МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет Информационных технологий
кафедра Информатики и информационных технологий

направление подготовки 09.03.02 «Информационные системы и технологии»

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

БАКАЛАВРСКАЯ РАБОТА

Тема: Разработка программного обеспечения игрового проекта «ЛАВЭ»

Исполнитель Курносова А.В. _____
(Фамилия И.О.) (Подпись)

Руководитель Семенов С.М., к.т.н., ст. преподаватель _____
(Фамилия И.О., степень, звание) (Подпись)

Нормоконтроль Семенов С.М., к.т.н., ст. преподаватель _____
(Фамилия И.О., степень, звание) (Подпись)

Антиплагиат Демидова А.М., ст. преподаватель _____
(Фамилия И.О., степень, звание) (Подпись)

«ДОПУЩЕНО К ЗАЩИТЕ»

Зав. кафедрой ИиИТ: к.т.н.; доц. Е.В. Булатников _____
(Подпись)

Москва

2024

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет Информационных технологий
кафедра Информатики и информационных технологий

направление подготовки 09.03.02 «Информационные системы и технологии»,

УТВЕРЖДАЮ
Зав. каф., к.т.н.; доц. Е.В. Булатников
« ___ » _____ 2024 г. _____
(Подпись)

ЗАДАНИЕ НА БАКАЛАВРСКУЮ РАБОТУ

Студент Курносова Арсения Валерьевна **группа 201-726**
(Фамилия, Имя, Отчество)

Тема Разработка программного обеспечения игрового проекта «ЛАВЭ»

Утверждена приказом по Университету № 6351-УД

1. **Срок представления работы к защите** « 15 » июня 2024 г.
2. **Исходные данные для выполнения работы:**

Техническое задание, техническая литература по теме ВКР

3. **Содержание бакалаврской работы:**

Введение, аналитическая часть, теоретическая часть, практическая часть, тестирование, заключение, список литературы, приложения

4. **Перечень графического материала** _____
5. **Консультанты по разделам:**

_____
наименование раздела (подпись, Фамилия И.О., ученая степень, звание)

_____
наименование раздела (подпись, Фамилия И.О., ученая степень, звание)

6. **Дата выдачи задания:** 28 октября 2023 г.
7. **Руководитель:** _____ / Семенов С.М. /
(Подпись)
8. **Задание к исполнению принял:** _____ / Курносова А.В. /
(Подпись)

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет Информационных технологий
кафедра Информатики и информационных технологий

направление подготовки 09.03.02 «Информационные системы и технологии»,

КАЛЕНДАРНЫЙ ПЛАН

Студент _Курносова Арсения Валерьевна_____ **группа 201-726_**
(Фамилия, Имя, Отчество)

Тема ВКР: Разработка программного обеспечения игрового проекта «ЛАВЭ»
_____

№ п/п	Наименование этапа ВКР	Срок выполнения этапа	Примечание
1.	Получение задания от компании	04.09.2023	Выполнено
2.	Изучение предметной области	16.09.2023	Выполнено
3.	Анализ предметной области	26.09.2023	Выполнено
4.	Проектирование информационной системы	24.10.2023	Выполнено
5.	Согласование проекта	12.11.2023	Выполнено
6.	Разработка основных информационных систем	04.02.2024	Выполнено
7.	Разработка демонстрационной модели – игрового проекта	06.02.2024	Выполнено
8.	Разработка второстепенных информационных систем	14.04.2024	Выполнено
9.	Тестирование	20.04.2024	Выполнено
10.	Подготовка документации	28.04.2024	Выполнено

Руководитель_Семенов С.М._____
(Фамилия И.О., степень, звание) **(Подпись)**

Зав. каф. ИиИТ _____ /Е.В. Булатников/
(Подпись)

« ___ » _____ 2024 г.

Аннотация

В данной выпускной квалификационной работе рассматривается создание программного обеспечения для игрового проекта. Ключевой особенностью разработанного программного обеспечения является высокий уровень адаптивности под различные игровые проекты и низкий уровень зависимости от мультимедийной оболочки игры.

Структура работы представлена введением, четырьмя главами, заключением, списком литературы и приложениями.

Первая глава посвящена изучению предметной области, согласованию технических требований и системному анализу.

Во второй главе рассматривается проектирование информационной системы и отдельных её частей, рассмотрение путей создания наиболее адаптивной информационной системы.

В третьей главе описывается процесс разработки информационной системы и её эксплуатации.

В четвёртой главе производится тестирование разработанных информационных систем на примере внедрения их в игровой программный продукт.

В заключении структурируются итоги проведённой работы.

Abstract

This final qualifying work examines the creation of software for a game project. The key feature of the developed software is a high level of adaptability for various game projects and a low level of dependence on the multimedia shell of the game.

The structure of the work is presented by an introduction, four chapters, a conclusion, a list of references and appendices.

The first chapter is devoted to the study of the subject area, coordination of technical requirements and system analysis.

The second chapter examines the design of an information system and its individual parts, considering ways to create the most adaptive information system.

The third chapter describes the process of developing an information system and its operation.

The fourth chapter tests the developed information systems using the example of their implementation in a gaming software product.

In conclusion, the results of the work done are structured.

Реферат

Объём пояснительной записки составляет 122 страницы, 4 приложения, 13 рисунков-схем, 30 листингов. При написании диплома использовалось 14 источников.

Тема: Разработка программного обеспечения игрового проекта «ЛАВЭ».

Ключевые слова: информационная система, C#, диалоговое дерево, вариативная диалоговая система, система квестов, система событий, адаптивность.

Оглавление

Введение 8
ГЛАВА 1. Аналитическая часть 10
1.1 Предметная область 10
1.1.1 Определение игры 10
1.1.2 Геймификация 11
1.1.3 Программирование 12
1.1.4 Игровые движки 13
1.2 Анализ технических требований 15
1.2.1 Анализ применимого игрового движка 16
1.2.2 Анализ поставленных задач 17
1.3 Вывод 18
ГЛАВА 2. Проектирование информационной системы 19
2.1 Выбор методов проектирования основных игровых систем 19
2.1.1 Разделение основных информационных систем 19
2.1.2 Диалоговая система 21
2.1.3 Теоретическое решение задачи по созданию диалоговой системы 23
2.1.4 Система квестов 26
2.1.5 Теоретическое решение задачи по созданию системы квестов 28
2.1.6 Теоретическое решение задачи объединения диалоговой системы и системы квестов 29
2.2 Проектирование второстепенных игровых систем 32
2.2.1 Система управления игровым персонажем 32
2.2.2 Система переходов между игровыми сценами 33
2.2.3 Система сохранения положения игрового персонажа при переходах между сценами 35
2.3 Проектирование систем управления игровым приложением 37
2.3.1 Система главного меню 38
2.3.2 Система внутриигрового меню 38
2.4 Итоги проектирования 39
ГЛАВА 3. Разработка информационной системы 41
3.1 Разработка структур диалоговой информационной системы 41

3.1.1 Разработка общей структуры диалоговой системы 41
3.1.2 Разработка системы хранения информации в диалоговой системе 43
3.1.3 Разработка схемы внедрения диалоговой системы в среду разработки Unity 48
3.2 Программная реализация диалоговой системы 56
3.2.1 Программная реализация системы хранения информации 56
3.2.2 Программная реализация смены диалоговых ветвей и системы событий 60
3.2.3 Программная реализация системы кнопок для смены диалоговой ветви 61
3.2.4 Программная реализация системы тегов в игровом пространстве 64
3.2.5 Программная реализация вывода текстовой информации 71
3.3 Программная реализация системы квестов 73
3.3.1 Программная реализация глобальной системы событий 73
3.3.2 Программная реализация ключей системы событий 76
3.3.3 Программная реализация общей системы событий 78
3.4 Программная реализация второстепенных информационных систем 80
3.4.1 Программная реализация систем управления 80
3.4.2 Программная реализация систем меню 87
3.4.3 Программная реализация внутриигровых информационных систем 94
3.4.4 Программная реализация систем геймплея 96
3.5 Итоги разработки 99
ГЛАВА 4. Тестирование информационных систем 101
4.1 Внедрение информационных систем 101
4.1.1 Внедрение диалоговой информационной системы 101
4.1.2 Внедрение системы квестов 114
4.2 Тестирование информационных систем 118
Заключение 120
Список литературы 121
Приложение А. Техническое задание 123
Приложение Б. Презентация проекта 126
Приложение В. Техническое задание компании-заказчика 133

Введение

В настоящее время большую популярность набирает геймификация образовательных и рабочих процессов. Под понятием «геймификации» подразумевается добавление развлекательных элементов в тот или иной процесс.

В настоящий момент, многие образовательные или рабочие процессы, подразумевают использование электронных ресурсов: сайтов, программ, приложений и т.д. Исходя из этого, когда речь заходит о геймификации образовательного или рабочего процессов, зачастую, подразумевается создание игровых программ или приложений, в геймплей которых встроены необходимые для освоения материалы.

Исследования доказывают, что геймификация образовательного или рабочего процессов приводят не только к проявлению большего интереса к самому материалу, но и к более быстрому его усвоению. Поэтому многие компании принимают решение о необходимости геймификации материалов для обучающихся или сотрудников.

Геймификация ряда образовательных или рабочих материалов имеет общий характер. Это значит, что техническая часть многих геймифицированных материалов имеет схожий принцип работы, независимо от различного визуального наполнения. Так возникает задача по созданию адаптивного «конструктора» - программного обеспечения для геймификации рабочих и образовательных материалов, имеющего минимальную привязку к мультимедийной составляющей. Решение этой задачи позволяет компании геймифицировать множество материалов с использованием одной адаптивной информационной системы.

