |
|  | Допуск работы к защите на ГЭК (нормоконтроль) | | 17.06.2024 | \_\_\_\_\_\_\_\_\_ | Нормоконтролер | Грошев С.В. |
|  | Внешняя рецензия | | 17.06.2024 | \_\_\_\_\_\_\_\_\_ |  |  |
|  | Защита работы на ГЭК | | 21.06.2024 | \_\_\_\_\_\_\_\_\_ |  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Студент | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Редколис Р.Р. | Руководитель ВКР | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Витюков Ф.А. |
|  | (подпись, дата) | (Фамилия И.О.) |  | (подпись, дата) | (Фамилия И.О.) |

**АННОТАЦИЯ**

Дипломная работа посвящена разработке виртуального помощника на базе платформы Unreal Engine 5. Она представляет собой исследование процесса создания и функционирования виртуальных персонажей в контексте современных технологий виртуальной реальности. Работа включает в себя анализ существующих методов и инструментов разработки виртуальных ассистентов, проектирование и реализацию модели виртуального помощника, а также анализ результатов и возможных перспектив применения разработанного виртуального помощника.

Тип работы: выпускная квалификационная работа.

Тема работы: «Разработка виртуального помощника на Unreal Engine 5».

Объекты исследований: 3d-моделирование, разработка системы на основе искусственного интеллекта.

**ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ**

**Game engine (игровой движок)** – набор ключевых компонентов программного обеспечения, используемых для разработки игр и иных 3d-приложений. Как правило, инструменты движка абстрагированы от специфики конкретной игры, но могут учитывать некоторые особенности жанра – они предоставляют «базис» для разработки, «надстройку» над которым создает его пользователь.

**Unreal Engine 5 (UE5)** – игровой движок, разрабатываемый и поддерживаемый компанией Epic Games.

**3d-model (3d-модель)** – математическое представление объекта в трехмерном пространстве.

**3d-modeling (3d-моделирование)** – процесс создания 3d-модели объекта.

**3d hard-surface modeling (3d-моделирование твердых поверхностей)** – процесс создания 3d-моделей объектов, имеющих гладкие поверхности и острые углы. Как правило, в данную категорию входят неживые объекты, например: механизмы, машины, здания, мебель.

**3d organic modeling (3d-моделирование органических объектов)** – процесс создания 3d-моделей объектов живой природы и объектов, имеющих похожие на них формы. В данную категорию входят такие объекты моделирования, как люди, животные, гуманоиды, растения.

**Текстурирование** – процесс создания текстур объекта.

**High-poly модель (высокополигональная модель)** – максимально детализированная версия 3d-модели. Имеет большое количество полигонов (от нескольких сотен тысяч до миллионов), в основном используется для дальнейшего запекания текстур. Как правило, высокополигональные модели не используются при отрисовке в реальном времени, поскольку для их обработки требуется слишком много вычислительных ресурсов.

**Actor (актёр)** – в рамках движка UE5 любой объект, который может быть размещен на уровне. Базовый класс всех actor’ов – AActor.

**Actor Component** – специальный тип объекта, который может быть присоединен к выбранному actor’у как подобъект (subobject). Как правило, используется для внедрения функциональности, общей для различных actor’ов. Базовый класс – UActorComponent.

**Rendering (рендеринг, отрисовка)** – процесс получения 2d-изображения по имеющейся 3d-модели.

**Polycount** – количество полигонов модели.

**Static Mesh (UStaticMesh)** – класс, представляющий собой статический геометрический объект. Хранит данные о полигональной сетке модели.

**Текстура** – изображение, накладываемое на поверхность 3d-модели. Может содержать одно или несколько свойств поверхности, например: цвет, жёсткость (roughness), смещение (displacement), направление нормалей (normal map), и т.д.

**Материал** – набор свойств, определяющих поведение света при отражении от поверхности. Материалы также могут использовать одну или несколько текстур.

**Material (UMaterial)** – класс, позволяющий хранить и модифицировать информацию о материале.

**Static Mesh Component (UStaticMeshComponent)** – компонент, который может иметь статический меш и набор материалов, применимых к нему.

**Geometry instancing (инстансинг, дублирование геометрии)** – техника, позволяющая отрисовывать множество однотипных элементов за один проход.

**Instanced Static Mesh Component (UInstancedStaticMeshComponent, ISMC)** – класс, производный от UStaticMeshComponent, позволяющий использовать механизм инстансинга геометрии.

**LOD (Level of Detail, уровни детализации)** – техника, позволяющая подменять разные по детализации версии модели в зависимости от дистанции между камерой и объектом, либо в зависимости от процента площади экрана, занимаемой моделью.

**Hierarchical Instanced Static Mesh Component (HISMC, UHierarchicalInstancedStaticMeshComponent)** – класс, производный от UInstancedStaticMeshComponent, позволяющий использовать LOD.

**Central Processing Unit (CPU)** – центральный процессор.

**Graphics Processing Unit (GPU)** – графический процессор (видеокарта).

**Frames per second (FPS)** – количество кадров в секунду.

**Rendering Hardware Interface (RHI)** – в UE4 надстройка над множеством графических API (например: DirectX, OpenGL, Vulkan), позволяющая писать независимый от платформы код.

**Asset (ассет)** – в контексте компьютерных игр, объект, представляющий собой единицу контента. Игровыми ассетами являются 3d-модели, текстуры, материалы, аудиофайлы, и т.д. Как правило, созданием ассетов занимаются художники, дизайнеры, музыканты.

**Reference (референс)** – изображение, используемое художником в процессе 3d-моделирования для получения дополнительной информации о моделируемом объекте.

**Lip synchronization** (синхронизация губ) – элемент анимации, при

котором движения губ совпадают с речью персонажа.

**API endpoint** — это конечная точка в API, к которой можно обратиться для выполнения нужного действия или получения данных.

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 10](#_Toc166772091)

[1. Создание цифрового аватара 12](#_Toc166772092)

[1.1. Создание проекта 13](#_Toc166772093)

[1.2. Создание персонажа в облачном сервисе MetaHuman Creator 14](#_Toc166772094)

[1.3. Импорт персонажа в Unreal Engine через Quixel Bridge 14](#_Toc166772095)

[1.4. Добавление персонажа на сцену 17](#_Toc166772096)

[2. Разработка системы обработки голоса пользователя 20](#_Toc166772097)

[2.1. Добавление плагина 22](#_Toc166772098)

[2.2. Установка параметров 24](#_Toc166772099)

[2.3. Оптимизация производительности. 28](#_Toc166772100)

[2.4. Запись и вывод переменной SpeechToText. 30](#_Toc166772101)

[2.5. Результаты разработки системы распознавания речи в Unreal Engine. 34](#_Toc166772102)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 36](#_Toc166772103)

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 дипломной 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. Cоздание 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.

1. 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. Разработка API запросов;

В данной части работы рассматривается практическое применение современных технологий и API для развития функциональности виртуального помощника. Использование API ChatGPT и elevenlabs открывает новые возможности для создания более реалистичного и интерактивного взаимодействия с виртуальным помощником. Также в этой части реализовано восприятие речи с переводом в текст speech-to-text и воспроизведение на основе текста text-to-speech.

4 Привязка lipsync и модели MetaHumans с использованием MetaHumanSDK;

В данном разделе исследуется процесс интеграции технологии lipsync с моделью MetaHuman при помощи MetaHumanSDK. Lipsync, или синхронизация речи с движением губ персонажа, является важным аспектом для достижения реалистичности в воспроизведении речи в виртуальной среде. Мы используем MetaHumanSDK, предоставленный Unreal Engine, для создания плавной и точной синхронизации между аудиофайлом и движением губ виртуального персонажа. Это позволяет обеспечить естественное и убедительное выражение лица персонажа во время речи, что повышает иммерсию и реализм взаимодействия с виртуальным помощником.

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.

