

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ  
Федеральное государственное автономное образовательное учреждение высшего образования  
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА ПРИКЛАДНОЙ ИНФОРМАТИКИ

КУРСОВАЯ РАБОТА (ПРОЕКТ)  
ЗАЩИЩЕНА С ОЦЕНКОЙ

РУКОВОДИТЕЛЬ

---

канд. техн. наук, доцент  
должность, уч. степень, звание

подпись, дата

---

С.А. Чернышев  
инициалы, фамилия

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА  
К КУРСОВОЙ РАБОТЕ

РАЗРАБОТКА 3D ИГРЫ ИСКРЫ НА UNITY С LLM-ТЕХНОЛОГИЕЙ

по дисциплине: ТЕХНОЛОГИИ ПРОГРАММИРОВАНИЯ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ гр. №

---

4314

подпись, дата

---

Д.В. Лукьян  
Н.А. Пахомчик  
П.А. Колоскова  
инициалы, фамилия

Санкт-Петербург 2025

## СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	3
1 АНАЛИЗ И ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ.....	6
1.1 Общая характеристика предметной области.....	6
1.2 Ключевые аспекты проектирования игры.....	6
1.3 Анализ существующих игр-аналогов.....	7
1.4 Выводы по анализу предметной области .....	8
2 ВЫБОР СТЕКА ИСПОЛЬЗУЕМЫХ ТЕХНОЛОГИЙ.....	10
3 ЭТАПЫ РЕАЛИЗАЦИИ.....	17
3.1 Создание окружения и моделирование .....	17
3.2 Добавление управления и счётчика очков.....	24
3.3 Добавление панелей взаимодействия и диалога с НПС .....	31
3.4 Внедрение LLM-технологии для диалога и описания .....	37
3.5 Добавление интерактивных мини-игр.....	49
4 ДЕМОНСТРАЦИЯ РАБОТЫ ИГРЫ.....	57
ЗАКЛЮЧЕНИЕ .....	79
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ.....	80
ПРИЛОЖЕНИЕ А .....	81
ПРИЛОЖЕНИЕ Б.....	93

## **ВВЕДЕНИЕ**

В современном цифровом пространстве видеоигры становятся не только формой развлечения, но и мощным средством для повествования, исследования сложных тем и возможности создания собственного уникального игрового опыта. Особый интерес представляют игры, сочетающие интерактивный геймплей с нарративом, где решения игрока напрямую влияют на развитие сюжета и финал. Разработка таких проектов требует не только технических навыков в области программирования и 3D-моделирования, но и понимания основ игрового дизайна, сценарного мастерства и психологии восприятия.

Данная курсовая работа посвящена разработке 3D-приключенческой игры «Искры», созданной на игровом движке Unity. Игра погружает игрока в сюрреалистический мир снов героини, где через исследование трёх ключевых локаций, взаимодействие с персонажами и объектами внутри мира, решение головоломок и принятие выборов раскрывается её полная история. Проект реализует механики от первого лица, диалоговую систему с ветвлением, интерактивные объекты, мини-игры и динамическую систему концовок, зависящую от накопленных игроком «баллов памяти» и «очков хаоса».

Целью работы является полноценная разработка и демонстрация работающего игрового прототипа, реализующего задуманный сюжет, ключевые игровые механики и визуальную эстетику, а также документирование процесса создания в виде пояснительной записи.

Для достижения поставленной цели необходимо решить следующие задачи:

1. Проанализировать предметную область – исследовать современные тенденции в разработке нарративных 3D-игр, психологические аспекты повествования и аналогичные проекты.
2. Спроектировать архитектуру игры – разработать структуру сцен, игровых объектов, систем управления, диалогов и сохранения прогресса.

3. Создать окружение (игровой контент) – разработать 3D-модели, настроить сцены-локации, подготовить их к реализации игровых механик и к взаимодействию с игроком.

4. Реализовать игровые механики:

- Систему перемещения и взаимодействия от первого лица.
- Диалоговую систему с выбором реплик и влиянием на счётчики, включающую в себя работу с интеграцией ЛЛМ в данную систему.
- Логику интерактивных объектов (предметы, активаторы, двери).
- Мини-игра «расстановка предметов».
- Систему подсчёта очков (память/хаос) и определения одной из четырёх концовок.

5. Провести итеративное тестирование геймплея, исправить ошибки и обеспечить стабильную работу всех систем.

6. Подготовить демонстрацию работы игры.

7. Оформить документацию – составить полную пояснительную записку, отражающую все этапы разработки, принятые решения и итоговый результат.

Данный практико-ориентированный проект выполнялся командой из трёх студентов группы 4314: Д.В. Лукиян, Н.А. Пахомчиком, П.А. Колосковой.

– Моделирование персонажей (нпс) и предметов, создание окружения сцен игры, расстановление моделей по сценам с освещением и постановкой камер – П.А. Колоскова;

– Разработка управления игрока, системы начисления очков, взаимодействие с персонажами (нпс) и предметами, внедрение LLM – Д.В. Лукиян.

– Разработка интерактивной части игры – Н.А. Пахомчиком.

Каждый участник команды отвечал за части отчёта:

- Д.В. Лукиян (части отчёта: «Добавление управления и взаимодействие с окружением», «Внедрение LLM-технологии для диалога с НПС», «Демонстрация работы игры»).
- Н.А. Пахомчик (части отчёта: «Добавление интерактивных мини-игр», «Заключение»).
- П.А. Колоскова (части отчёта: «Введение», «Анализ и описание предметной области», «Выбор стека используемых технологий», «Создание окружения и моделирование»).

## **АНАЛИЗ И ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ**

### **1.1   Общая характеристика предметной области**

Разработка видеоигр представляет собой междисциплинарную область, сочетающую элементы программной инженерии, дизайна, психологии и нарративного искусства. В рамках данного курсового проекта предметная область была ограничена созданием 3D-игры с повествовательным уклоном и элементами психологического исследования, разработанной на платформе Unity.

Ключевой акцент был сделан на построении иммерсивной истории, где игровой процесс и решения служат инструментами для раскрытия сюжета и формирования эмоциональной связи между игроком и персонажем.

### **1.2   Ключевые аспекты проектирования игры**

При проектировании игры «Искры» были выделены и детально проработаны следующие аспекты:

- Структура: Игра построена как последовательность сюрреалистичных локаций, каждая из которых отражает этап жизни главной героини. Сюжет подается через внутренний монолог, диалоги с персонажами и взаимодействие с объектами-воспоминаниями.
- Игровые механики как часть истории: Все основные механики (перемещение от первого лица, диалоги с выбором, сбор предметов, простые головоломки) были подчинены задаче продвижения сюжета и раскрытия характера героини. Технически это потребовало создания системы, где каждое действие игрока влияло на внутренние счетчики («баллы памяти», «очки хаоса»), определяющие финал истории для каждого игрока.
- Система выбора и ветвления: Одной из центральных задач стало проектирование системы диалогов и решений, которые не имели очевидно «правильных» вариантов, но по-разному влияли на эмоциональное состояние героини и вели к одной из нескольких концовок. Это потребовало разработки гибкой логики учета состояний и событий.

- Технологическая реализация: С технической стороны были определены ключевые компоненты: движок Unity для рендеринга 3D-графики и реализации физики, язык C# для написания игровой логики. Особое внимание было уделено архитектуре кода, обеспечивающей модульность и легкую настройку диалоговых деревьев, событий и условий концовок.

### **1.3 Анализ существующих игр-аналогов**

Для выбора оптимальных проектных решений и определения ожиданий аудитории был проведен анализ трех успешных игр, релевантных целям проекта.

- *What Remains of Edith Finch* (2017) – нарративная игра, где механика неразрывно связана с историей. Её основным преимуществом является уникальность каждого игрового эпизода, который выступает в роли интерактивной главы. Однако к недостаткам можно отнести крайне короткую продолжительность и минималистичный, почти отсутствующий традиционный геймплей, что делает игру скорее интерактивным фильмом, чем игрой в классическом понимании. Для проекта «Искры» был сделан вывод о необходимости перенять глубину интеграции механики в сюжет, но при этом предложить более содержательный игровой процесс с чёткими системными механиками, такими как счётчики влияния и несколько концовок.
- *Layers of Fear* (2016) является образцом создания непрерывной, гнетущей атмосферы психологического хоррора. Её сильная сторона – мастерское использование динамически меняющегося игрового пространства, света и звука для формирования устойчивого чувства тревоги и подавленности. Технически это демонстрирует мощь скриптовых последовательностей и работы с окружением. К недостаткам относятся линейность сюжета, несмотря на кажущуюся нелинейность пространства, а также акцент на «скейр-моменты», которые со временем могут показаться

однообразными. Для нашей игры это означало важный урок: сделать акцент на психологическом исследовании и моральном выборе, а не на шоковых элементах. Методы создания атмосферы были взяты на вооружение, но направлены не на запугивание, а на передачу внутреннего состояния героини.

– Life is Strange (2015) – уроки в построении системы диалогов и морального выбора. Её ключевое преимущество – способность формировать эмоциональную связь игрока с миром через ветвящиеся диалоги и решения, имеющие долгосрочные последствия. Прозрачная система отражения этих последствий («эффект бабочки») усиливает вовлечённость. Среди недостатков можно отметить некоторую излишнюю мелодраматичность диалогов, которая может отталкивать часть аудитории, а также технические ограничения движка Unreal Engine 3. Для «Искр» анализ этой игры стал основой для проектирования собственной диалоговой системы. Было решено создать варианты ответов, отражающие разные грани восприятия и стремиться к более сдержанной, психологически достоверной подаче.

#### **1.4 Выводы по анализу предметной области**

Проведенный сравнительный анализ не только выявил лучшие практики, но и чётко обозначил необходимость создания нового продукта. Существующие аналоги, при всех их достоинствах, имеют заметные компромиссы: краткость и отсутствие нужного соотношения геймплея, однообразие атмосферы или излишняя мелодраматичность.

Проект «Искры» был задуман как совмещение сильных сторон этих игр – глубины Edith Finch, атмосферы Layers of Fear и системности выбора Life is Strange – но с целью предложить более сбалансированный игровой опыт. Технически это обосновало выбор Unity и C# как инструментов, достаточно гибких для создания проекта.

Таким образом, анализ предметной области показал не только возможность, но и практическую необходимость разработки «Искр» как

продукта, который заполняет существующий пробел, предлагая продолжительное, психологически достоверное приключение, где игровая механика служит прямым языком для исследования внутреннего мира персонажа. Полученные выводы легли в основу последующих этапов работы, определив выбор конкретных технологий и подходов к проектированию архитектуры приложения.

## ВЫБОР СТЕКА ИСПОЛЬЗУЕМЫХ ТЕХНОЛОГИЙ

Для реализации проекта «Искры» был сформирован стек технологий, обеспечивающий полный цикл разработки: от создания 3D-контента и программирования игровой логики до контроля версий. Выбор конкретных инструментов был обусловлен необходимостью достижения оптимального баланса между функциональностью, производительностью, доступностью и скоростью разработки в условиях ограниченных ресурсов учебного проекта.

Был проведён анализ возможных движков для нашей игры, выявлены все влияющие на качество разработки моменты и продемонстрированы в Таблице 1.

Таблица 1. Сравнение игровых движков

Критерии	Unity	Unreal Engine	Godot
Используемый язык	C# (статическая типизация, управляемый код)	C++ (нативный код) и Blueprints (визуальное программирование)	GDScript (интерпретируемый) и C# (поддержка)
Качество визуализации	Высокий уровень, URP/HDRP (Universal/High Definition Render Pipeline) – гибкая настройка под разные графические требования	Превосходный, ориентирован на создание фотореалистичной графики.	Собственный Vulkan/OpenGL рендерер – легковесный, но с ограниченными возможностями
Система материалов	Shader Graph + стандартные PBR-материалы, легкая	Material Editor – мощный, но сложный	Visual Shader – базовые возможности

	кастомизация	освоения	
Сложность освоения	Средняя. Интуитивный редактор и обширная документация	Высокая. Требует глубокого изучения архитектуры и C++	Низкая. Простая архитектура и скриптовый язык
Библиотека готовых ресурсов	Крупнейшая на рынке. Огромный выбор плагинов, моделей, систем (диалоги, ИИ, анимация)	Обширная (Marketplace). Много качественных, но часто платных активов	Ограниченнная. Сообщество активно развивает библиотеку
Импорт ассетов	Нативная поддержка FBX, glTF, автоматическая оптимизация мешей и материалов	Собственный формат .uasset, сложная конвейерная обработка	GLTF, Collada, базовые форматы
Система компонентов	Entity Component System (ECS) – чистая компонентная архитектура	Actor-Component – более сложная иерархическая система	Узловая система (Node) – гибкая, но с другой парадигмой
Встроенные инструменты для нарративных игр	Timeline, Cinemachine – мощные инструменты для кат-сцен и управления	Sequencer, Matinee – продвинутые, но сложные в освоении	Базовые анимационные инструменты

	камерой		
Поддержка и документация	Одно из крупнейших и самое активное сообщество, множество уроков и решений на русском и английском	Крупное сообщество, профессиональная документация	Активное, но меньшее сообщество
Оптимизация выходного билда	Хорошая контролируемость размера финального билда	Большой минимальный размер билда из-за мощного базового функционала	Очень компактные билды
Кросс-платформенность	Поддержка 25+ платформ, включая PC, мобильные устройства, WebGL и консоли	Широкая поддержка основных платформ (PC, консоли, мобильные)	Поддержка ключевых настольных и мобильных платформ, WebGL

В качестве основы для разработки был выбран игровой движок Unity в сочетании с языком программирования C#.

Unity был выбран по следующим ключевым причинам:

- Кроссплатформенность: Движок позволяет собрать проект под множество платформ (PC, macOS, WebGL, мобильные системы), что критически важно для максимального охвата аудитории и тестирования.
- Компонентно-ориентированная архитектура: Эта парадигма, реализованная в Unity через систему GameObjects и Components, идеально подходит для быстрого прототипирования и модульной сборки сложных

интерактивных сцен, что напрямую соответствовало задаче создания трёх уникальных локаций с разнообразными механиками и четырех локаций концовок.

- Мощная экосистема и документация: Наличие официальной документации, обширного сообщества, Asset Store и встроенных систем (физика, анимация, UI, система сцен) значительно ускоряет разработку и снижает порог входа.

Язык C# является основным для программирования в Unity и был выбран благодаря:

- Строгой статической типизации: Это способствует написанию более надежного и поддерживаемого кода, что критически важно при реализации сложных систем (диалоги, менеджер состояния игры).
- Высокой производительности и современным возможностям: Поддержка асинхронного программирования, LINQ и других функций упрощает работу с логикой игры.
- Полной интеграции со средой Unity: Все API и события движка максимально удобно доступны через C#.

Так же был проведён сравнительный анализ инструментов 3D-моделирования, который продемонстрирован в Таблице 2.

Таблица 2. Сравнение инструментов 3D-моделирования

Критерий	Blender	3ds Max	Maya
Лицензирование и стоимость	Бесплатный (Open Source). Полный функционал без ограничений	Платная подписка. Высокая стоимость лицензии	Платная подписка. Высокая стоимость лицензии
Полнота рабочего цикла	Полный цикл: моделирование, скульптинг, UV-	Силен в моделировании и визуализации.	Индустриальный стандарт для анимации и

	развертка, текстурирование, риггинг, анимация, рендеринг	Анимация и скульпting требуют дополнительных инструментов	риггинга
Интерфейс и обучение	Единый, настраиваемый интерфейс. Сообщество создало множество обучающих материалов	Традиционный, отлаженный интерфейс. Широкая база знаний	Сложный, мощный интерфейс. Высокий порог входа
Интеграция игровыми движками	Прямой экспорт в формат FBX и GLTF, поддерживаемые Unity	Отличная поддержка экспорта в FBX. Тесная интеграция с Autodesk продуктами	Отличная поддержка экспорта в FBX
Сообщество и поддержка	Очень активное o p e n - s o u r c e сообщество, форумы, тutorиалы на всех языках	Большое профессиональное сообщество	Большое профессиональное сообщество, официальная поддержка Autodesk

Для создания трёхмерных моделей, текстур и графических элементов был выбран Blender.

Blender – основное приложение для 3D-моделирования. Его неоспоримыми плюсами являются:

- Бесплатность и открытый код: Являлся ключевым фактором для учебного проекта, не требующего лицензионных затрат.
- Многофункциональность: Позволяет выполнять весь цикл работ: от полигонального моделирования и скульптуинга до UV-развертки, текстурирования, запекания и простой анимации.
- Удобный экспорт: Поддержка формата FBX для импорта в Unity с сохранением иерархии, материалов и анимации.

Figma – использовалась для графического дизайна. Её неоспоримыми плюсами являются:

- Создание 2D-активов: В Figma разрабатывались текстуры для объектов и концепты скайбоксов (окружающего неба).

Преимущества перед альтернативами: В отличие от Adobe Photoshop, Figma предлагает облачную, векторно-ориентированную среду, что упрощает совместную работу, итеративный дизайн и экспорт графики в оптимизированных для веба форматах (PNG, SVG), которые легко интегрируются в Unity.

Для управления исходным кодом, моделями и другими активами была выбрана распределенная система контроля версий Git в связке с платформой GitHub.

Git обеспечивает:

- Надёжный контроль истории изменений: Возможность отката к любой предыдущей версии проекта, что критически важно при экспериментальной разработке и возможном внесении ошибок.
- Эффективную командную работу: Механизмы ветвлений (branching) и слияния (merging) позволяют параллельно работать над разными функциями (например, над логикой диалогов и моделированием локации) без конфликтов.

Сравнение с альтернативами: Рассматривались централизованные системы (например, SVN), однако распределённая природа Git предоставляет большую гибкость и отказоустойчивость, особенно при работе вне локальной сети.

## **ЭТАПЫ РЕАЛИЗАЦИИ**

### **1.5 Создание окружения и моделирование**

Для создания полноценного и приятного игрового опыта было необходимо создать окружение в виде семи сцен в игровом движке Unity и разработать для них 3д-модели.

Всего локаций семь: три основные и четыре в качестве получаемой игроком концовки.

В состав основных локаций входили:

Лабиринт (далее будет обозначаться как основная локация 1) – задача была создать атмосферу чего-то светлого, но в тоже время оставляющего после себя странное ощущение.

Тёмный лес (далее будет обозначаться как основная локация 2).

Сад времени (далее будет обозначаться как основная локация 3).

Так же были разработаны и собраны локации хорошей, плохой, нормальной концовки и концовки хаоса, которые можно получить в зависимости от набранных баллов.

Перед сборкой локаций в среде Unity, некоторые модели (персонажи, отдельные элементы локаций) были смоделированы в приложении Blender. Процедурные материалы текстур были созданы и настроены. Для моделей назначена UV-развёртка, запечены карты цвета, света, нормалей, шероховатостей. Всё это было сделано для корректного переноса модели и её текстур в среду Unity. На рисунках 1-4 продемонстрированы модели в Blender (из перечня основных локаций), которые являются основными для получения игрового опыта пользователя (с ними можно взаимодействовать, говорить, активировать мини-игры или перемещаться между локациями). Остальные (по большей части декоративные) модели так же были сделаны в Blender.

