|  |  |
| --- | --- |
| 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» (РК) |
| КАФЕДРА | «Системы автоматизированного проектирования» (РК6) |
|  | |
| **РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА** | |
| ***К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ*** | |
| ***НА ТЕМУ:*** | |
| ***«Разработка метода интеграции больших языковых моделей средствами representational state transfer application programming interface (REST API) для управления виртуальными агентами в Unreal Engine 5»*** | |
|  | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | РК6-84Б |  | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | **А.С. Больных** |
| (Группа) | (Подпись, дата) | (И.О.Фамилия) |
| Руководитель ВКР | | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | **Ф.А. Витюков** |
| (Подпись, дата) | (И.О.Фамилия) |
| Нормоконтролёр | | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | **С.В. Грошев** |
| (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 |
| (Индекс) |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | Карпенко А.П. | |
| (Фамилия И.О.) | |
| «\_\_» \_\_\_\_\_\_\_ 202\_ г. | | | |

|  |  |
| --- | --- |
| **ЗАДАНИЕ** | |
| **на выполнение выпускной квалификационной работы бакалавра** | |
| Студент группы | РК6-84Б |
| Больных Андрей Сергеевич | |
| (фамилия, имя, отчество) | |
| Тема квалификационной работы: «Разработка метода интеграции больших языковых моделей средствами representational state transfer application programming interface (REST API) для управления виртуальными агентами в Unreal Engine 5» | |
| При выполнении ВКР:   |  |  | | --- | --- | | Используются / Не используются | Да/Нет | | 1. Литературные источники и документы, имеющие гриф секретности | нет | | 1. Литературные источники и документы, имеющие пометку «Для служебного пользования», иных пометок, запрещающих открытое опубликование | нет | | 1. Служебные материалы других организаций | нет | | 1. Результаты НИР (ОКР), выполняемой в МГТУ им. Н.Э.Баумана | нет | | 1. Материалы по незавершенным исследованиям или материалы по завершенным исследованиям, но ещё не опубликованные в открытой печати | нет | | |
| Тема квалификационной работы утверждена распоряжением по факультету \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ № \_\_\_\_\_\_\_\_\_\_\_от « \_\_\_ » \_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_ г. | |
| **Часть 1.** *Аналитическая часть* | |
| *Изучить средства разработки, предоставляемые трехмерным движком Unreal Engine 5. Провести продуктовое исследование существующих LLM моделей, сравнить их характеристики. Провести анализ реализации взаимодействия выбранной LLM модели с виртуальными агентами.* | |
| **Часть 2.** *Практическая часть 1. 3d-моделирование* | |
| *Создать трёхмерную сцену игрового мира с различными параметрами для агентов. Найти трёхмерные модели виртуальных агентов и настроить их анимации. Выполнить поиск и импорт ассетов в движок Unreal Engine 5.* | |
| **Часть 3.** *Практическая часть 2. Разработка внутриигровых систем.* | |
| *Интегрировать LLM модель в движок Unreal Engine 5. Разработать систему общения LLM модели с виртуальными агентами. Провести программную реализацию взаимодействия генеративных виртуальных агентов с внешним миром. Создать интерфейс, показывающий лог событий и данные о выбранных объектах, для конечного пользователя.* | |
| **Оформление выпускной квалификационной работы** | |
| Расчетно-пояснительная записка на 44 листах формата A4. | |
| Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.): | |
| |  | | --- | | *Работа содержит 7 графических листов формата A4. Лист 1* – *модель и компоненты виртуального агента «Крокодил». Лист 2* – *Модель и компоненты виртуального агента «Зебра». Лист 3* – *Общая часть дерева поведения персонажей, примеры поддеревьев действий виртуальных агентов. Лист 4* – *Биомы игровой сцены. Лист 5* – *Граф поз и анимации виртуальных агентов. Лист 6* – *Компоненты интерфейса для конечного пользователя. Лист 7* – *Примеры анимированных действий виртуальных агентов.* | | |
| Дата выдачи задания: «10» февраля 2024 г. | |
| В соответствии с учебным планом выпускную квалификационную работу выполнить в полном объеме в срок до «01» июня 2024 г. | |

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

|  |
| --- |
| 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** **\_\_\_\_\_\_\_\_\_\_\_** УТВЕРЖДАЮ

**КАФЕДРА** **\_\_\_\_\_\_\_\_\_\_\_\_\_\_** Заведующий кафедрой \_\_\_\_\_\_\_\_\_\_

(Индекс)

**ГРУППА** **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** \_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(И.О.Фамилия)

« \_\_\_\_\_ » \_\_\_\_\_\_\_\_\_\_\_\_ 20 \_\_\_\_ г.

**КАЛЕНДАРНЫЙ ПЛАН**

**выполнения выпускной квалификационной работы** студента:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(фамилия, имя, отчество)

Тема квалификационной работы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **№ 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*  *Планируемая дата* |  | Руководитель ВКР | Витюков Ф.А. |
|  | 1 часть Аналитическая часть | *18.02.2024*  *Планируемая дата* |  | Руководитель ВКР | Витюков Ф.А. |
|  | Утверждение окончательных формулировок решаемой проблемы, цели работы и перечня задач | *28.02.2024*  *Планируемая дата* |  | Заведующий кафедрой | Карпенко А.П. |
|  | 2 часть Практическая часть 1 | *21.04.2024*  *Планируемая дата* |  | Руководитель ВКР | Витюков Ф.А. |
|  | 3 часть Практическая часть 2 | *23.05.2024*  *Планируемая дата* |  | Руководитель ВКР | Витюков Ф.А. |
|  | 1-я редакция работы | *28.05.2024*  *Планируемая дата* |  | Руководитель ВКР | Витюков Ф.А. |
|  | Подготовка доклада и презентации | *04.06.2024*  *Планируемая дата* |  |  |  |
|  | Отзыв руководителя | *10.06.2024*  *Планируемая дата* |  | Руководитель ВКР | Витюков Ф.А. |
|  | Нормоконтроль | *15.06.2024*  *Планируемая дата* |  | Нормоконтролер | Грошев С.В. |
|  | Внешняя рецензия | *17.06.2024*  *Планируемая дата* |  |  |  |
|  | Защита работы на ГЭК | *20.06.2024*  *Планируемая дата* |  |  |  |

*Студент \_\_\_\_\_\_\_\_\_\_\_*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ *Руководитель работы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

(подпись, дата) (подпись, дата)

**АННОТАЦИЯ**

Выпускная квалификационная работа посвящена подходам для создания генеративных виртуальных агентов в движке Unreal Engine 5. Описана разработка системы взаимодействия с большой языковой моделью (large language models, LLM) для получения управляющих команд агентов.

В первой главе рассматривается создание игровой системы с окружающей средой, в которой виртуальные агенты будет осуществлять свою деятельность. Осуществляется подготовка и импорт трёхмерных моделей в движок Unreal Engine 5, настройка анимации для агентов.

Во второй главе описываются этапы разработки основных компонентов проекта для дальнейшего использования в движке, а именно: окружающая среда с различными параметрами, функции и дерева поведений виртуальных агентов, графический интерфейс для конечного пользователя с целью мониторинга объектов трёхмерной сцены.

В третьей главе показывается механизм «общения» LLM модели с виртуальными агентами, а также анализ эффективности использования LLM для управления ими.

В пояснительной записке 43 страницы, 24 рисунка, 7 графических листов.

Тип работы: выпускная квалификационная работа.

Тема работы: «Разработка метода интеграции больших языковых моделей средствами representational state transfer application programming interface (REST API) для управления виртуальными агентами в Unreal Engine 5».

Объекты исследований: 3d-моделирование, разработка внутриигровых систем, LLM, БЯМ.

**ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ**

**UE5** – трёхмерный движок Unreal Engine 5.

**Blueprints (BP) –** это система визуального скриптинга Unreal Engine 5.

**AActor (актёр)** – один из основных классов в UE5. Представляет собой любой объект, который может быть размещен на уровне.

**APawn** **(пешка)** – это класс, который является физическим и визуальным представлением того, чем управляет игрок или искусственный интеллект.

**ACharacter (персонаж)** – это наследник класса APawn, который, по умолчанию, имеет некоторые базовые функции движения. В отличие от пешек, у персонажей есть SkeletalMeshComponent – возможность включить расширенную анимацию, использующую скелет. К классам, производным от персонажа, можно добавлять другие скелеты, но это основной компонент, связанный с персонажем.

**USkeletalMeshComponent** –используются для всего, что имеет сложные анимации и использует скелет. Скелет (внешний вид) имеет внутри сложную конструкцию (взаимосвязанные кости), которая помогает перемещать отдельные вершины скелетной сетки в соответствии с текущей воспроизводимой анимацией.

**UStaticMeshComponent (SMC)** – компонент в UE5, позволяющий использовать Static Mesh.

**UInstancedStaticMeshComponent (ISMC)** – компонент в UE5, позволяющий использовать инстансинг.

**UHierarchicalInstancedStaticMeshComponent (HISMC)** – компонент в UE4, позволяющий использовать инстансинг. В отличие от ISMC, имеет поддержку LOD.

**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.

**Полигон** – многоугольник, являющийся базовым компонентом 3D-сетки. Основные типы: треугольник (tri), четырёхугольник (quad) и n-gon (5 или более вершин).

**Polycount** – количество полигонов модели.

**Текстура** – изображение, накладываемое на поверхность 3D-модели. Может содержать различные свойства поверхности, например: цвет, жёсткость (roughness), смещение (displacement), направление нормалей (normal map), и т.д.

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

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

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

**LLM** **(Large Language Model)** ­– большая языковая модель.

**Intelligent game agent (виртуальный агент) –** это программное обеспечение или часть программного обеспечения, которое наблюдает за своей средой, разумно мыслит и действует в соответствии с этой средой.

**Token (токен)** **–** это специальные маркеры, которые используются для обозначения определенных элементов в тексте, например, слов, начала и конца предложения, начала и конца абзаца, маркеров времени и т.д.

**Генеративный игровой агент** **–** это вид агентов, которые создаются на генеративных моделях для моделирования правдоподобного человеческого поведения, они демонстрируют симуляцию как индивидуального, так и группового поведения.

**AnimInstance –** это элемент, который используются для создания и управления сложным поведением анимации.

**AnimGraph** **–** элемент сграфом, используемый для выборки, смешивания и управления позами, которые будут применяться к скелетным сеткам с помощью Animation Blueprint (**AnimInstance BluePrint**).