Целью работы является разработка адаптивного программного обеспечения, предназначенного для геймификации образовательных материалов. В дальнейшем, предполагается использование разработанного программного обеспечения другими разработчиками, что обуславливает

необходимость создания наиболее низкого порога вхождения для применения систем.

Для демонстрации работоспособности информационной системы, разработана демонстрационная модель – игра, в основе которой лежит разработанное программное обеспечение.

Разработанная игра выступает как самостоятельный продукт, имеет уникальный сюжет и мультимедийное наполнение. Для создания разнообразного геймплея, разработаны дополнительное, второстепенные информационные системы. Второстепенные информационные системы не предполагают дальнейшего использования другими разработчиками и выступают в качестве элементов программного обеспечения уникального игрового приложения.

Результатом работы являются пакет информационных систем, предназначенный для геймификации прикладных образовательных материалов, и самостоятельное игровое приложение, имеющее уникальный сюжет, мультимедийное и геймплейное наполнение.

ГЛАВА 1. Аналитическая часть

1.1 Предметная область

Разработка информационной системы начинается с анализа применяемых навыков и технологий. Для разработки программного обеспечения игрового проекта необходимо применение навыков программирования, работы с игровыми движками, а также первоначальное понимание определения «игры».

Поскольку в ходе разработки также затрагиваются и вопросы геймификации прикладных образовательных материалов, аналитика также включает и их рассмотрение.

1.1.1 Определение игры

Игра (компьютерная игра) – программа, служащая для организации игрового процесса, досуга.

Внутри игрового приложения, игровой процесс так же может называться геймплеем. Геймплей представляет из себя набор функций игровой программы, при помощи которых организуется сам игровой процесс.

Во время игры может происходить взаимодействие с игровым партнёром (в таком случае, игра называется многопользовательской), в качестве игрового партнёра может выступать сама игровая программа (тогда игра является однопользовательской).

В настоящее время, в качестве синонима «компьютерной игре» употребляется термин «видеоигра». Эти понятия могут использоваться как синонимичные, поскольку в большинстве случаев, отображение взаимодействия с игровым миром происходит при помощи вывода изображения на экран монитора или телевизора. Но в более широком смысле, компьютерные игры могут быть телетайповыми, звуковыми и другими.

Компьютерные игры могут разрабатываться на основе каких-либо фильмов, литературных произведений. В последнее время активно наблюдается и обратная тенденция, когда оригинальная игра служит основой для создания

фильма или написания произведения. Ряд игр могут содержать в себе отсылки на культурные явления, отражать тенденции общества. Особо популярные и крупные проекты могут даже порождать новые культурные течения. С 2011 года компьютерные игры признаны в США отдельным видом искусства.

Компьютерные игры имеют множество различных жанров. Жанры могут быть разделены на три основные категории:

- Динамические игры – игры, требующие от игрока быстрой реакции, но, зачастую, не заставляющие серьёзно размышлять над игровыми действиями;
- Игры планирования – игры, направленные на логику и стратегию, требующие от игрока вдумчивых, взвешенных игровых решений;
- Сюжетные игры – игры, в основе которых лежит сюжет, заложенный разработчиком. Такие игры являются полноценной историей и могут восприниматься как «интерактивная книга».

Исходя из своих запросов, потребностей и желаний, пользователь может самостоятельно выбрать наиболее подходящий ему жанр игр. Такое разнообразие жанров, позволяет не только организовывать различный досуг и удовлетворять запросам множества пользователей, но и наделять игровые программы разными качествами. Это, в свою очередь, позволяет не только создавать изначально образовательные игровые программы, но и геймифицировать неигровое прикладное программное обеспечение.

1.1.2 Геймификация

Геймификация – процесс внедрения игровых элементов в неигровой процесс.

В вопросе компьютерных игр геймификация подразумевает внесение игровых элементов в неигровое программное обеспечение.

Процесс геймификации осуществляется для привлечения новых пользователей и потребителей, а также для вовлечения пользователей в решение прикладных задач.

Помимо большего уровня вовлечённости пользователей в решение поставленной задачи, при геймификации неигрового процесса может наблюдаться нахождение более эффективных, творческих путей решения задачи или более высокий уровень усвоения информации.

При решении задач геймификации в цифровой среде, как правило, используется внедрение элементов компьютерных игр в неигровое программное обеспечение. При этом процессе, могут быть использованы элементы игр, свойственные различным жанрам.

Геймификация является одной из основных тенденций развития современного дистанционного электронного образования. С 2010-х годов геймификация считается одним из ключевых направлений информационных технологий для организаций в целом.

1.1.3 Программирование

Программирование – процесс создания компьютерных программ.

Программирование может быть разделено на различные группы, исходя из ряда особенностей, таких как: язык программирования, область применения программных продуктов, тип обрабатываемых конечным продуктом данных и т.д. В случае разработки программного обеспечения для игрового проекта, речь пойдёт о программировании для игровых платформ.

Разработка информационной системы для геймификации неигрового программного обеспечения во многом схожа с разработкой информационной системы для игровой программы. В некоторых случаях, эти процессы полностью идентичны (в том случае, если уровень геймификации прикладной программы достаточно высокий). В таком случае, игровая программа наполняется материалами, изначально находящимися в прикладном виде. Материалы могут

быть изменены по необходимости: добавлены в игровой сюжет или внедрены в решение геймплея, но при этом они сохраняют своё начальное образовательное значение.

Навыки программирования могут быть получены различными путями: при обучении в общеобразовательных учебных заведениях, обучении в высших учебных заведениях, прохождении специализированных курсов или самостоятельном обучении. Несмотря на различные источники получения навыков программирования, уровень самих навыков больше зависит от пройденной образовательной программы и степени её усвоения. Так, для оценки степени освоения навыков программирования, существует общая система «уровня» владения программированием. По этой системе существует несколько основных уровней:

→ Новичок (Junior);

→ Продвинутый (Middle);

→ Эксперт (Senior);

→ Технический лидер (Tech Lead);

→ Архитектор (Architect).

В большинстве случаев оценки уровня владения программированием, ключевыми уровнями являются первые три.

1.1.4 Игровые движки

Под «игровыми движками» обычно понимаются программы, служащие для более простой и комфортной (с точки зрения разработчика) разработки игр. Такие программы содержат множество готовых инструментов, применение которых упрощает процесс создания игрового приложения. Эти наборы инструментов, в большинстве случаев, решают проблему многократного повторения решения однотипных задач.

Множество различных игровых проектов содержат схожие по механике элементы. Так, например, управление персонажем схоже в большинстве игровых

приложений (если только игровое приложение не использует уникальных методов). Решение однотипных для огромного множества проектов задач, занимает много времени у разработчиков, но не привносит новых знаний в глобальном смысле. Наличие ряда инструментов в игровых движках позволяет применять наиболее эффективные способы решения таких задач, без повторной самостоятельной реализации со стороны разработчиков.

Среди наиболее популярных игровых движков – Unreal и Unity.

Каждый из этих игровых движков имеет свои преимущества и недостатки. Так, Unity считается более интуитивно понятным с точки зрения интерфейса и более простым с точки зрения написания кода. Но, в свою очередь, Unreal привлекает разработчиков способностью работы с более высококачественной графикой.

Отдельное внимание при сравнении игровых движков, стоит уделить вопросу времени обработки и генерации созданного игрового мира (рендеру). Под временем рендера понимается время, которое требуется движку для обработки полигонов моделей, загрузки и генерации элементов игрового окружения. Чем выше сложность моделей, тем сложнее считается рендер.

Помимо двух перечисленных движков, существуют так же и чуть менее популярные игровые движки, такие как Godot. Несмотря на меньшую известность, они также пользуются высоким спросом в среде разработчиков игровых проектов и приложений. Такие игровые движки могут иметь более узкую специфику и предназначаться для разработки игр определённого жанра или только под определённые платформы. Но при этом, конечные продукты, разработанные на таких движках, не редко не уступают по качеству играм, разработанных на более широко используемых площадках.

Каждый игровой движок имеет свои особенности, которые формируют его приспособленность решать те или иные задачи. В различных игровых движках существуют не только разные наборы инструментов, но и разные системы формирования взаимодействий между элементами разрабатываемого продукта. Так, к примеру, могут использоваться различные системы взаимодействий

скриптов внутри программы: от более стандартных (как, например, в Unity), когда порядок взаимодействия скриптов прописывается внутри кода; до более редких (как, например, в Godot), когда взаимодействие скриптов устанавливается при помощи построения дерева элементов.

1.2 Анализ технических требований

Основой для реализации проекта являются технические требования, предоставленные компаний. Более подробно технические требования описаны в разделе «Техническое задание».

Резюмируя технические требования к проекту, можно выделить следующие пункты:

- → Информационная система должна быть разработана для двумерного игрового пространства;
- → Подразумевается использование информационной системы для двумерных игровых проектов в жанре «квест»;
- → Необходима наиболее доступная степень адаптивности программной части информационной системы под различные графические и текстовые содержания;
- → Необходима разработка системы, позволяющей реализовать диалоги с вариантами ответов в игровом приложении;
- → Необходима разработка информационной системы, позволяющей реализовать механику получения и выполнения заданий в игровом приложении в жанре «квест».

Исходя из этих требований, необходим анализ, выбор и согласование используемого игрового движка, а также теоретический анализ информационной системы.

1.2.1 Анализ применимого игрового движка

При выборе игрового движка, применимого для разработки информационной системы, опорой будет служить краткий обзор игровых движков, проведённый в анализе предметной области.