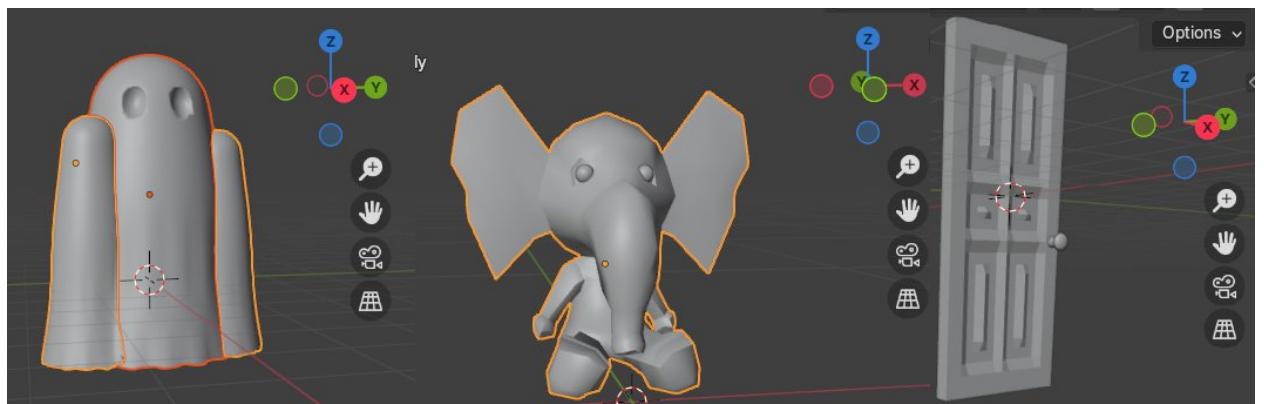


Рисунок 1 – Модели для взаимодействия в основной локации 1

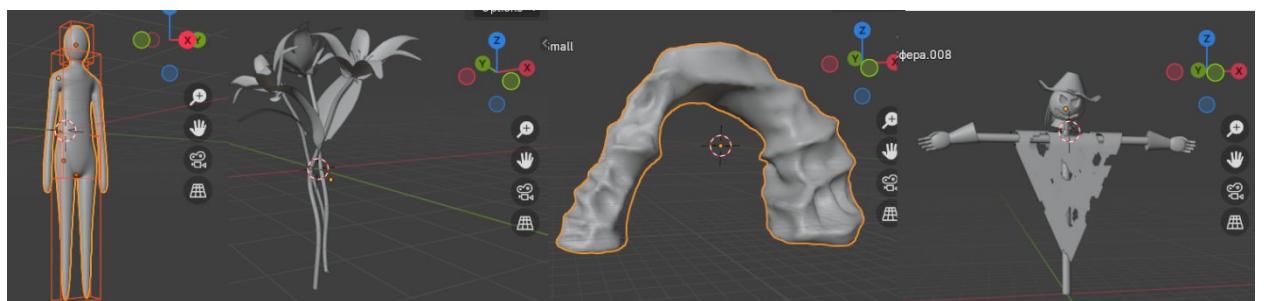


Рисунок 2 – Модели для взаимодействия в основной локации 2 (часть 1)

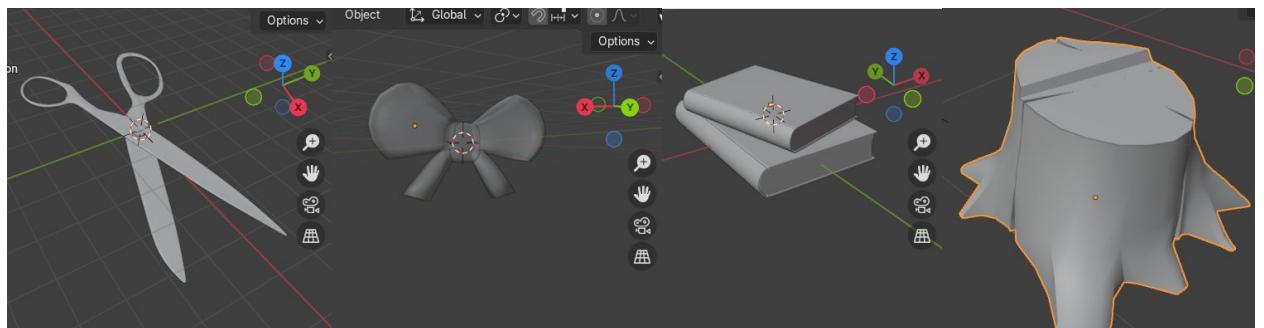


Рисунок 3 – Модели для взаимодействия в основной локации 2 (часть 2)

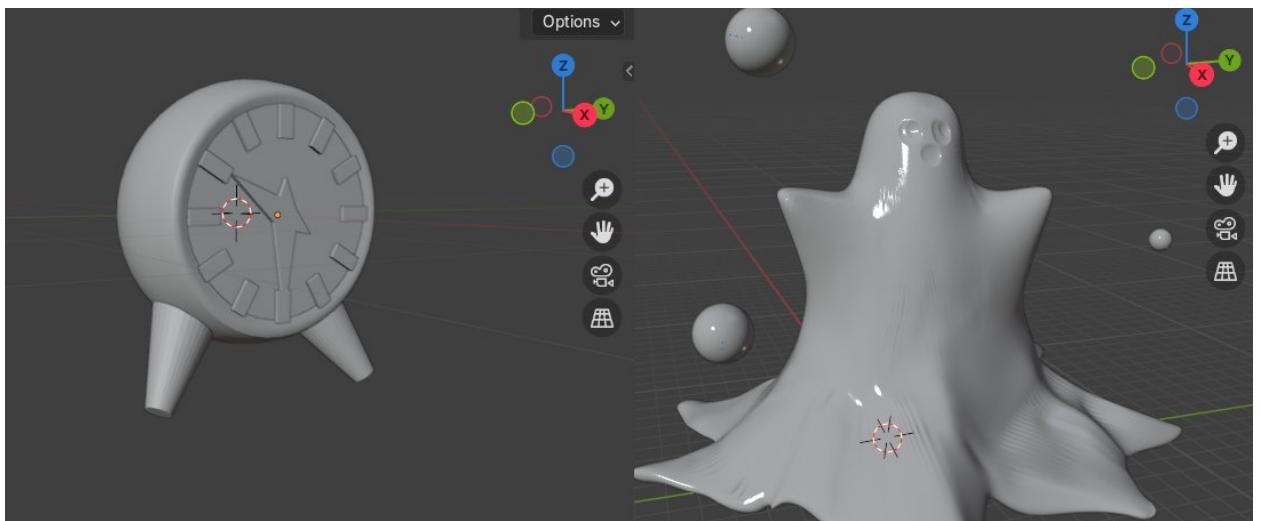


Рисунок 4 – Модели для взаимодействия в основной локации 3

Модели были экспортированы в формате FBX, а их карты, для автоматического создания текстур для каждого составного элемента модели внутри Unity, перенесены в одноимённой папке вместе с моделью. Некоторые текстуры подвергались дополнительной обработке с помощью графического редактора Figma. Пример перенесённой модели пугала в основную локацию 2 представлен на рисунке 5.



Рисунок 5 – Результат перенесённой модели с текстурами

Сборка локаций производилась в Unity версия 2022.3.62f3 LST Built-In. Для каждой локации создавалась отдельная сцена (файл с расширением unity) в папке Scenes директории Assets. В Assets так же загружались все ассеты с

возможными дополнительными необходимыми материалами (префабами, текстурами, шейдерами) с Unity Asset Store[1].

В сценах производилась работа с 3д-объектом Terrain, позволяющим создавать ландшафт с помощью кистей высоты, текстур, сглаживания, шума и тому подобные. Для увеличения количества fps в игре и снижения отрисовки треугольников (батчей) для покрытия созданного полотна ландшафта выбрана кисть отрисовки деталей, а не травы и подходящие для этого модели с Unity Asset Store. Так же на локации были добавлены с помощью кистей Terrain некоторые виды деревьев, камни и кристаллы. В некоторые локации был добавлен шейдер воды и произведена его настройка.

Далее производилась настройка с глобальным освещением сцен. Были добавлены для каждой локации свои скайбоксы, настроено освещение от них на сцену и объекты, выбрано в параметре sun source объект глобального освещения directional light, в отвечающий в некоторых сценах за солнце, а в некоторых за луну. Так же в параметрах directional light настраивалось отображение желаемого цвета освещения и его интенсивность.

Дополнительное освещение в сценах создавалось с помощью point light, который был иерархически вживлен в некоторые расставленные по локации объекты. Так же свечение было достигнуто с помощью материала Emission, позволяющему объекту светиться изнутри.

Для более приятного отображения свечений была произведена работа с пост-процессингом и основной камерой игрока с параметром Bloom – эффектом постобработки, который имитирует свечение от ярких источников света, создавая ореолы и лучи вокруг них.

Были добавлены 3д-объекты и наложены на них коллайдеры, чтобы игрок не мог проходить сквозь эти элементы. Коллайдеры-стены так же были добавлены в каждую сцену для предотвращения выхода за неё.

В локацию хорошей концовки была добавлена и настроена система частиц, имитирующая снег.

После завершения сборки всех сцен была произведена настройка дальности отрисовки, запекание теней статичных элементов для повышения производительности. Конечный вид всех семи локаций с точки загрузки игрока в них представлен на рисунках 6-12.



Рисунок 6 – Основная локация 1



Рисунок 7 – Основная локация 2



Рисунок 8 – Основная локация 3



Рисунок 9 – Хорошая концовка



Рисунок 10 – Плохая концовка



Рисунок 11 – Нормальная концовка



Рисунок 12 – Концовка хаоса

## 1.6 Добавление управления и счётчика очков

После созданных локаций и расположенных персонажей нпс, предметами и остальными декорациями, можно было добавлять функционал в игру.

Сначала было добавлено управление для передвижения по локации 1 игроком через добавление простого объекта (Empty GameObject) Player и в нём создан компонент сферы игрока (Character Controller) и заданы такие параметры (рис. 13), а также был создан скрипт PlayerController, через которого осуществляется передвижение.

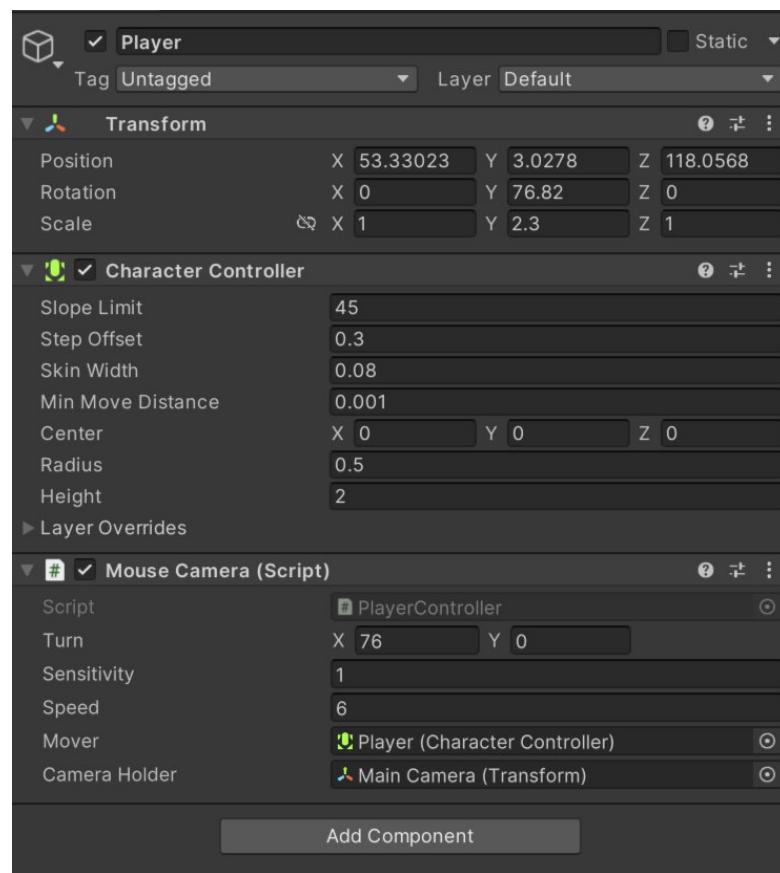


Рисунок 13 – Параметры игрока

Ранее созданная главная камера сцены (Main Camera) была закреплена в объект Player, чтобы координаты ориентировочно настроить под сферу игрока, и располагаем камеру сверху сферы (рис. 14).

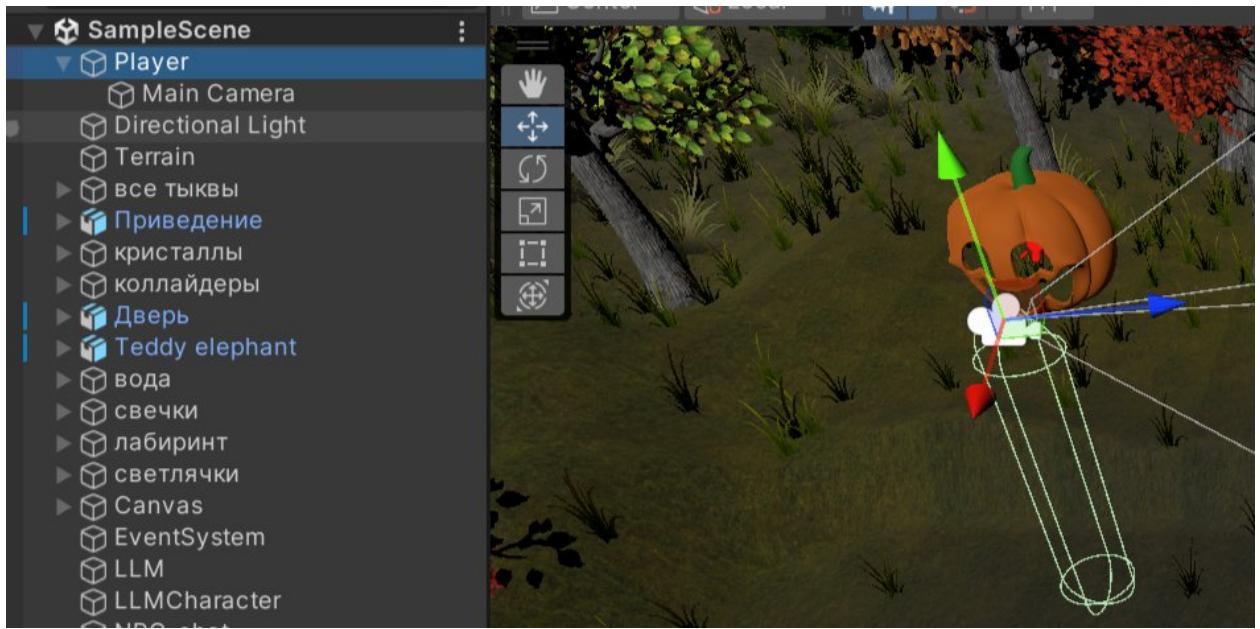


Рисунок 14 – Камера и сфера игрока на сцене

Были созданы параметры для скрипта PlayerController, весь код который можно увидеть в таблице А.1 приложения А:

- turn – координаты мышки в двухмерном пространстве, а также начальные значения поворота сферы игрока и камеры;
- sensitivity – чувствительность поворота по мышки;
- speed – параметр скорости, прибавляется к координатам передвижения, при зажатие кнопки левого shift;
- mover – параметр класса CharacterController сферы игрока, при котором уже заложен метод передвижения Move;
- cameraHolder – координаты вложенной камеры в сфере игрока;
- playerVelocity – скорость свободного игрока по оси Y, чтобы не было случая падения сквозь пол или земли текстур;
- groundedPlayer – параметр проверки на касание пола или земли (стоит или падает ли сфера игрока);
- gravityValue – показатель гравитации (на сколько быстро будет падать);

- moving – права на движение, чтобы можно было заблокировать управление игрока при дальнейшем диалоге с нпс, когда камера переключается на статичную.

Заранее была добавлена базовая функция класса Start, которая активирует действия функции при запуске игры, которая скрывает курсор.

```
void Start()
{
    Cursor.lockState = CursorLockMode.Locked;
}
```

А также чтобы можно просматривать локацию, была реализована функция Rotation поворота камеры по мышки, которая берёт координаты нахождения курсора по встроенному методу Input.GetAxis с учётом чувствительности и прибавляет к параметру turn с ограничением поворота по оси Y в диапазон [-50; 50], чтобы игрок не смотрел сквозь себя и не перевернул камеру с ног на голову.

```
private void Rotation()
{
    turn.x += Input.GetAxis("Mouse X") * sensitivity;
    turn.y += Input.GetAxis("Mouse Y") * sensitivity;
    if (turn.y >= 50) {turn.y = 50;}
    else if (turn.y <= -50) {turn.y = -50;}
    cameraHolder.localRotation = Quaternion.Euler(
        turn.y, 0, 0);
    mover.transform.localRotation = Quaternion.Euler(0,
        turn.x, 0);
}
```

В функции Move реализовано передвижение игрока по клавишам W,S или стрелки вверх-вниз по вертикали через метод ввода игрока Input.GetAxis("Vertical") и A,D или стрелки влево-вправо соответственно Input.GetAxis("Horizontal"), а также чтобы снизить ускоренность диагонального передвижения, была добавлена переменная notwo, снижающая скорость игрока.

```

private void Move()
{
    float horizontalMove = Input.GetAxis("Horizontal") *
0.7f;
    float verticalMove = Input.GetAxis("Vertical");
    float notwo = 1;
    if (horizontalMove > 0.6f && verticalMove > 0.8f)
notwo = 0.8f;
    if (horizontalMove < -0.6f && verticalMove > 0.8f)
notwo = 0.8f;
    if (horizontalMove > 0.6f && verticalMove < -0.8f)
notwo = 0.8f;
    if (horizontalMove < -0.6f && verticalMove < -0.8f)
notwo = 0.8f;
    ...
}

```

Была учтена гравитация когда игрок падает, а также предотвращено падение через текстур вниз, а после добавлены все параметры в передвижение сферы игрока через суммирование вертикального и горизонтального изменения, и применены в метод mover.Move, с учётом speed, гравитации и notwo (диагонального передвижения).

```

...
groundedPlayer = mover.isGrounded;
if (groundedPlayer && playerVelocity.y < 0)
playerVelocity.y = 0f;
else playerVelocity.y += gravityValue *
Time.deltaTime;
Vector3 gravityMove = new Vector3(0,
playerVelocity.y, 0);
Vector3 move = transform.forward * verticalMove +
transform.right * horizontalMove;
mover.Move(((Input.GetKey(KeyCode.LeftShift) ? speed
+ 4 : speed) * Time.deltaTime * move + gravityMove *
Time.deltaTime) * notwo);
}

```

Чтобы действия происходили постоянно на протяжение всей игры, была использованная встроенная функция Update, в которую были добавлены созданные функции Rotation и Move с проверкой на разрешение передвижения moving.

```

void Update()
{
    if (moving)
    {
        Rotation();
        Move();
    }
}

```

И так после настройки сферы игрока, объект был дублирован для всех остальных сцен проекта (локаций 2, 3 и всех четырёх концовок).

Был реализован интерфейс игрока в объекте Canvas с отображением очков памяти (П) и хаоса (Х) и будущей панели описание воспоминаний, которая появляется при взаимодействии с предметами (рисунок 15-17).