**Representational State Transfer (REST) –** архитектурныерекомендации (стиль) по взаимодействию компонентов распределённого приложения в сети.

**Application protocol interface** **(API)** **–** описание способов взаимодействия одной компьютерной программы с другим**.**

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 9](#_Toc169627591)

[1. Создание игровой системы 11](#_Toc169627592)

[1.1. Правила игры 11](#_Toc169627593)

[1.2. Структура проекта 14](#_Toc169627594)

[1.3. Импорт ассетов в Unreal Engine 5 16](#_Toc169627595)

[1.4. Моделирование игрового поля 17](#_Toc169627596)

[1.5. Настройка анимации у моделей 18](#_Toc169627597)

[2. Разработка внутриигровых систем 20](#_Toc169627598)

[2.1. Описание классов 20](#_Toc169627599)

[2.2. Разработка виртуального агента «Крокодил» 24](#_Toc169627600)

[2.2.1. Разработка интерфейса виртуального агента 25](#_Toc169627601)

[2.2.2. Функции вычисления параметров 27](#_Toc169627602)

[2.2.3. Разработка узлов для дерева поведения 28](#_Toc169627603)

[2.3. Разработка виртуального агента «Зебра» 29](#_Toc169627604)

[2.3.1. Функции вычисления параметров 31](#_Toc169627605)

[2.4. Реализация камеры 32](#_Toc169627606)

[3. Разработка системы взаимодействия с LLM 34](#_Toc169627607)

[3.1. Обзор существующих моделей 34](#_Toc169627608)

[3.2. Принцип работы LLM 35](#_Toc169627609)

[3.3. Интеграция модели в движок 37](#_Toc169627610)

[3.4. Реализация общения с моделью 37](#_Toc169627611)

[4. Тестирование 39](#_Toc169627612)

[ЗАКЛЮЧЕНИЕ 41](#_Toc169627613)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 42](#_Toc169627614)

[ПРИЛОЖЕНИЕ А. Графическая часть выпускной квалификационной работы 44](#_Toc169627615)

ВВЕДЕНИЕ

Большие языковые модели (large language models, LLM) – это мощные модели искусственного интеллекта, которые используют возможности обработки естественного языка для эффективного понимания, создания и управления человеческим языком. Эти модели обучаются на огромных объемах текстовых данных, что позволяет им преуспеть в различных языковых задачах, таких как контекстно-зависимое понимание естественного языка, генерация текста, перевод и обобщение.

Данный проект является прототипом внедрения LLM модели внутри трёхмерного движка Unreal Engine 5 (UE 5). В контексте компьютерных игр можно отметить, что создание ботов - модели программных компонентов для игровых агентов – использует разные подходы и методы. Например, конечно-автоматная модель, модель поведения с применением машинного обучения, структура с расписанием действий. В данной работе, в качестве модели поведения игрового агента используются текстовые запросы и ответы LLM модели.

Для реализации эффективного взаимодействия с LLM моделью необходимо выбрать нужную модель, правильно формировать вопрос и ответ к ней, реагировать на некорректное поведение, продумывать различные сценарии использования модели, исследовать предметную область и рынок, анализировать качество уже придуманных решений.

Для быстрого создания прототипов игр в Unreal Engine 5 существует визуальная система скриптинга – Blueprint (BP). Система Blueprint основана на концепции использования интерфейса на основе узлов для создания элементов игрового процесса из редактора Unreal Editor. Как и многие распространенные скриптовые языки, Blueprint используется для определения объектно-ориентированных классов или объектов в движке. При запуске игры все Blueprint-скрипты переводятся на язык C++, ввиду чего игра на Blueprint может работать менее продуктивно, чем на C++. Система Blueprint чрезвычайно гибкая и мощная, поскольку дает дизайнерам возможность использовать практически весь спектр концепций и инструментов, обычно доступных только программистам. Кроме того, разметка для Blueprint, доступная в реализации Unreal Engine на C++, позволяет программистам создавать базовые системы, которые могут быть расширены дизайнерами [1].

В данном проекте рассматривается разработка элементов взаимодействия LLM модели с движком и пользовательским интерфейсом для игры жанра «Выживание».

Затрагивается и вопрос рациональности применения LLM моделей в UE и игровой индустрии, в частности.

1. Создание игровой системы
   1. Правила игры

Симулятор выживания ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) survival sim) — жанр [компьютерных игр](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%B8%D0%B3%D1%80%D0%B0), разновидность [симуляторов жизни](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BC%D1%83%D0%BB%D1%8F%D1%82%D0%BE%D1%80_%D0%B6%D0%B8%D0%B7%D0%BD%D0%B8), в которых основной целью игрока является сохранение жизни виртуального персонажа на фоне множества угрожающих ему опасностей. Элементы выживания содержатся практически во всех компьютерных играх, но в симуляторах выживания эта задача выдвинута на передний план и является главной в игре [2].

В работе были созданы два противоборствующих виртуальных агента: крокодил и зебра. Эти персонажи имеют следующие параметры:

* Здоровье (HP) – от очков здоровья зависит жив персонаж или нет. Здоровье медленно восстанавливается. Чем больше параметров находятся в критическом состоянии, тем быстрее уменьшается здоровье. Крокодил может восстанавливать здоровье быстрее, находясь в своём гнезде (блок nest). Диапазон значений – от 0 до 100 для крокодила и от 0 до 75 для зебры.
* Сытость (Satiety) – очки сытости от 0 до 100. При значении сытости меньше 0, персонаж теряет здоровье. Зебра может восполнить сытость на блоке травы (блок grass). Крокодил может пополнить сытость, если он пересечётся с зеброй при охоте. В таком случае, крокодил получает +25 к сытости, а зебра умирает. Если крокодил пересекается с зеброй, но не охотится, – ничего не произойдёт.
* Жажда (Thirst) – очки жажды от 0 до 100. Этот параметр есть только у зебры. При значении меньше 15, персонаж теряет здоровье. Зебра может пополнить жажду, находясь на блоке воды (блок water). Этот параметр очень быстро восполняется и уменьшается в процессе симуляции.
* Энергия (Energy) – очки бодрости от 0 до 100. Бодрость равная 100 означает, что персонаж полон сил. При 0 и меньше – виртуальный агент теряет здоровье. Крокодил может пополнить энергию только в своём гнезде. Зебра может пополнить энергию находясь везде, кроме блока воды.
* Температура (Temperature) – очки температуры. Максимальная температура равна 60 – это температура пустынного региона (блока). Минимальная температура равна -20 – это температура снежного региона. Изначально персонаж начинает с температурой равной 35. В зависимости от местонахождения, игровой агент понижает или повышает свою температуру. Температура меняется по закону охлаждения Ньютона:

,

где означает конечную температуру агента через 1 сек., – начальную температуру персонажа, – температура блока, k – коэффициент теплообмена. В проекте у крокодила он равен 0,19; у зебры – 0,25. При перегреве (по умолчанию 40), или переохлаждении (по умолчанию 14), персонаж теряет здоровье. Значения границ перегрева, переохлаждения могут быть изменены в настройках проекта.

* Скорость – скорость перемещения персонажа. У зебры максимальная скорость уменьшается только в пустыне или снежном регионе. Скорость крокодила же, в свою очередь, зависит от того, улучшил ли он своё гнездо или нет. Если гнездо не было модернизировано, то максимальная скорость крокодила будет зависеть от его местонахождения на игровом поле. Если гнездо было улучшено, то максимальная скорость крокодила будет выше на любом блоке, кроме пустыни и снега.
* Дерево – этот параметр есть только у крокодила. Дерево нужно для улучшения гнезда, которое даст бонус к максимальной скорости. Его можно добыть в блоке леса (блок forest). Крокодил может иметь с собой только одну единицу древесины. Данный параметр задаётся переменной типа bool. Диапазон значений – от 0 до 1. По умолчанию, улучшение гнезда требует только одно дерево.

Ниже приведён список возможных действий агента «Крокодил»:

1. go\_near\_<имя\_игровой\_области> – персонаж ищет ближайший от него блок нужного типа. Затем он начнёт движение к локации по прямой. Если виртуальный агент уже находится в данной локации, он не переместится, а система проинформирует персонажа об этом.
2. go\_random – персонаж выберет произвольную доступную точку на игровом поле и пойдёт к ней.
3. rest – виртуальный агент попробует отдохнуть и восстановить энергию до 100. Если крокодил не был в гнезде, система проинформирует персонажа об ошибке.
4. hunt – виртуальный агент начнёт охоту и будет следовать за ближайшей от него зеброй в течение 5 секунд. Выбранной жертве система отправит сообщение о том, что на неё начали охоту. Если за время охоты крокодил пересечётся с зеброй, она погибнет, а он пополнит сытость на 25 единиц.
5. go\_maxforest – персонаж отправится в лесную область с самым большим количеством дерева (wood).
6. get\_wood – виртуальный агент попробует получить 1 единицу дерева. Если крокодил не был в лесу, система проинформирует персонажа о том, есть ли у него сейчас единица дерева, а также где его можно получить или потратить.
7. upgrade\_nest – персонаж попробует потратить единицу древесины и улучшить гнездо. Если крокодил не был в гнезде, или у него не было дерева, – то система проинформирует виртуального агента нужным сообщением.

Также приведён список возможных действий агента «Зебра»:

1. go\_near\_<имя\_игровой\_области> – персонаж ищет ближайший от него блок нужного типа. Затем он начнёт движение к локации по прямой. Если виртуальный агент уже находится в данной локации, он не переместится, а система проинформирует персонажа об этом.
2. go\_random – персонаж выберет произвольную доступную точку на игровом поле и пойдёт к ней.
3. rest – виртуальный агент попробует отдохнуть и восстановить энергию до 100. Если зебра находилась в воде (на блоке water), система проинформирует персонажа об ошибке.
4. eat – виртуальный агент попробует поесть и восстановить сытость до 100. Если зебра находилась не на блоке травы, система проинформирует персонажа об ошибке.
5. drink – виртуальный агент попробует попить и восстановить жажду до 95. Если зебра находилась не в воде, система проинформирует персонажа об ошибке.

Примечание**:** зебра может отдыхать в пустыне, в снежном регионе и в лесу. Первые два региона несут потенциальный риск для виртуального агента. При этом, находясь в лесу зебра имеет больше шансов пережить охоту от крокодила (см. пункт Разработка виртуального агента «Зебра»).