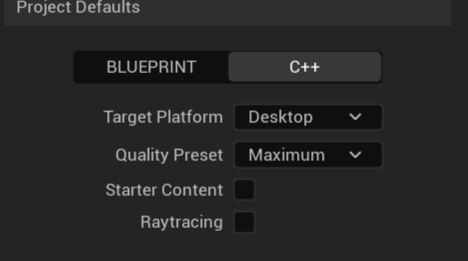


Рисунок 1 Найстроки проекта

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.

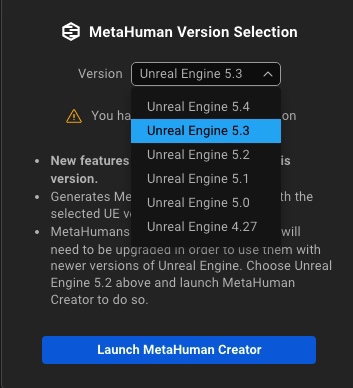


Рисунок 2 выбор версии под проект

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.

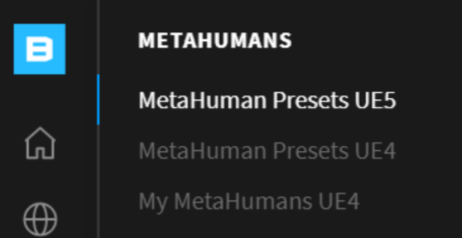


Рисунок 3 Quixel Bridge вкладка MetaHumans

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

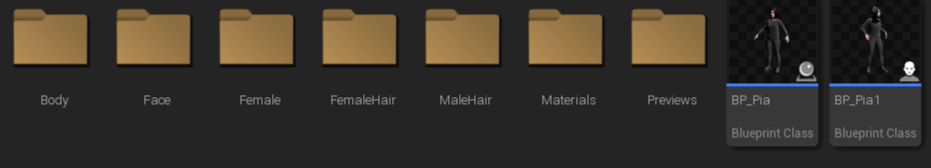


Рисунок 4 среда файлов проекта

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.

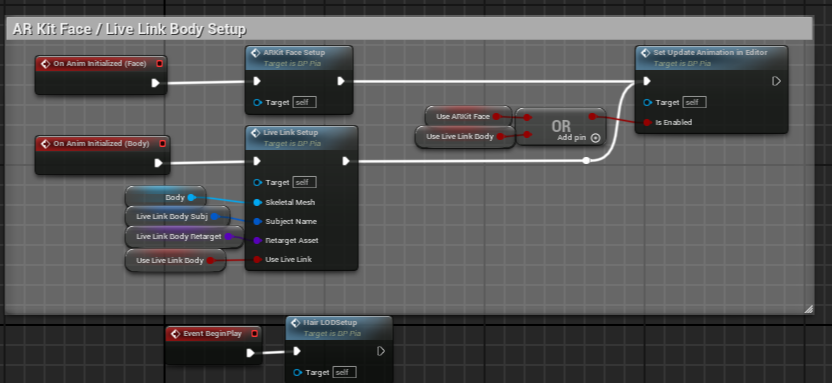


Рисунок 5 blueprint набор привязок тела и головы

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.

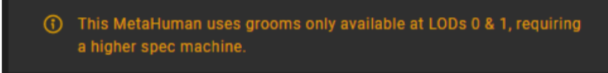


Рисунок 7 Предупрждение к персонажу

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

Далее требуется перейти в компоненты модели и выбрать LODSync

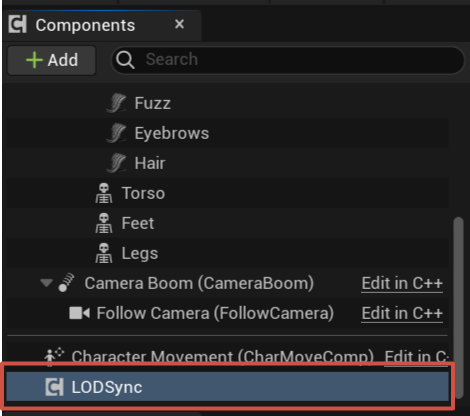


Рисунок 8 компонент детализации

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.

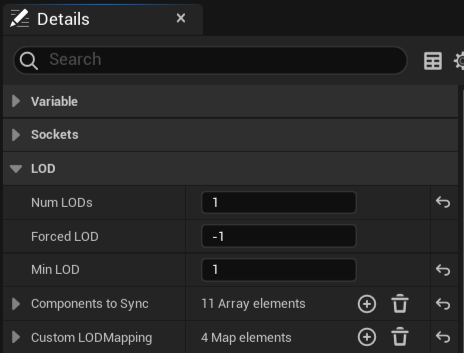


Рисунок 9 Окно настроек детализации

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

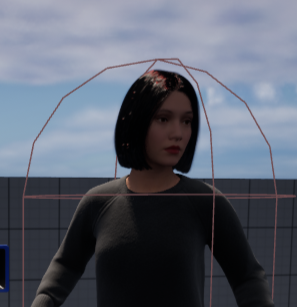


Рисунок 10 результат изменения параметра

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.