Выбор игрового движка осуществляется из двух наиболее популярных игровых движков для создания игровых проектов на персональные компьютеры – Unity и Unreal. Использование других игровых движков является менее предпочтительным, поскольку дальнейшая эксплуатация разработанной информационной системы предполагает владение пользователем начальными навыками разработки на актуальном для информационной системы игровом движке. Выбор непопулярных, более сложных для освоения игровых движков может привести к затруднениям при эксплуатации пользователем информационной системы. Также, более популярные игровые движки, имеют большее количество информации в сети Интернет об их эксплуатации. Их использование позволит не только снизить порог вхождения для пользователя-разработчика, но и «увеличить» количество доступной информации по теме, в случае необходимости обращения к сторонним источникам.

Важную роль при выборе движка имеет предполагаемая сложность рендеринга. Поскольку информационная система предназначена для использования в двумерных игровых проектах (исходя из технического задания), при эксплуатации игрового движка не предполагается наличие сложных для рендеринга графических элементов. При этом, использование движка, разработанного под более сложный рендер, может привести к излишней массивности итогового продукта и дополнительно затруднить процесс разработки информационной системы. В таком случае страдает оптимизация итогового продукта и повышается порог вхождения для пользователя-разработчика.

Исходя из этого, в качестве основного игрового движка был выдвинут движок Unity.

1.2.2 Анализ поставленных задач

Игровое приложение или приложение, являющееся продуктом геймификации прикладных образовательных материалов, представляется единой информационной системой, объединяющей в себе ряд подсистем.

Исходя из задания на разработку, конечный продукт (информационная система) может быть разбит на две основные системы: вариативную диалоговую систему и систему квестов (систему событий).

Вариативная диалоговая система – система, позволяющая реализовать систему диалогов с вариантами ответов. Основным типом данных, хранимых в такой информационной системе, является текст. Часть хранимого в системе текста представляет собой «диалог» – основной массив передаваемой информации; другая часть – варианты ответов, которые пользователь может выбрать из перечня. В зависимости от выбранного пользователем ответа, диалоговая система может или изменять своё состояние, или передавать информацию во внешние, глобальные системы игрового приложения.

Система квестов (система событий) – система, позволяющая реализовать функции получения и выполнения пользователем заданий, в рамках игрового окружения. При получении пользователем задания, система квестов принимает состояние ожидания выполнения задания – ожидание получения определённой информации от внешних систем. При выполнении пользователем задания, изменяется состояние системы квестов, и поставленная задача снимается. При выполнении задания, происходит отправка сигнала во внешние системы. Задания могут формулироваться как прямым, для игрока, образом, так и скрытым. При прямой постановке задачи, пользователь получает текст задания и/или указания к определённым действиям во внутриигровом пространстве. При скрытой постановке, задание не формулируется для пользователя, а его получение и выполнение является следствием совершения действий в игровом пространстве, как правило, связанных с исследованием игрового мира и сюжета.

1.3 Вывод

Осуществлён обзор предметной области работы. Даны определения понятиям «игры», «геймификации», «программирования», «игровые движки».

Был осуществлён анализ поставленной задачи и технических требований. Также, был проведён обзор инструментов, применимых для решения поставленной задачи.

Среди инструментов, отдельное внимание было уделено выбору игрового движка. В качестве наиболее подходящего для решения поставленных задач, был выбран игровой движок Unity.

При анализе поставленных задач, дано определение вариативной диалоговой системе и системе квестов (системе событий). Рассмотрены основные функции, выполняемые разрабатываемыми системами. Обозначено значение разрабатываемых функций в общей системе игрового приложения.

ГЛАВА 2. Проектирование информационной системы

2.1 Выбор методов проектирования основных игровых систем

Информационные системы в игровом проекте могут быть разделены на основные и второстепенные. Основные системы выполняют функции, лежащие в основе геймплея и определяют основные игровые механики. Для данного проекта основными являются диалоговая система и система квестов. Проектирование остальных систем необходимо для реализации полноценного, завершённого игрового приложения и их проектирование рассматривается отдельно.

Перед началом проектирования информационной системы, необходим более детальный теоретический анализ методов решения поставленной задачи. При первичном обзоре методов решения, проведённом в аналитической части работы, были обозначены доступные пути. Также, были выбраны наиболее предпочтительные пути решения задачи, на основе теоретического анализа.

При проектировании информационных систем, устанавливаются наиболее выгодные стратегии построения информационных структур. Выделяются основные функции каждой информационной системы. Для каждой из функций устанавливаются отдельные компоненты разработки, которые в совокупности формируют единую информационную систему.

2.1.1 Разделение основных информационных систем

При создании общей информационной системы, применимой для разработки двумерного игрового проекта, необходимо определить ключевые задачи информационной системы и разбить общую информационную систему на функциональные сегменты.

Информационная система содержит два основных типа функций: установление взаимодействия между пользователем-игроком и информационной системой при помощи текстовой информации и при помощи системы внутриигровых задач.

Под функциями взаимодействия с пользователем-игроком при помощи текстовой информации понимается создание диалоговой системы для игрового приложения. Диалоговая система позволяет пользователю-игроку получать отклик от игрового приложения, взаимодействовать с игровым миром, а также выполняет образовательные функции в рамках геймификации прикладного образовательного материала. При помощи текстов, содержащихся в информационной системе, игрок получает необходимые теоретические знания или взаимодействует с программой в игровых целях.

Под функциями установления взаимодействия между пользователем-игроком и информационной системой при помощи системы игровых заданий, понимается система квестов. Система квестов представляет из себя линейную систему взаимодействия игрока с игровым миром. При прохождении игроком определённого игрового этапа, информационная система принимает состояние «изменения», после чего игроку становится доступен следующий игровой этап, а информационная система принимает следующее положение. Изменение состояния информационной системы может быть достигнуто или прохождением определённой зоны в игровом пространстве, или выполнением определённых действий игроком. При этом действия игрока могут носить разнообразный характер – выполнение внутриигровых заданий, прохождение определённой диалоговой ветви и т.д.

Поскольку основные информационные системы образуют собой единую структуру, состоящую из двух самостоятельных компонентов, к основным системам также может быть добавлена и система взаимосвязи диалоговой информационной системы и информационной системы квестов. Система взаимосвязи двух систем может быть определена как обязательный компонент системы квестов, так как система квестов строится на принципе отслеживания состояний как внутренних, так и внешних компонентов, а в число внешних компонентов также входит и диалоговая система.

2.1.2 Диалоговая система

Основным требованием к диалоговой системе, как и ко всем основным информационным системам, является её адаптивность под различные двумерные игровые проекты.

Задачами диалоговой системы является отображение текстовой информации в заданном порядке, переход между различными диалоговыми узлами при изменении состояния диалоговой системы и отслеживание ключевых элементов.

Вывод текстовой информации – процесс в диалоговой системе, при котором пользователь получает выведенный на экран текст, содержащийся в информационной системе. Предполагается отображение текстовой информации в выделенных для этих задач элементах интерфейса. Под выделенными элементами интерфейса понимаются установленные текстовые фреймы и шаблоны отображения.

Отображение текста в игровом пространстве происходит при определённом взаимодействии игрока с игровым пространством. Так, вывод текста может происходить при прохождении игроком участка, содержащего триггер – участок игрового пространства, реагирующий на определённые изменения игрового окружения.

Вывод текста также может происходить и при взаимодействии игрока с активными элементами игры. При этом, для получения отображения текста, игроку необходимо взаимодействовать с активным элементом игрового пространства.

Порядок отображения текста задаётся исходя из смысловой нагрузки текста и игрового сюжета. Отображаемый текст, как правило, соотносится с настоящим состоянием игрового окружения. В тексте могут содержаться описательные элементы, позволяющие игроку ознакомиться с игровым миром и получить дополнительную информацию о нём. Так же, в тексте могут содержаться и указания к дальнейшим игровым действиям – задания для игрока.

Последнее обуславливает взаимосвязь между диалоговой системой и системой квестов.

В диалоговой системе предусматривается наличие элементов, позволяющих изменять состояние информационной системы. Одному интерактивному объекту может соответствовать несколько узлов диалоговой системы. Под узлом системы понимается компонент, содержащий текст и варианты ответа, соответствующие тексту. При изменении состояния информационной системы, происходит смена актуального узла, соответствующего интерактивному объекту в игровом пространстве.

Также, к некоторым узлам диалоговой системы, могут быть привязаны игровые события. Если игровое событие привязано к узлу диалоговой системы, оно становится актуальным в момент прохождения данного узла. Это обуславливает необходимость установления взаимосвязи диалоговой системы не только с системой квестов, но и с остальными игровыми системами, которые могут быть затронуты системой событий.

В рамках разработки информационной системы требуется разработка вариативной диалоговой системы. Под вариативностью понимается возможность игрока влиять на ход диалога. С точки зрения технической реализации, такая система подразумевает влияние действий игрока на выводимую в дальнейшем текстовую информацию.

Вариативность диалоговой системы представляется в виде различных вариантов ответов, доступных игроку при прохождении диалога. Это реализуется путём создания соответствующего пользовательского интерфейса. Интерфейс содержит окно с основным текстом и набор кнопок с вариантами ответов.

Также, в разрабатываемой системе, предполагается возможность вариативного исследования пользователем-игроком игрового пространства. Это значит, что каждый интерактивный объект в игровом окружении, по умолчанию

имеет несколько привязанных к нему диалоговых ветвей, каждая из которых отвечает за свой способ исследования объекта.

В рамках проекта предполагается наличие двух способов исследования игрового окружения – «осмотр» и «диалог». С точки зрения игрового процесса, осмотр объекта предполагает использование диалоговой ветви, в которой хранится информация, связанная с визуальным описанием исследуемого объекта. Диалог же предполагает открытие ветви, в которой хранится информация с разговором с данным объектом или попыткой данного разговора, если как таковой диалог не предусмотрен логически.