Как видно, данная симуляция достаточно похожа на другие игры жанра выживания. У зебры есть преимущество в виде начальной скорости, при этом у неё есть дополнительный параметр жажды. Также у неё есть потенциальная угроза в виде охоты от крокодила. У крокодила есть возможность успешно охотиться на виртуальных агентов «Зебра», если он добудет и использует дерево для улучшения гнезда.

Однако модель LLM может допустить ряд ошибок или неправильно отвечать на запросы, что может привести к гибели любого виртуального агента.

* 1. Структура проекта

Файлы проекта имеют строгую иерархию. Все файлы с исходным кодом на C++ расположены в папке «Source», файлы конфигурации – в папке «Config», основной контент проекта и Blueprint-ы – в папке «Blueprints».

Иерархия файлов с исходным кодом изображена на рисунке 1:

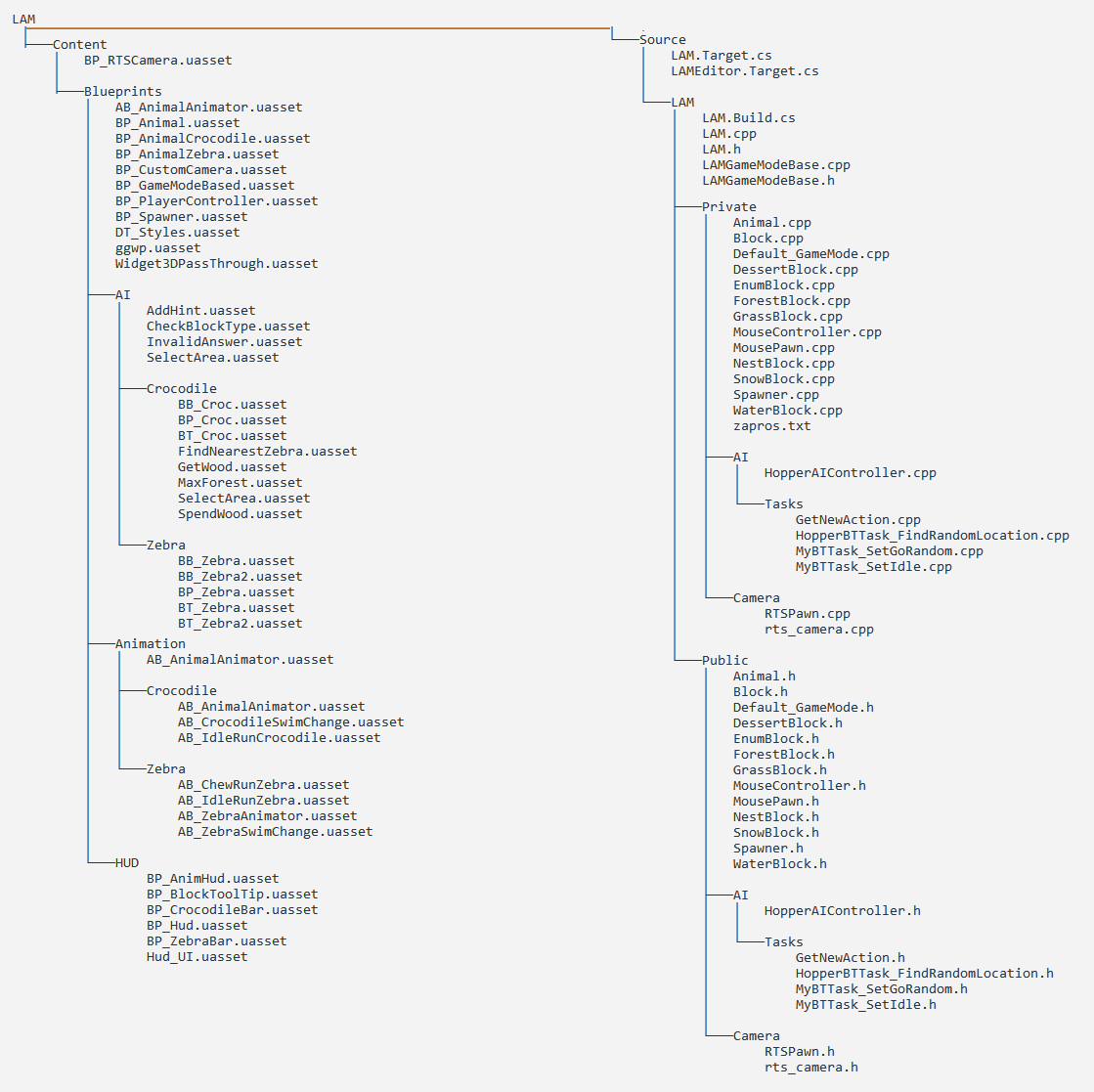


Рисунок 1 – Структура корневой папки проекта

В папке «Public» содержатся заголовочные файлы (.h), в «Private» – исходные (.cpp). Файлы в папке «Blueprints» содержат основные классы проекта, обязательные для его функционирования.

Файлы внутри папки «Blueprints» разделены по следующему принципу:

1. HUD: файлы с классами, отвечающими за интерфейс;
2. AI: файлы с классами, хранящие данные и дерево поведения виртуальных агентов; также там находятся разработанные компоненты для дерева поведения;
3. Animation: файлы с классами, отвечающими за анимации моделей персонажей;
4. Прочие файлы – классы виртуальных агентов, камеры и других вспомогательных объектов.

Аналогичный подход применяется к разделению контента проекта. К ранее описанному добавляются папки, содержащие различные ассеты, например: «Maps» (карты, уровни), «Widgets» (UMG-виджеты), «Materials» (материалы и текстуры моделей).

* 1. Импорт ассетов в Unreal Engine 5

Для визуального обогащения трехмерной сцены необходимы ассеты для игровых агентов и окружающей среды. Они были найдены в официальном магазине от Epic Games. Теперь их необходимо импортировать в движок. Для этого воспользуемся встроенной в Unreal Engine 5 утилитой. Вызвать ее можно через меню Add/Import в Content Browser’е (рис. 2):

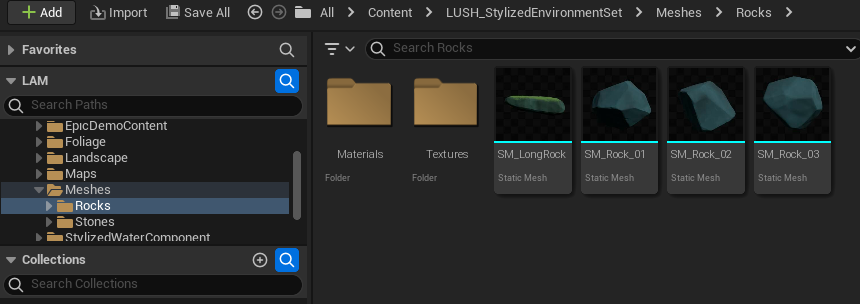


Рисунок 2 – Вызов утилиты импорта ассетов в Unreal Engine 5

Далее необходимо выбрать все необходимые файлы. В данном случае нас интересуют текстуры (набор .png файлов), файлы Static Mesh-ей, а также файлы моделей (.fbx).

Результат после импорта изображен на рисунках 3,4:

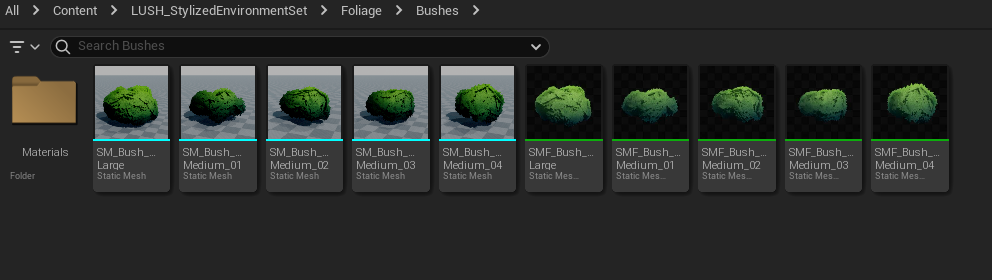


Рисунок 3 – Ассеты для окружающего мира в UE Content Browser после импорта

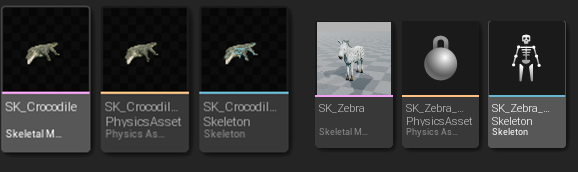


Рисунок 4 – Ассеты игровых агентов в UE Content Browser после импорта

* 1. Моделирование игрового поля

За генерацию игрового поля отвечает функция GenerateMap() объекта класса Spawner. По заранее заготовленной двумерному массиву строк WorldMap создаётся сцена из блоков различных классов. Все mesh-ы блоков имеют тип ISMC [6, 11].

Ниже даны эквиваленты символами строки карты:

“g” – зеленый блок травы класса AGrass. С вероятностью 75% для него создаётся модель травы, которая имеет случайный относительный поворот.

“w” – синий блок воды класса AWater. С вероятностью 50% для него создаётся модель кувшинки, имеющая случайную относительную позицию и поворот.

“f” – зеленый блок с лесом класса AForest. Создаётся вместе с моделями деревьев на нём. Параметр wood генерируется случайно в диапазоне от 1 до 15. Модель дерева зависит от числа древесины (wood).

“d” – желтый блок пустыни класса ADesert.

“s” – белый блок снега класса ASnow.

“n” – оранжевый блок гнезда класса ANest. Создаётся вместе с моделями камней.

Размеры блоков настраиваемые, как и модели их компонентов. Можно генерировать карты разных форм, если на определенной позиции в строке стоит неразрешённый символ. Тогда куб не поместится в сцену на нужной позиции.