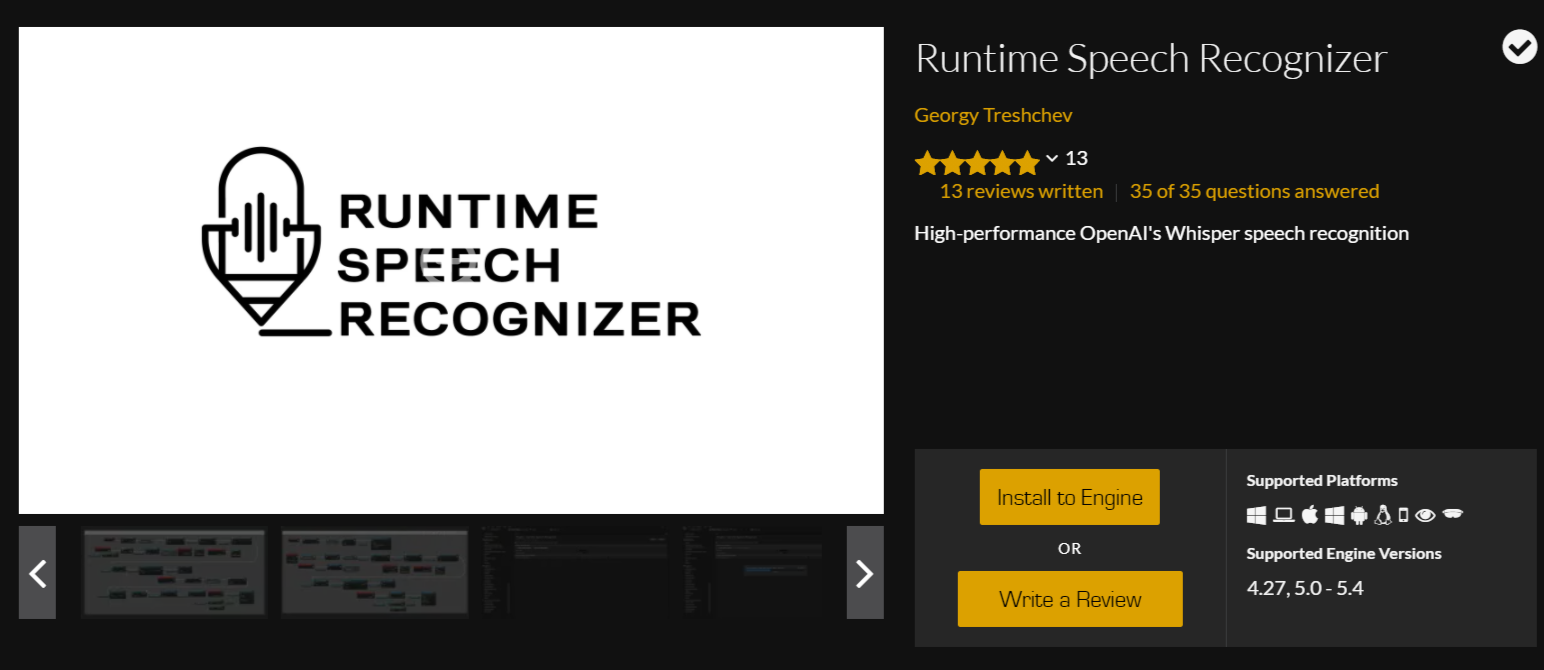


Рисунок 11 установка плагина через сервис

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

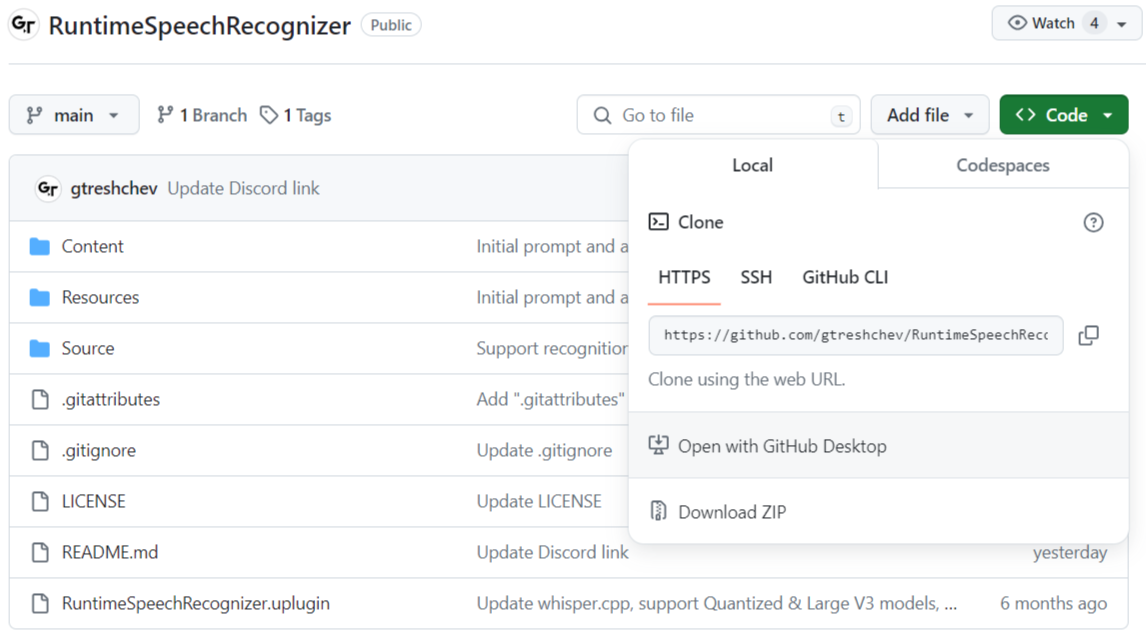


Рисунок 12 Репозиторий автора плагина

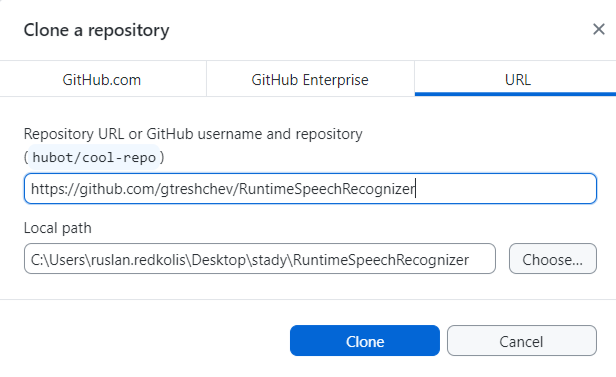


Рисунок 13 клонирование репозитория

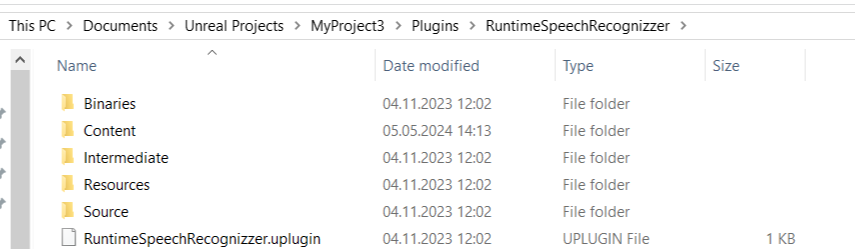
Далее требуется импортировать репозиторий в папку проекта. 

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.

* 1. 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.

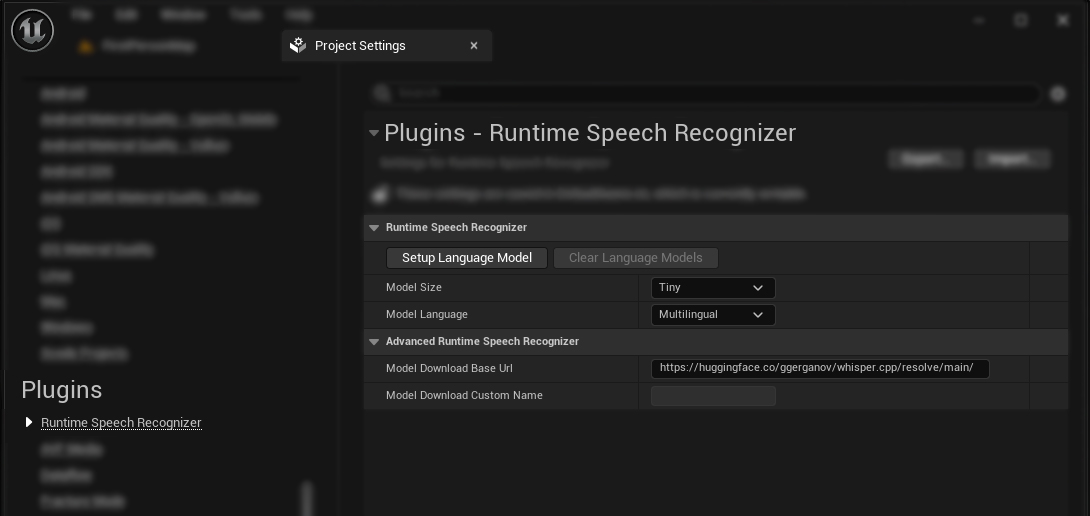
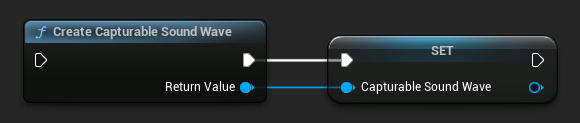


Рисунок 15 выбор размера модели

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.

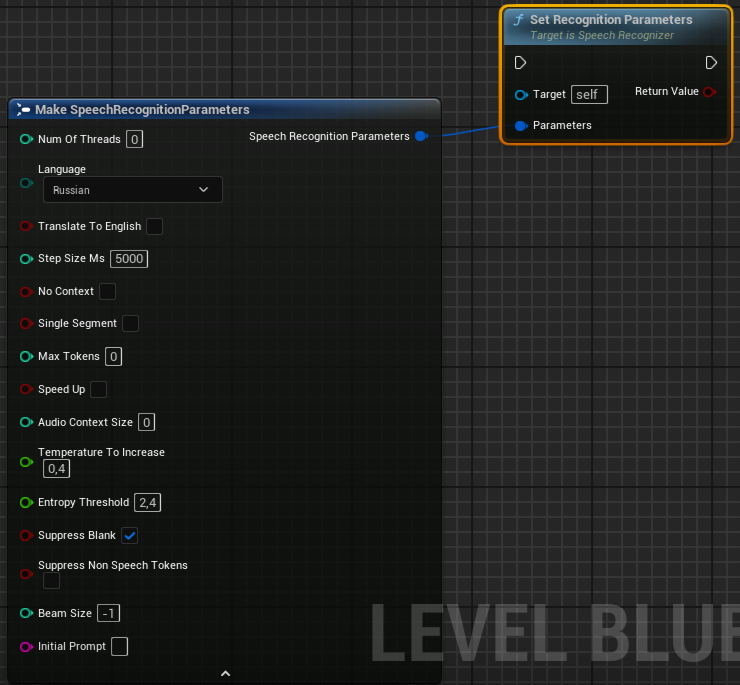


Рисунок 16 параметры класса speechrecognizer

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.