2.1.3 Теоретическое решение задачи по созданию диалоговой системы

Для создания вариативной диалоговой системы задействуется информационная структура дерева. Структура дерева позволяет сделать разветвлённую систему. Дерево состоит из структурных компонентов. Структурные компоненты имеют одинаковый шаблон, но содержат в себе различную информацию. Каждый предстоящий структурный элемент является родительским для последующего. Примитивный принцип создания диалогового дерева возможно рассмотреть на рисунке 2.1.

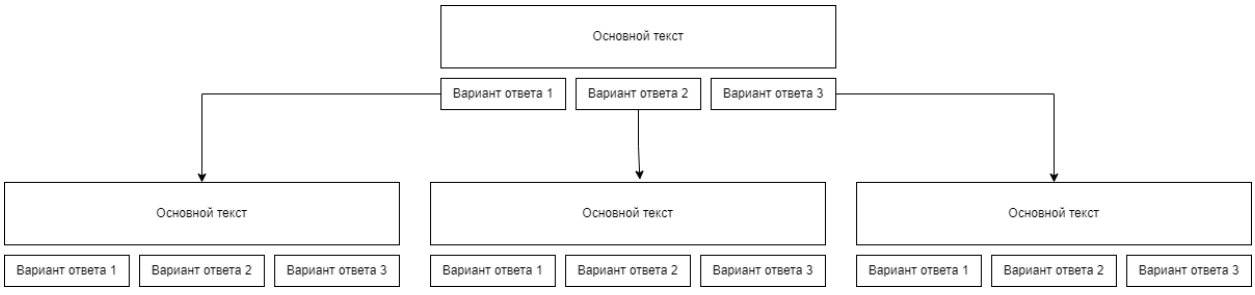

Рисунок 2.1 – Примитивное диалоговое дерево

В качестве структурного элемента выступает элемент интерфейса, представленный в виде совокупности нескольких элементов – основного текстового фрейма и вариантов ответов. Основной текстовый фрейм содержит основной текст диалога или описания. Варианты ответов содержат текст,

который по смысловой нагрузке соотносится с текстом из основного текстового фрейма.

При рассмотрении одной ветви такой системы следует:

→ Вход в диалоговую систему и начало взаимодействия с ней возможно только при обращении к одному из родительских элементов. При этом, данный родительский элемент должен являться родительским для всей выделенной ветви, так как текст, содержащийся в информационной системе, имеет смысловую нагрузку;

→ Выход из диалоговой системы и окончание взаимодействия с ней возможно только при переходе к конечному узлу выбранной ветви.

Также, отдельное внимание стоит уделить методам перехода от одного узла диалогового дерева, к другому.

Переход между узлами происходит при выборе одного из вариантов ответов. При этом, не исключается соответствие разных вариантов ответов одному исходу. Одному основному текстовому фрейму может принадлежать неограниченное количество вариантов ответов. В приведённом на рисунке 2.1. примере, количество вариантов ответов для каждого основного текстового фрейма соответствует трём, но это является частным случаем.

При создании адаптивной диалоговой системы, необходимо учитывать возможность соответствия разным основным текстовым фреймам разного количества вариантов ответов, даже в рамках одной информационной системы. Так, например, текстовый фрейм, содержащий три варианта ответа, при выборе одного из вариантов, может приводить к переходу к текстовому фрейму, содержащий один вариант ответа.

При такой вариативности системы как глобально – в способах эксплуатации, так и в частных случаях – внутреннем устройстве одной, выделенной системы, необходимо определение основного, универсального принципа сообщения структурных компонентов внутри системы.

Наиболее оптимальным способом сообщения компонентов является присвоение каждому структурному компоненту уникального тега. По тегу может происходить обращение к узлу как внутри диалоговой системы, так и внутри общей игровой системы. Такая система позволяет избежать установления жёстких соединений между различными структурными компонентами и сохранить вариативность как глобально, так и в частных случаях.

На этом этапе проектирования информационной системы, стоит совершить разделение компонентов системы на «элементы диалога» и «элементы переадресации». Под элементами диалога понимаются основные текстовые фреймы, содержащие основной текст и имеющие привязку к набору вариантов ответов. Под элементами переадресации понимаются сами варианты ответов.

Стоит более детально рассмотреть принцип работы диалогового дерева. Для этого будет выделена одна ветвь. На выделенной ветви диалогового дерева происходит последовательный переход между структурными элементами. Принцип работы представлен на рисунке 2.2.

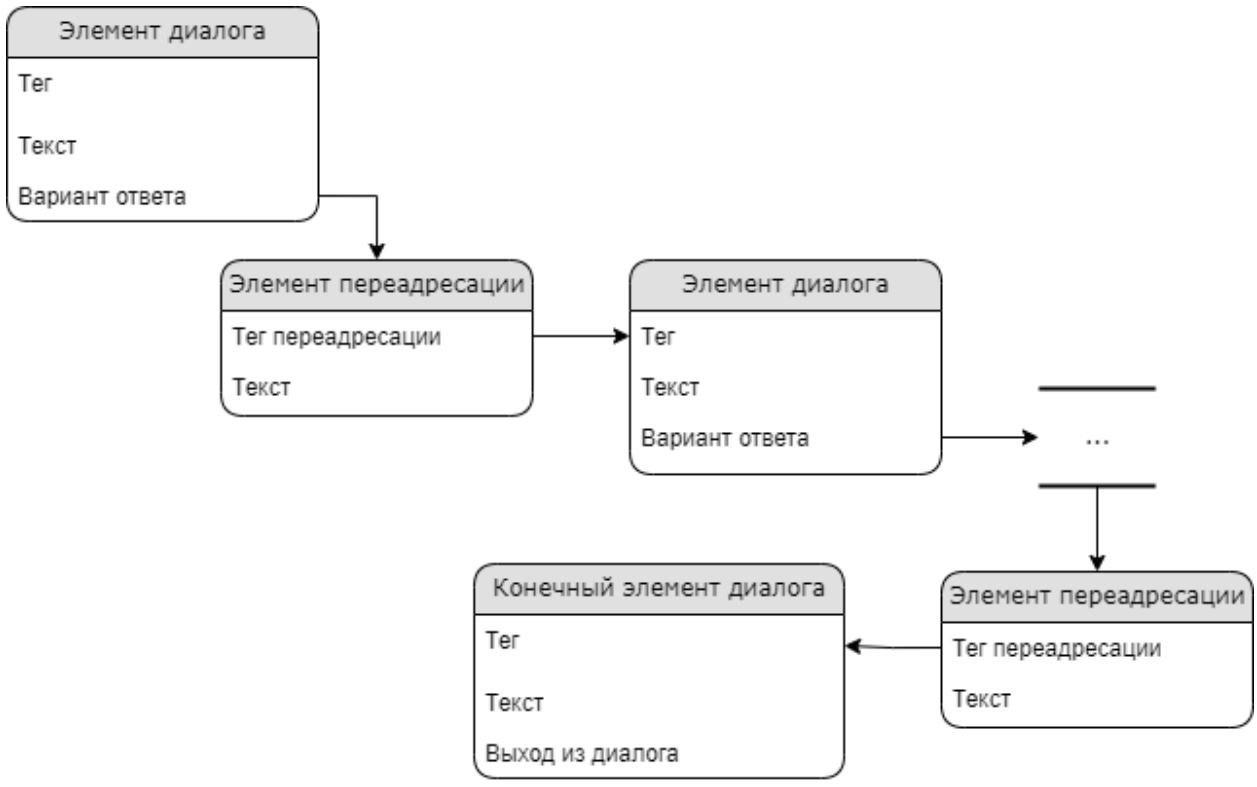

Рисунок 2.2 – Принцип работы ветви вариативной диалоговой системы

Элемент диалога и элемент переадресации в совокупности представляют структурный элемент диалогового дерева.

Элемент диалога содержит тег, текст и набор вариантов ответов. По тегу происходит сообщение между структурными элементами диалогового дерева. Текст представляет собой текстовую информацию, выводимую на экран для пользователя-игрока. Набор вариантов ответов – тот набор элементов переадресации, который соответствует данному элементу диалога.

Элемент переадресации содержит тег переадресации и текст. Текст представляет собой текстовую информацию, доступную для пользователя-игрока. Тег переадресации – тег последующего элемента диалога.

Тег представляется текстовой строкой. Для элемента диалога и элемента переадресации, ведущего к данному элементу диалога, текстовое поле тега и тега переадресации имеет одинаковые значения.

Конечный элемент диалога имеет такую же структуру, как и все остальные элементы диалога, только имеет лишь один вариант ответа, приводящий к завершению диалога и выключению диалоговой системы.

2.1.4 Система квестов

Поскольку основным способом взаимодействия с игровым миром для пользователя-игрока является получение текстовой информации, система квестов неразрывно связана с диалоговой системой.

При помощи системы квестов, пользователь-игрок получает внутриигровые задачи и оповещается о их завершении или завершении одного из этапов их выполнения.

Оповещения о получении внутриигровых заданий и оповещения о их выполнении реализуются путём вывода соответствующей текстовой

информации. Как правило, пользователь-игрок получает данную информацию в виде текстов, смысловая нагрузка которых соотносится со смысловой нагрузкой остальных текстов в диалоговой системе. Иными словами, получение игрового задания, оповещение о прохождении определённого этапа по выполнению поставленной задачи или оповещение о завершении задания, для игрока является частью игрового сюжета, а информация вписана в контекст диалогов и описаний, открываемых по мере прохождения игры.

В общем принципе работы системы квестов, в рамках поставленной на проект задачи, диалоговая система служит компонентом, реализующим взаимодействие между пользователем-игроком и системой квестов. Диалоговая система служит компонентом, реализующим получение пользователем информации о получении внутриигровых задач, информации о выполнении задачи или одного из этапов её решения.