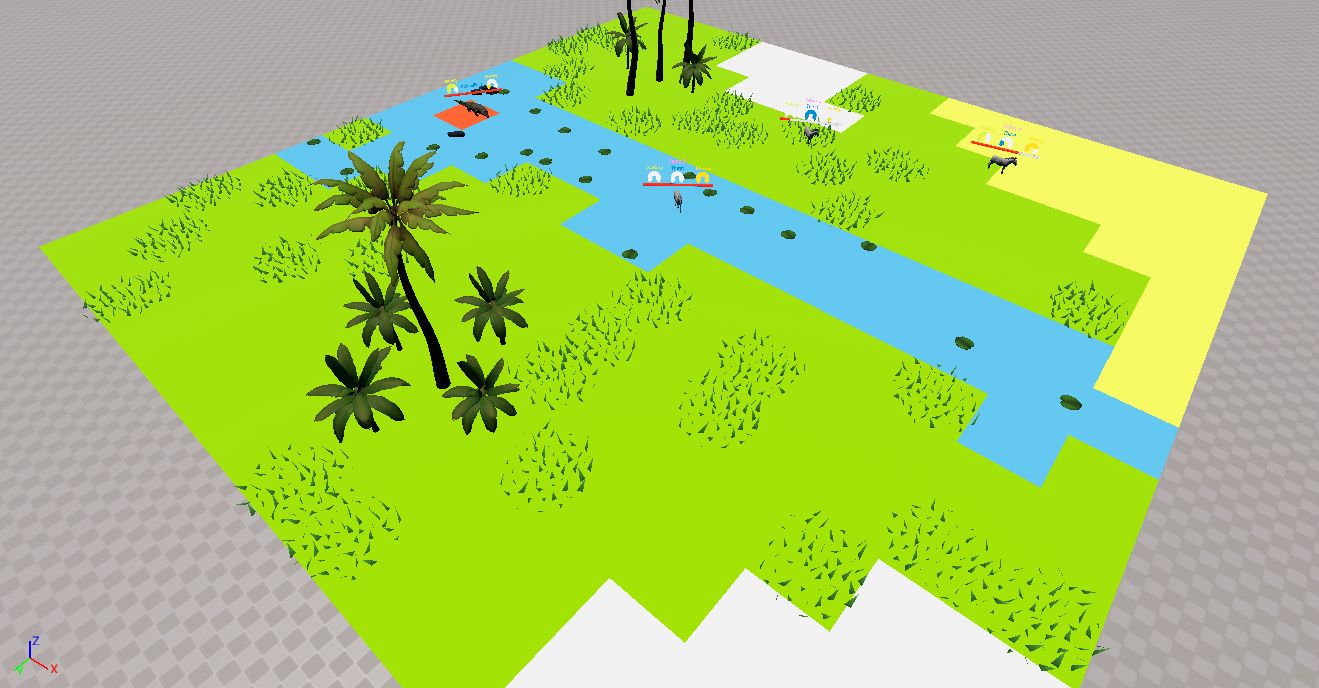


Рисунок 5 – Модель карты с различными блоками

* 1. Настройка анимации у моделей

За анимацию моделей отвечают объекты класса Animation Blueprint (AnimInstance). Они включают в себя граф поз – StateMachine (см. рис.6). В зависимости от параметров виртуальных агентов данные компоненты переключают состояния графа. За различные состояния (states), узлы графа, отвечают объекты типа BlendSpace.

Например, состояние позы Idle/Walk агента «Зебра» контролирует объект AB\_IdleRun\_Zebra. В зависимости от скорости компонент переключает анимацию модели между ожиданием, ходьбой, рысью, галопом.

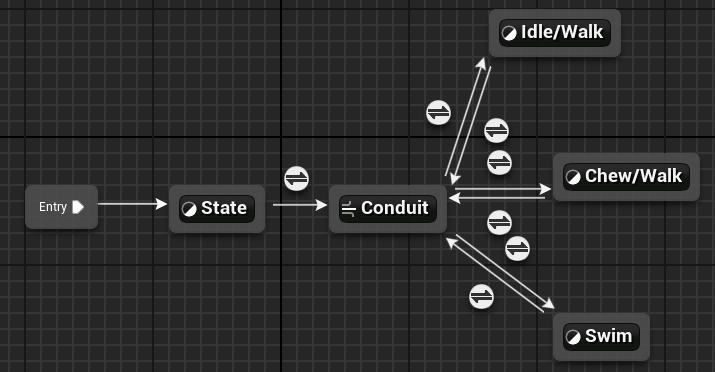


Рисунок 6 – Граф виртуального агента «Зебра»

1. Разработка внутриигровых систем
   1. Описание классов

Рассмотрим классы, созданные в рамках проекта.

Классы, размещаемые на 3D сцене (производные от AActor): [3, 5, 10]

1. ASpawner – класс, генерирующий игровое поле из кубов. Основными его целями являются размещение различных игровых областей в зависимости от символа из текстовой карты, сбор информации о симуляции от агентов, формирование запросов к LLM и приём ответов от неё. После инициализации объект формирует массив всех виртуальных агентов на сцене, запускает собственный таймер с периодом в 1 секунду, в котором проверяется готовность всех персонажей. Если на поле остался только один вид виртуальных агентов (только зебры или только крокодилы), симуляция завершается и ASpawner передает сообщение о победителе объекту BP\_HUD. В противном случае, собирается информация от каждого агента, формируется общий запрос, который отправляется большой языковой модели. После этого таймер ставится на паузу, идёт ожидание ответа от LLM. Если код запроса неуспешный (не равен 200), сформируется новый запрос с прежним текстом о текущем состоянии симуляции. Если запрос прошёл успешно, – полученный текст отправится на анализ каждому виртуальному агенту, а таймер снова запустится;
2. BP\_Animal – класс генеративного виртуального агента. От него наследуются все классы других персонажей. Содержит в себе модель, капсулу (capsule component), сферу (sphere component), скелет (skeletal mesh), анимацию движения, контроллер (AIController). Капсула представляет собой сферу, отвечающую за коллизию модели с землёй и другими объектами. Внутри капсулы также создан маленький параллелепипед, который будет соприкасаться с игровым блоком и отсылать информацию агенту о том, в какой территории он сейчас находится. Сфера представляет собой невидимую оболочку большого радиуса вокруг персонажа. Является триггером для проигрывания анимации смерти и уничтожения виртуального агента со сцены. Анимация движения (Animation Blueprint), в свою очередь, контролирует тип перемещения агента (ходьба или плавание) в зависимости от информации из капсулы. Также этот компонент обеспечивает плавность анимации передвижения и поворота в зависимости от скорости агента. Данный подобъект несёт исключительно декоративную функцию. Также класс BP\_Animal содержит в себе следующие функции: BeginPlay(), ReceiveTick(), CheckHP(), AnalyzeText, SetGameMoveText. Первая функция – является аналогом конструктора класса, вызывается при создании или «спавне» объекта на сцене. В ней инициализируются все компоненты, также она запускает внутренний таймер для расчёта параметров виртуального агента. Функция ReceiveTick() вызывается каждый игровой тик (период ~ 0,17 секунд, зависит от FPS) и синхронизирует с остальными подсистемами важные параметры персонажа, такие как координаты текущего местоположения модели, игровая область, где находится виртуальный агент. Также ReceiveTick проверяет не опустилось ли здоровье персонажа ниже 0. Функция AnalyzeText фильтрует текст запроса, отправленному агенту, и добавляет проанализированное действие в очередь. SetGameMoveText отвечает за формирование текста о текущем состоянии персонажа, который получит объект Spawner.
3. ABlock – класс, производный от ISMC. Представляет собой блок, по умолчанию область с травой, от которого наследуются все остальные классы блоков. Блок имеет параметры max\_speed, max\_temperature – первый отвечает за максимально допустимую скорость игрового агента, второй за его температуру воздуха. Далее описаны все типы блоков.

Классы, производные от ABlock:

1. AGrassBlock – класс-наследник от ABlock. Это обычная область со степью. Не имеет штрафов к максимальной скорости, температура чуть ниже начального значения персонажа. С вероятностью 75% для неё создаётся модель травы, которая имеет случайный относительный поворот;
2. AForestBlock – класс-наследник от ABlock. Это область с лесом. Не имеет штрафов к максимальной скорости, температура такая же как у области с травой. Содержит в себе целочисленную переменную wood, сгенерированную заранее со значениями от 1 до 15. От этой переменной зависит модель дерева, которая расположена на этом блоке. Дерево несёт декоративную функцию, также позволяет визуально отличить блоки травы от блоков с лесом;
3. ADesertBlock – класс-наследник от ABlock. Это пустынная область. Температура гораздо выше начальной температуры игрового агента. Также имеет штраф к максимальной скорости. Является основным препятствием, находясь на ней агент может терять здоровье из-за перегрева;
4. ASnowBlock – класс-наследник от ABlock. Это заснеженная область. Не имеет штраф к скорости. Имеет отрицательную температуру, игровой агент может терять здоровье из-за переохлаждения;
5. AWaterBlock – класс-наследник от ABlock. Это область с водой. С вероятностью 50% для неё создаётся модель кувшинки, имеющая случайную относительную позицию и поворот. Персонаж, находясь в этой области, начинает плавать. Скорость плавания чуть ниже максимальной скорости ходьбы. ;
6. ANestBlock – класс-наследник от ABlock. Это гнездо для виртуального агента «Крокодил». Находясь в нём, персонаж восстанавливает здоровье и усталость. Не имеет штрафов к скорости. Имеет температуру равную начальной температуре персонажа. Область создаётся вместе с моделями камней на границах.

Классы, расширяющие функциональность проекта:

1. BP\_HUD – класс интерфейса. Содержит в себе объект типа UI и компонент LLM модели. Обеспечивает основной бизнес процесс по взаимодействию с нейросетью.
2. BP\_AnimHud – класс UMG Widget, позволяет в режиме реального времени выводить в правом верхнем углу состояние его показатели выбранного игрового агента.
3. BP\_BlockToolTip – класс UMG Widget, появляется при нажатии левой кнопкой мыши по игровому блоку. Позволяет в режиме реального времени выводить имя, температуру и др. показатели куба.
4. Hud\_UI - класс UI. Содержит в себе поле с textbox и scrollbar, отображающим все входные и выходные текстовые запросы в виде чата. Под этим объектом находится поле text input, которое позволяет корректировать действия виртуальных агентов, по желанию. Слева от text input расположена кнопка класса RadioButton, позволяющая скрывать/раскрывать чат. Справа от поля запроса расположена кнопка restart. При нажатии на неё все ранние запросы к LLM завершаются, и симуляция начинается заново.
5. AI\_Anim\_Controller – класс типа AIController. Взаимодействует с объектами типа BlackBoard и BehaviourTree. BehaviourTree – это компонент, позволяющий создавать дерево поведения персонажа . BlackBoard – хранилище данных для персонажа, управляющее поведением в BehaviourTree. Каждый BlackBoard виртуального агента содержит следующие компоненты: вектор TargetLocation – координаты конечной точки маршрута, строку Action – выбранное действие персонажа, переменную типа actor SelfActor – указатель на персонажа, а также переменную AllowedToMove типа bool – её значение меняет объект Spawner.