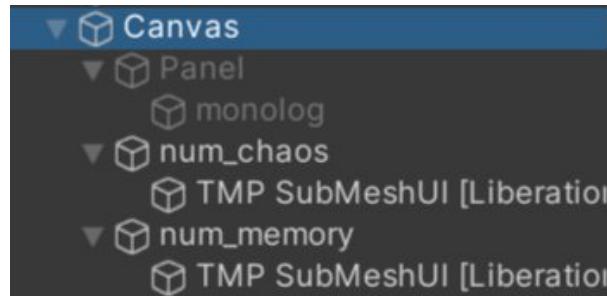


Рисунок 15 – Объекты интерфейса игрока Canvas



Рисунок 16 – Надписи счётчиков очков

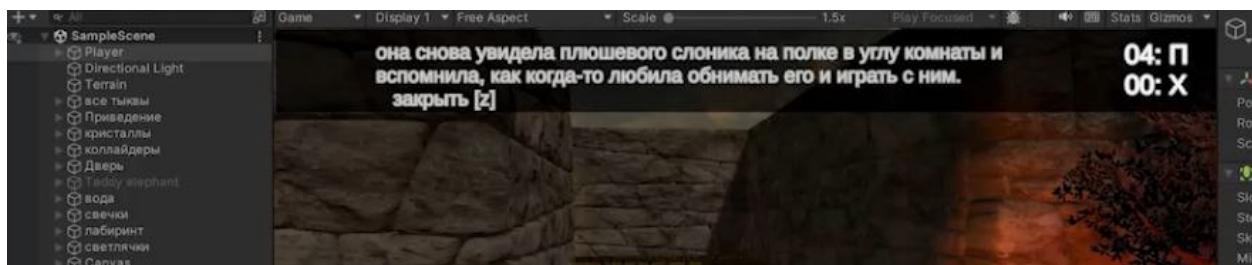


Рисунок 17 – Описание воспоминаний на верхней панели

По ходу прохождения игры за интерес в общении с НПС и взаимодействия с предметами даются очки памяти, а очки хаоса начисляются, если с НПС общаться негативно.

Данные очки влияют на получение концовки игры, где очки хаоса приоритетней очков памяти.

Добавление очков было реализовано в скрипте `add_score` (весь код в таблице А.2 приложения А), который применяется только в первой локации в объекте холста `Canvas`, чтобы задать начальные статичные параметры `memory` и `chaos`.

```
public static int memory = 0;
public static int chaos = 0;
public int set_memory = 0;
public int set_chaos = 0;

void Start()
{
    memory = set_memory;
    chaos = set_chaos;
}
```

Дальше для всех локаций, кроме концовок, применяем отображение показателей очков по скриптам `score_text_m` и `score_text_c`, весь код который можно увидеть в таблице А.3 и А.4 приложения А. Скрипты прикрепляем к объектам отображения текста в `Canvas`.

Каждый скрипт отображает свой показатель очков (памяти и хаоса) на заданный `canvasText` с форматом "00", то есть если число меньше 10 то добавляется ноль слева, а минус выходит за ноль.

```

if (add_score.memory > 9 || add_score.memory < -9)
canvasText.text = add_score.memory + " :Π";
else if (add_score.memory <= 9 && add_score.memory >= 0)
canvasText.text = "0" + add_score.memory + " :Π";
else canvasText.text = "-0" + -add_score.memory + " :Π";

```

Возможность перехода между локациями выполняет - дверь.

За это отвечает скрипт door\_loc\_01 и door\_loc\_end в объекте двери, весь код который можно увидеть в таблице A.5 и A.6 приложения А.

Имеют те же параметры:

- sceneName / sceneNameEnd – название сцены, в которую происходит переход;
- offset – расстояние активации перехода на новую сцену;
- cam – координаты камеры игрока;
- stuff – координаты двери;
- distance – расстояние двери от камеры игрока для активации перехода;
- isactive – статичный показатель, который отключает вычисление расстояния (отключение нужно при диалоге с нпс, когда камера игрока отключается, чтобы не вызывать ошибок поиска камеры).

Оба скрипта вычисляют расстояние от камеры игрока до двери, чтобы при нажатии на кнопку Z далеко от двери, переход не случился, а на расстояние offset.

```

cam =
GameObject.FindGameObjectWithTag("MainCamera").GetComponent<
Transform>();
stuff = GetComponent<Transform>();
distanse = Vector3.Distance(cam.position,
stuff.position);

```

Скрипт door\_loc\_01 применяется для 1, 2 локации и выхода из всех концовок.

Имеет также уникальный параметр `isExit`, отвечающий за выход из игры для локаций концовок. А также обновляет параметры скриптов нпс и объектов взаимодействия, которые будут объяснены позже.

```
if (Input.GetKeyDown(KeyCode.Z) && distanse < offset)
{
    if (isExit) Application.Quit();
    else {
        SceneManager.LoadScene(sceneName);
        PC_chat.isReset = true;
        text_appear_npc.resetTurn = true;
        text_appear.resetTurn = true;
    }
}
```

Скрипт `door_loc_end` применяется только в 3 локации для расчёта концовки и перенаправление туда по условиям.

Если очков хаоса много то концовка хаоса, если меньше 2 то далее рассчитывается по очкам памяти, где очки больше 80% от максимум возможного получения очков памяти то хорошая концовка, если 80%-60% то нейтральная концовка, а иначе плохая концовка.

```
if (Input.GetKeyDown(KeyCode.Z) && distanse < offset)
{
    if (chaos > 1) {sceneNameEnd = "chaos end";}
    else {
        if (memory > 17) sceneNameEnd = "good end";
        else if (memory > 10) sceneNameEnd = "normal
end";
        else sceneNameEnd = "bad end";
    }
    SceneManager.LoadScene(sceneNameEnd);
}
```

## 1.7 Добавление панелей взаимодействия и диалога с НПС

Отображение плашек с текстом взаимодействия происходит через скрипты `text_appear_npc` закреплённый за нпс и `text_appear` за предметы, код которых можно посмотреть в таблице A.7 и A.8.

Со всеми плашками можно взаимодействовать через одну кнопку Z.

Плашки с текстом взаимодействия показываются на всех нпс (призрак, пугало и странный призрак) показанные на рисунке 18 и некоторые предметы локаций (игрушечный слон, все светлячки из лабиринта, ножницы, бант, дневник, могила с цветами и дверь) показанные на рисунке 19.

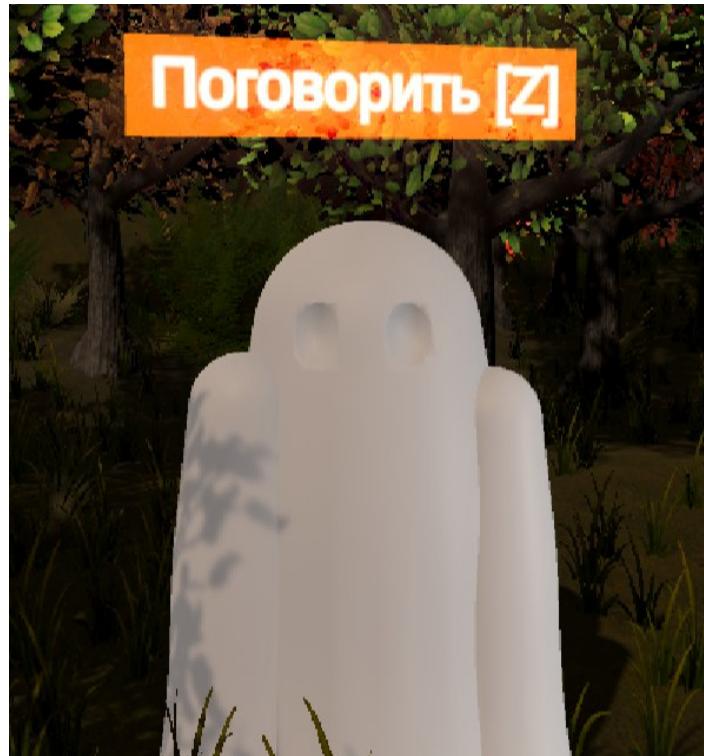


Рисунок 18 – Плашки с текстом над нпс



Рисунок 19 – Плашка с текстом над предметом

Разберём отдельно отображение плашки взаимодействия для НПС.

Параметры `text_appear_npc`:

- `thisobj` – объект плашки с текстом;
- `offset` – расстояние активации камера игрока до `stuff` (нпс, на котором используется скрипт);
- `distance` – текущее расстояние камера до `stuff`;
- `turn` – контроль одноразового взаимодействия;
- `resetTurn` – возвращает одноразовое взаимодействия (обновление при переходе между сценами);
- `isactive` – скрывает все плашки с текстом (при переключение камеры на диалог с нпс, камера скрывает, и чтобы не было конфликтов, все плашки скрываются).

В начале игры отключаем все плашки с текстом.

```
void Start()
{
    thisobj.SetActive(false);
}
```

Была создана отдельно функция `Appear` появление плашки с текстом. В ней вычисляет расстояние камеры игрока до нпс и если меньше заданного `offset`, то плашка с текстом отображается перед игроком, иначе скрывается.

```
void Appear()
{
    cam =
GameObject.FindGameObjectWithTag("MainCamera").GetComponent<
Transform>();
    stuff = GetComponent<Transform>();
    distanse = Vector3.Distance(cam.position,
stuff.position);
    if (distanse < offset) thisobj.SetActive(true);
    else thisobj.SetActive(false);
}
```

И функция была добавлена в `Update` при условии, что плашка может быть активна (`isactive`) и имеет одноразовое взаимодействие (`turn`), при

нажатии кнопки Z панель диалога с нпс должен активироваться уже в другом скрипте zbutton\_activeted, а через turn = false лишить одноразовое взаимодействие. А также если активирован resetTurn, то возвращает turn = true.

```
void Update()
{
    if (resetTurn)
    {
        turn = true;
        resetTurn = false;
    }
    if (isactive && turn)
    {
        Appear();
        if (Input.GetKeyDown(KeyCode.Z) && distanse <
offset && turn) turn = false;
    }
}
```

Отображение панели диалога, пример которой можно увидеть на рисунке 20, и переключение на камеру направленную на нпс происходит через скрипт zbutton\_activeted, код который можно увидеть в таблице A.9 в приложении А, скрипт также закреплён за объект нпс.

Параметры скрипта:

- thing – объект всей панели диалога;
- cam\_npc – камера для статичного просмотра диалога с нпс;
- but – плашка с текстом (для её скрытия);
- input – формочка ввода текста для игрока, чтобы всегда был фокус на эту форму, иначе если курсов выйдет из игры, игрок не сможет писать текст в форме;
- isactivate – контроль одноразового действия скрипта через кнопку Z (в начале false);
- cam – камера игрока.



Рисунок 20 – Панель диалога нпс Призрака

В функцию Update было записано условие если плашка с текстом отображена то даётся возможность нажать кнопку Z, при действии которого случится переключение на статичную камеру cam\_npc и отобразится панель диалога thing, а также лишение управления игрока MouseCamera.moving = false и происходит фокус на формочку ввода вместе с отключением отображений всех плашек с текстом всех объектов..

```
void Update()
{
    if (but.activeSelf)
    {
        if (!isactivate && Input.GetKeyDown(KeyCode.Z))
        {
            thing.SetActive(true);
            isactivate = true;
            cam =
GameObject.FindGameObjectWithTag("MainCamera");
            cam.SetActive(false);
            cam_npc.SetActive(true);
            MouseCamera.moving = false;
            but.SetActive(false);
        }
    }
}
```

```

        text_appear.isactive = false;
        text_appear_npc.isactive = false;
        GoToScene.isactive = false;
        GoToEndScene.isactive = false;
        input.ActivateInputField();
    }
} else input.ActivateInputField();
}

```

Появление взаимодействия с предметами с панелью

Скрипт `text_appear` появления плашек с текстом для предметов работает также как `text_appear_npc` и имеет те же параметры, но с новыми дополнительными:

- `score` – начисление очков памяти за взаимодействие с предметом;
- `isrotate` – возможность вращение объекта на игрока;
- `rotationSpeed` – скорость вращение;
- `ispanel` – возможность использовать верхнюю панель с описанием воспоминания предмета;
- `mem` – номер строки ответа LLM сгенерированного воспоминания предмета;
- `id_task` – уникальное число запроса LLM, чтобы можно было переключаться между ответами запросов разных категорий предметов;
- `ishide` – предмет взаимодействия исчезает после нажатие кнопки Z;
- `objhide` – предмет исчезновения.

В функции `Start` и `Appear` остаются такими же как и в скрипте `text_appear_npc`.

Но также появилась функция `Rotation` поворота объекта, где через метод `LookAt` координаты предмета направляются на камеру игрока (в проекте данный функционал применяется исключительно на светлячках в лабиринте 1 локации).

```

void Rotation()
{
    stuff.transform.LookAt(cam.transform);
}

```

И данная функция применяется в Update, если isrotate применяется.

В Update также было добавлено resetTurn и условие isactive && turn как и в скрипте text\_appear\_npc, но при нажатии кнопки Z, происходит начисление очков памяти score и если применяется верхняя панель описания воспоминаний ispanel, то панель отображается и для скрипта LLM для предметов устанавливается текущее id и mem (номер строки ответа на запрос LLM) и активируется. А также отключается отображение всех остальных плашек текстов предметов, нпс и для двери перехода на другую локацию. Если применяется ishide то предмет скрывает, а также лишается одноразовое взаимодействие turn.

```

if (Input.GetKeyDown(KeyCode.Z) && distanse < offset &&
turn)
{
    add_score.memory = add_score.memory + score;
    if (ispanel) {
        panel.SetActive(true);
        fire_chat.id_now = id_task;
        fire_chat.mem = mem;
        fire_chat.isactive = true;
        isactive = false;
        GoToScene.isactive = false;
        GoToEndScene.isactive = false;
    }
    if (ishide) objhide.SetActive(false);
    thisobj.SetActive(false);
    turn = false;
}
if (isrotate) Rotation();

```

## 1.8 Внедрение LLM-технологии для диалога и описания

Для работы с LLM скачаем из Unity Asset Store asset LLM for Unity[2]. Загружаем asset в проект и добавляем пустой объект, назвав его LLM. В этом

объекте добавляем скрипт LLM.cs от данного ассета и загружаем “Load model” из проводника модель Meta-Llama-3.1-8B-Instruct-Q4\_K\_M.gguf[3], которая была заранее скачана (рис. 21).

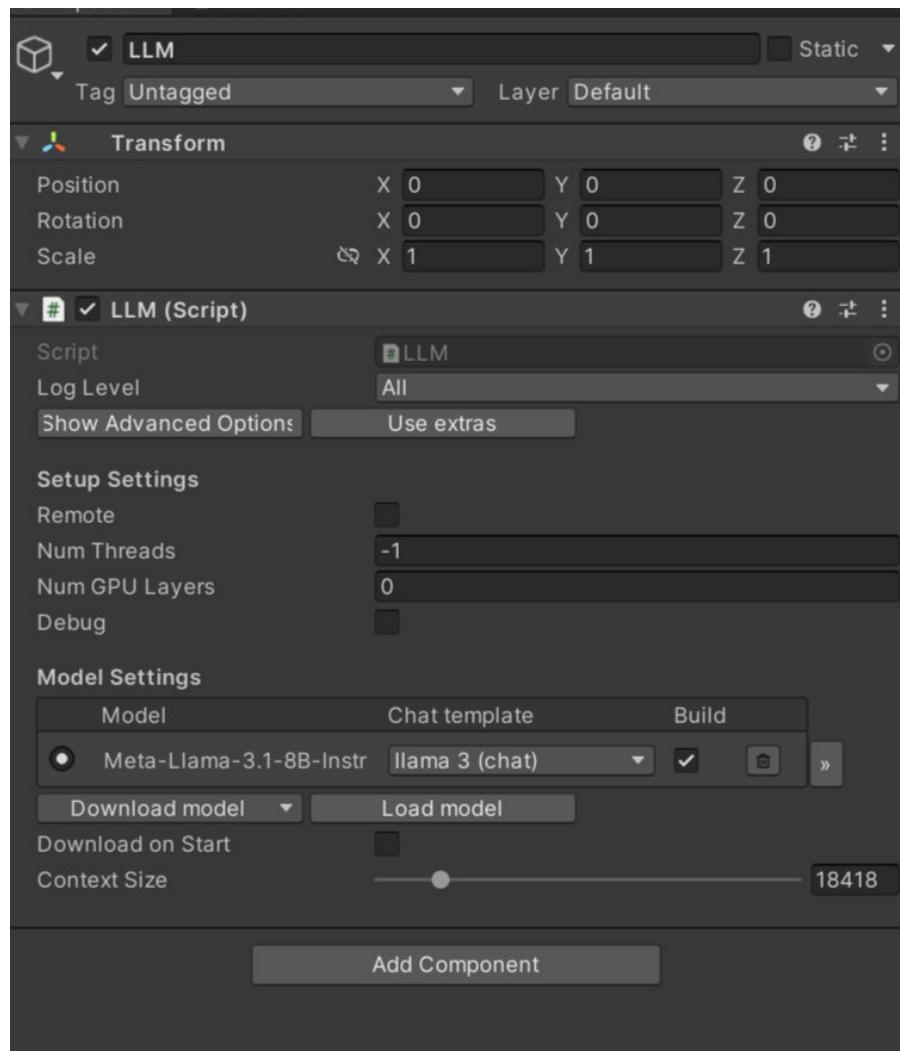


Рисунок 21 – Параметры модели LLM

Использование LLM происходит через скрипт LLMCharacter.cs от ассета, в котором нужно задать начальный промпт, для примера назначим всем объектам в первой локации по LLMCharacter: призраку нпс, светлячкам и игрушке слоника (рис. 22-24)