Несмотря то, что часть задач системы квестов, реализуется компонентами диалоговой системы, сохраняется необходимость разработки остальных компонентов. Основной принцип работы системы квестов заключается в фиксировании состояния информационной системы на момент получения игрового задания пользователем-игроком и сравнении текущего состояния информационной системы с ожидаемым состоянием. Под ожидаемым состоянием понимается такое состояние информационной системы, которое соответствует выполнению игрового задания игроком или выполнения одного из этапов задания.

На этом этапе стоит заметить, что выполнение игрового задания пользователем-игроком, может затрагивать не только изменение состояния системы квестов, но изменение состояний других систем. Исходя из этого, при поиске решения по созданию системы квестов, необходимо учитывать возможность видимости других систем для системы квестов.

2.1.5 Теоретическое решение задачи по созданию системы квестов

Основная задача информационной системы квестов – отслеживать состояние внутри системы и состояния других систем. При этом, в системе хранится понятие об ожидаемом состоянии. Как только внутреннее состояние системы или состояние внешних систем приходит в состояние «ожидаемого», происходит фиксация достижение этого состояния и изменение информационной системы квестов.

Под изменением состояния информационной системы квестов, понимается отправка сигнала во внешние системы, о достижении «ожидаемого» состояния. При отправке сигнала, внешние системы информируются о полученном событии, после чего происходят необходимые изменения состояний внешних систем.

Более подробно этот принцип можно рассмотреть на примере рисунка 2.3.

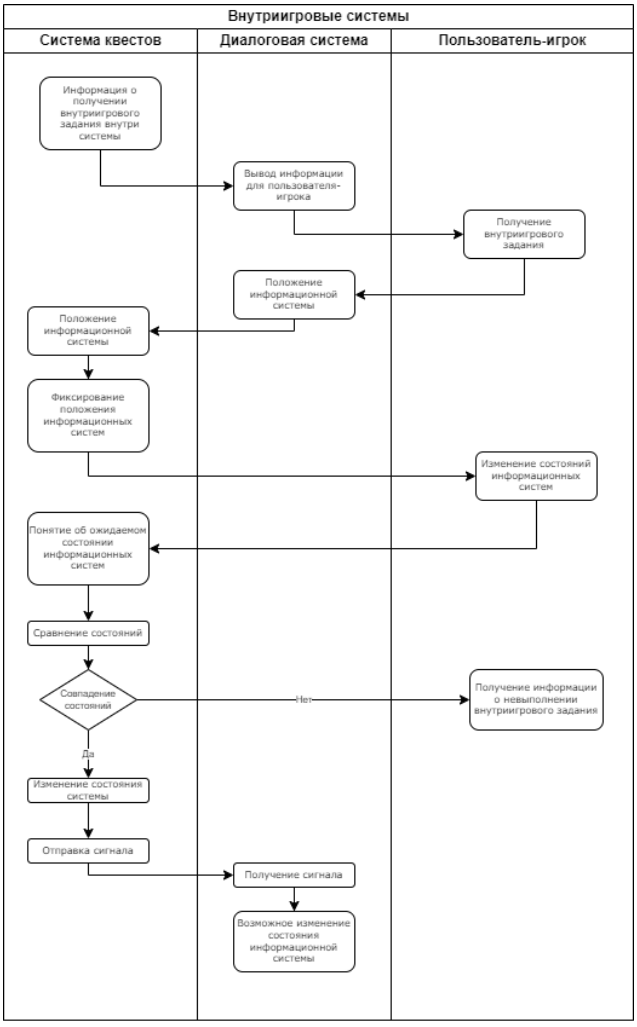

Рисунок 2.3 – Принцип работы системы квестов

Внутри системы квестов находится внутриигровая задача. При помощи диалоговой системы, происходит вывод соответствующей информации на экран и оповещение пользователя-игрока о получении задания. Далее, происходит фиксация состояний информационных систем. На данном этапе информационные системы находятся в состоянии получения игроком задания, но отсутствии факта его выполнения. Далее игроком совершаются изменения информационных систем. Изменения могут затрагивать как систему квестов и диалоговую систему, так и остальные внутриигровые системы. Полученные изменения сравниваются с понятием ожидаемого состояния, хранящимся в информационной системе квестов. При совпадении ожидаемого состояния и текущего состояния систем, происходит изменение состояния системы квестов и отправка сигнала для других систем. Сиглан информирует другие системы о завершении задания игроком и возможной необходимости изменения состояний. Если ожидаемое состояние не совпадает с текущим состоянием систем, игрок оповещается о невыполнении текущего задания.

В рассматриваемом примере, сигнал о выполнении задания и необходимости изменения состояния системы, принимается только диалоговой системой. При фактической работе информационных систем, сигнал принимается всеми системами, затрагиваемыми внутриигровой системой заданий.

2.1.6 Теоретическое решение задачи объединения диалоговой системы и системы квестов

Поскольку в рамках установленной задачи, система квестов неразрывно связана с диалоговой системой, необходима разработка системы взаимодействия двух вышеперечисленных систем.

Основным способом координации между элементами диалога являются теги. По тегам происходит вызов тех или иных ветвей диалога и переход между

ними. Исходя из этого, наиболее эффективным способом установления взаимосвязи между диалоговой системой и системой квестов будет установление системы событий при передаче определённых тегов внутри диалоговой системы.

В рамках рассматриваемого теоретического решения, предполагается создание глобальной системы событий. Под «глобальной» системой понимается такая система, значения, функции и результаты работы которой видны для всех систем.

Поскольку система квестов в значительной степени определяет игровые события, а основной её функцией является отслеживание и фиксация состояний внутри системы и внешних систем, глобальную систему событий можно рассматривать как составляющую системы квестов.

Система событий может быть основана на принципе переключателя – при свершении какого-либо события, глобальная система событий изменяет своё состояние. Изменение состояния глобальной системы событий является сигналом об изменении остальных систем.

Более подробно рассмотреть предполагаемый принцип работы глобальной системы событий возможно на рисунке 2.4.

Каждый столбец схемы отображает информационную систему, внутри которой происходит процесс. При этом, все рассматриваемые системы относятся к одной, общей игровой информационной системе, выраженной совокупностью всех разработанных систем.

Переходы между процессами обозначены стрелками. Если процесс, порождаемый работой алгоритма одной информационной системы, вызывает работу других алгоритмов, в других информационных системах, на схеме это отражается в виде изменение расположения последующего процесса относительно выделенных столбцов.

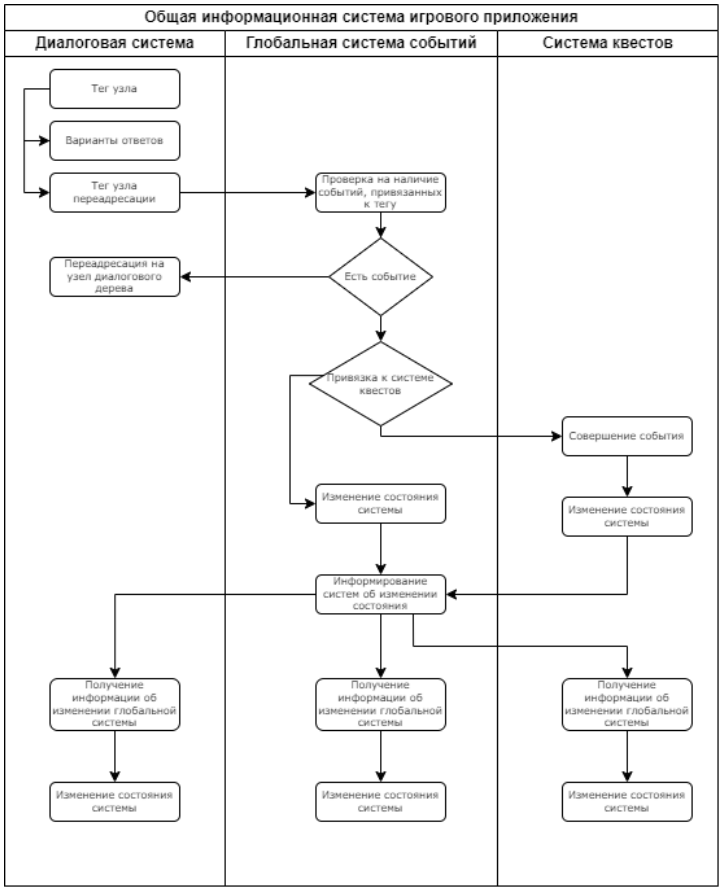

Рисунок 2.4 – Принцип работы глобальной системы событий

За основу берётся работа диалоговой системы, поскольку взаимодействие с диалоговой системой является основным принципом взаимодействия с программным продуктом и игровым миром. Основной передаваемой информацией в системе, является тег. Так, внутри диалоговой системы происходит передача тега, для вызова следующего узла диалогового дерева. При передаче тега, в глобальной системе событий происходит проверка на наличие привязки к тегу каких-либо событий. События могут быть связаны с системой квестов, или относиться напрямую к глобальной системе событий. При отсутствии привязанных к тегу событий, происходит возвращение тега в диалоговую систему и переадресация на следующий узел диалогового дерева. При наличии привязанных событий, в глобальной системе событий происходит проверка на наличие связи события с системой квестов. Если событие не связано с системой квестов и относится к глобальной системе событий, происходит изменение последней. При наличии связи события с системой квестов,

происходит изменение только в системе квестов. Изменение состояния глобальной системы событий приводит к изменению состояния всех остальных систем.