Базовое дерево помимо стандартных компонентов MoveTo и Wait имеет созданные узлы GetActionName, FindRandomLocation, SetActionToIdle, InvalidAnswer. Стоит отметить, что большую часть времени агент находится в состоянии idle и ждёт следующее действие. Параметры виртуального агента не меняются при action = idle. Если персонаж получит неправильное действие, то сработает узел InvalidAnswer. После него агент отправится в случайное место (action = go\_random). Данное решение мотивировано тем, чтобы персонаж долго не стоял на одном и том же месте в состоянии idle.

Сам контроллер имеет в себе строковый массив Actions, с очередью действий.

* 1. Разработка виртуального агента «Крокодил»

Для персонажа «Крокодил» был разработан класс BP\_Animal\_Crocodile, который наследуется от класса BP\_Animal (см. рис. 7).



Рисунок 7 – Трёхмерная модель крокодила с его компонентами

На рис. 7 представлены следующие компоненты:

* Капсула (capsule component)
* Сфера (sphere component)
* Параллелепипед (collision box component)
* Модель крокодила (skeletal mesh)
* Модель стрелки (arrow component)
* HUD интерфейс с параметрами агента (Bar widget)

Параллелепипед отвечает за коллизию с игровым блоком, сфера – за пересечение с другими моделями персонажей. Стрелка несёт декоративную функцию, чтобы видеть направление поворота модели виртуального агента.

* + 1. Разработка интерфейса виртуального агента

Для удобного мониторинга параметров виртуальных агентов, необходимого конечному пользователю, разработаны 3D Widget-ы. Данные компоненты входят в состав моделей игровых агентов (рис. 8). Они состоят из объекта ProgressBar для отображения параметра здоровья, а также из объектов RadialSlider для прочих параметров. Объект TextBox нужен для отображения имени виртуального агента.

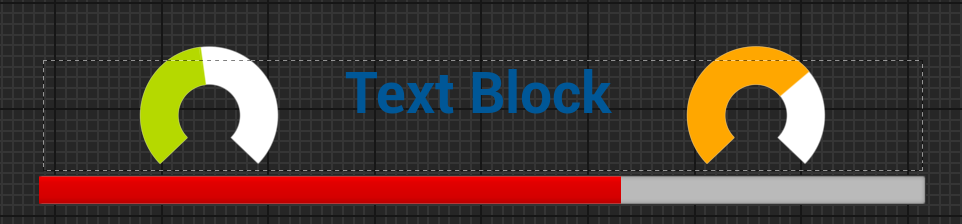
****

Рисунок 8 – Объект BarWidget для агента «Крокодил»

Архитектура интерфейса для крокодила представлена на рисунке 9.

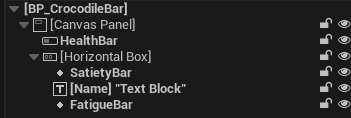
****

Рисунок 9 – Архитектура BarWidget для агента «Крокодил»

Параметры визуальных компонентов изменяются в функции Tick(). Там же происходит расчёт положения Widget-а относительно экрана.

Стоит упомянуть, что существует два типа отрисовки Widget-а: World и Screen. При значении World объект помещается на сцену, в соответствии с настройками модели. При значении Screen объект отрисовывается непосредственно на экране у игрока (в данном случае игрок представляет собой подвижную камеру сверху), также он масштабируется в зависимости от настроек экрана (рис. 10). Опция Screen работает корректно в исключительных случаях, поэтому для объекта установлен режим рендера World с последующей программной реализации поворота, в соответствии с камерой игрока (рис. 11). Результат показан на рисунке 12.



Рисунок 10 – Отображение 3D Widget в режиме Screen

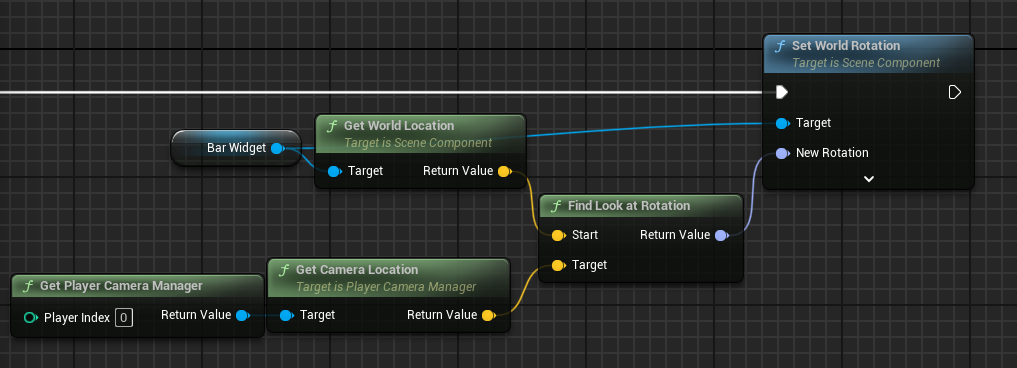


Рисунок 11 – Поворот объекта 3D Widget относительно местоположения камеры игрока

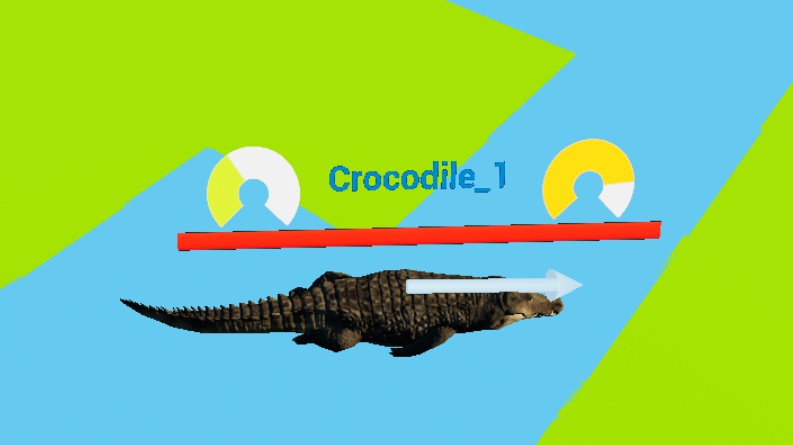


Рисунок 12 – Отображение 3D Widget в режиме World с программной реализацией поворота

* + 1. Функции вычисления параметров

Так как жанр игры «Выживание», игровой агент должен терять очки своих потребностей исходя из условий внешней среды. Чтобы параметры персонажа не изменялись слишком быстро, был создан таймер с периодом (тиком) 3 секунды. Он работает только, если виртуальный агент находится в состоянии отличном от idle и значение переменной AllowedToMove от BlackBoard из BehaviorTree равно true.

Ниже представлены функции, вычисляющие параметры данного агента. Функция CalcTemp считает температуру агента каждый тик таймера.

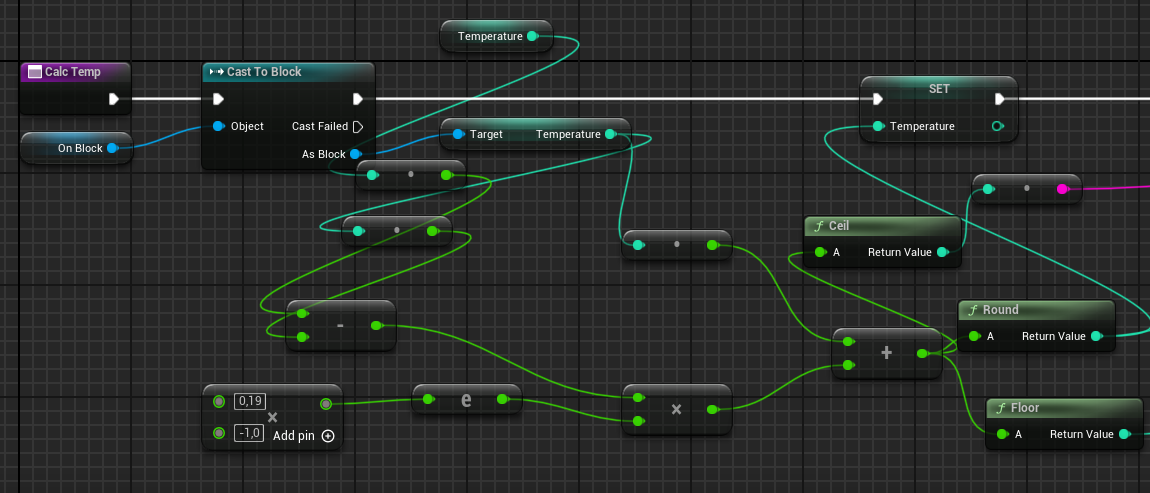


Рисунок 13 – BP функции CalcTemp

Функция CalcFatigue вычитает из параметра усталости игрового агента определённое значение каждый тик таймера (см. рис. 14). В зависимости от типа блока, на котором стоит агент (Вода, Снег, Пустыня, Гнездо) меняется вычитаемая/прибавляемая величина. Также для агента «Крокодил» эта функция из-за обращения к блоку гнезда (Nest), может инкрементировать значение здоровья персонажа. Сделано в целях оптимизации.

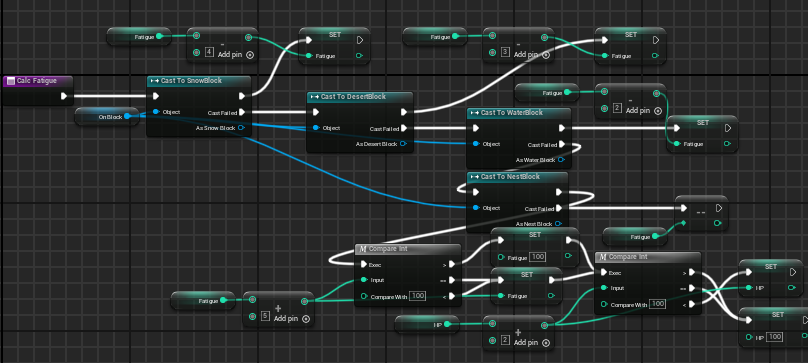


Рисунок 14 – BP функции CalcFatigue

Функция CalcHP исходя из значения всех трёх величин потребностей игрового агента, а именно (энергия, сытость, температура) изменяет значение здоровья каждый тик таймера (см. рис. 15). Стоит отметить, что данный персонаж всегда регенерирует 1 очко здоровья вне гнезда.