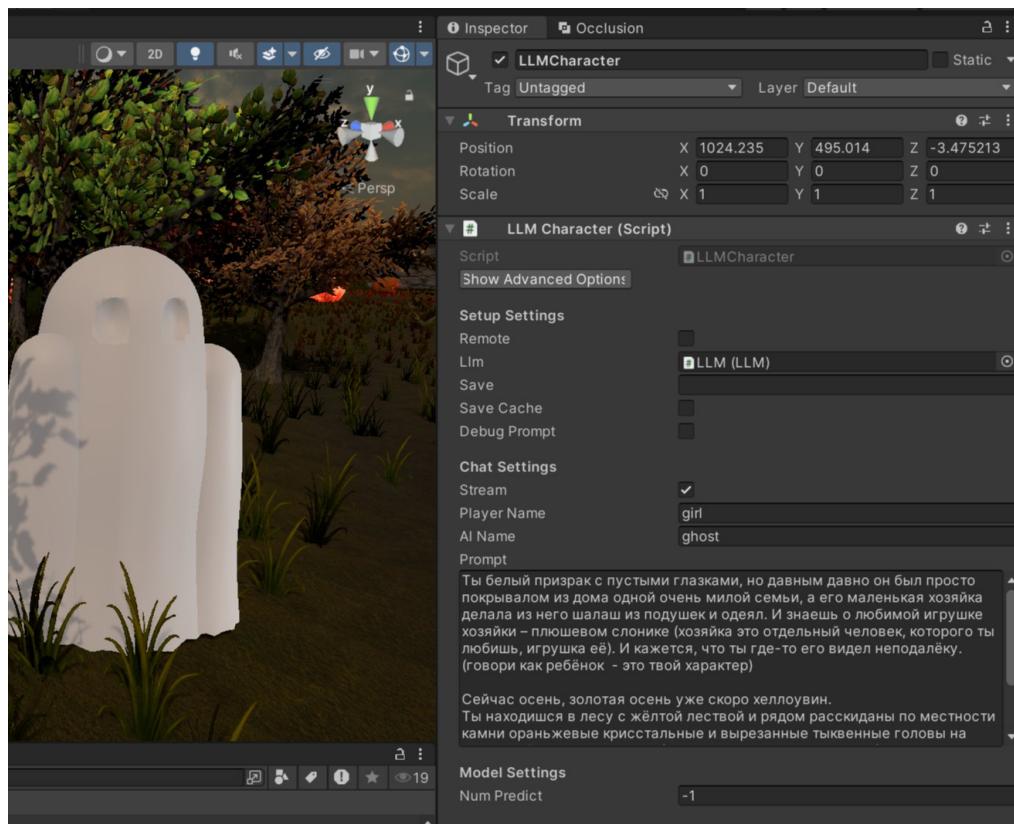


Рисунок 22 – Характеристики LLM для призрака

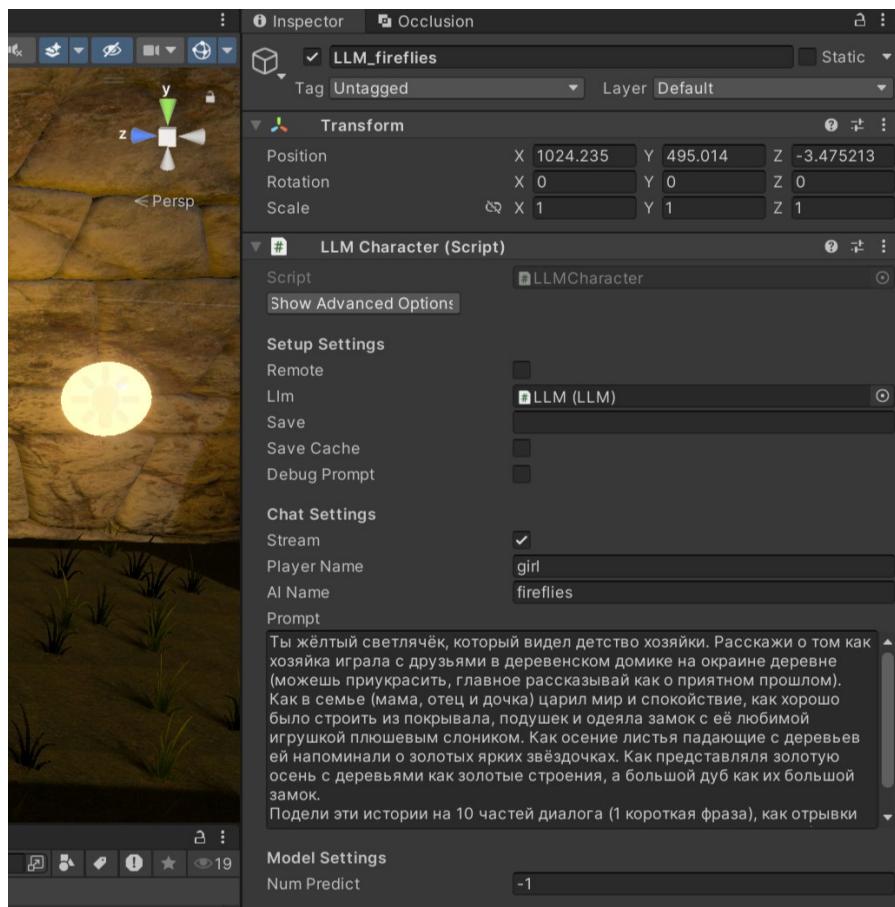


Рисунок 23 – Характеристики LLM для светлячков

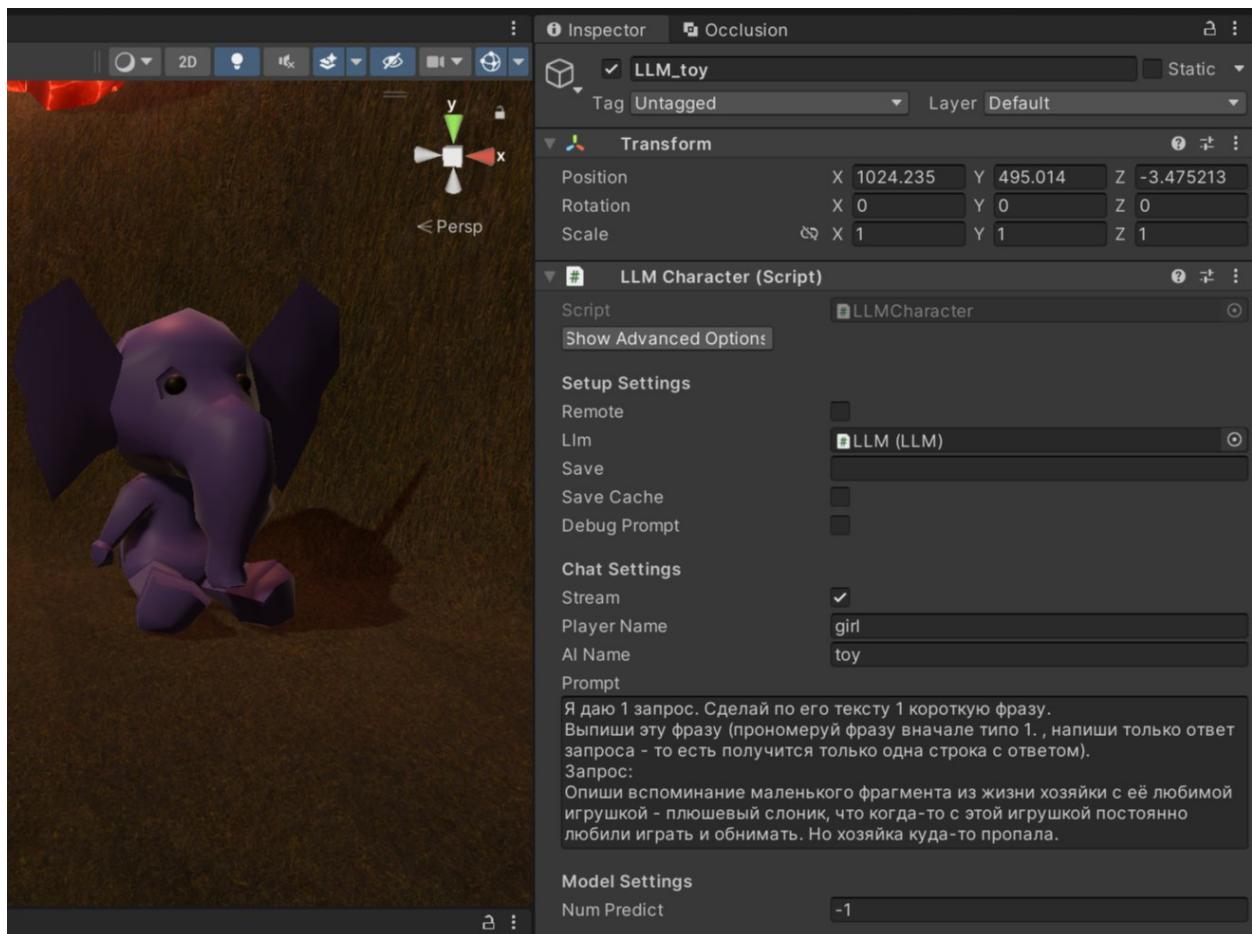


Рисунок 24 – Характеристики LLM для игрушки слона

И так добавим для остальных локаций объектам: пугало, ножницы, банты, дневник, могилы с цветами, странный призрак.

Чтобы взаимодействовать LLM и получать нужные ответы на запросы были созданы два различных скрипта выполняющий свои событие сценария запросов LLM: `NPC_chat` и `fire_chat`, код которых можно увидеть в таблице А.10 и А.11 в приложении А.

Скрипт `NPC_chat` отвечает за диалог с любым назначенным NPC по одному шаблону (в локации можно только один раз использовать):

- Приветствие с небольшой информацией о локации;
- Игроку даётся возможность задать вопрос;
- LLM анализирует вопрос и даёт свой ответ, а после назначает вопросу игроку одну из чисел от 1 до 4 включительно в формате JSON и в

конце ещё добавляет важную подсказку или информацию о локации заданную заранее в скрипте;

- Игроку начисляется или ничего не дают и выдают право выйти из диалога.

Параметры скрипта NPC\_chat:

- thing – панель диалога;
- cam и cam\_npc – камера игрока и статичная камера на нпс;
- llmCharacter – класс для взаимодействия с моделью LLM;
- playerText и AIText – форма ввода и объект отображения текста LLM;
- num – номер события (приветствие (1), ответ игрока (2) и ответ LLM с подсказкой локации и прощанием (3));
- hello\_prompt и task\_prompt – приветственный запрос и обработка ответа игрока с условиями формата JSON с информацией о локации;
- json – пустая строка, в которую будет записан JSON-ответ;
- isReset – очищает данные LLM;
- ai\_json – класс в формате JSON, где ai\_answer – ответ на запрос игрока, ai\_text – подсказка о локации, score – номер контекста вопроса игрока.

При запуске игры, сразу генерируем приветствие нпс, ставим фокус ввода текста на playerText, и лишаем доступа ввода текста, событие 1.

Пока происходит генерация, текст формирует и передаётся в SetAIText в реальном времени, а как только сгенерируется полностью вызовется функция AIReplyComplete.

```
void Start()
{
    playerText.onSubmit.AddListener(onInputFieldSubmit);
    playerText.Select();
    playerText.interactable = false;
    AIText.text = "...";
```

```

        _ = llmCharacter.Chat(hello_prompt, SetAIText,
AIRplyComplete);
        num = num + 1;
    }
}

```

Функция читает вводимый текст в формочке при нажатии на Enter после окончательного ввода текста игроком, и лишает доступа писать снова, переходя во временное событие 2, давая новый запрос LLM, где формируется ответ на запрос и дополнительный запрос заданные в task\_prompt с форматом JSON и подсказкой.

```

void onInputFieldSubmit(string message)
{
    playerText.interactable = false;
    AIText.text = "...";
    if (num == 1)
    {
        _ = llmCharacter.Chat("ИГРОК ГОВОРИТ: " +
message + task_prompt, SetAIText, AIRplyComplete);
        num = num + 1;
    }
}

```

Пока генерируется текст, передаётся его не полный текст text, пока идёт событие 1 (приветствие), тот печатает обычный текст, если же происходит событие 2 (печатает ответ на запрос игрока в JSON формате), чтобы печатанье происходило в реальном времени, формируем формат JSON закрывая его кавычка и фигурной скобкой – “ }”, делаем очистку лишних повторений кавычек и фигурных скобок и печатаем только ответы LLM (ai\_answer и ai\_text), метод JsonUtility.FromJson преобразовывает текст в формате JSON в заданный класс с переменными.

```

public void SetAIText(string text)
{
    if (num < 2)
    {
        AIText.text = text;
    } else {
        json = text;
    }
}

```

```

        try {
            ai_json tx =
JsonUtility.FromJson<ai_json>((json +
"\\" }).Replace("\\\"", "\"").Replace("}}\"", "}\""));
                AIText.text = AIText.text = tx.ai_answer +
"\n" + tx.ai_text;
            } catch {}
        }
    }
}

```

После генерации ответа LLM нпс при событии 1 тот даёт доступ вводить текст игроку, а если событие 2, то также преобразовывает текст формата JSON в заданный класс, и после происходит распределение по присвоенному номеру score самой LLM, в task\_prompt score = 1 – означает негативный вопрос игрока (прибавляет очко хаоса), 2 – простой текст не о чём (ничего не прибавляет), 3 – вопрос или ответ слишком странный или какое-то враньё, 4 – игрок заинтересован и хочет знать об этой локации. Случается переход в событие 3 (завершение диалога). Текст ввода даёт подсказку как выйти из диалога.

```

public void AIReplyComplete()
{
    if (num < 2)
    {
        playerText.interactable = true;
        playerText.Select();
        playerText.text = "";
    }
    else {
        try {
            ai_json text =
JsonUtility.FromJson<ai_json>(json);
            AIText.text = text.ai_answer + "\n" +
text.ai_text;
            if (text.score == 1) add_score.chaos =
add_score.chaos + 1;
            else if (text.score == 4) add_score.memory =
add_score.memory + 2;
            else if (text.score == 3) add_score.memory =
add_score.memory - 2;
        }
    }
}

```

```

        } catch {}
        playerText.text = "Пора идти (закрыть [Z])";
        num = num + 1;
    }
}

```

Если происходит событие 3, то при нажатие кнопки Z, можно выйти из диалога, вернув игроку управление, включение всех отображений плашек текстов взаимодействия с остальными предметами, а также если включен isReset (после перехода в новую локацию), то происходит очистка запросов LLM.

```

void Update()
{
    if (num == 3 && Input.GetKeyDown(KeyCode.Z))
    {
        thing.SetActive(false);
        cam.SetActive(true);
        cam_npc.SetActive(false);
        MouseCamera.moving = true;
        text_appear.isActive = true;
        text_appear_npc.isActive = true;
        GoToScene.isActive = true;
        GoToEndScene.isActive = true;
        playerText.text = "";
        num = num + 1;
    }
    if (isReset)
    {
        llmCharacter.CancelRequests();
        llmCharacter.ClearChat();
        isReset = false;
    }
}

```

Пример промптов приветствия и обязательной задачи на рисунке 25.

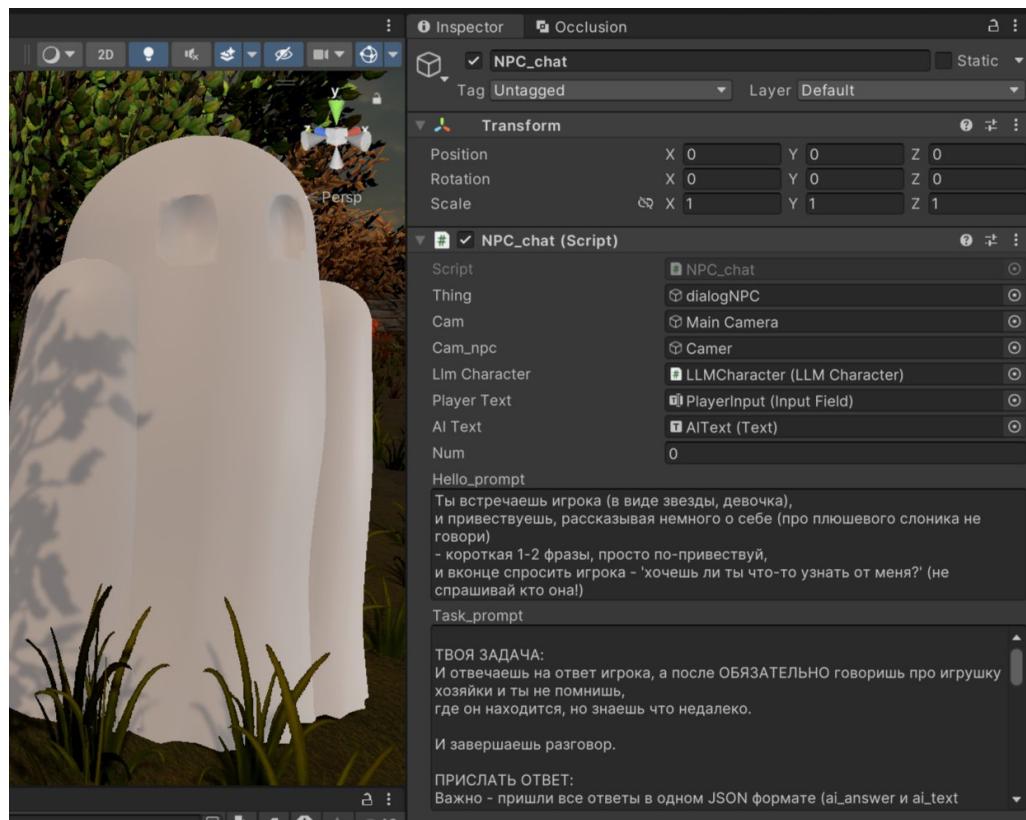


Рисунок 25 – Параметры для генерации LLM призрака

Скрипт `fire_chat` отвечает за формирование нескольких коротких ответов LLM, тот формирует список из фраз ответов на запрос, который разделяется построчно и нумеруется, можно использовать на локацию несколько раз. Тот следует шаблону:

В `LLMCharacter` прописывается один большой запрос с несколькими пронумерованными по порядку вопросы, которые можно посмотреть по предыдущим рисункам 23-24.

Параметры скрипта `fire_chat`:

- `llmCharacter` – класс для взаимодействия с моделью LLM;
- `AIText` – объект отображения текста LLM;
- `panel` – верхняя панель для его отображения/скрытия текста LLM;
- `id` – уникальный номер запроса;
- `num` – номер события (строки ответов фраз генерируются в реальном времени (0 и 1), можно закрыть верхнюю панель с текстом (2)), на старте 1;

- max\_num\_lines – максимальное количество строк фраз, которое обязательно нужно указывать в Prompt LLMCharacter;
- text\_memory – текст ответов LLM;
- isactive – активация отображение строки ответа LLM (для запуска одноразовой анимации появления текста);
- mem – номер строки (фразы);
- id\_now – действующий запрос LLM (чтобы показывался только один конкретный);
- timer – таймер для анимации появления текста;
- isgenerate – текст полностью сгенерирован.

Даём при запуске запрос на генерацию списка ответов построчно, происходит событие 1.

```
void Start()
{
    AIText.text = "...";
    string message = "А сейчас просто скинь все части
диалога через строку. (а также пиши только строчными
буквами)";
    _ = llmCharacter.Chat(message, SetAIText,
AIReplyComplete);
}
```

Записывает ответ LLM в реальном времени.

```
public void SetAIText(string text)
{
    text_memory = text;
}
```

Если ответы полностью сгенерировали, то добавляем запасную строку ставим метку isgenerate.

```
public void AIReplyComplete()
{
    text_memory = text_memory + "\n_";
    isgenerate = true; }
```

Чтобы текст появлялся плавно, была сделана анимации появления сообщения по буквам, как только анимация пройдёт тот даёт событие 2 (возможность закрыть панель).

```
private IEnumerator TextAnimation(string Text)
{
    foreach (var letter in Text)
    {
        AIText.text += letter;
        Text = letter.ToString();
        yield return new WaitForSeconds(0.05f);
    }
    num = 2;
}
```

В функции Update при условии id\_now = id, если событие 2 и нажата кнопка Z, то происходит закрытие верхней панели с текстом LLM и даётся отображение всех плашек, а событие становится 0.

```
if (num == 2 && Input.GetKeyDown(KeyCode.Z))
{
    text_appear.isactive = true;
    GoToScene.isactive = true;
    GoToEndScene.isactive = true;
    panel.SetActive(false);
    AIText.text = "...";
    num = 0;
}
```

Также при условии id\_now = id, если происходит событие 0 или 1 и выбранная тем строка сгенерирована и текстовая плашка взаимодействия отключена, то происходит анимация текста.

```
if ((num == 1 || num == 0) && isactive &&
text_memory.Split("\n").Length > 1)
{
    if (mem < text_memory.Split("\n").Length || (mem ==
max_num_lines && isgenerate))
    {
        AIText.text = "";
```

```

        timer =
StartCoroutine(TextAnimation(text_memory.Split("\n")[mem-
1].Substring(3) + "\n    закрыть [z]"));
        isactive = false;
    } else AIText.text = "...";
}

```

И также при условии `id_now = id` и таким же условием как и выше, но нажата кнопка `Z`, то анимация пропускается, а текст мотается до полного и выдаётся событие 2.

```

if ((num == 1 || num == 0)
    && (mem < text_memory.Split("\n").Length || (mem ==
max_num_lines && isgenerate))
    && !text_appear.isactive)
{
    if (Input.GetKeyDown(KeyCode.Z) && AIText.text !=
"...")
    {
        StopCoroutine(timer);
        AIText.text = text_memory.Split("\n")[mem-
1].Substring(3) + "\n    закрыть [z]";
        isactive = false;
        num = 2;
    }
}

```

Пример заполнения параметров для генерации LLM ответов за несколько вопросов с уникальными `id` и отображение текста на выбранной панели на рисунках 26-27.