2.2 Проектирование второстепенных игровых систем

Для создания полноценного программного обеспечения игрового проекта, помимо реализации основных систем – диалоговой системы и системы квестов, необходима разработка и второстепенных информационных систем, позволяющих пользователю полноценно взаимодействовать с игровым продуктом.

К второстепенным информационным системам можно отнести:

→ Систему управления игровым персонажем;

→ Систему переходов между игровыми сценами;

→ Систему простых триггеров.

Каждая из перечисленных систем представляет из себя совокупность информационных систем, выполняющих различные функции.

2.2.1 Система управления игровым персонажем

Система управления игровым персонажем предназначена для передвижения модели игрового персонажа в игровом пространстве, и обеспечения взаимодействия игрока с игровым миром.

С точки зрения программной реализации информационной системы управления игровым персонажем, система предполагает изменение и сохранение координат модели персонажа в системе координат игрового пространства.

Также, стоит взять во внимание возможность изменения скорости передвижения игровой модели в системе координат. Скорость передвижения игрового персонажа, с точки зрения программной реализации информационной системы, влияет на скорость изменения координат модели в системе координат игрового пространства.

Система управления игровым персонажем частично должна пересекаться с системой переходов между сценами. Для реализации правильных переходов между сценами, необходимо включение системы, сохраняющей положение игрового персонажа при переходе. Такая система затрагивает как систему переходов между сценами, так и систему управления персонажем.

Таким образом, информационная система управления персонажем должна включать:

→ Систему изменения и сохранения координат модели персонажа в системе координат игрового пространства;

→ Систему изменения скорости передвижения модели;

→ Систему сохранения положения модели в системе координат при изменении игровой сцены.

2.2.2 Система переходов между игровыми сценами

Система переходов между игровыми сценами предназначена для реализации смены игровых сцен.

Игровой проект может включать в себя множество игровых сцен. Как правило, сцены образуют собой отдельную систему – каждая сцена включает в себя возможность перехода на другую сцену или на ряд других сцен.

Такая система позволяет решить сразу несколько задач:

→ Логическое разбиение игрового пространства и упрощение восприятия игрового мира игроком;

→ Искусственное ограничение игрока в игровом пространстве;

→ Облегчение нагрузки на устройство при процессах генерации игрового пространства и обработке игровых объектов.

Логическое разбиение игрового пространства позволяет управлять восприятием игрового мира, создавать уникальную атмосферу, конкретизировать игровые задачи и локализировать игровые события.

При помощи логического разбиения игрового пространства на сцены, возможно управление восприятием игрового мира. Так, для игрока, сюжет обретает логически завершённые части развития. Возможно проведение аналогии с книгами, где каждая глава – отдельная часть одного сюжета.

Искусственное ограничение игрока в игровом пространстве позволяет установить определённые условия или ограничения на переходы между сценами для игрока. Переход на дальнейшую сцену для игрока невозможен до тех пор, пока не будет удовлетворено условие для перехода.

Такая система позволяет конкретизировать игровые задачи, уточнять требования, ограничивать пространство, в котором игрок может искать решение поставленной задачи. Эта система также позволяет избежать непредусмотренных действий со стороны игрока. При наличии условий, соблюдение которых необходимо для перехода, игрок может совершить переход на следующую игровую сцену только в таком состоянии информационной системы, которое предусмотрено игровым продуктом.

Облегчение нагрузки на устройство, при обработке объектов, позволяет снизить затраты памяти устройства для полноценной работы игрового продукта. Также, это позволяет сделать игровой продукт более эффективным по памяти и по времени.

Облегчение нагрузки достигается за счёт уменьшения необходимых для обработки элементов в установленную единицу времени. Так, например, при разбиении единой игровой сцены на две, равных по объёму, устройству потребуется единовременная обработка вдвое меньшего количества информации.

2.2.3 Система сохранения положения игрового персонажа при переходах между сценами

При смене игровых сцен, необходим корректный перенос координат игрового персонажа в игровом пространстве. Переход между сценами может включать в себя как перенос старой модели игрового персонажа, так и генерацию новой, аналогичной модели.

Поскольку в рамках проекта, переход между сценами предполагает генерацию новой аналогичной модели игрового персонажа, необходима система, позволяющая изменять координаты первоначальной генерации модели, в зависимости от сцены.

Модель игрового персонажа представляется в виде единого шаблона для генерации, исходя из этого, без дополнительных условий или функций, изменение положения генерации на разных сценах невозможно. Иными словами, при генерации, модель персонажа будет получать одинаковые координаты, независимо от сцены.

При добавлении условий генерации с прямой привязкой к модели игрового персонажа, изменение передаваемых при генерации координат будет невозможно, без изменений строк кода программной реализации информационной системы. Это приведёт к утрате адаптивности информационной системы и сделает её актуальной только для текущего игрового проекта.

Для сохранения адаптивности, необходима система, не имеющая прямой привязки к игровой модели персонажа, но видимая для систем, расположенных на модели и передающая в эти системы соответствующие значения координат. Реализация подобной системы возможна путём создания глобальной системы, видящей остальные игровые компоненты.

Более подробно принцип работы такой системы можно рассмотреть на примере рисунка 2.4.

Рисунок 2.4 – Принцип работы системы переноса координат

Игровой персонаж имеет текущие координаты и координаты, установленные для модели игрового персонажа при первичной генерации. Координаты для первичной генерации необходимы для реализации генерации модели игрового персонажа в игровом пространстве, вне зависимости от сцены. При переходе между сценами, происходит обновление загрузки скриптов. Этот процесс приводит к перезапуску скриптов, расположенных на модели игрового персонажа. При этом, процесс перезапуска касается и скриптов генерации модели персонажа в игровом пространстве. Также, при этом происходит отправка и загрузка координат первичной генерации. В атрибутах перехода между сценами имеются атрибуты генерации. При переходе между сценами, происходит запрос на актуальные координаты для генерации модели игрового персонажа. Запрос отправляется в систему переноса координат, в которой хранятся актуальные значения координат. Актуальные значения присваиваются текущим координатам игрового персонажа. Таким образом, при запуске новой сцены, модель игрового персонажа получает актуальные для данной сцены значения.

2.3 Проектирование систем управления игровым приложением

Для реализации полноценной работы игрового приложения, необходимы системы, обеспечивающие корректное начало и окончание игрового процесса, а также системы, обеспечивающие воспроизведение мультимедийного наполнения игрового приложения

Для реализации данных задач предполагается разработка основного игрового меню, внутриигрового меню и внутренних игровых систем.

Основной задачей главного меню является обеспечение корректного начала и окончания игрового процесса. Через главное меню происходит выполнение таких функций, как:

→ Начало игры;

→ Переход к последнему сохранению игры;

→ Завершение игрового процесса.

Выполнение всех этих функций представляется для пользователя-игрока в виде набора кнопок. На кнопки могут быть добавлены различные визуальные эффекты и графическое наполнение.

Внутриигровое меню обеспечивает переход к другим внутриигровым системам – к главному меню, настройкам и т.п. Предполагается осуществление функций корректного перехода к главному меню, а также функций возвращения к игровому процессу и выходу из игры. Процесс сохранения игры также может фиксироваться в разделе внутриигрового меню или же происходить автоматически.

Внутриигровые системы, предназначенные для работы с мультимедийным наполнением игрового проекта, выполняют функции воспроизведения звуковой информации.

Так, системы могут выполнять функции воспроизведения фоновых звуков или звуковых эффектов, связанных с игровым процессом.

Каждая из описанных систем является стандартным решением поставленных задач в игровых приложениях, но при этом системы могут иметь различную программную реализацию.

2.3.1 Система главного меню

Система главного меню может быть представлена в виде набора кнопок, каждая из которых выполняет определённые задачи. На рисунке 2.5 каждая кнопка представлена в виде набора функций, выполняемых при её нажатии.

Рисунок 2.5 – Принцип работы главного меню

При нажатии кнопки начала игры, происходит полное обновление игрового процесса, его перезагрузка и переход к игровой сцене. При нажатии на кнопку продолжения игры, происходит аналогичный переход к игровой сцене, но при этом не происходит полного обновления игрового процесса, а прогресс, полученный ранее, сохраняется. При нажатии на кнопку выхода из игры, происходит завершение работы игрового приложения.

2.3.2 Система внутриигрового меню

Функции, выполняемые внутриигровым меню, могут быть представлены в том же виде, что и функции главного меню. На рисунке 2.6 представлен

принцип работы внутриигрового меню. Каждая кнопка также представлена в виде набора выполняемых ею функций.

Рисунок 2.6 – Принцип работы внутриигрового меню

При нажатии на кнопку возвращения к игре, происходит закрытие окна внутриигрового меню и переход к текущей игровой сцене. При нажатии на кнопку перехода в главное меню, текущая игровая сцена сменяется сценой главного меню. При нажатии на кнопку выхода из игры, происходит корректное завершение работы программы.

2.4 Итоги проектирования

Определено разделение разрабатываемой информационной системы на основные и второстепенные компоненты. Информационные системы, выполняющие задачи основополагающих игровых механик, выделены в категорию основных информационных систем. Также, совокупность основных информационных систем определена как пакет программ, необходимых для разработки в рамках задания от компании-заказчика.

Второстепенные информационные системы определены как компоненты, выполняющие внутриигровые функции, предназначенные для текущего игрового проекта, разработанного на базе основных информационных систем. Второстепенные информационные системы не требуют высокой степени адаптивности, но являются важными элементами для разработки геймплея текущего игрового проекта.

Рассмотрены теоретические решения по разработке основных и второстепенных информационных систем. Отдельное внимание уделено рассмотрению теоретического решения задачи по настройке коммуникации между системами.

Также были выделены системы, выполняющие функции управления самой игровой программной на устройстве пользователя-игрока. Рассмотрено теоретическое решение по реализации данных систем.