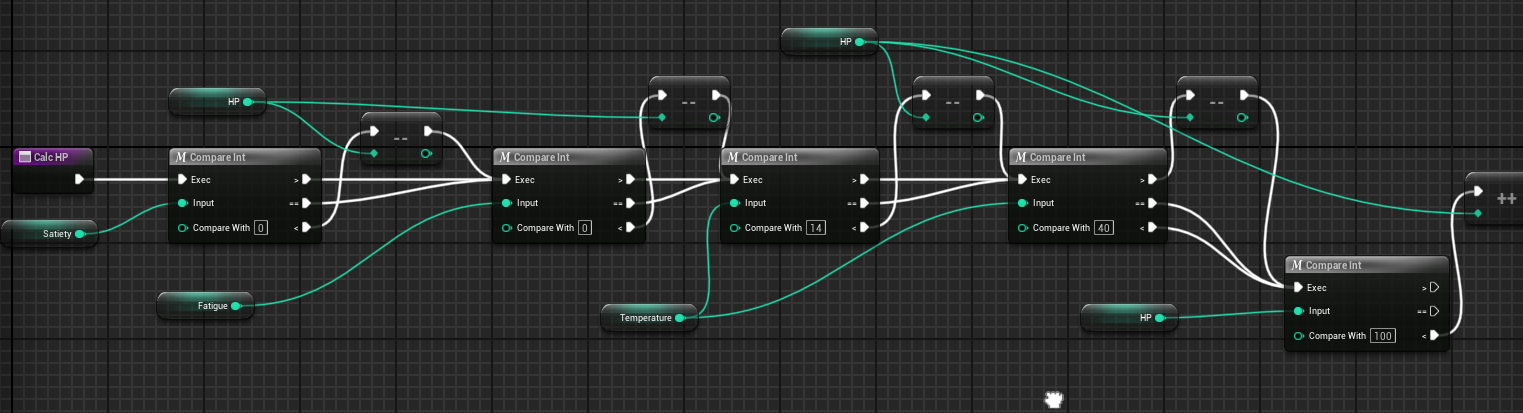


Рисунок 15 – BP функции CalcHP

* + 1. Разработка узлов для дерева поведения

Для дерева поведения персонажа «Крокодил» был разработаны собственные действия, соответствующие правилам игры. Для реализации охоты разработано поведение (task, узел дерева) под названием FindNearestZebra (см. рис. 16).

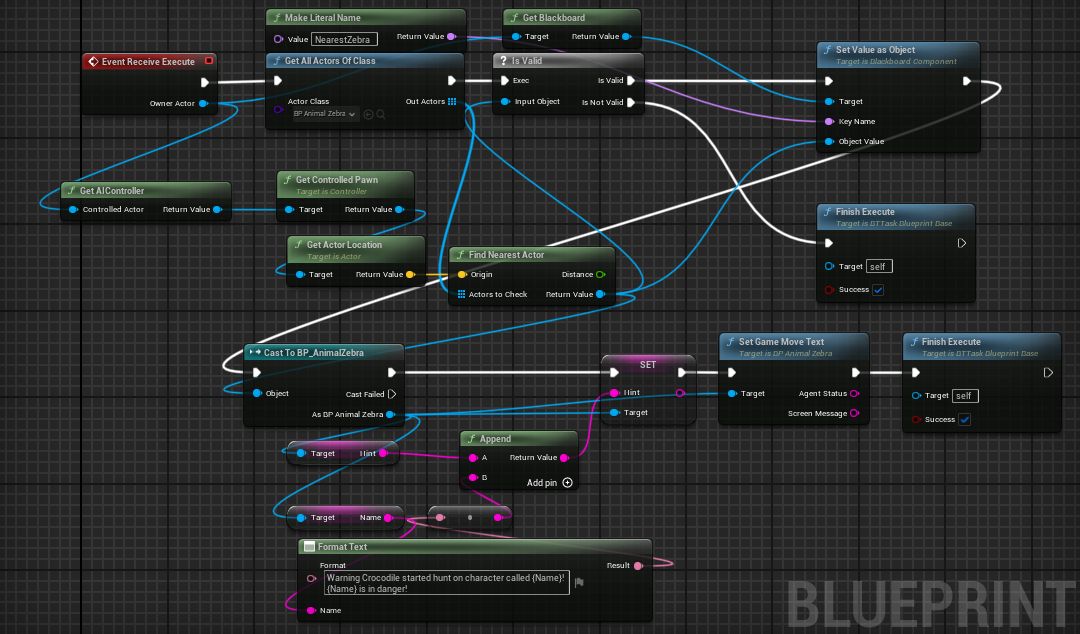


Рисунок 16 – Действие FindNearestZebra

В данной функции, во-первых, находятся координаты ближайшей зебры по отношению к крокодилу, совершающему данное действие. Во-вторых, агент зебры (жертвы) получает подсказку о том, что на него начали охоту.

Для реализации механики с получением дерева и улучшением гнезда были разработаны 2 task-а TryGetWood, TrySpendWood. В зависимости от значения переменной Wood система выдаёт агенту подсказку о том, удалось ли получить/потратить единицу дерева.

* 1. Разработка виртуального агента «Зебра»

Для персонажа «Зебра» был разработан класс BP\_Animal\_Zebra, который наследуется от класса BP\_Animal (см. рис. 17).

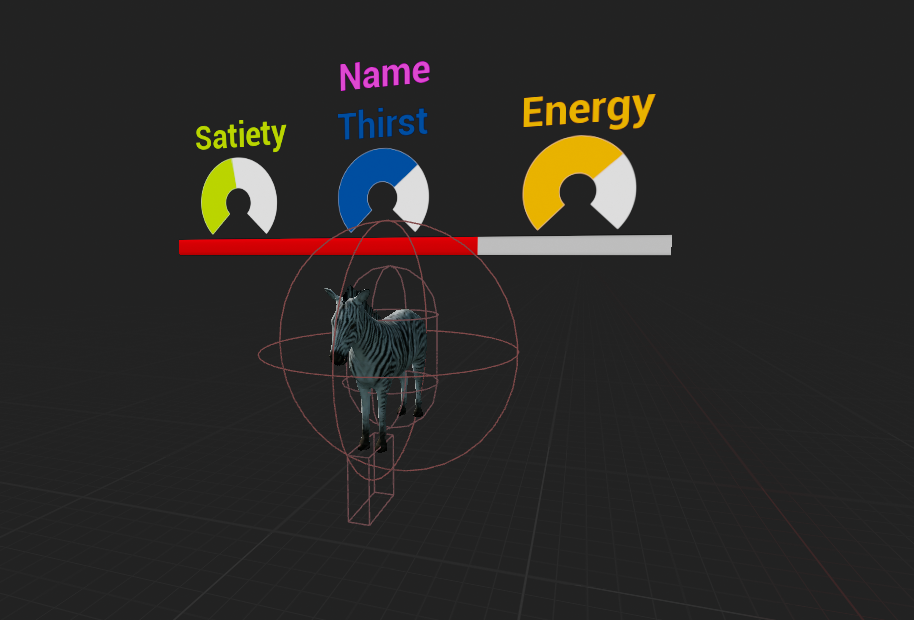


Рисунок 17 – Трёхмерная модель зебры с его компонентами

Компоненты имеют такой же функционал, как и у агента «крокодил». Разница лишь в том, что компонент сферы имеет меньший радиус. Также он уменьшает свои размеры, если виртуальный агент находится в регионе леса.

Компонент BarWidget видоизменён (см. рис. 18), его архитектура представлена на рисунке 19.

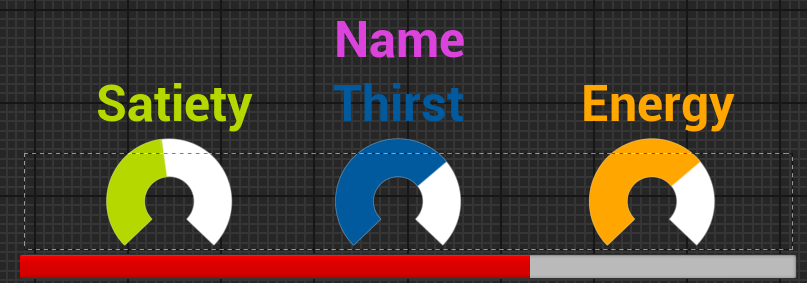


Рисунок 18 – Объект BarWidget для агента «Зебра»

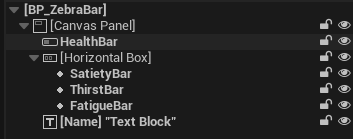
****

Рисунок 19 – Архитектура BarWidget для агента «Зебра»

* + 1. Функции вычисления параметров

Функции, вычисляющие параметры данного агента не отличаются от компонентов персонажа «Крокодил». Однако у зебры есть характеристика «Жажда» (Thirst). Для расчёта данной характеристики вместо функции для BP\_Animal\_Zebra был разработан узел дерева поведения AddWater (рис. 20).

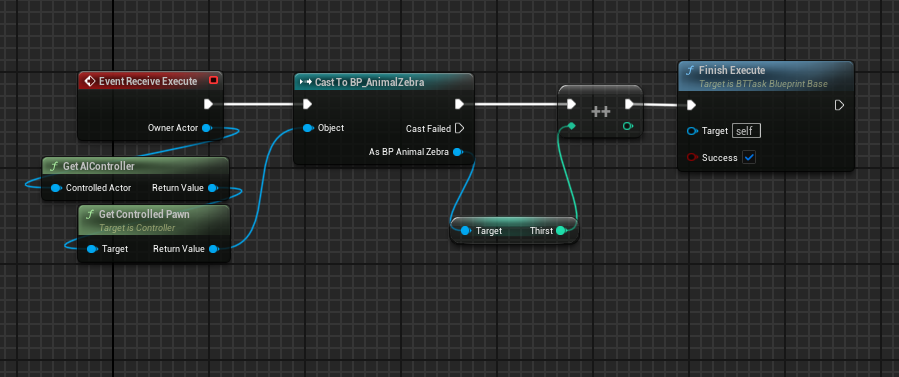


Рисунок 20 – Действие AddWater

Важно отметить, что данный task вызывается в поддереве поведения виртуального агента (рис. 21). Если предусловия сработают положительно (модель находится на блоке воды и параметр жажды меньше 95), то цепочка выполнения замкнётся в цикле до тех пор, пока параметр не будет больше или равен 95. Благодаря этому, параметр thirst заполняется каждый игровой тик.