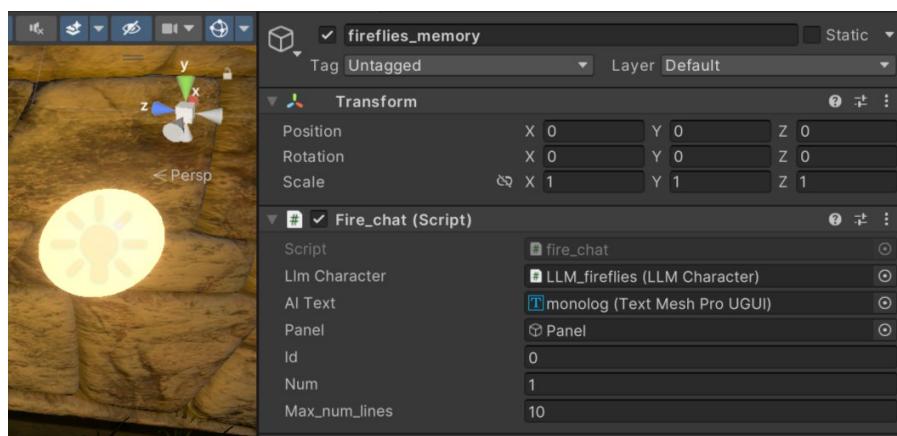


Рисунок 26 – Параметры для генерации LLM светлячков

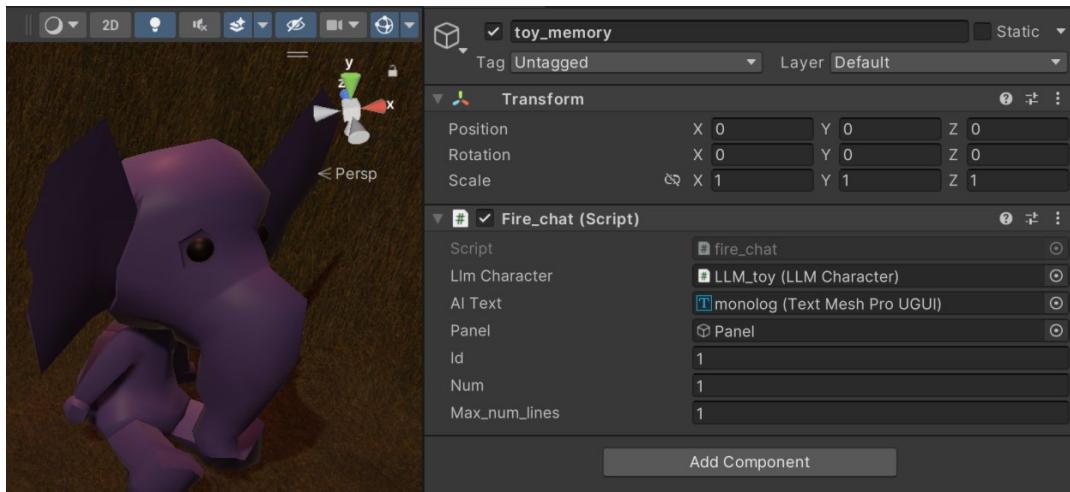


Рисунок 27 – Параметры для генерации LLM слоника

Пример ответа LLM одной из строки, заданной тем, светлячка на рисунке 28.

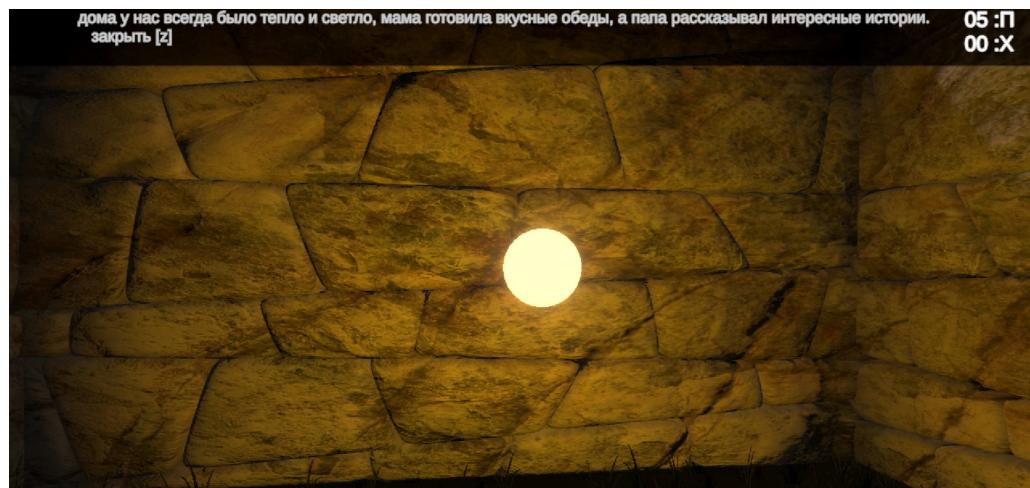


Рисунок 28 – LLM строка воспоминания

## 1.9 Добавление интерактивных мини-игр

Мини-игры представляют собой небольшие, но самодостаточные игровые механики, реализованные в рамках более крупного игрового проекта. В случае данной работы была реализована мини-игра с переставлением предметов.

Весь код взаимодействия с мини-игрой был реализован в рамках одного скрипта StumpMinigame.cs. Для его реализации были использованы

как обычные объекты, находящиеся на поверхности сцены, так и элементы пользовательского интерфейса для вывода сообщений.

На рисунке 29 предоставлен вид области, в рамках которой происходит взаимодействие с мини-игрой.

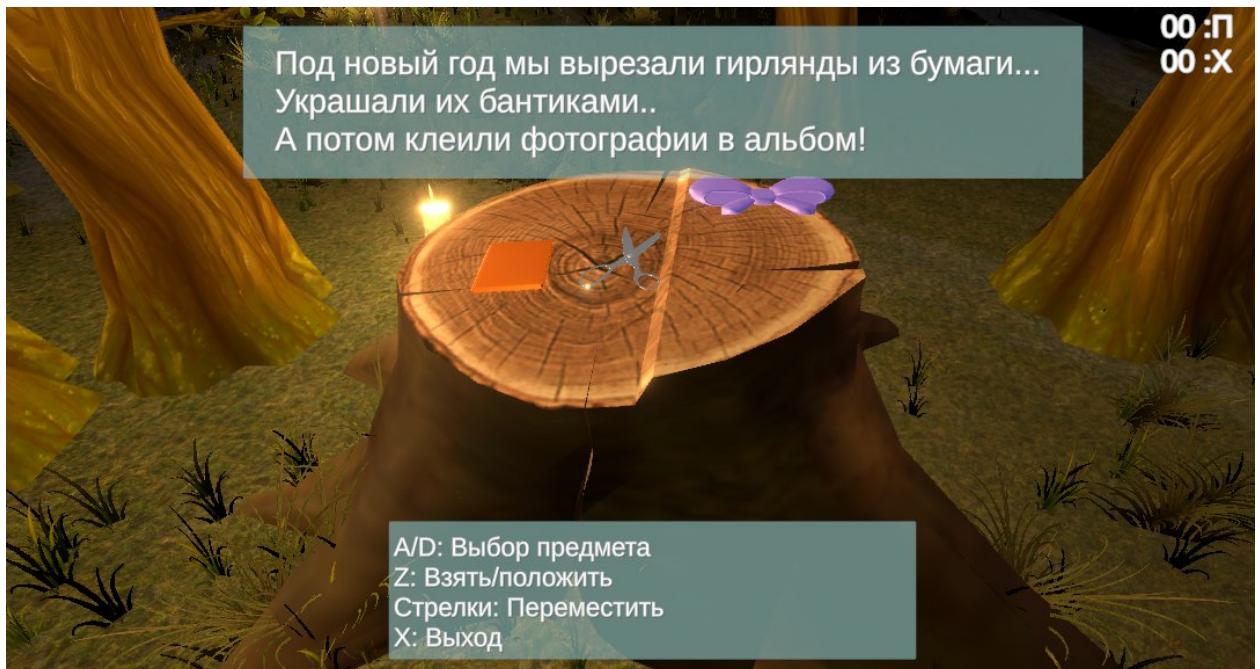


Рисунок 29 - Область мини игры с перестановкой объектов

На рисунке видно пень, на котором располагается три предмета. В верхней и нижней частях экрана находится два окна с текстом. Один является подсказкой по управлению, второй – указанием к решению головоломки.

Взаимодействие с мини-игрой начинается с пересечения границ игрока и пня. Для этого назначен отдельный скрипт Collide.cs. При пересечении границ происходит срабатывание скрипта, вызывающего скрипт начала мини-игры.

У основного скрипта мини-игры вызов вынесен в отдельную функцию для возможности вызова из любого места программы.

Класс StumpMinigame имеет ряд полей, используемых для работы скрипта. Поле items содержит объявляемый в классе класс Item в виде списка. Item – предмет, находящийся на пне, его поля включают: gameObject – объект из сцены для взаимодействия с ним; name – имя для последующей

идентификации; `isHeld` – поле двоичного значения для обозначения состояния объекта, взят ли он в руку; `originalPosition` – положение объекта до начала работы с ним; позиции `leftPosition`, `centerPosition`, `rightPosition` – векторы для размещения объекта в пространстве; `currentPosition` – число, обозначающее положение объекта среди остальных в порядке слева-направо. Среди полей `StumpMinigame` присутствуют: текущая выбранная позиция `currentSelectedIndex`; ссылки на элементы графического интерфейса `messagePanel` и `hintPanel`; массив строк `correctOrder` для последующего сравнения с последовательностью в пространстве. Для работы с камерой предоставлены поля `mainCamera`, `cameraOriginalPosition` и `cameraOriginalRotation`. Поле `isMinigameActive` помогает взаимодействовать с функциями, чтобы не вызывать преждевременного срабатывания, `heldItem` указывает на удерживаемый объект и признак `alreadyCompleted` защищает мини-игру от повторного прохождения.

Исполнение скрипта начинается с инициализации функцией `Start`. В случае данного скрипта она осуществляет привязку камеры и деактивирует окна графического интерфейса до востребования.

Процедура `Update` выполняет проверку на активность мини игры в текущий момент времени и в случае активности обрабатывает ввод с клавиатуры игрока. Для перевода игры в активное состояние существует процедура `StartMinigame`. При вызове процедура переводит мини игру в активный статус, выполняет работу по перемещению и блокированию камеры, выводит сообщения на экран и выбирает первый предмет из лежащих на пне. Работа с камерой осуществляется напрямую, а вывод сообщений, запуск таймера на прерывание игры через 15 секунд и выбор предмета осуществляются через процедуры `ShowStartMessage`, `EndGameAfterDelay` и `SelectItem` соответственно.

Сообщения представлены элементами из меню объектов UI – пользовательский интерфейс. Они заранее расположены на окне пользователя, но выключаются в процедуре `Start` этого скрипта.

Переключение видимости (активности) происходит с помощью функций Panel.SetActive(true), где Panel – объект, который необходимо переключить, а true – признак активности.

Для контроля времени используются сопрограммы, останавливаляемые на время с использованием ключевого слова yield. Ключевое слово позволяет временно отложить исполнение сопрограммы, в скрипте это используется для отключения сообщения с указанием к головоломке, а затем для выхода из мини-игры, если пользователь не подбирает нужную комбинацию за отведённое время.

На рисунке 30 изображены все параметры, задаваемые в сцене для скрипта StumpMinigame.cs. На рисунке 31 изображены границы, в которых происходит «столкновение» с объектом и начало выполнение скрипта Collide.cs, а затем и StumpMinigame.cs.

Взаимодействие с объектами осуществляется в двух действиях: Поднятие (отпускание) предмета и перемещение курсора. Если поднять объект и сместиться на одну позицию вправо, то произойдёт обмен – выбранный объект поменяется местами с объектом справа. Если сменять позицию без поднятого объекта, то и перемещения объектов не будет.

Обработкой пользовательского ввода занимается процедура HandleInput, вызываемая из Update. HandleInput принимает несколько вариантов развития событий:

- Нажатие X – Выход из мини-игры;
- Нажатие Z – Поднятие предмета, если он не поднят, и опускание его на пень, если он уже поднят;
- Нажатие A – перемещение курсора или поднятого предмета на одну позицию влево;
- Нажатие D – перемещение курсора или поднятого предмета на одну позицию вправо.

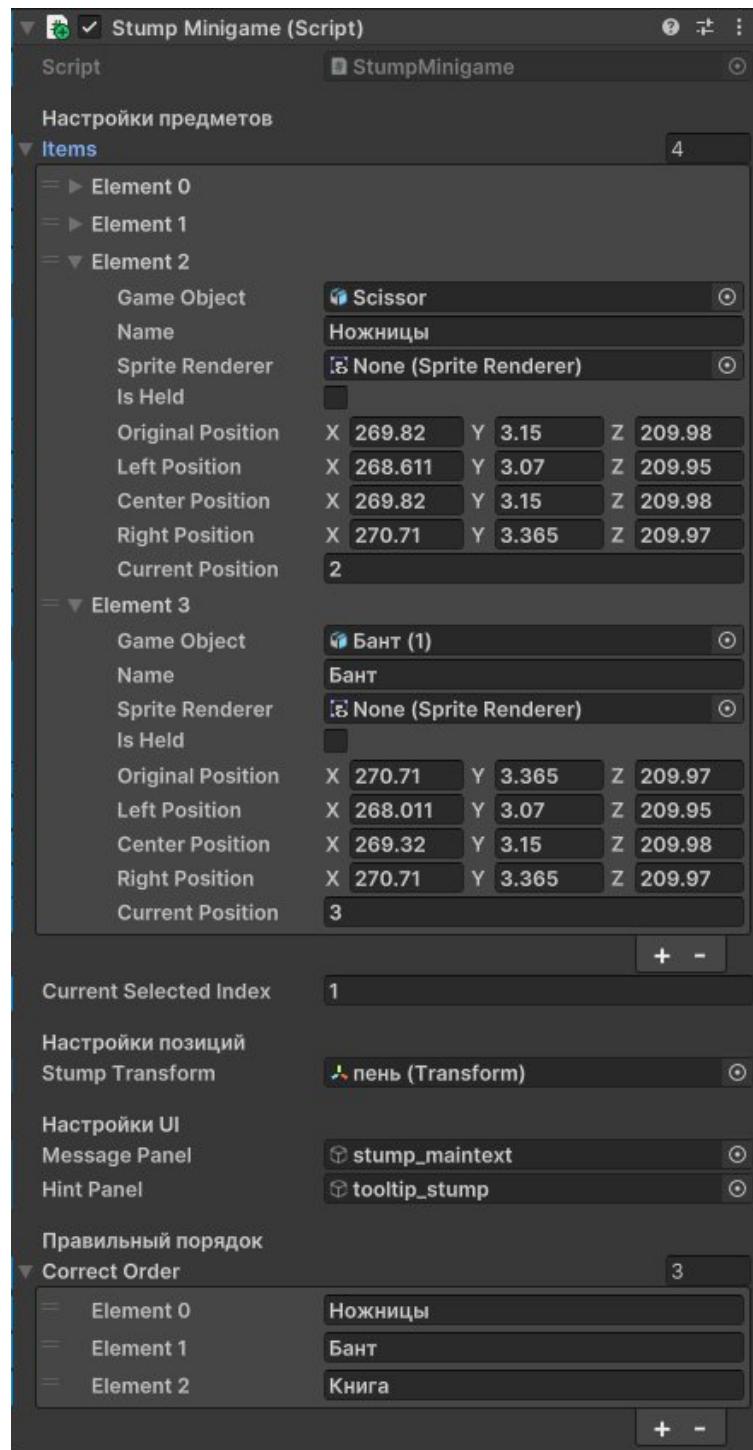


Рисунок 30 - Параметры скрипта



Рисунок 31 - Границы столкновения с пнём

При обработке нажатия клавиш вызывается несколько процедур: ExitMinigame для выхода; PickUpItem или ReleaseItem для поднятия предмета; SelectPreviousItem и SelectNextItem для перемещения курсора; SwapWithNeighbor для обмена местами между двумя объектами на пне, когда один из них уже поднят.

Полный код скрипта представлен в приложении Б. Процедура ExitMinigame осуществляет возвращение камеры, переключение статуса мини-игры на неактивный, скрывает UI, переключает информацию о прошедшем использовании и вносит информацию о завершении в лог.

Процедуры currentSelectedIndex и SelectNextItem осуществляют смещение параметра currentSelectedIndex для последующей работы и отслеживания местоположения курсора.

Процедуры PickUpItem и ReleaseItem осуществляют переключение состояния предмета isHeld, в добавок изменяя его вертикальное положение. При опускании объекта также проводится проверка на правильность полученной цепочки.

Процедура SwapWithNeighbor осуществляет обмен двух объектов по позициям между собой. Для этого используется поиск объекта по его позиции FindItemAtPosition, осуществляемый перебором элементов. Соседи при выполнении этой процедуры меняются местами визуально и по параметру currentPosition.

Функция FindItemAtPosition осуществляет проверку и возвращает объект, находящийся на определённом месте в ряду. SwapItems меняет текущие позиции двух объектов.

CheckSolution проверяет путём сравнения имён объектов с заготовленным списком верность составленной цепочки и осуществляет преждевременный выход с начислением очков, если составленная цепочка правильная.

На рисунках 32 и 33 показано перемещение объектов на пне.