ГЛАВА 3. Разработка информационной системы

3.1 Разработка структур диалоговой информационной системы

Для разработки диалоговой системы, используются материалы проектирования диалоговой информационной системы.

Основным принципом при разработке информационной системы является создание адаптивных, расширяемых структурных элементов, при помощи которых может быть создана полная информационная система. Это позволяет создавать используемую информационную систему по принципу конструктора.

3.1.1 Разработка общей структуры диалоговой системы

Для реализации диалоговой системы, необходима пошаговая разработка отдельных её частей. Каждой части диалоговой системы отведён свой спектр задач. Первичным шагом в разработке диалоговой системы, является выделение и разбиение единой системы на составляющие. Компоненты диалоговой системы могут быть выделены, исходя из общих групп задач, выполняемых диалоговой системой.

Для определения задач, выполняемых информационной системой, необходимо рассмотрение состояний системы на каждом из этапов работы:

1. Информационная система добавлена в среду разработки. На данном этапе, в информационной системе не хранится пользовательская информация;
2. Добавление пользовательской информации в систему. На данном этапе, в информационную систему вносится пользовательская информация. Пользовательская информация может представлять собой текстовую информацию, предназначенную для пользователя-игрока, или текстовую информацию, представляющую теги узлов диалогового дерева. Происходит построение диалогового дерева;
3. Запуск информационной системы. На данном этапе происходит вывод информации на экран. Информация, выводимая на экран,

предназначена для пользователя-игрока и не включает в себя текстовую информацию, предназначенную для установки внутренней работы системы. При данном действии, необходим последовательный вывод информации, с соотнесением текстовых полей и полей вариантов ответов.

Исходя из рассмотренных функций информационной системы, могут быть выделены три основных компонента, совокупность которых представляет диалоговую систему:

→ Система записи и хранения информации;

→ Система соотнесения информации;

→ Система вывода информации.

Перед началом разработки компонентов, необходимо более точное их рассмотрение и установление основных функций.

Система записи и хранения информации

Данный компонент системы реализует функции конструктора. Основной задачей компонента является корректная запись вносимой информации. Информация записывается в соответствующие массивы, в установленном порядке. Этот компонент позволяет создать структуру хранения информации внутри системы.

Система соотнесения информации

Данный компонент позволяет соотносить информацию в информационной системе между собой. Информация в информационной системе хранится в различных ячейках различных массивов. Функцией данного компонента является установление связей между ячейками.

Система вывода информации

Данный компонент позволяет установить взаимосвязь между информацией в информационной системе и компонентами интерфейса, при выводе информации для пользователя-игрока.

Каждый компонент информационной системы должен быть видим для других компонентов этой же системы.

3.1.2 Разработка системы хранения информации в диалоговой системе

При организации системы хранения информации в диалоговой системе, необходимо обращение к системе вывода информации на экран.

Хранящаяся в системе информация разделяется на текстовую информацию, выводимую в текстовое поле – основной текстовый фрейм, и на текстовую информацию, выводимую в поля вариантов ответов – кнопки вариантов ответов.

Для реализации системы хранения информации, представленной в виде программного кода на языке программирования C#, был задействован метод создания структур.

Создание структуры позволяет вносить в информационную систему информацию, записываемую по принципу, заданному в структуре. При этом, при внесении информации, используется обращение к самой структуре или скрипту, содержащему структуру.

При создании структуры, был создан отдельный скрипт для её хранения. Структура хранения информации задействуется множеством других скриптов, отвечающих за получение и передачу информации в системе. Для более удобного использования скрипта, хранящего структуру, для него была создана инициализированная система – единое пространство имён. Все файлы (скрипты), внесённые в единое пространство имён, становятся видимыми друг для друга. Это позволяет использовать функции и структуры, описанные в одних файлах, в других файлах этой же инициализированной системы.

При создании структуры хранения информации в информационной системе, был использован основной принцип деления информации по принципу вывода и принципу обращения к информационным полям при помощи тегов.

Таким образом, система хранения информации содержит два ключевых структурных поля: для текстового фрейма и для фреймов вариантов ответа.

Структурное поле текстового фрейма

Структурное поле для текстового фрейма названо «Story». Данное поле предназначено для хранения основной текстовой информации. При помощи данных текстов передаётся основной массив информации – описание игровых объектов, большая часть текстов диалогов.

Для пользователя-игрока данное структурное поле представляется в виде основного текстового фрейма в диалоговой системе. Основной фрейм передаёт большую часть информации. Пример основного фрейма представлен на рисунке 3.1.

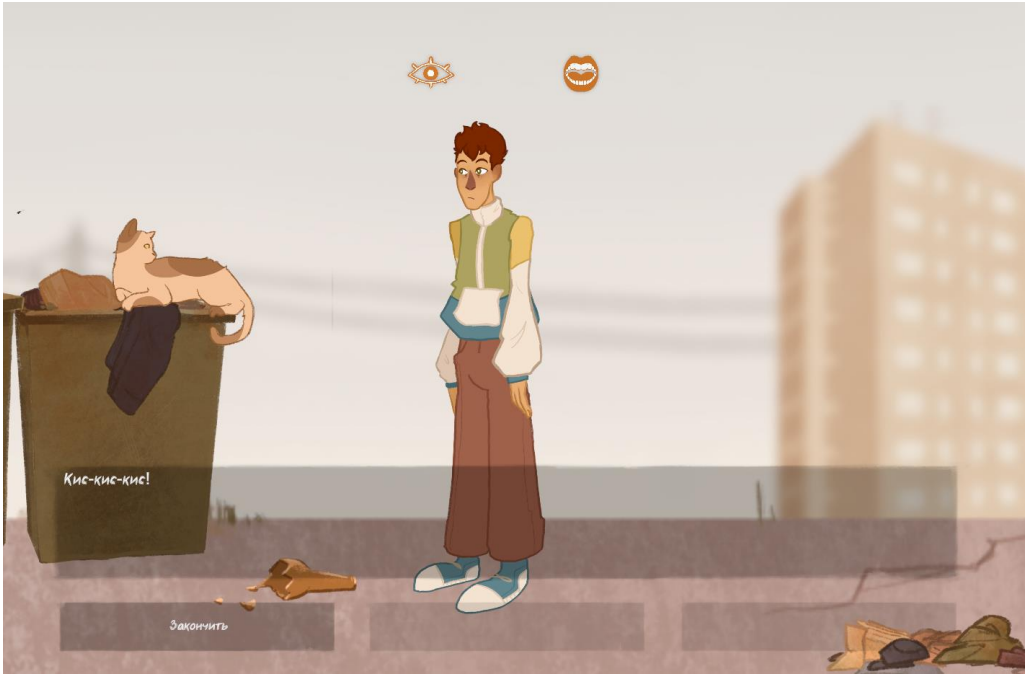

Рисунок 3.1 – Пользовательское представление структур диалоговой системы

Основными атрибутами структурного поля являются тег, текст и варианты ответов.

Тег – уникальное имя, которым обладает структурный элемент, как элемент диалогового дерева. При помощи тега возможно обращение к данному структурному элементу и, как следствие, обращение к диалоговой ветви.

Текст – атрибут, определяющий возможность структурного элемента хранить какую-либо текстовую информацию. Атрибут отвечает за привязку отдельных текстовых элементов к структурным элементам. Это, в свою очередь, позволяет связать текст и тег, по которому возможно обращение к данному тексту.

Варианты ответов – атрибут, позволяющий установить связь между структурным элементом текстового поля и набором структурных элементов полей вариантов ответов. Для каждого текста отведён один или несколько вариантов ответов.

Таким образом, текстовый фрейм представляется связкой текста, вариантов ответов и тега, по которому возможно обращение к элементу.

Структурное поле варианта ответа

Структурное поле варианта ответа названо «Answer». При помощи данного структурного элемента, игроку предоставляется возможность выбрать дальнейший путь развития диалога или дальнейшую ветку описания объекта игрового окружения.

Для пользователя-игрока данное поле представляется в виде набора кнопок с вариантами ответов. Варианты ответов имею сюжетно обусловленный текст. Пример принципа отображения вариантов ответов в диалоговой системе представлен на рисунке 3.2.

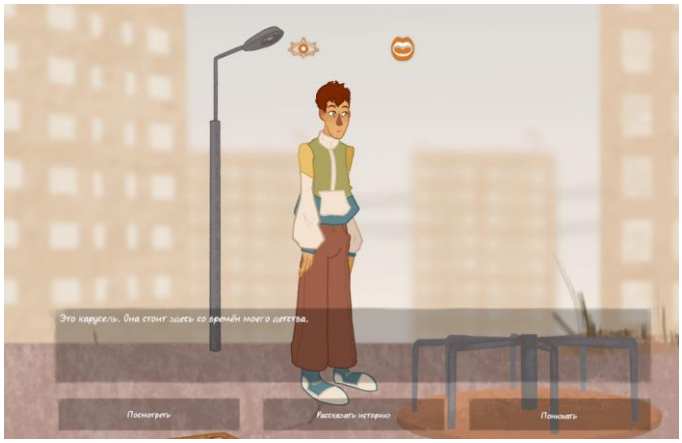

Рисунок 3.2 – Пользовательское представление структур вариантов ответа

Данное поле содержит два основных атрибута: текст и переадресацию.

Текст – атрибут, отвечающий за текстовую информацию, выводимую на соответствующем варианте ответа. Этот текст отображается для пользователя, как текст варианта ответа. Как правило, текстовая информация варианта ответа, связана по смыслу с сюжетным текстом в текстовом поле.

Переадресация – атрибут, в котором содержится тег элемента текстового поля, к которому происходит обращение при выборе игроком соответствующего варианта ответа. Текст переадресации (тег) не виден для игрока.