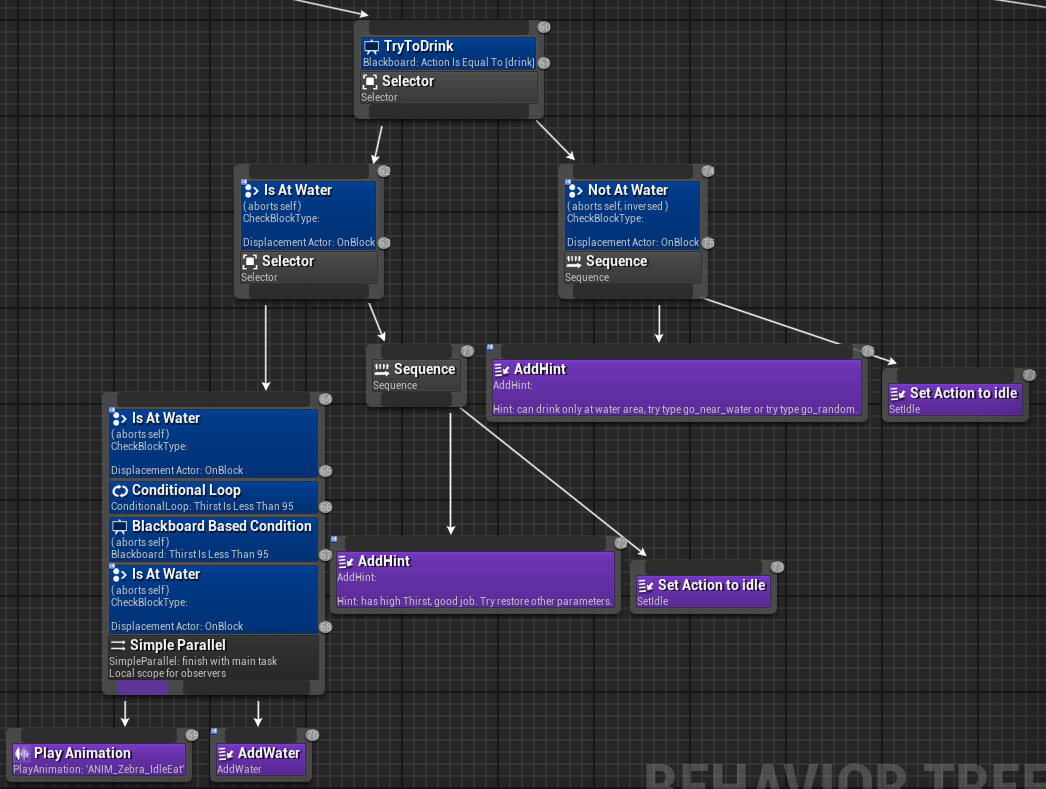


Рисунок 21 – Поддерево поведения TryToDrink агента «Зебра»

* 1. Реализация камеры

Для удобства тестирования проекта и улучшения визуальной составляющей проекта была создана подвижная камера. Перемещение её осуществляется при помощи клавиш клавиатуры W,A,S,D; поворот против часовой стрелки на Q, по часовой – на E. Колесо мыши отвечает за приближение/отдалении камеры. При нажатии на X камера перемещается на начальное положение, поворот и приближение сбрасывается до значений по умолчанию. Привязки на нажатия клавиш можно поменять в файлах типа Data Asset (см. рис. 22).

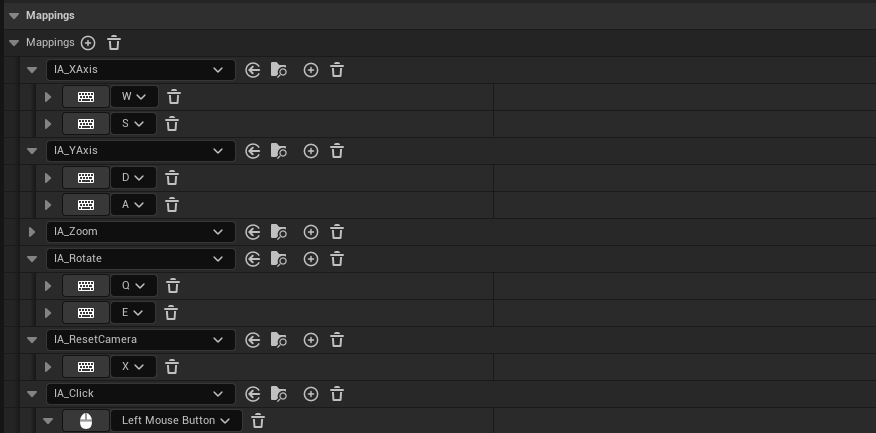


Рисунок 22 – Изменение конфигурации управления камерой

Также в функционал камеры добавлена возможность левой кнопкой мыши выделять цветные блоки из трёхмерной сцены. Результат данного действия описан в предыдущем пункте про интерфейс.

За программную реализацию камеры отвечает класс ARTSPawn. Для удобного изменения параметров внутри UE используются производные от C++ классов компоненты BP\_CustomCamera, BP\_PlayerController, BP\_GameModeBase. В последнем из них выбирается нужный класс игровой сессии, интерфейса игрока, контроллера игрока, класс камеры по умолчанию и другие (рисунок 23).

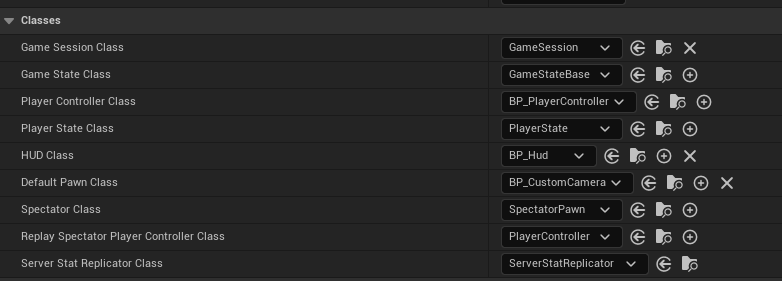


Рисунок 23 – Настройки класса GameModeBase

1. Разработка системы взаимодействия с LLM
   1. Обзор существующих моделей

В настоящее время доступно несколько LLM, и новые разрабатываются по мере роста спроса на возможности обработки естественного языка. Некоторые популярные примеры включают в себя: [12]

* GPT-4 от OpenAI
* BERT от Google
* Gemma от Google
* RoBERTa от Facebook
* Microsoft Turing-NLG
* LLaMA (Large Language Model Meta AI)
* Mistral (Mistral AI)

Некоторые из моделей требуют постоянное подключение к интернету. Исходя из потребностей проекта, принято решение скачать ряд моделей, а именно: LLaMa2-7b, Gemma-7b, LLaMa3-8b.

LLaMa2-7b – это простая модель, настройки которой позволяют задавать системный контекст (в нашем случае – это роль игрока, управляющего виртуальным агентом). Gemma-7b – открытая модель от компании Google, разработанная на базе закрытой Gemini. Она более «тяжёлая», т.к. она обучена на большем числе параметров. LLaMa3-8b, как утверждают разработчики: «Самая мощная из открытых LLM на сегодняшний день».

Почти все модели умеют общаться в json формате и других грамматиках. В данной работе в общении предусмотрен формат json для повышения качества ответа у LLM модели. Стоит отметить, что на этапе создания проекта был найден единственный доступный плагин в UE5 с офлайн моделью llama. Однако практически любую модель в офлайн и онлайн версиях, работающую с API, можно интегрировать в UE.

* 1. Принцип работы LLM

LLM Модели работают с помощью текстовых запросов. Принцип работы состоит в следующем:

1. на вход модели подается контекст (любой текст);
2. модель анализирует весь контекст и генерирует всего один токен (токен - слово, часть слова или символ);
3. модель добавляет токен к контексту и передает новый контекст в пункт 1, до тех пор, пока не будет получен ответ.

Языковая модель имеет ограничение на суммарное количество токенов, которое мы можем передать на вход и получить на выходе. В основном, это зависит от размера модели. Токен это слово или часть слова, символ и тд. 1 токен не всегда равен 1 слову. Для английского языка 1000 токенов в среднем равны 750 словам. Для русского языка 1000 токенов это всего около 375 слов [7, 8].

Также стоит отметить различные параметры LLM моделей:

1. **Temperature** (Температура). Чем ниже температура, тем более детерминированными являются результаты в том смысле, что всегда выбирается наиболее вероятный следующий токен. Повышение температуры может привести к большей случайности, что будет способствовать более разнообразным и творческим результатам. По сути, вы увеличиваете вес других возможных токенов. С точки зрения применения вы можете использовать более низкое значение температуры для таких задач, как контроль качества на основе фактов, чтобы стимулировать более фактические и краткие ответы. Для создания стихотворений или других творческих задач может быть полезно увеличить значение температуры.
2. **Top P** — метод выборки с температурой, называемый выборкой ядра, при котором можно контролировать степень детерминированности модели. Если нужны точные и фактические ответы, данный параметры должен быть на низком уровне. Если требуются более разнообразные ответы, значение следует увеличить. Если в модели есть параметр Top P, это означает, что для ответов учитываются только токены, составляющие вероятностную массу top\_p, поэтому низкое значение top\_p выбирает наиболее уверенные ответы. Это означает, что высокое значение top\_p позволит модели просмотреть больше возможных слов, в том числе менее вероятных, что приведет к более разнообразным результатам. Общая рекомендация – изменить температуру или Top P, но не оба сразу.
3. **Max Length (**максимальная длина). Позволяет управлять количеством токенов, генерируемых моделью, регулируя максимальную длину. Указание максимальной длины помогает предотвратить длинные или ненужные ответы и контролировать затраты.
4. **Stop sequence** (стоп-последовательности). Это строка, которая останавливает генерации токенов у модели. Указание последовательности остановки — это еще один способ контролировать длину и структуру ответа модели. Например, можно указать модели генерировать списки, содержащие не более 10 элементов, добавив «11» в качестве стоп-последовательности.
5. **Frequency Penalty** (штраф за частоту). Штраф за частоту накладывает штраф на следующий токен, пропорциональный тому, сколько раз этот токен уже появлялся в ответе и подсказке. Чем выше штраф за частоту, тем меньше вероятность того, что слово появится снова. Этот параметр уменьшает повторение слов в ответе модели, назначая более высокий штраф за лексемы, которые кажутся более выраженными.
6. **Presence Penalty (**штраф за присутствие). Штраф за присутствие также применяет штраф к повторяющимся токенам, но, в отличие от штрафа за частоту, штраф одинаков для всех повторяющихся токенов. Токен, который появляется дважды, и токен, который появляется 10 раз, наказываются одинаково. Эта настройка не позволяет модели слишком часто повторять фразы в ответе. Если требуется, чтобы модель генерировала разнообразный или креативный текст, можно использовать более высокий штраф за присутствие. Чтобы модель оставалась детерминированной, используется более низкий штраф за присутствие [4].
   1. Интеграция модели в движок

Для проекта было принято решение работать с API различных моделей. В UE5 для этого есть бесплатный плагин VaRest. Для тестирования локальных LLM был установлен кроссплатформенное ПО Ollama [14]. Данная программа предоставляет возможность работать как с известными моделями, так и с собственными. Для тестов использовались встроенные в Ollama модели Gemma-7b и LLaMa3-8b, а также собственноручно скачанная модель LLaMa2-7b – она была также внесена в список моделей, запускаемых с помощью Ollama.