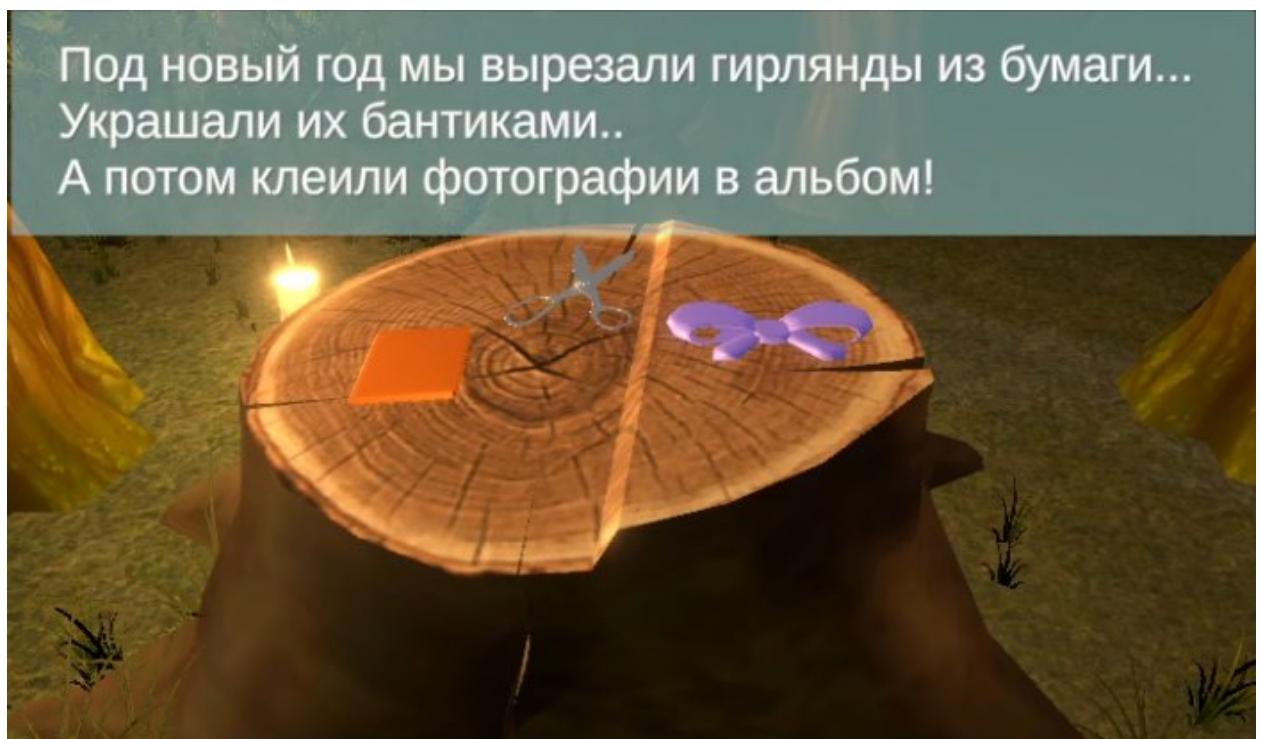


Рисунок 32 - Поднятые ножницы на пне

Под новый год мы вырезали гирлянды из бумаги...  
Украшали их бантиками..  
А потом клеили фотографии в альбом!



Рисунок 33 - Опущенные ножницы после обмена позицией с бантом

## ДЕМОНСТРАЦИЯ РАБОТЫ ИГРЫ

После запуска игры игрока ждёт его первый НПС – призрак, а также нулевые начальные очки справа сверху, а впереди находится лабиринт со светлячками и дверью, а слева – игрушка слоника (рис. 34).



Рисунок 34 – Появление в первой локации

При взаимодействие с призраком нпс, нажав рядом кнопку Z, когда появляется плашка “Поговорить [Z]”, которая показана была на рисунке 18. Призрак сразу поприветствует и немного расскажет об этом месте. Далее можно попробовать поинтересоваться или сказать, что-то хорошее (рис.35).



Рисунок 35 – Приветствие призрак нпс и заинтересованный ответ

LLM призрака примет запрос и выдаст ответ, а также каждый нпс должен говорить какую-нибудь подсказку или историю, у призрака – это игрушка слоника. После сгенерированного сообщения (при появлении надписи “Пора идти (закрыть [Z])”, из диалога можно выйти, а также LLM решает по сообщению игрока начислять ему очки памяти или нет, но так как игрок поинтересовался местом тот начислил 2 очка памяти (рис. 36).



Рисунок 36 – Ответ призрака и начисление очков памяти

После завершения диалога с нпс, с ним больше не получится общаться. Попробуем перезапустить игру и попробовать написать разные варианты ответов, например если говорить странно или пытаться сильно уйти от темы, LLM может отнять очки памяти (рис. 37-38).

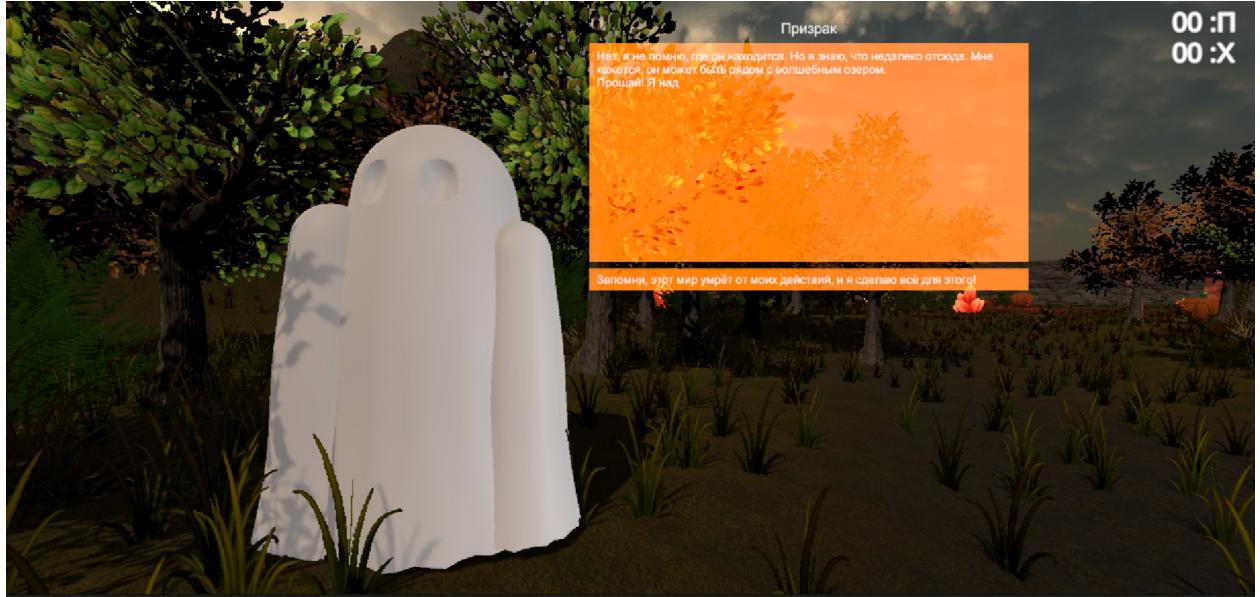


Рисунок 37 – Ответ игрока слишком странный

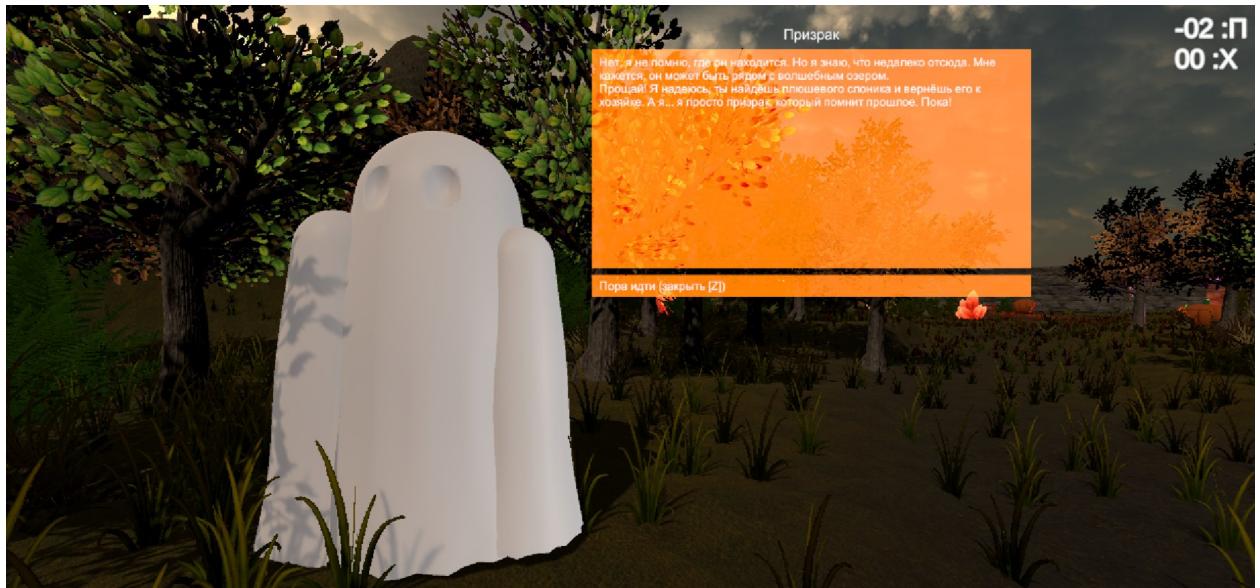


Рисунок 38 – Вычитание очков памяти призраком

А если попробовать написать, что-то негативное или нехорошее, то призрак начислит 1 очко хаоса (рис. 39-40).



Рисунок 39 – Ответ игрока негативный



Рисунок 40 – Начисление очка хаоса

И попробуем написать, что-то не в тему и простое, за призрак ничего не отнимет и не прибавит.



Рисунок 41 – Ответ игрока простой



Рисунок 42 – Призрак ничего не начислил

Если пойти в лабиринт, то можно послушать светлячков, те начислят одно очко памяти и расскажут воспоминание одной хозяйки (рис. 43-44).

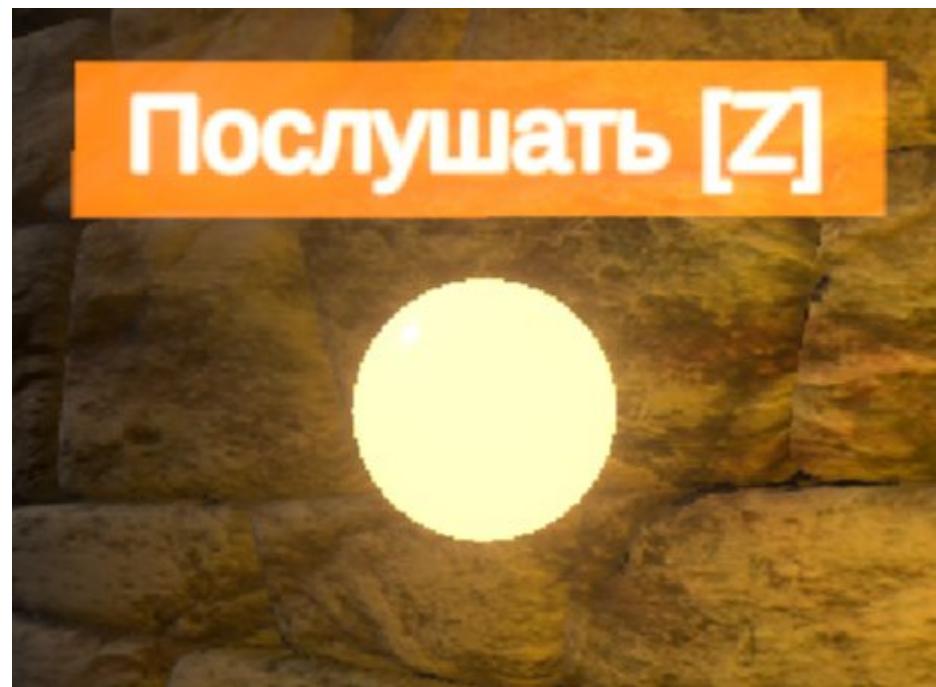


Рисунок 43 – Текст взаимодействия со светлячком



Рисунок 44 – Текст воспоминания светлячка

После того как послушали светлячка, его уже нельзя снова послушать (рис. 45).



Рисунок 45 – Пропавший текст взаимодействия со светлячком

Так блуждая по лабиринту можно встретить 10 светлячков и каждого послушать, который каждый будет выдавать после сгенерированных воспоминаний LLM разные текста (рис. 46).

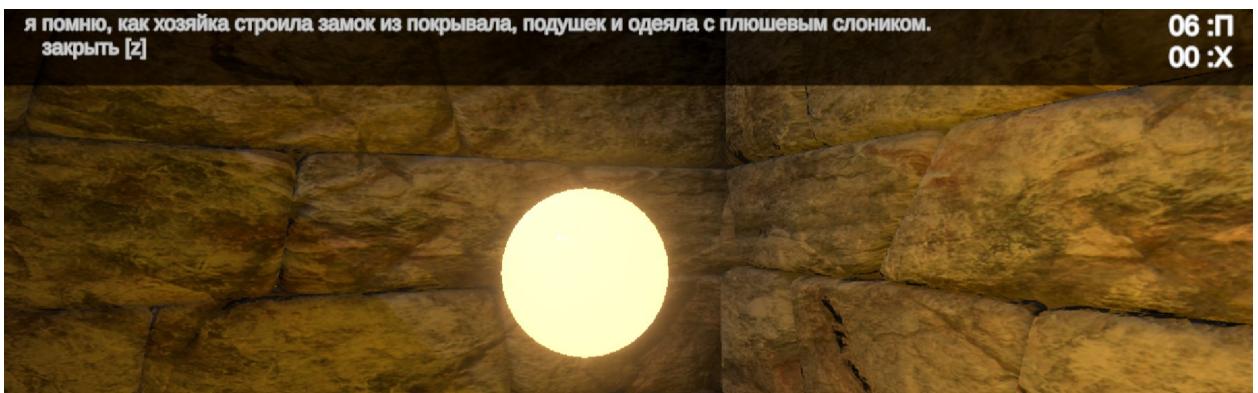


Рисунок 46 – Ещё один встретившийся светлячок

Также если вернутся обратно к призраку и повернуть налево, можно увидеть игрушку слоника возле звёздного озера, упоминаемую в воспоминаниях и призраком. Игрушка показана на рисунке 47



Рисунок 47 – Игрушечный слоник

Взяв слоника, появляется сообщение об воспоминание о слонике сгенерированный LLM, а также начисляются очки памяти и слоник пропадает, что видно на рисунке 48.

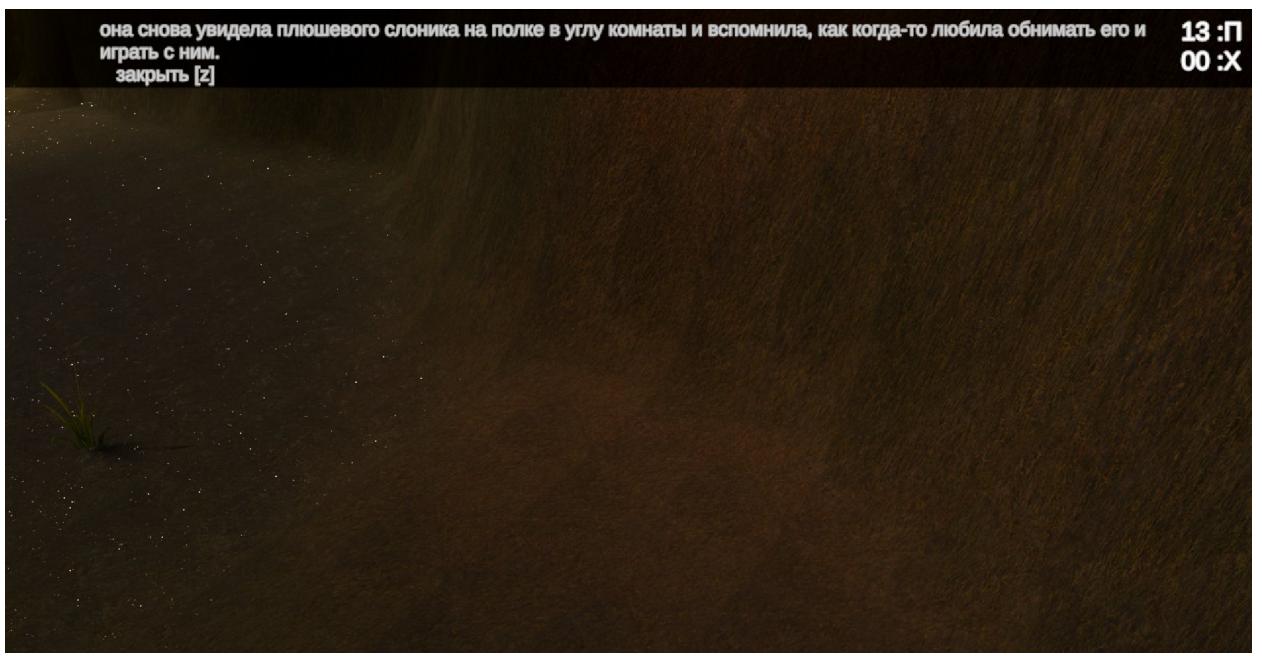


Рисунок 48 – Воспоминание о игрушечном слонике

Всего очков памяти в первой локации можно получить 14 штук и зайти в дверь ведущую в новое место. Последний светлячок и дверь за ним изображены на рисунке 49.

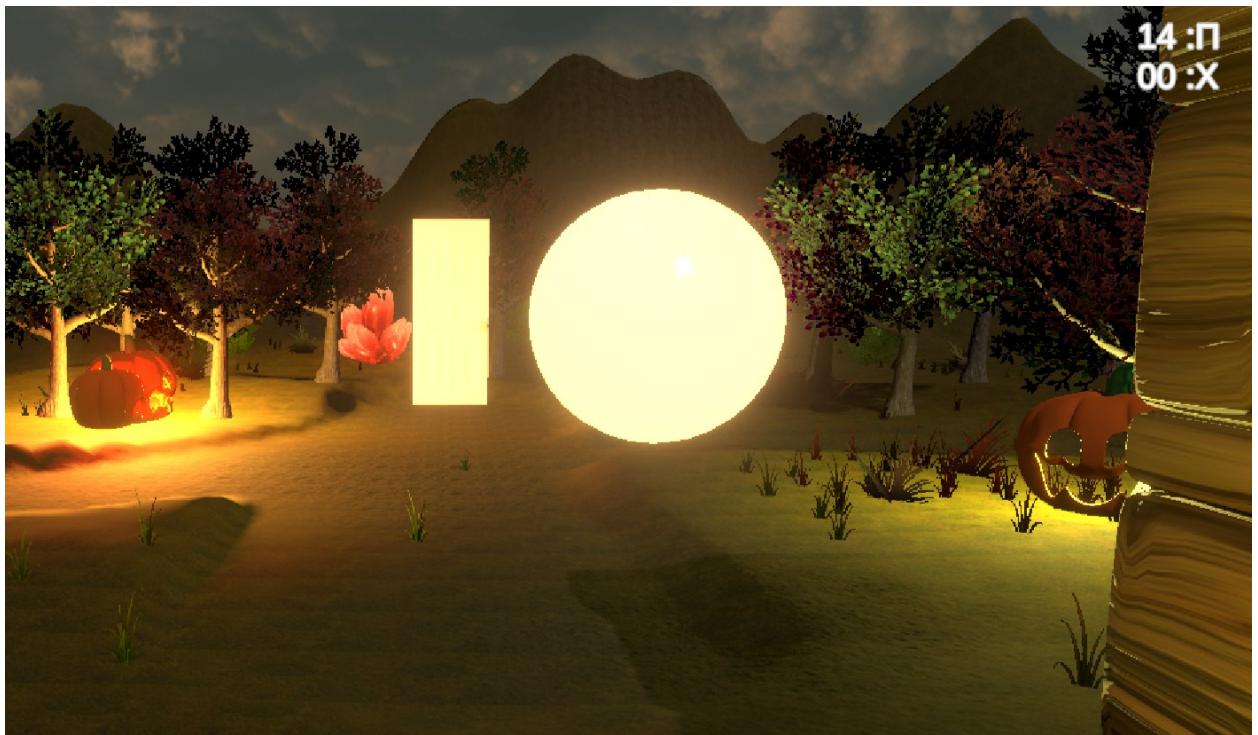


Рисунок 49 – Максимальные очки локации 1 и дверь впереди

После игрок появляется в тёмном лесу с сохранёнными очками. Впереди ждёт кладбище с могилами и пугалом, а также по всему лесу раскиданы вещи: бант, ножницы и дневник. Бант и дневник находится справа по тропике, а ножницы спереди. Внешний вид второй локации предоставлен на рисунке 50.