* 1. 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.

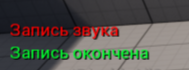


Рисунок 17 результат нажатия клавиши

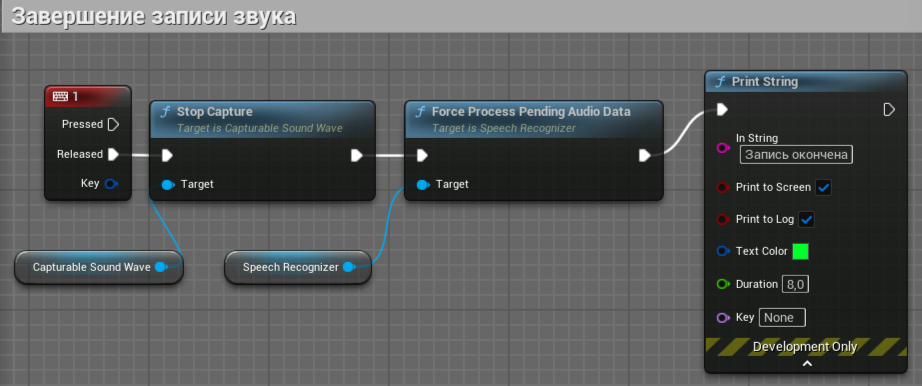


Рисунок 18 логика остановки записи звука

Вызывая StartSpeechRecognition, произходит 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.

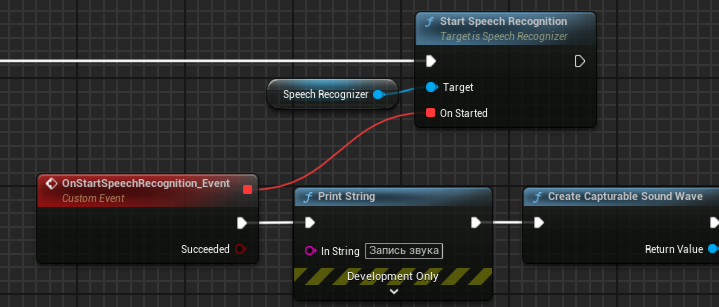
Для устранения этой задержки придерживайтесь того же принципа, описанного выше: вызывайтся StartSpeechRecognition в момент окончания записи звука (рисунок 19). 

Рисунок 19 Логика запуска функции распознания речи

* 1. 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.

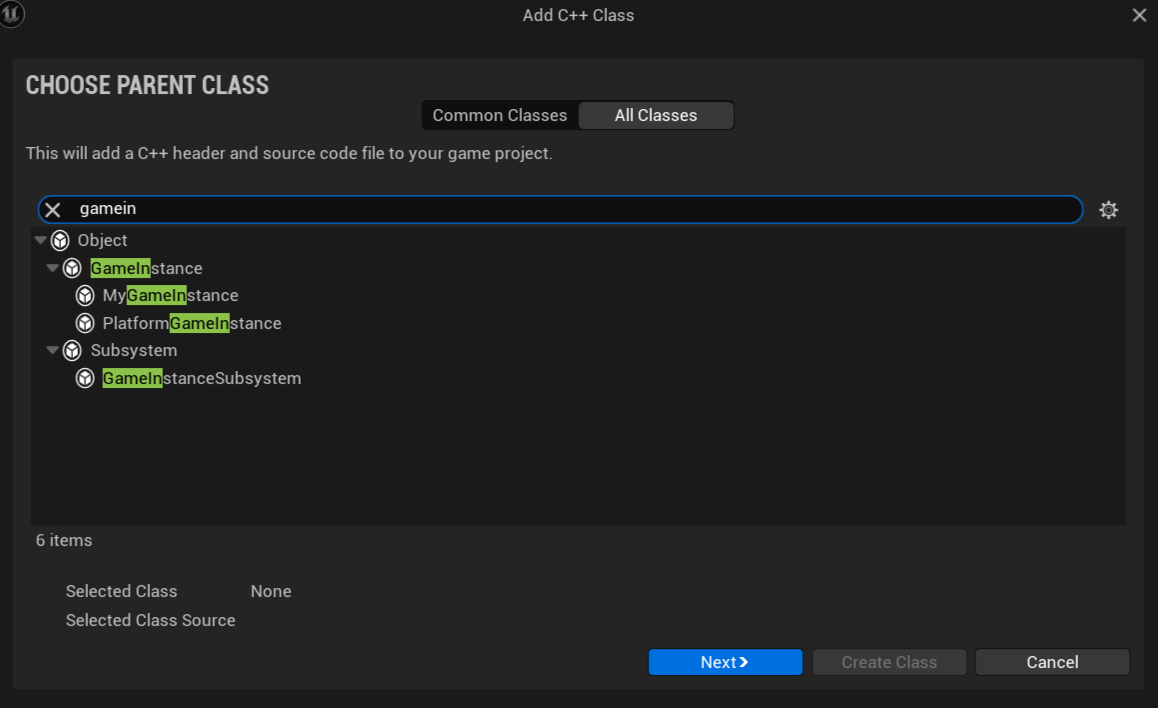


Рисунок 20 Класс GameInstance

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.

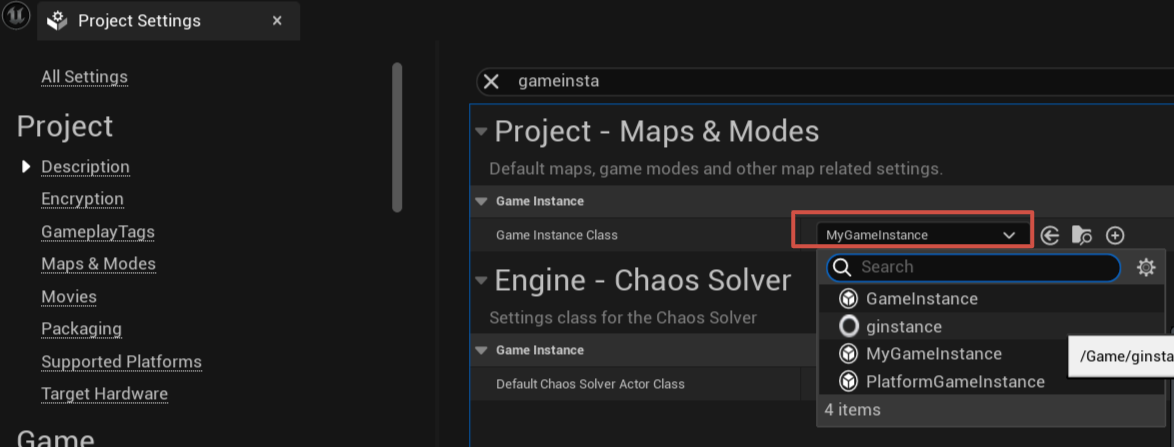


Рисунок 21 Найстрока GameInstance

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:

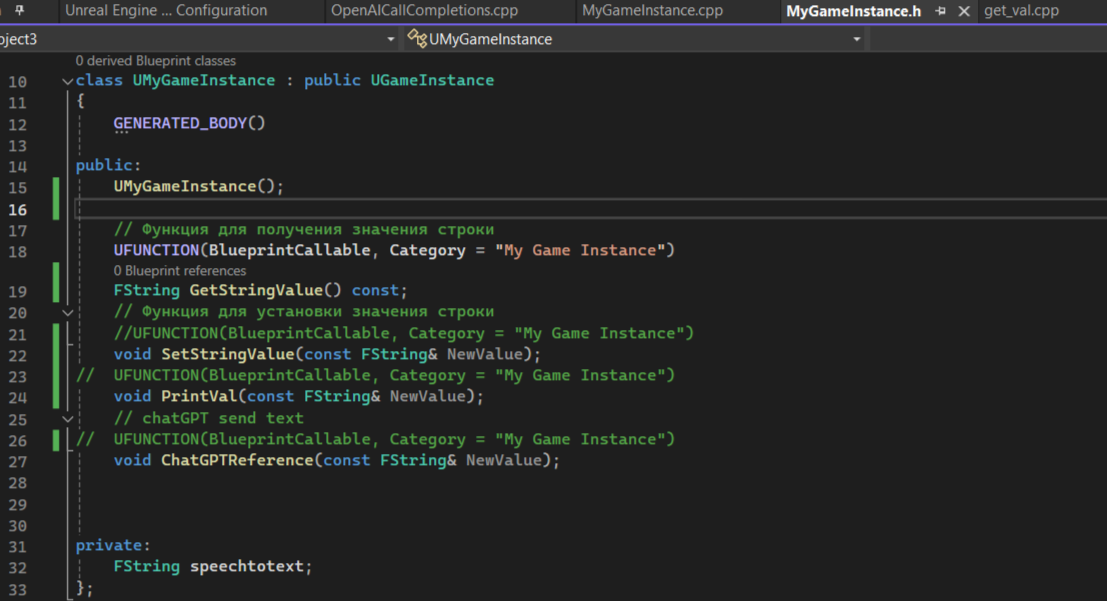


Рисунок 22 Заголовочный файл

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.

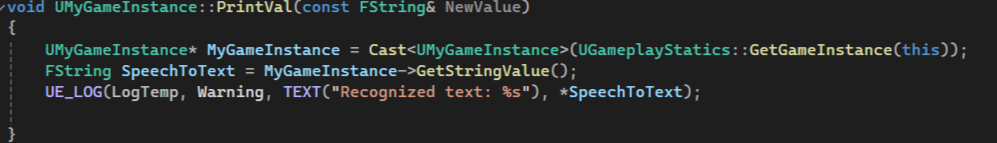


Рисунок 23 Реализация вывода в консоль

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.

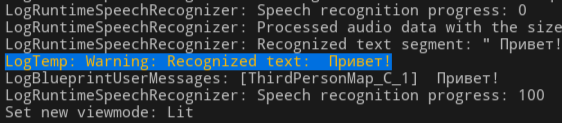


Рисунок 25 Результат вывода

* 1. 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.

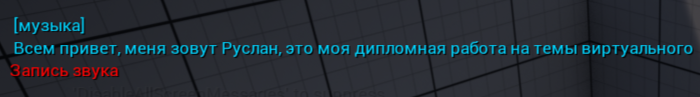


Рисунок 27 Вывод обработки голоса

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

1. Разработка API запросов

API-сервисы предоставляют возможность взаимодействия с облачными ресурсами и сервисами через программные интерфейсы. Эти сервисы позволяют разработчикам создавать, развивать и масштабировать свои приложения без необходимости управления физической инфраструктурой. Они предоставляют широкий спектр функций, включая хранение данных, вычислительные мощности, искусственный интеллект, аналитику, машинное обучение и многое другое. Облачные API-сервисы позволяют разработчикам создавать инновационные приложения и сервисы, используя гибкость и масштабируемость облачных ресурсов, что способствует повышению производительности, снижению затрат и ускорению развертывания новых продуктов на рынке.

Разработка API запросов играет ключевую роль в усовершенствовании функциональности виртуального помощника, позволяя ему взаимодействовать с пользователем более естественным и эффективным способом. В данной части работы исследуются современные технологии и применения к ним API, такие как ChatGPT и elevenlabs, которые предоставляют широкий спектр инструментов для создания более реалистичного и интерактивного взаимодействия с виртуальным помощником. Использование этих API открывает новые возможности для расширения функциональности виртуального помощника и повышения его эффективности. В рамках этой части работы уже реализованы функции распознавания речи с последующим переводом в текст (speech-to-text) будет использованы в качестве запроса, результат запроса будет воспроизводится виртуальным голосом (text-to-speech), это значительно улучшает взаимодействие пользователя с виртуальным помощником. Таким образом, разработка API запросов играет важную роль в развитии и совершенствовании функциональности виртуального помощника, открывая перед ним новые перспективы для более эффективного и удобного использования.

В рамках дипломной работы было реализовано взаимодействие с внешними API в Unreal Engine с использованием запросов на языке C++. Результаты этих запросов были успешно интегрированы и воспроизведены в проекте, что позволило расширить функционал виртуального помощника.

* 1. Разработка API запроса в ChatGPT

API-запросы к ChatGPT пишутся в виде HTTP-запросов с использованием методов POST и GET. В запросе указывается URL endpoint API, а также параметры, необходимые для выполнения определенной функции, такие как текст для генерации ответа или параметры модели.

POST: Этот метод используется для отправки данных на сервер для обработки. В отличие от GET, данные в запросе передаются в теле запроса, что делает его более безопасным для передачи больших объемов данных или конфиденциальной информации. POST-запросы часто используются для создания или обновления данных на сервере, простой API запрос представлен на рисунке .

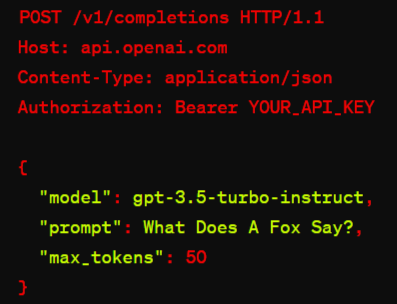


Рисунок 28 пример POST-запроса

Этот пример представляет собой POST-запрос к API OpenAI для получения завершения текста с использованием конкретной модели " gpt-3.5-turbo-instruct". В запросе указывается хост (api.openai.com), тип содержимого (application/json) и авторизационный токен с помощью которого открывается доступ к модели. В теле запроса передаются параметры, такие как выбранная модель, начальная фраза (prompt) и максимальное количество токенов, которые должны быть сгенерированы в ответе.

POST-запрос выполняется в методе Activate(). В этом методе сначала заполняется персональный ключ сервиса, который был указан в ноде blueprint.

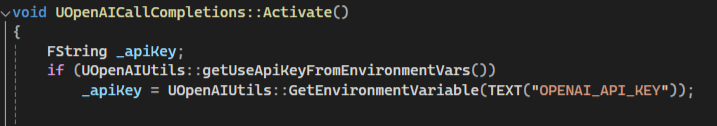


Рисунок 29 установка API ключа

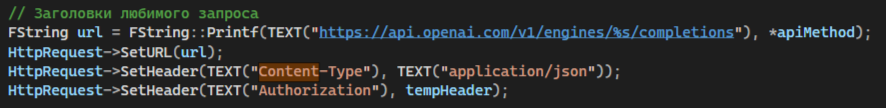
Затем формируется объект HTTP-запроса и устанавливаются заголовки, такие как Content-Type и Authorization, включая API-ключ в заголовке Authorization. 

Рисунок 30 установка заголовков запроса

Затем создается JSON-объект, который содержит параметры запроса, такие как текстовый запрос (prompt), максимальное количество токенов (max\_tokens), температура (temperature) и другие.