* 1. Реализация общения с моделью

Для взаимодействия с моделью внутри UE 5 используется плагин VaRest. Он включает в себя Blueprint-ы для работы с запросами и ответами в формате JSON. Данные компоненты внесены в функции и события объекта Spawner. Функция CreateRequest() создаёт запрос включающий в себя: url-адрес, заголовки headers, строковые поля ключей JSON и их значения. Основным полем является prompt, в котором содержится отправляемое сообщения для LLM. Также можно настроить системное сообщение для модели, где формируются инструкции и роль для неё. В проекте системный контекст заполняется правилами игры и описанием каждого из персонажей. В обычное сообщение каждого запроса помещается текущее состояние каждого из виртуальных агентов.

Чтобы дождаться ответа LLM, в классе Spawner реализованы события-делегаты OnLLMAnswer и OnRequestFail. В конце тела функции CreateRequest() привязываются эти события-обработчики (рис. 24). При получении запроса срабатывает один из двух обработчиков.

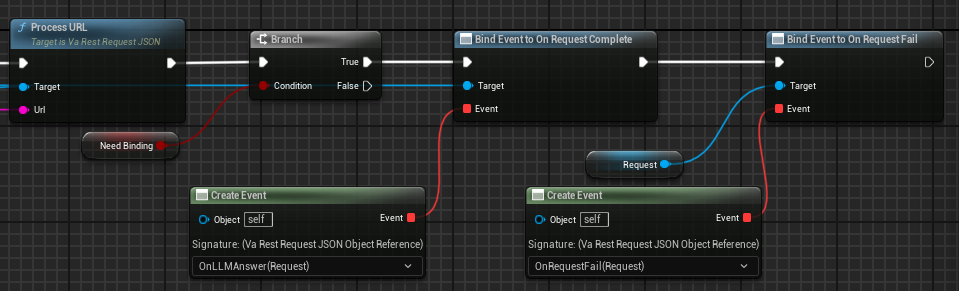


Рисунок 24 ­– События-делегаты у функции CreateRequest.

Если сработает обработчик OnLLMAnswer (код ответа равен 200), то объект Spawner разошлёт всем виртуальным агентам ответ модели, а его таймер продолжит свою работу до тех пор, пока симуляция не завершится чьей-либо победой или пока агенты не остановятся. Если действие в тексте не было распознано виртуальным агентом, то он отправится в случайную точку на поле. При этом, он добавит к своему сообщению-подсказке информацию для LLM, что её действие не было воспринято правильно. К тексту также прикрепится список всех возможных действий данного агента. Виртуальные агенты после действий перейдут в состояние idle и сформируют сообщение о своих характеристиках. К их сообщениям может быть прикреплены и другие подсказки. Объект Spawner соберёт сообщения со всех существующих агентов, сформирует запрос и отправит его LLM модели.

Если сработает обработчик OnRequestFail, Spawner отправит предыдущее сообщение о состоянии симуляции по тому же адресу. Также выведется системное предупреждение о неудачном запросе.

1. Тестирование

Проект запускался на ПК со следующими характеристиками:

Процессор Intel(R) Core(TM) i5-2400 CPU @ 3.10GHz 3.30 GHz

Оперативная память 16,0 ГБ

Видеокарта NVIDIA GeForce GTX 1050 Ti

Тип системы 64-разрядная операционная система, процессор x64

В процессе симуляции возникали проблемы с нехваткой видеопамяти, при этом диспетчер ресурсов показывал, что видеокарта почти не используется. Консольные команды для Ollama показывали, что процессор использовался на 60%, видеокарта на 40%. Среднее время успешного ответа от LLM составляло от 30 секунд до 2,5 минут. Максимальная задержка ответа достигалась в начале симуляции, когда модели передавался системный контекст (system prompt).

Возможными причинами таких задержек являются:

1. Недостаточные мощности ПК.
2. Недостаточно тонкая настройка среды UE 5.

Также проект запускался на ПК с другими характеристиками:

Процессор Intel(R) Core(TM) i7-7700K @ 4.50 GHz

Оперативная память 16,0 ГБ

Видеокарта NVIDIA GeForce RTX 3060 Ti

Тип системы 64-разрядная операционная система, процессор x64

В процессе работы проблем с производительностью не было. Среднее время успешного ответа от LLM составляло от 3 секунд до 8 секунд. При этом из трёх локальных моделей лучше всего справилась Gemma-7b, она сразу формировала команды для всех виртуальных агентов. LLaMa3-8b нередко выдавала неполноценные ответы. При этом все три модели очень плохо реагировали на подсказки в сообщениях. Изменение параметров моделей не дало значительных успехов. Стоит упомянуть, что и локальные, и онлайн LLM (GPT 3.5 turbo) часто выдавали одни и те же команды для одинаковых агентов, игнорируя подсказки.

В результате можно сказать, что применение LLM в играх может быть обусловлено жанром разрабатываемого проекта. По понятным причинам, модель LLM не подойдут для интеграции с активными играми (например, FPS жанра). Также нежелательно использование таких моделей в играх с реальным течением игрового времени. LLM является универсальным инструментом, который, в свою очередь, нужно правильно настраивать и корректировать, чтобы не получать на выходе неадекватные ответы и действия. Очень важна техника *«промпт-инжиниринга»* [13]. Модели можно дообучать на больших данных для конкретизации её назначения. Для LLM можно настроить хорошее воображение (temperature), чтобы выдавать необычное поведение для игровых агентов. Однако любая модель имеет лимит на память предыдущих токенов, поэтому ей периодически придётся напоминать о правилах игры и предыдущих действиях.

Стоит упомянуть, что LLM на данный момент чаще используют для генерации реплик, картинок, текстов, уровней; нежели чем для более строгих и математически обусловленных операций [9].

ЗАКЛЮЧЕНИЕ

В результате выполнения работы были получены навыки работы с Unreal Engine 5, получен опыт разработки на C++. Освоены навыки 3D-моделирования, анимирования, создания дерева поведения. Изучены способы взаимодействия, настройки и интеграции модели LLM в движок Unreal Engine 5.

В работе описан принцип работы LLM моделей, способы и особенности взаимодействия с ними. Рассмотрены популярные модели LLM, которые можно использовать для генерации управляющих команд игровых агентов. Изучены возможности, достоинства, недостатки применения LLM в игровой индустрии.

В процессе работы выполнены следующие задачи:

1. создание сцены из ISMC кубов;
2. создание персонажей с различными характеристиками;
3. интеграция различных LLM моделей в движок;
4. синхронизация LLM модели с симуляцией;
5. синхронизация действий виртуальных агентов;
6. работа с анимацией.

Итогом выпускной квалификационной работы стал проект, симулирующий деятельность генеративных виртуальных агентов в разнообразном мире под контролем LLM модели. Такой же принцип взаимодействия с LLM можно использовать и в других работах.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Unreal Engine 5 Documentation [электронный ресурс] // Unreal Engine Documentation URL: <https://docs.unrealengine.com/>. Дата обращения: 11.03.2024.
2. Симулятор выживания [электронный ресурс] // URL: <https://en.wikipedia.org/wiki/Survival_game> Дата обращения: 01.02.2024.
3. Основа геймплея игры на C++ для Unreal Engine [электронный ресурс] // URL: <https://habr.com/ru/articles/357976/>. Дата обращения: 11.03.2024.
4. Настройка параметров БЯМ [электронный ресурс] // URL: <https://obnimorda.ru/guides/prompt/introduction/settings/>. Дата обращения: 11.03.2024.
5. Display aspect ratio [электронный ресурс] // Wikipedia, the free encyclopedia URL: <https://en.wikipedia.org/wiki/Display_aspect_ratio/>. Дата обращения: 05.02.2024.
6. Geometry instancing [электронный ресурс] // Wikipedia, the free encyclopedia URL: <https://en.wikipedia.org/wiki/Geometry_instancing>. Дата обращения: 01.02.2024.
7. Использование токенов в LLM [электронный ресурс] // URL: <https://dzen.ru/a/ZBKqdO5TiGGPDjJN>. Дата обращения: 05.03.2024.
8. ChatGPT: влияем на галлюцинации [электронный ресурс] // URL: [https://habr.com/ru/articles/727458/](https://habr.com/ru/articles/727458/%20) Дата обращения 11.03.2024.
9. Using Large-Language Models (LLM) in Game Development [электронный ресурс] // URL: <https://gamedevacademy.org/using-large-language-models-llm-in-game-development-tutorial-list/#How_Can_Large_Language_Models_Be_Used_for_Games>. Дата обращения 12.03.2024.
10. Generative Agents: Interactive Simulacra of Human Behavior [электронный ресурс] // URL: <https://arxiv.org/abs/2304.03442>. Дата обращения 11.04.2024.
11. Инстансинг [электронный ресурс] // Habr: интернет-портал. URL: <https://habr.com/ru/articles/352962/>. Дата обращения: 03.04.2024.
12. Большие языковые модели: полное руководство в 2024 году [электронный ресурс] // URL: <https://research.aimultiple.com/large-language-models/>. Дата обращения: 05.03.2024.
13. Настройки LLM [электронный ресурс] // URL: <https://www.promptingguide.ai/ru/introduction/settings>. Дата обращения: 05.03.2024.
14. Ollama [электронный ресурс] // URL: <https://ollama.com/> . Дата обращения: 05.03.2024.

ПРИЛОЖЕНИЕ А. Графическая часть выпускной квалификационной работы

В графическую часть выпускной квалификационной работы входят 7 чертежей.

* Модель и компоненты виртуального агента «Крокодил».
* Модель и компоненты виртуального агента «Зебра».
* Общая часть дерева поведения персонажей, примеры поддеревьев действий виртуальных агентов.
* Биомы игровой сцены.
* Граф поз и анимации виртуальных агентов.
* Компоненты интерфейса для конечного пользователя.
* Примеры анимированных действий виртуальных агентов.