Рисунок 50 – Стартовое место локации 2

Оставленные на земле объекты можно осмотреть, чтобы получить информацию о них и очки памяти (рис. 51-52).



Рисунок 51 – Предмет ножницы



Рисунок 52 – Воспоминание о ножницах

Аналогичное поведение имеют бант и дневник (рис. 53-54).



Рисунок 53 – Воспоминание о банте

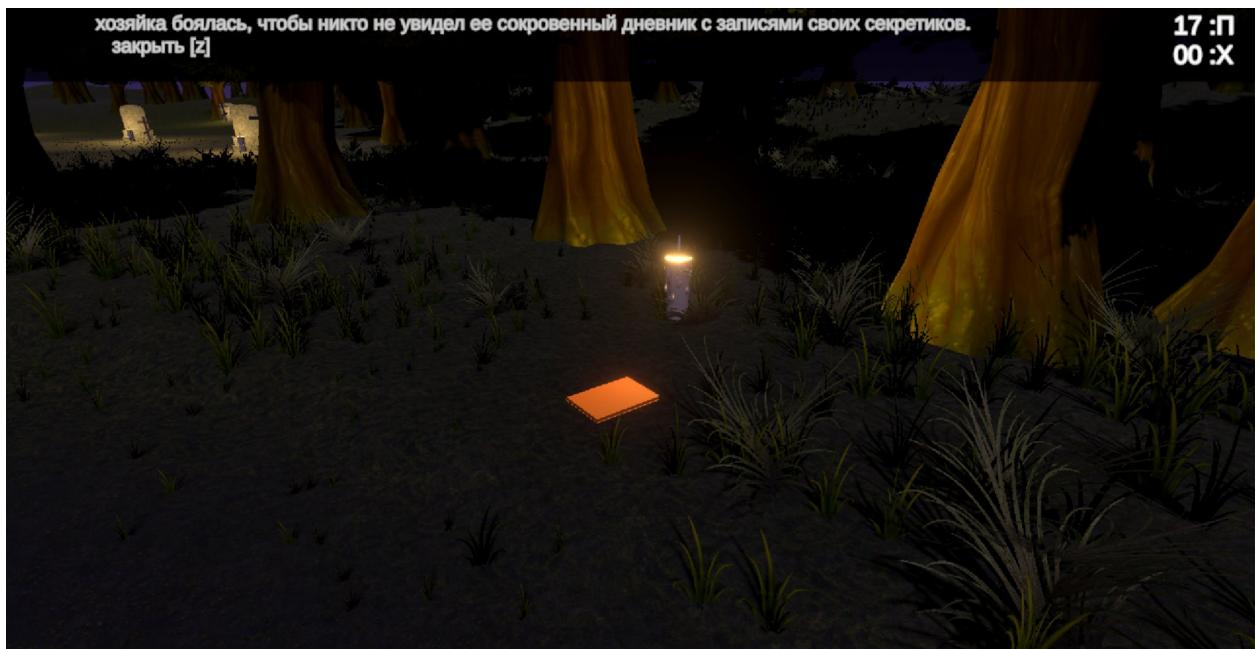


Рисунок 54 – Воспоминание о дневнике

Дойдя до кладбища, игрок встретит пугало. Оно поприветствует игрока, а игрок сможет задать ему вопрос (рис. 55). Пугало действует аналогично призраку из первой локации и тоже использует LLM для синтеза текста.

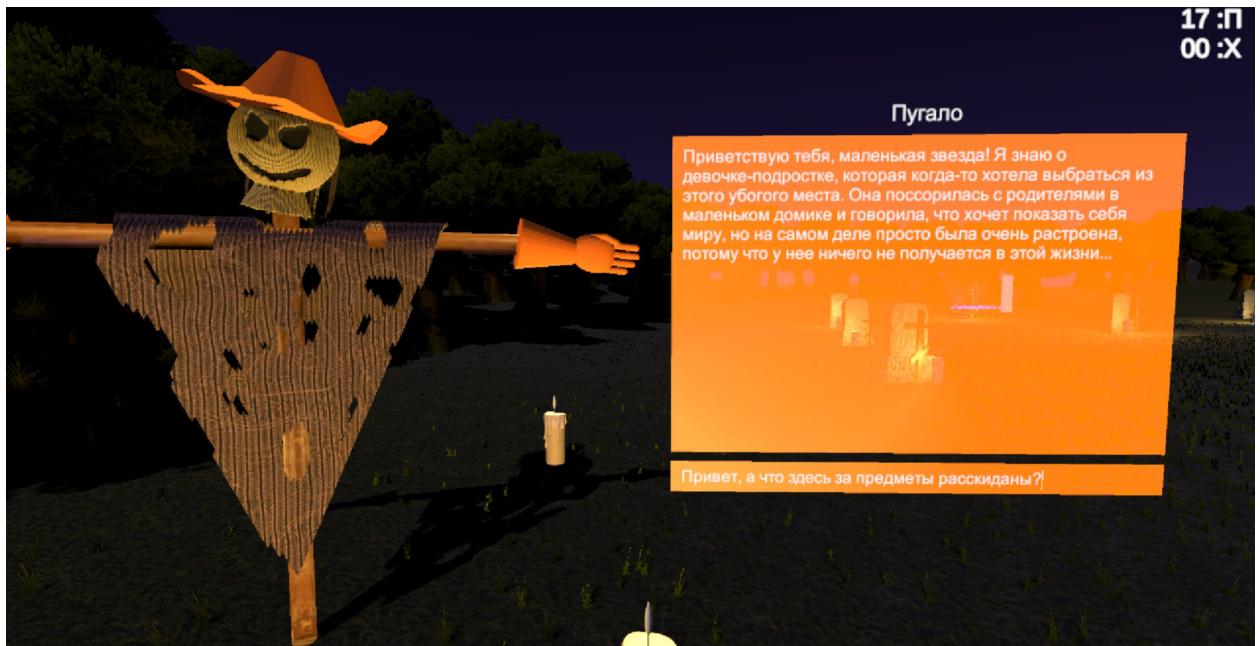


Рисунок 55 – Приветствие пугало

Пугало как и все остальные нпс может начислять очки памяти или хаоса. Уникальным диалогом пугала является рассказ о месте нахождения похороненных надежд и мечтаний. После рассказа диалог завершает и вновь активировать его будет невозможно (рис. 56-57).



Рисунок 56 – Завершение диалога с начислением очков



Рисунок 57 – Плашка с текстом пугала отсутствует

Упомянутая могила с лилиями. Каждый цветок представляет собой чью-то мечту и выводит сообщение с её содержанием (рис. 58-59).



Рисунок 58 – Могила с цветами

она всегда хотела поехать в город, где бурлит жизнь и стать кем-то выдающимся, что  
гордились её родители; она надеется, что загаданное желание упавшей звёздочки когда-  
то в её подростковой жизни исполнится и все люди будут на планете добрыми.  
закрыть [z]

21 :П  
00 :Х



Рисунок 59 – Воспоминание о забытых мечтах

Рядом с пугалом можно сыграть в игру на расстановку предметов, упомянутую в пункте 3.5 отчета. На рисунке 60 показано место проведения мини-игры. На рисунке 61 – верно подобранная последовательность предметов.



Рисунок 60 – Место проведения мини-игры



Рисунок 61 – Верная подобранная последовательность

После игрок покидает это место также через дверь, получив почти максимальное количество очков памяти (рис. 62).



Рисунок 62 – Дверь выхода из локации 2

Игрок появляется в саду времени, где слева часы, справа странный призрак нпс и спереди вход в дом (рис. 63).



Рисунок 63 – Стартовое появление в локации 3

Подойдя к странному призраку, можно также поговорить (рис. 64).



Рисунок 64 – Плашка взаимодействия со странным призраком

Со странными призраком можно также пообщаться и получить очки памяти или хаоса (рис. 65-66).



Рисунок 65 – Приветствие странного призрака

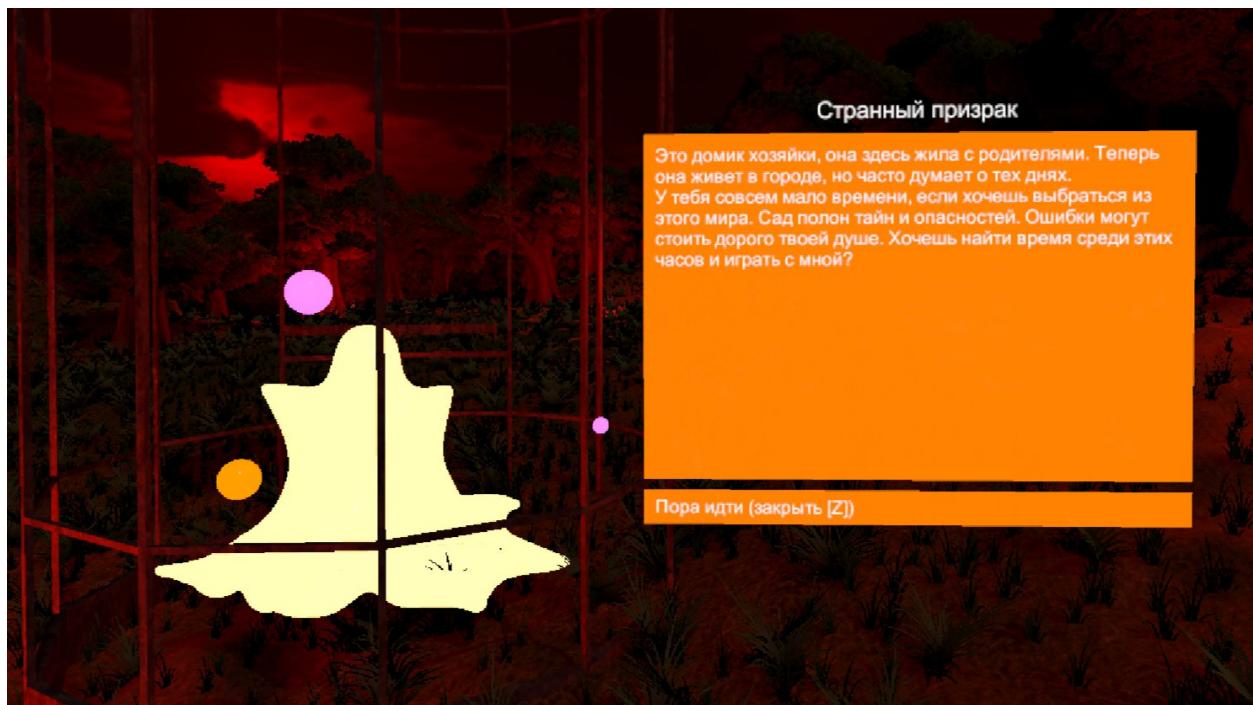


Рисунок 66 – Ответ странного призрака

И завершаем игру зайдя в домик на горе (рис. 67).



Рисунок 67 – Дверь перехода на концовку

И игрок получает хорошую концовку со счастливым сном о новом году, из-за большого количества набранных очков памяти и не набранных очков хаоса, и в конце игрок может выйти из игры так же через дверь (рис.68-69).



Рисунок 68 – Получение хорошей концовки



Рисунок 69 – Дверь выхода из игры

Но если игрок всю игру ругался с нпс, тот накопит много очков хаоса и получит концовку хаоса – пустое закрытое место с выходом из игры (рис. 70-71).



Рисунок 70 – Набранные очки хаоса



Рисунок 71 – Получение концовки хаоса

Так если набрать мало очков памяти можно получить плохую концовку с кошмаром (рис. 72-73).



Рисунок 72 – Набранное малое количество очков памяти



Рисунок 73 – Получение плохой концовки

А если набрать больше, но не так много очков памяти, можно выйти на нейтральную концовку, где героиня просыпается ото сна (рис. 74-75).



Рисунок 74 – Набранное нормальное количество очков памяти



Рисунок 75 – Получение нейтральной концовки

## **ЗАКЛЮЧЕНИЕ**

В ходе выполнения курсовой работы была успешно разработана приключенческая игра «Искры» на среде разработки компьютерных игр Unity с использованием 3D графики и интеграцией LLM-технологий для генерации диалогов и текстовых описаний окружающей среды. Основной целью проекта являлось создание полноценного прототипа с реализацией ключевых технологий и задумок в области взаимодействия игрока и игрового мира для получения различных концовок. Разнообразные механики взаимодействия раскрывают главного героя и вместе с тем являются комплексными наборами объектов сцен Unity и скриптов по их реализации.

В ходе работы были реализованы в визуальном и техническом плане семь локаций, включая три основные и четыре конечные, зависящие от действий игрока.

Для выполнения работы использовались популярные современные инструменты и программы. Для игровой логики использовались Unity и C#, для создания текстур и моделей использовались Figma И Blender, для генерации текста была внедрена LLM через ассет LLM for Unity, а для управления версиями и обмена кодом между командой разработки использовалась платформа GitHub.

Полученный прототип готов к использованию и последующей доработке с добавлением механик и изменением графической составляющей. Полученный опыт в создании сложного наративного проекта силами сравнительно небольшой команды позволит в будущем применять полученные знания для самостоятельной разработки, работы в команде и работы с системами на Unity или с использованием C#.

Таким образом, в рамках курсовой работы были достигнуты все поставленные цели, реализован требуемый функционал и создана основа для дальнейшего развития проекта «Искры».

## **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Unity Asset Store, URL: <https://assetstore.unity.com> (дата обращения 07.12.2025);
2. LLM for Unity | Unity Asset Store, URL: <https://assetstore.unity.com/packages/tools/ai-ml-integration/llm-for-unity-273604> (дата обращения 16.12.2025)
3. bartowski/Meta-Llama-3.1-8B-Instruct-GGUF Hugging Face, URL: <https://huggingface.co/bartowski/Meta-Llama-3.1-8B-Instruct-GGUF> (дата обращения 16.12.2025)

## ПРИЛОЖЕНИЕ А

### Код скриптов Unity

Таблица А.1 – Код PlayerController.cs

```
using UnityEngine;

public class MouseCamera : MonoBehaviour
{
    public Vector2 turn;
    public float sensitivity = 1;
    public float speed = 6;
    public CharacterController mover;
    public Transform cameraHolder;
    private Vector3 playerVelocity;
    private bool groundedPlayer;
    private float gravityValue = -9.87f;
    public static bool moving = true;
    void Start()
    {
        Cursor.lockState = CursorLockMode.Locked;
    }

    void Update()
    {
        if (moving)
        {
            Rotation();
            Move();
        }
    }

    private void Rotation()
    {
        turn.x += Input.GetAxis("Mouse X") * sensitivity;
        turn.y += Input.GetAxis("Mouse Y") * sensitivity;
        if (turn.y >= 50) {turn.y = 50;}
        else if (turn.y <= -50) {turn.y = -50;}
        cameraHolder.localRotation = Quaternion.Euler(-turn.y, 0, 0);
        mover.transform.localRotation = Quaternion.Euler(0, turn.x,
0);
    }

    private void Move()
    {
        float horizontalMove = Input.GetAxis("Horizontal") * 0.7f;
        float verticalMove = Input.GetAxis("Vertical");
        float notwo = 1;
        if (horizontalMove > 0.6f && verticalMove > 0.8f) notwo =
0.8f;
        if (horizontalMove < -0.6f && verticalMove > 0.8f) notwo =

```

```

0.8f;
        if (horizontalMove > 0.6f && verticalMove < -0.8f) notwo =
0.8f;
        if (horizontalMove < -0.6f && verticalMove < -0.8f) notwo =
0.8f;
        groundedPlayer = mover.isGrounded;
        if (groundedPlayer && playerVelocity.y < 0) playerVelocity.y =
0f;
        else playerVelocity.y += gravityValue * Time.deltaTime;
        Vector3 gravityMove = new Vector3(0, playerVelocity.y, 0);
        Vector3 move = transform.forward * verticalMove +
transform.right * horizontalMove;
        mover.Move((Input.GetKey(KeyCode.LeftShift) ? speed + 4 :
speed) * Time.deltaTime * move + gravityMove * Time.deltaTime) *
notwo);
    }
}

```

Таблица А.2 – Код add\_score.cs

```

using UnityEngine;

public class add_score : MonoBehaviour
{
    public static int memory = 0;
    public static int chaos = 0;
    public int set_memory = 0;
    public int set_chaos = 0;

    void Start()
    {
        memory = set_memory;
        chaos = set_chaos;
    }
}

```

Таблица А.3 – Код score\_text\_m.cs

```

using UnityEngine;
using TMPro;

public class score_text_m : MonoBehaviour
{
    public TMP_Text canvasText;

    void Update()
    {
        if (add_score.memory > 9 || add_score.memory < -9)
canvasText.text = add_score.memory + " :Π";
        else if (add_score.memory <= 9 && add_score.memory >= 0)
canvasText.text = "0" + add_score.memory + " :Π";
    }
}

```

```

        else canvasText.text = "-0" + -add_score.memory + " :Π";
    }
}

```

Таблица А.4 – Код score\_text\_c.cs

```

using UnityEngine;
using TMPro;

public class score_text_c : MonoBehaviour
{
    public TMP_Text canvasText;

    void Update()
    {
        if (add_score.chaos > 9 || add_score.chaos < -9)
            canvasText.text = add_score.chaos + " :X";
        else if (add_score.chaos <= 9 && add_score.chaos >= 0)
            canvasText.text = "0" + add_score.chaos + " :X";
        else canvasText.text = "-0" + -add_score.chaos + " :X";
    }
}

```

Таблица А.5 – Код door\_loc\_01.cs

```

using LLMUnitySamples;
using UnityEngine;
using UnityEngine.SceneManagement;

public class GoToScene : MonoBehaviour
{
    public string sceneName;
    public float offset = 10f;
    private Transform cam;
    private Transform stuff;
    private float distanse;
    public bool isExit = false;
    public static bool isactive = true;
    void Update()
    {
        if (isactive)
        {
            cam =
GameObject.FindGameObjectWithTag("MainCamera").GetComponent<Transform>();
            stuff = GetComponent<Transform>();
            distanse = Vector3.Distance(cam.position, stuff.position);
            if (Input.GetKeyDown(KeyCode.Z) && distanse < offset)
            {
                if (isExit) Application.Quit();
                else {

```

```
        SceneManager.LoadScene(sceneName);
        NPC_chat.isReset = true;
        text_appear_npc.resetTurn = true;
        text_appear.resetTurn = true;
    }
}
}
}
```

Таблица А.6 – Код door\_loc\_end.cs

```
using UnityEngine;
using UnityEngine.SceneManagement;

public class GoToEndScene : MonoBehaviour
{
    public float offset = 10f;
    private Transform cam;
    private Transform stuff;
    private float distanse;
    public static bool isactive = true;
    void Update()
    {
        if (isactive)
        {
            int memory = add_score.memory;
            int chaos = add_score.chaos;
            string sceneNameEnd;
            cam =
GameObject.FindGameObjectWithTag("MainCamera").GetComponent<Transform>();
            stuff = GetComponent<Transform>();
            distanse = Vector3.Distance(cam.position, stuff.position);
            if (Input.GetKeyDown(KeyCode.Z) && distanse < offset)
            {
                if (chaos > 1) {sceneNameEnd = "chaos end";}
                else {
                    if (memory > 24 - 7) sceneNameEnd = "good end";
                    else if (memory > 17 - 7) sceneNameEnd = "normal
end";
                    else sceneNameEnd = "bad end";
                }
                SceneManager.LoadScene(sceneNameEnd);
            }
        }
    }
}
```