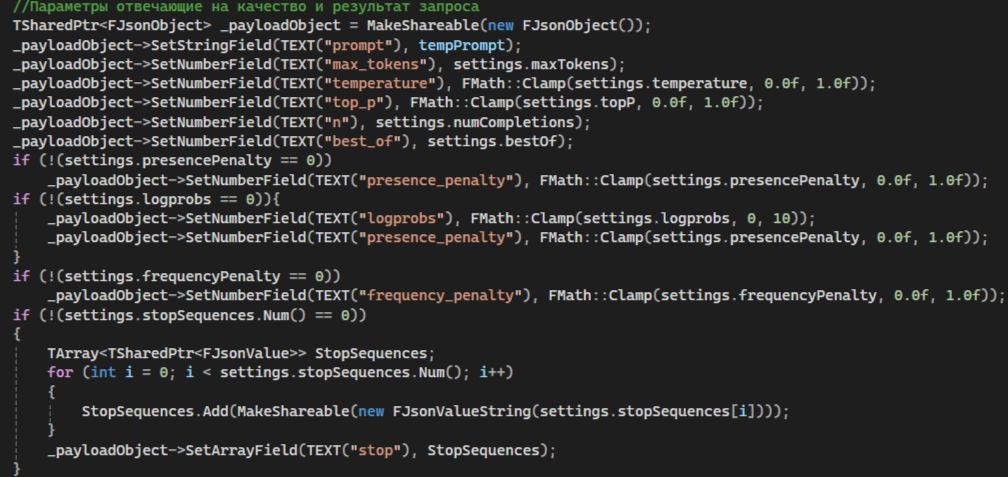


Рисунок 31 параметры модели

Этот JSON-объект сериализуется в строку и устанавливается как содержимое (Content) HTTP-запроса. Затем запрос отправляется на сервер API OpenAI с помощью метода ProcessRequest().



Рисунок 32 NNNNNN

GET: Этот метод используется для запроса данных от сервера. В запросе параметры передаются в URL в виде строки запроса. Этот метод обычно используется для получения данных и не должен изменять состояние сервера.

Мы получаем ответ с сервера, используя объект FHttpResponsePtr Response, который представляет HTTP-ответ от сервера. Для получения текстового содержимого ответа, которое содержит данные в формате JSON

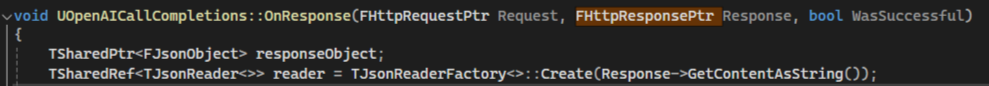


Рисунок 33 NNNN

Далее вызывается метод GetContentAsString() на объекте Response. В результате получаем JSON файл. Данный JSON содержит не только ответ на запрос но иную информацию, десериализация JSON файла представлена на рисунке NNN.

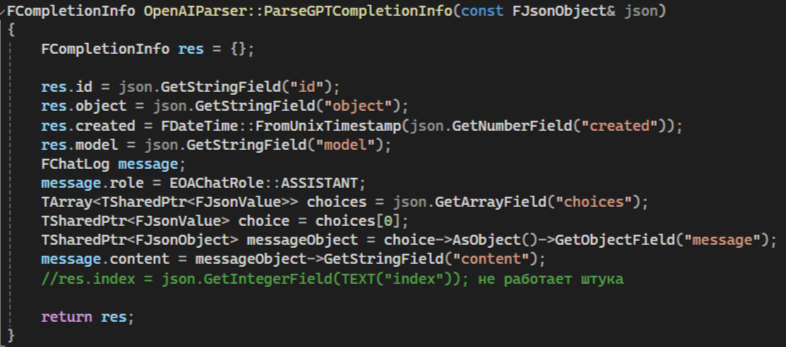


Рисунок 34 десериализация JSON файла

Из responseObject извлекается массив, который находится в поле Choices. Из данного поля извлекаются массивы с содержащие требуемый ответ на запрос, которые добавляется в переменную массива объекта FCompletions.

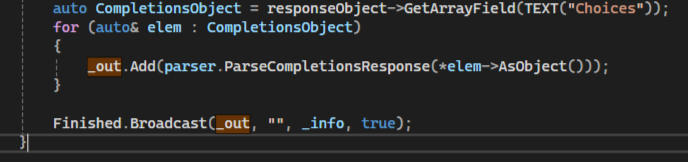


Рисунок 35 Запись ответа в массив

На рисунке NNN представлено извлечение текста из переменной массива FString.

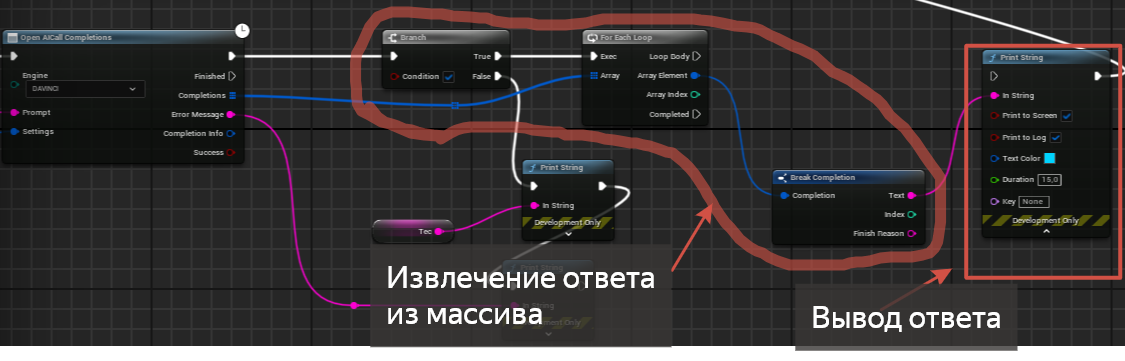


Рисунок 36 Вывод response сервера

В дальнейшем, полученный ответ будет использован в качестве запроса для получения аудиофайла, который будет воспроизводиться голосовым помощником. Этот подход позволит обеспечить более интерактивное и удобное взаимодействие пользователя с системой, предоставляя ему возможность получать необходимую информацию в аудиоформате.

* 1. Разработка API запроса в ElevenLabs

В рамках дальнейшего развития проекта планируется использовать полученные текстовые ответы для создания запросов на получение аудиофайлов через сервис ElevenLabs, которые будут воспроизводиться голосовым помощником, что существенно повысит удобство и эффективность взаимодействия пользователя с системой. Несмотря на то, что у нас имеется возможность использовать функцию преобразования текста в речь (TTS) от MetaHuman SDK, данный сервис не поддерживает необходимый язык и производит более роботизированное звучание. В отличие от этого, сервис ElevenLabs предоставляет высококачественные, естественно звучащие аудиофайлы, что делает его более подходящим для реализации нашего проекта. В этом разделе будет представлена логика, реализованная с использованием blueprint, описывающая процесс отправки текста на сервис ElevenLabs, получения аудиофайла и его воспроизведения, что позволит значительно улучшить пользовательский опыт.

В предшествующем разделе была разработана функциональность для выполнения API-запросов на языке C++. Однако, следует отметить, что Varest предлагает аналогичный функционал. Этот плагин также обеспечивает обработку ответов от серверов, включая чтение JSON-данных и других форматов ответов.

VaRest - это плагин, который предоставляет разработчикам удобные средства для работы с веб-запросами и взаимодействия с внешними веб-сервисами прямо из проектов, созданных на Unreal Engine. Этот плагин обеспечивает возможность отправки HTTP-запросов (GET, POST, PUT, DELETE и других) и обработки ответов от серверов, что позволяет интегрировать ваши игры и приложения с различными сетевыми сервисами и API.

Для выполнения преобразования текста в речь с использованием API сервиса ElevenLabs используется HTTP POST-запрос, включающий различные параметры настройки процесса синтеза речи. Структура запроса представлена на рисунке NNN



Рисунок 37 Post запрос в сервис

Исходя из заголовка можно заметить, что тело запроса содержит данные в формате JSON. Тело запроса представляет собой JSON-объект следующего вида:

"text": Текст, который необходимо преобразовать в речь.

"model\_id": Идентификатор модели синтеза речи, которая будет использоваться.

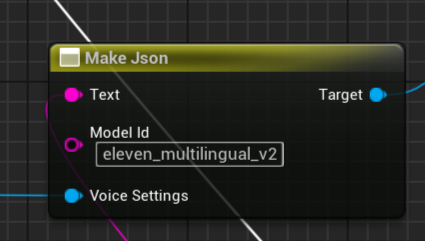


Рисунок 38 версия модели и текста

"voice\_settings": Объект с настройками голоса (см Рисунок NNN) указывается 2 параметра:

"stability": Параметр стабильности голоса.

"similarity\_boost": Параметр усиления сходства.



Рисунок 39 параметры голоса модели

Остальные параметры устанавливаются в самом сервисе мануально, но по желанию их также можно передать в POST-запрос.

"style": Параметр стиля речи.

"use\_speaker\_boost": Логическое значение, указывающее на необходимость использования усиления громкости.

"pronunciation\_dictionary\_locators": Массив объектов, указывающих на словари произношения:

"pronunciation\_dictionary\_id": Идентификатор словаря произношения.

"version\_id": Идентификатор версии словаря.

"seed": Значение случайного числа для генерации уникального синтеза речи.

"previous\_text": Предыдущий текст, используемый для контекстуального синтеза речи.

"next\_text": Следующий текст, используемый для контекстуального синтеза речи.

"previous\_request\_ids": Массив идентификаторов предыдущих запросов для сохранения контекста:

"string": Идентификатор предыдущего запроса.

"next\_request\_ids": Массив идентификаторов следующих запросов для сохранения контекста:

"string": Идентификатор следующего запроса.

Итоговый POS

Этот запрос позволяет гибко настроить параметры синтеза речи, включая настройки голоса, словари произношения, а также контекстуальные параметры, такие как предыдущий и следующий текст. Возможность использования настроек голоса и контекстуальных параметров позволяет достичь высокой точности и естественности синтезируемой речи, что является ключевым фактором для создания качественных голосовых аудиозаписи для виртуального помощника. На рисунке NNN показано, как данный API POST запрос можно реализовать с использованием Blueprints в Unreal Engine.

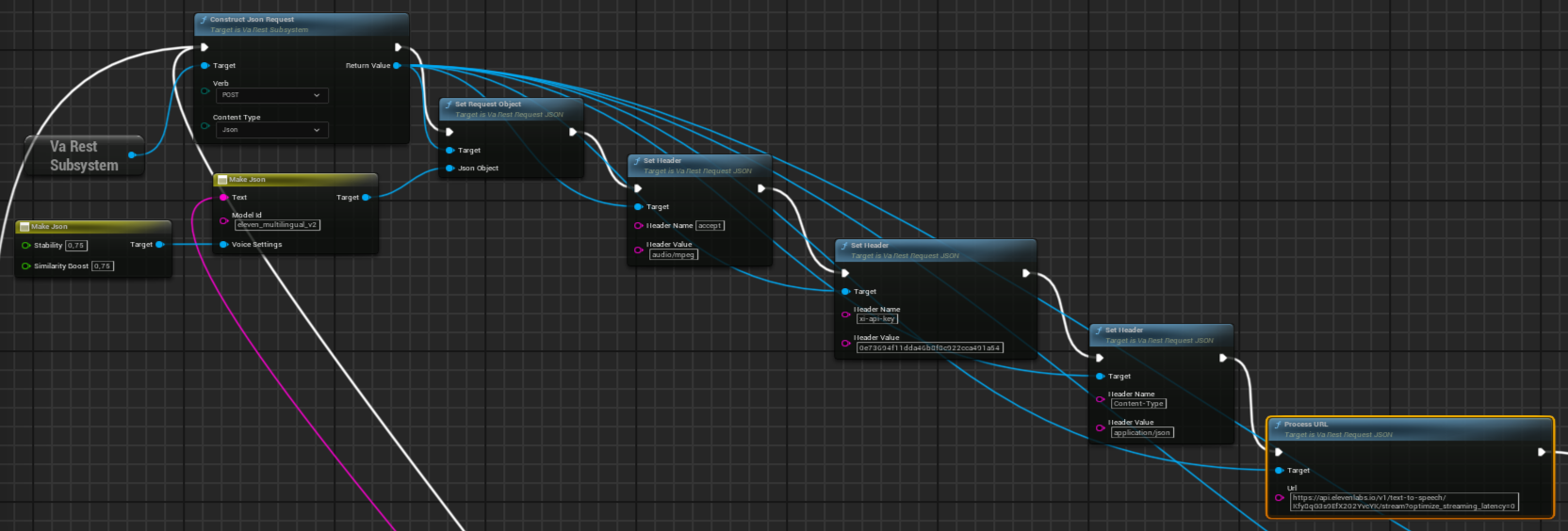


Рисунок 40 Логика POST-запроса в сервис

В дальнейшем на представленной диаграмме Blueprints (Рисунок NNN) выполняется процесс получения аудиозаписи в ответ на API-запрос, проверки содержимого ответа и последующего импорта аудиоданных с использованием плагина Runtime Audio Importer. После получения содержимого ответа в виде строки, проверяется наличие определенной подстроки. В зависимости от результата, создается объект Runtime Audio Importer, который используется для импорта аудиоданных из буфера, передаваемых в формате MP3.

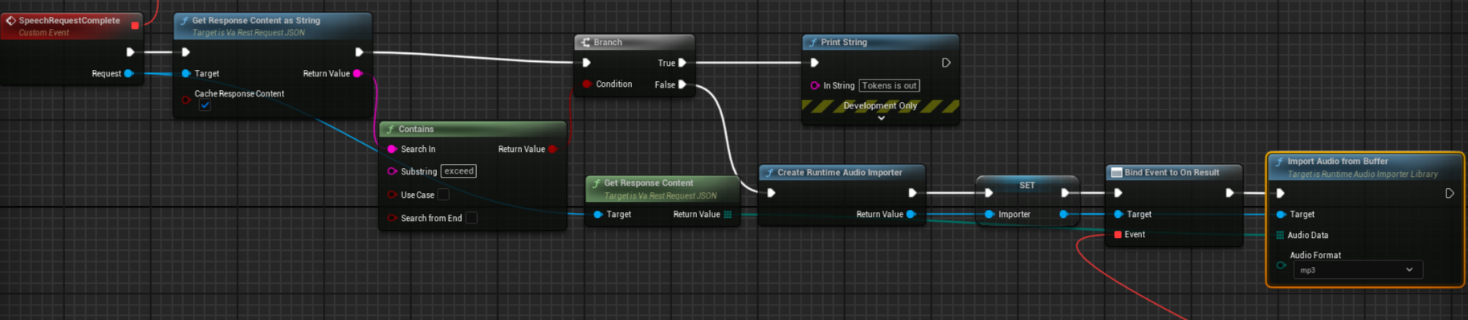


Рисунок 41 Логика обработки Response

1. Воспроизведение аудиозаписи голосовым помощником

В данном разделе рассматривается процесс воспроизведения речи голосовым помощником, который включает несколько ключевых этапов: получение аудиофайлов через API-запросы, их обработку и интеграцию в игровое или интерактивное приложение на базе Unreal Engine. Воспроизведение речи играет важную роль в создании реалистичных и интерактивных персонажей, способных озвучивать ответы на запросы пользователей в реальном времени.

В этой работе используется плагин Runtime Audio Importer для динамического импорта аудиофайлов, что позволяет загружать и воспроизводить звуковые данные на лету. Также задействованы Blueprints для упрощения процесса настройки и интеграции аудиофайлов. Синхронизация воспроизведения речи с анимацией мимики виртуального помощника обеспечивает более естественное и правдоподобное взаимодействие с пользователем. Такой подход не только улучшает пользовательский опыт, но и расширяет возможности для разработки интерактивных и адаптивных игровых сценариев.

Особое внимание уделено использованию MetaHuman SDK, предоставляющего мощные инструменты для создания высококачественных цифровых людей в Unreal Engine. Интеграция MetaHuman с синтезом речи позволяет создавать реалистичных виртуальных помощников, чья анимация лица и губ синхронизируется с воспроизводимой речью, что делает взаимодействие максимально естественным и интерактивным.