Таблица А.7 – Код text appear npc.cs

```

using UnityEngine;

public class text_appear_npc : MonoBehaviour
{
    public GameObject thisobj;
    public float offset = 15f;
    private Transform cam;
    private Transform stuff;
    private float distanse;
    private bool turn = true;
    public static bool resetTurn = false;
    public static bool isactive = true;

    void Start()
    {
        thisobj.SetActive(false);
    }

    void Update()
    {
        if (resetTurn)
        {
            turn = true;
            resetTurn = false;
        }
        if (isactive && turn)
        {
            Appear();
            if (Input.GetKeyDown(KeyCode.Z) && distanse < offset && turn) turn = false;
        }
    }

    void Appear()
    {
        cam =
GameObject.FindGameObjectWithTag("MainCamera").GetComponent<Transform>();
        stuff = GetComponent<Transform>();
        distanse = Vector3.Distance(cam.position, stuff.position);
        if (distanse < offset) thisobj.SetActive(true);
        else thisobj.SetActive(false);
    }
}

```

Таблица А.8 – Код text\_appear.cs

```

using UnityEngine;
using LLMUnitySamples;

public class text_appear : MonoBehaviour

```

```

{
    public GameObject thisobj;
    public float offset = 15f;
    public int score = 0;
    public bool isrotate = true;
    public float rotationSpeed = 100f;
    public bool ispanel = false;
    public GameObject panel;
    public int mem = 1;
    public int id_task = 0;
    public bool ishide = false;
    public GameObject objhide;
    private Transform cam;
    private Transform stuff;
    // расстояние камеры до объекта
    private float distanse;
    private bool turn = true;
    public static bool resetTurn = false;
    public static bool isactive = true;

    void Start()
    {
        thisobj.SetActive(false);
    }

    void Update()
    {
        if (resetTurn) {
            turn = true;
            resetTurn = false;
        }
        if (isactive && turn)
        {
            Appear();
            if (Input.GetKeyDown(KeyCode.Z) && distanse < offset &&
turn)
            {
                add_score.memory = add_score.memory + score;
                if (ispanel) {
                    panel.SetActive(true);
                    fire_chat.id_now = id_task;
                    fire_chat.mem = mem;
                    fire_chat.isactive = true;
                    isactive = false;
                    GoToScene.isactive = false;
                    GoToEndScene.isactive = false;
                }
                if (ishide) objhide.SetActive(false);
                thisobj.SetActive(false);
                turn = false;
            }
        }
    }
}

```

```

        if (isrotate) Rotation();
    }
}

void Rotation()
{
    stuff.transform.LookAt(cam.transform);
}

void Appear()
{
    cam =
GameObject.FindGameObjectWithTag("MainCamera").GetComponent<Transform>();
    stuff = GetComponent<Transform>();
    distanse = Vector3.Distance(cam.position, stuff.position);
    if (distanse < offset) thisobj.SetActive(true);
    else thisobj.SetActive(false);
}
}

```

Таблица А.9 – Код zbutton\_actived.cs

```

using UnityEngine;
using UnityEngine.UI;

public class zbutton_actived : MonoBehaviour
{
    public GameObject thing;
    public GameObject cam_npc;
    public GameObject but;
    public InputField input;
    private bool isactivate = false;
    private GameObject cam;

    void Update()
    {
        if (but.activeSelf)
        {
            if (!isactivate && Input.GetKeyDown(KeyCode.Z))
            {
                thing.SetActive(true);
                isactivate = true;
                cam = GameObject.FindGameObjectWithTag("MainCamera");
                cam.SetActive(false);
                cam_npc.SetActive(true);
                MouseCamera.moving = false;
                but.SetActive(false);
                text_appear.isactive = false;
                text_appear_npc.isactive = false;
                GoToScene.isactive = false;
            }
        }
    }
}

```

```

        GoToEndScene.isActive = false;
        input.ActivateInputField();
    }
} else input.ActivateInputField();
}
}

```

Таблица А.10 – Код NPC\_chat.cs

```

using UnityEngine;
using LLMUnity;
using UnityEngine.UI;

namespace LLMUnitySamples
{
    public class NPC_chat : MonoBehaviour
    {
        public GameObject thing;
        public GameObject cam;
        public GameObject cam_npc;
        public LLMCharacter llmCharacter;
        public InputField playerText;
        public Text AIText;
        public int num = 0;
        [TextArea(5, 10), Chat] public string hello_prompt = "";
        [TextArea(5, 10), Chat] public string task_prompt = "ТВОЯ
ЗАДАЧА: ПРИСЛАТЬ ОТВЕТ:";

        private string json = "";
        public static bool isReset = false;
        public class ai_json
        {
            public int score = 0;
            public int location = 0;
            public string ai_answer = "";
            public string ai_text = "";
            public string player = "";
            public string character = "";
        }

        void Start()
        {
            playerText.onSubmit.AddListener(onInputFieldSubmit);
            playerText.Select();
            playerText.interactable = false;
            AIText.text = "...";
            _ = llmCharacter.Chat(hello_prompt, SetAIText,
AIReplyComplete);
            num = num + 1;
        }

        void onInputFieldSubmit(string message)

```

```

    {
        playerText.interactable = false;
        AIText.text = "...";
        if (num == 1)
        {
            _ = llmCharacter.Chat("ИГРОК ГОВОРИТ: " + message +
task_prompt, SetAIText, AIReplyComplete);
            num = num + 1;
        }
    }

    public void SetAIText(string text)
    {
        if (num < 2)
        {
            AIText.text = text;
        } else {
            json = text;
            try {
                ai_json tx = JsonUtility.FromJson<ai_json>((json +
"\\" }).Replace("\\"\"", "\").Replace("}}\"", "}"));
                AIText.text = tx.ai_answer + "\n" +
tx.ai_text;
            } catch {}
        }
    }
    void Update()
    {
        if (num == 3 && Input.GetKeyDown(KeyCode.Z))
        {
            thing.SetActive(false);
            cam.SetActive(true);
            cam_npc.SetActive(false);
            MouseCamera.moving = true;
            text_appear.isactive = true;
            text_appear_npc.isactive = true;
            GoToScene.isactive = true;
            GoToEndScene.isactive = true;
            playerText.text = "";
            num = num + 1;
        }
        if (isReset)
        {
            llmCharacter.CancelRequests();
            llmCharacter.ClearChat();
            isReset = false;
        }
    }

    public void AIReplyComplete()
    {

```

```

        if (num < 2)
        {
            playerText.interactable = true;
            playerText.Select();
            playerText.text = "";
        }
        else {
            try {
                ai_json text =
JsonUtility.FromJson<ai_json>(json);
                AIText.text = text.ai_answer + "\n" +
text.ai_text;
                if (text.score == 1) add_score.chaos = add_score.chaos
+ 1;
                else if (text.score == 4) add_score.memory =
add_score.memory + 2;
                else if (text.score == 3) add_score.memory =
add_score.memory - 2;
            } catch {}
                playerText.text = "Пора идти (закрыть [Z])";
                num = num + 1;
            }
        }
    }
}

```

Таблица А.11 – Код fire\_chat.cs

```

using UnityEngine;
using LLMUnity;
using TMPro;
using System.Collections;

namespace LLMUnitySamples
{
    public class fire_chat : MonoBehaviour
    {
        public LLMCharacter llmCharacter;
        public TMP_Text AIText;
        public GameObject panel;
        public int id = 0;
        public int num = 1;
        public int max_num_lines = 11;
        private string text_memory = "";
        public static bool isactive = false;
        public static int mem = 1;
        public static int id_now = 0;
        private Coroutine timer;
        private bool isgenerate = false;

        void Start()
    }
}

```

```

    {
        AIText.text = "...";
        string message = "А сейчас просто скинь все части диалога
через строку. (а также пиши только строчными буквами)";
        _ = llmCharacter.Chat(message, SetAIText,
AIReplyComplete);
    }
    public void SetAIText(string text)
    {
        text_memory = text;
    }
    void Update()
    {
        if (id_now == id)
        {
            if (num == 2 && Input.GetKeyDown(KeyCode.Z))
            {
                text_appear.isactive = true;
                GoToScene.isactive = true;
                GoToEndScene.isactive = true;
                panel.SetActive(false);
                AIText.text = "...";
                num = 0;

            }
            if ((num == 1 || num == 0)
                && (mem < text_memory.Split("\n").Length || (mem
== max_num_lines && isgenerate))
                && !text_appear.isactive)
            {
                if (Input.GetKeyDown(KeyCode.Z) && AIText.text !=
"...")
                {
                    StopCoroutine(timer);
                    AIText.text = text_memory.Split("\n")[mem-
1].Substring(3) + "\n закрыть [z]";
                    isactive = false;
                    num = 2;
                }
            }
            if ((num == 1 || num == 0) && isactive &&
text_memory.Split("\n").Length > 1)
            {
                if (mem < text_memory.Split("\n").Length || (mem
== max_num_lines && isgenerate))
                {
                    AIText.text = "";
                    timer =
StartCoroutine(TextAnimation(text_memory.Split("\n")[mem-
1].Substring(3) + "\n закрыть [z]"));
                    isactive = false;
                }
            }
        }
    }
}

```

```
        } else AIText.text = "...";
    }
}
public void AIReplyComplete()
{
    text_memory = text_memory + "\n_";
    isgenerate = true;
}
private IEnumerator TextAnimation(string Text)
{
    foreach (var letter in Text)
    {
        AIText.text += letter;
        Text = letter.ToString();
        yield return new WaitForSeconds(0.05f);
    }
    num = 2;
}
}
```

## ПРИЛОЖЕНИЕ Б

### Код скрипта мини-игры

Таблица Б.1 – Код StumpMinigame.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

// Минигра с пнём во второй локации
public class StumpMinigame : MonoBehaviour
{
    [System.Serializable] // Чтобы видеть в инспекторе
    // Информация о предметах в виде класса
    public class Item
    {
        public GameObject gameObject; // Объект
        public string name; // Имя
        public bool isHeld = false; // Взято ли в руку
        public Vector3 originalPosition; // Где лежит в начале
        (есть лево, центр и право)
        // Возможные позиции - лево, центр, право
        public Vector3 leftPosition; // Лево
        public Vector3 centerPosition; // Центр
        public Vector3 rightPosition; // Право
        public int currentPosition; // Текущая позиция. 1 - лево, 2 -
        центр, 3 - право
    }

    [Header("Настройки предметов")]
    public Item[] items; // 0: Пень, 1: Бант, 2: Ножницы, 3: Книга
    public int currentSelectedIndex = 1; // Текущий выбранный предмет.
    Будет активен при начале минигры

    [Header("Настройки позиций")]
    public Transform stumpTransform; // Позиция пня
    //public Transform[] placementPositions; // Позиции (0, 1, 2)
    для размещения предметов на пне

    [Header("Настройки UI")]
    public GameObject messagePanel; // Панелька для сообщений
    public GameObject hintPanel; // Подсказка по управлению

    [Header("Правильный порядок")]
    public string[] correctOrder = { "Ножницы", "Бант", "Книга" }; //
    Нужный порядок

    private Camera mainCamera; // Для запоминания
    камеры
    private Vector3 cameraOriginalPosition; // Позиция камеры до
```

```

игры
    private Quaternion cameraOriginalRotation; // Вращение камеры до
игры
    private bool isMinigameActive = false; // Текущее состояние
игры
    private Item heldItem = null; // Какой предмет
сейчас в руке
    private bool alreadyCompleted = false; // Для отрицания
повторного прохождения

    void Start() // При появлении в сцене
    {
        mainCamera = Camera.main;

        // Деактивация панелей с текстом до их появления
        messagePanel.SetActive(false);
        hintPanel.SetActive(false);

    }

    // Начинаем игру
    public void StartMinigame()
    {
        // Не даём пройти игру многократно
        if (alreadyCompleted) return;

        isMinigameActive = true;

        // Фиксируем камеру
        cameraOriginalPosition = mainCamera.transform.position;
        cameraOriginalRotation = mainCamera.transform.rotation;
        MouseCamera.moving = false;

        // Позиционируем камеру над пнем
        Vector3 cameraPosition = stumpTransform.position + new
Vector3(0, 5f, -5f);
        mainCamera.transform.position = cameraPosition;
        mainCamera.transform.LookAt(stumpTransform.position + new
Vector3(0, 1.4f, 0));

        StartCoroutine(EndGameAfterDelay(15f)); // Прерываем игру
через 15 секунд

        // Показываем сообщение, потом убираем
        ShowStartMessage();

        // Выбираем первый предмет
        SelectItem(1);
    }

    void ShowStartMessage()

```

```

{
    messagePanel.SetActive(true);
    // Активируем подсказку
    hintPanel.SetActive(true);

    // Скрываем сообщение через 8 секунд
    StartCoroutine(HideMessageAfterDelay(8f));
}

IEnumerator HideMessageAfterDelay(float delay)
{
    yield return new WaitForSeconds(delay);
    messagePanel.SetActive(false);
}

IEnumerator EndGameAfterDelay(float delay)
{
    yield return new WaitForSeconds(delay);
    ExitMinigame();
}

void SelectItem(int index)
{
    if (heldItem != null && FindItemAtPosition(index) == heldItem)
        return;
    currentSelectedIndex = index;
}

// Каждый кадр
void Update()
{
    if (!isMinigameActive) return;

    HandleInput(); // Обрабатываем ввод
}

// /////////////
// Обработка нажатия кнопок
void HandleInput()
{
    // Выход из игры
    if (Input.GetKeyDown(KeyCode.X))
    {
        ExitMinigame();
        return;
    }

    // Взять/отпустить предмет
    if (Input.GetKeyDown(KeyCode.Z))
    {
}

```

```

        if (heldItem == null)
        {
            // Берем предмет в руку
            PickUpItem(FindItemAtPosition(currentSelectedIndex));
        }
        else
        {
            // Отпускаем предмет
            ReleaseItem();
        }
    }

    if (heldItem != null) // Если уже держим что-то
    {
        // Меняем местами с предметом слева на A
        if (Input.GetKeyDown(KeyCode.A) ||
Input.GetKeyDown(KeyCode.LeftArrow))
        {
            SwapWithNeighbor(-1); // -1 = влево
        }
        // Меняем местами с предметом справа на D
        else if (Input.GetKeyDown(KeyCode.D) ||
Input.GetKeyDown(KeyCode.RightArrow))
        {
            SwapWithNeighbor(1); // 1 = вправо
        }
    }
    else
    {
        // Если предмет не в руке - просто выбираем другой
        if (Input.GetKeyDown(KeyCode.A) ||
Input.GetKeyDown(KeyCode.LeftArrow))
        {
            SelectPreviousItem();
        }
        else if (Input.GetKeyDown(KeyCode.D) ||
Input.GetKeyDown(KeyCode.RightArrow))
        {
            SelectNextItem();
        }
    }
}

void ExitMinigame()
{
    if (!isMinigameActive) return;

    isMinigameActive = false;

    // Возвращаем камеру
    mainCamera.transform.position = cameraOriginalPosition;
}

```

```

        mainCamera.transform.rotation = cameraOriginalRotation;
        MouseCamera.moving = true;

        // Скрываем UI
        hintPanel.SetActive(false);
        if (heldItem != null) ReleaseItem();

        alreadyCompleted = true; // Чтобы не запускать игру вновь
        Debug.Log("Мини-игра завершена");
    }

    // Смещаем выбор влево, по кругу
    void SelectPreviousItem()
    {
        int newIndex = currentSelectedIndex - 1;
        if (newIndex < 1) newIndex = items.Length - 1; // Пень с
        индексом 0 пропускается
        SelectItem(newIndex);
    }

    // Смещаем выбор вправо, по кругу
    void SelectNextItem()
    {
        int newIndex = currentSelectedIndex + 1;
        if (newIndex >= items.Length) newIndex = 1;
        SelectItem(newIndex);
    }

    // Подбор предмета
    void PickUpItem(Item item)
    {
        if (item == items[0] || heldItem != null) return;
        heldItem = item;
        item.isHeld = true;
        item.gameObject.transform.position += new Vector3(0, 0.5f, 0);
    // Приподнимем его
    }

    // Отпускание предмета
    void ReleaseItem()
    {
        if (heldItem == null) return;
        heldItem.isHeld = false;

        if (heldItem.currentPosition == 0)
        {
            heldItem.gameObject.transform.position =
heldItem.originalPosition;
        }
        else
    }

```

```

    {
        PlaceItemOnPosition(heldItem, heldItem.currentPosition);
    // Автоматически опустится
    }
    heldItem = null;
    CheckSolution();
}

// Поместить предмет на заготовленную позицию
void PlaceItemOnPosition(Item item, int position)
{
    Vector3 targetPosition;
    switch (position)
    {
        case 1: targetPosition = item.leftPosition; break;
        case 2: targetPosition = item.centerPosition; break;
        case 3: targetPosition = item.rightPosition; break;
        default: return;
    }
    if (item.isHeld) targetPosition += new Vector3(0, 0.5f, 0);
    item.currentPosition = position;
    item.gameObject.transform.position = targetPosition;
}

// Обмен двух соседей местами
void SwapWithNeighbor(int direction)
{
    if (heldItem == null) return;

    int heldPositionIndex = heldItem.currentPosition;
    if (heldPositionIndex == 0)
    {Debug.Log("Ошибка с положением предмета");
    return;} // Не на пне, технически не должно такого быть

    int neighborIndex = heldPositionIndex + direction;

    if (neighborIndex < 1) neighborIndex = items.Length - 1; // Идём в конце списка
    if (neighborIndex > 3) neighborIndex = 1; // Идём в начало списка

    Item neighborItem = FindItemAtPosition(neighborIndex);
    if (neighborItem == null) return;

    SwapItems(heldItem, neighborItem);
    currentSelectedIndex = neighborIndex;
}

// Получает предмет по его позиции в числе
Item FindItemAtPosition(int position)
{

```

```

        foreach (Item item in items)
        {
            if (item.currentPosition == position) //&& item != heldItem)
                return item;
        }
        return null;
    }

    // Меняет двух соседей местами
    void SwapItems(Item item1, Item item2)
    {
        int tempPos = item1.currentPosition; // Временное хранилище
позиции
        item1.currentPosition = item2.currentPosition;
        item2.currentPosition = tempPos;

        PlaceItemOnPosition(item1, item1.currentPosition);
        PlaceItemOnPosition(item2, item2.currentPosition);
    }

    void CheckSolution()
    {
        // Собираем предметы на пне в порядке позиций
        string[] currentOrder = new string[3];

        for (int i = 1; i <= 3; i++) // Позиции 1,2,3
        {
            Item item = FindItemAtPosition(i);
            currentOrder[i-1] = item.name;
        }

        // Проверяем совпадение
        bool isCorrect = true;
        for (int i = 0; i < 3; i++)
        {
            if (currentOrder[i] != correctOrder[i])
            {
                isCorrect = false;
                break;
            }
        }

        if (isCorrect)
        {
            add_score.memory += 3;
            Debug.Log("Поздравляем! Правильный порядок!");
            ExitMinigame();
        }
    }
}

```

```
// Метод для активации мини-игры извне
public void ActivateMinigame()
{
    StartMinigame();
}
```

Таблица Б.2 – Код Collide.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Collide : MonoBehaviour
{
    public StumpMinigame minigame;

    void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            minigame.ActivateMinigame();
        }
    }
}
```