* 1. Воспроизведение мимики голосовым помощником

В данном разделе рассматривается процесс создания базовой анимации лица для виртуального помощника MetaHuman, включая анимации нейтрального и радостного выражения лица. Анимация лица играет ключевую роль в обеспечении реалистичного и выразительного взаимодействия с пользователем, особенно в контексте воспроизведения синтезированной речи.

Для создания нейтрального выражения лица используется MetaHuman Creator, который предоставляет широкий спектр инструментов для настройки лицевой анимации.

Итоговые анимация улыбки и печали представлена на рисунке nnn и рисунке nnn



Рисунок Анимация улыбки



Рисунок 43 анимация грусти

На рисунке 44 разработана система управления мимикой виртуального помощника, включающая три основных состояния: Idle, Thinking и Talk. Каждое из этих состояний характеризуется уникальным выражением лица. Переходы между состояниями определяются событиями, такими как начало или завершение общения и обработка запросов пользователя.

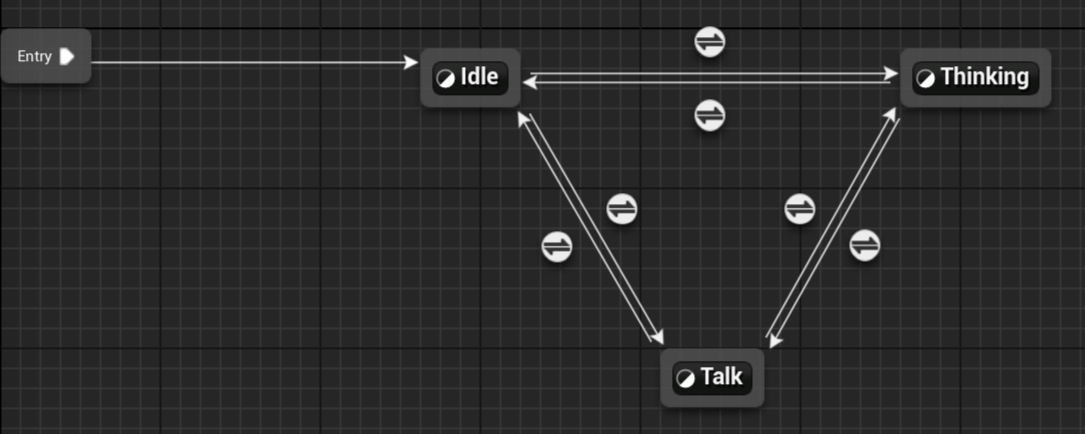


Рисунок 44 Логика смены выражения лица

Состояние Idle является стандартным и не требует дополнительных условий для своего поддержания или перехода в него. Состояние Thinking представляет собой переменное состояние, когда виртуальный помощник думает над ответом. для его управления используется булевая переменная bThinking (рисунок 45), которая, при значении true, переводит помощника в это состояние. Состояние Talk активируется, когда виртуальный помощник произносит ответ пользователю, для неё создана такая же булевая переменная для определения того, что помощник говорит. длительность этого состояния зависит от длительности аудиозаписи, после чего помощник возвращается в состояние Idle. Переходы между этими состояниями обеспечивают динамическое и реалистичное поведение виртуального помощника.

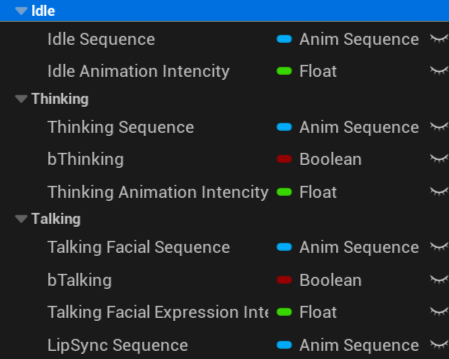


Рисунок 45 Переменные для определения мимики

Для плавного перехода между анимациями используется метод смешивания (blend), как показано на рисунке 46. Ранее, на рисунке 45, представлена переменная intensity, которая используется для различных состояний с целью регулирования мимики виртуального помощника. Первоначально выражения лица были слишком экспрессивными, и использование переменной intensity позволило смягчить переходы и сделать выражения лица более естественными и сбалансированными. Этот подход обеспечивает более реалистичное отображение эмоций виртуального помощника в каждом из состояний: Idle, Thinking и Talk.

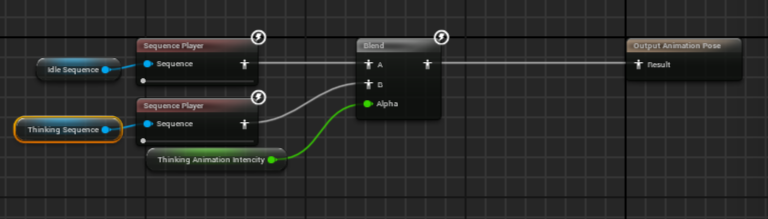


Рисунок 46 Логика смешивания анимаций

* 1. Lip sync виртуального помощника с помощью MetahumanSDK

Анимация лица является важным компонентом при создании реалистичного виртуального помощника. Использование технологий синтеза речи (TTS) и алгоритмов анализа и синхронизации анимации с аудиофайлом позволяет добиться высокой степени реализма и выразительности. В прошлом разделе виртуальный помощник был оснащен такими анимациями, теперь он может обеспечить более естественное и привлекательное взаимодействие с пользователем, что является важным аспектом в современном развитии голосовых интерфейсов.

Для синхронизации речи персонажа с движениями его губ (Lip sync) необходим плагин MetaHumanSDK, который автоматически генерирует эту синхронизацию. Плагин можно скачать бесплатно из Epic Store Marketplace (Рисунок 47). Рекомендуется добавлять его на свой аккаунт Epic Store через браузер, так как в самом приложении он может не отобразиться.

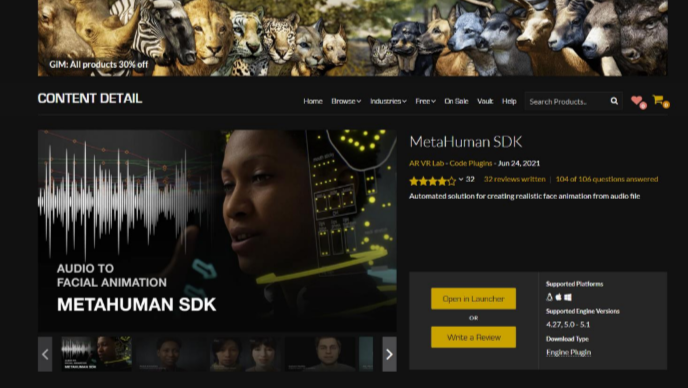


Рисунок плагин в Epic Games Store

ЗАКЛЮЧЕНИЕ

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.
7. ElevenLabs API // ElevenLabs Documentation URL: https://api.elevenlabs.io/docs. Дата обращения: 22.05.2024.
8. MetaHuman SDK // Unreal Engine Documentation URL: https://docs.unrealengine.com/5.3/en-US/metahuman-sdk/. Дата обращения: 22.05.2024.
9. Gaelle Morand, Charles de Rousiers, Michael Forot Unreal Engine Hair and

Fur – GROOMING FOR REAL-TIME REALISM: HAIR AND FUR

WITH UNREAL ENGINE: учебное пособие // Gaelle Morand, Charles de Rousiers, Michael Forot; Дата обращения: 07.05.2024;

1. Playground – Как создать лицевую анимацию в играх // Как создать

лицевую анимацию URL:

https://www.playground.ru/misc/news/kak\_sozdayut\_litsevuyu\_animatsiyu\_v\_

igrah-356274 Дата обращения: 20.02.2024;

1. Unreal Engine 5 Documentation // Unreal Engine Documentation URL:

https://docs.unrealengine.com/. Дата обращения: 28.12.2023;