Таким образом, структура вариантов ответов содержит текст – видимую для игрока информацию, и тег переадресации – скрытую для игрока информацию, предназначенную только для работы информационной системы.

При целостном рассмотрении структуры хранения информации получим:

Единый элемент дерева информационной системы представлен в виде текстового поля, принадлежащего ему массива вариантов ответов и назначенного ему тега. В массиве вариантов ответов содержится один или более вариант ответа. Каждый вариант ответа представлен совокупностью текста варианта ответа и тега переадресации.

Схема записи и хранения информации в диалоговой информационной системе представлена на рисунке 3.3.

Схема частично представлена в виде элементов пользовательского интерфейса, а частично – в виде компонентов, формирующих информационную систему. Представление части информации с использованием обращения к элементам пользовательского интерфейса позволяет отобразить процессы, происходящие внутри системы, при использовании более понятных для пользователя-игрока систем управления.

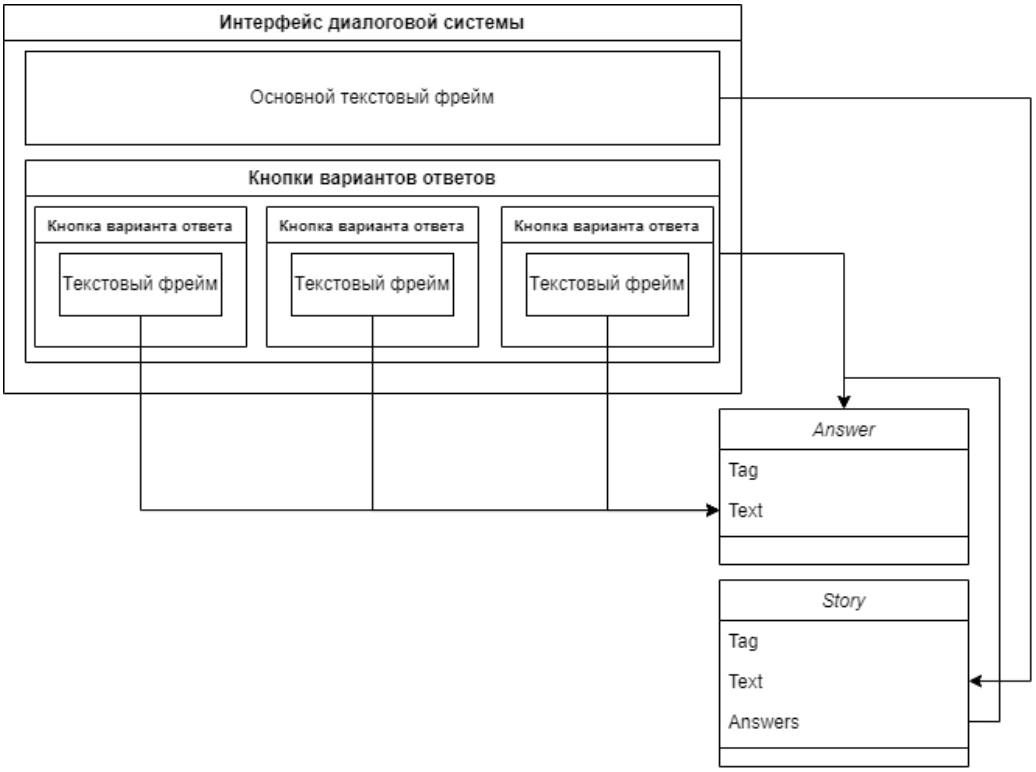

Рисунок 3.3 – Система хранения информации в диалоговой системе

Структурный элемент в данном примере представлен в виде одного элемента пользовательского интерфейса. Пользовательский интерфейс диалоговой системы представлен в виде совокупности основного текстового фрейма и вариантов отсвета. Каждый вариант ответа представлен в виде интерфейса кнопки варианта ответа и текстового фрейма. Основной текстовый фрейм выступает в качестве отображения элемента диалога, а каждая кнопка варианта ответа – в виде элемента переадресации. В совокупности, элементы переадресации, принадлежащие одному элементу диалога, представляют собой массив вариантов ответа, принадлежащий одному структурному элементу. Текст основного текстового фрейма хранится в классе Story, атрибуте Text. Текст вариантов ответов, находящийся в текстовых фреймах интерфейса кнопок, хранится в классе Answer, атрибуте Text. Все тексты вариантов ответов хранятся в едином массиве класса. При этом, в атрибуте Answers, класса Story, содержится весь массив соответствующих данному структурному элементу вариантов ответов. Это может быть представлено и как хранение совокупности кнопок

вариантов ответов в классе Answer, так и как хранение элемента класса Answer, в атрибуте Answers, класса Story.

3.1.3 Разработка схемы внедрения диалоговой системы в среду разработки Unity

Поскольку диалоговая информационная система предназначена для использования другими разработчиками в различных проектах со схожими техническими требованиями, система должна обладать простой схемой применения.

Для применения диалоговой системы в проектах, необходимо:

→ Добавление диалоговой системы в структуру проекта;

→ Добавление новых ветвей в структуре информационной системы;

→ Внесение текстовой информации в информационную систему;

→ Создание выходных точек из диалоговой системы;

→ Установление связей между узлами информационной системы и системой квестов;

→ Вывод текстовой информации из диалоговой системы.

Каждый из указанных пунктов должен быть реализован наиболее простым путём, для снижения порога вхождения при использовании разработанной системы.

Задача добавления системы в проект

Наиболее простым способом добавления объектов в среду разработки Unity является внесение объектов в папки проекта. При этом возможен дальнейший перенос объекта из папок проекта на сцену среды разработки. Эти действия являются равными по сложности и не требуют высоких навыков владения Unity.

Следуя из этого, при разработке диалоговой системы, был задействован метод создания объектов типа Prefab. Такой метод позволяет представить разработанную систему в виде объекта, перенесённого в папку проекта. Из папки

проекта, объект может быть скопирован в папку другого проекта и перенесён на сцену (рисунок 3.4).

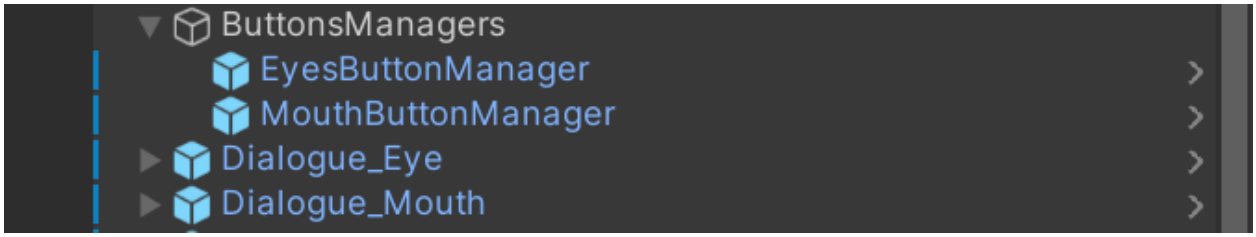

Рисунок 3.4 – Представление системы на сцене проекта

Таким образом, распространение разработанной системы, внутри среды разработки, между разными проектами не представляется сложной задачей и не требует высоких навыков владения разработкой на Unity.

Задача добавления новых ветвей

Для решения данной задачи, необходимо обратиться к структуре процесса создания новых диалоговых ветвей.

При создании, разработчику необходимо:

→ Установить интерактивный объект, с точки зрения взаимодействия с игровым пространством;

→ Установить соответствующий диалог для данного интерактивного объекта;

→ Добавить текстовую информацию.

На данном этапе стоит отметить, что диалоговая система не требует постоянной работы. Работа диалоговой системы необходима только в моменты взаимодействия с интерактивными объектами игрового пространства. Стоит уточнить, что на стадии проектирования информационной системы (описание см. «глава 2»), структурной единицей информационной системы был установлен элемент диалога, при установлении связи между парой которых, происходит формирование диалоговой ветви.

Исходя из этого, представляется возможным представить процесс создания новой диалоговой ветви при помощи добавления объектов типа Trigger и Script.

Trigger – встроенный в систему Unity объект, позволяющий устанавливать взаимодействие между объектами в игровом пространстве. Как правило, триггер представлен областью (видимой или не видимой для игрока), при взаимодействии с которой (прохождении через неё, нахождении в ней или её покидании), происходит установленное действие.

Script – объект в среде разработки Unity, представленный в виде объекта в папке проекта, позволяющий установить связь между объектами в игровом пространстве и информационной системой (программным кодом игры).

При помощи этих двух компонентов, возможно установление системы работы и добавления новых ветвей диалога в диалоговой системе.

В диалоговой системе, в единый промежуток времени, может быть задействована только одна ветвь. Помимо отсутствия необходимости в постоянной работе всех ветвей диалоговой системы, есть необходимость в отключении неактуальных на данный промежуток времени элементов, в целях экономии ресурсов устройства. Эта задача решается при помощи объединения в систему элементов триггера и скрипта.

Триггер позволяет включать только актуальные на данный момент времени ветви диалоговой системы. Скрипт, в свою очередь, позволяет установить связь между работой триггера и передачей информации в теле информационной системы.

Таким образом, процесс создания новой диалоговой ветви, с точки зрения пользователя-разработчика может быть описан следующим образом:

→ Выбор интерактивного объекта из объектов игрового пространства;

→ Добавление объекта Trigger в иерархию интерактивного объекта;

→ Добавление объекта Script в иерархию объекта;

→ Внесение значения типа string (текст) в поле тега объекта.

Добавленный элемент является корневым для диалоговой ветви данного объекта. Установленный на объект триггер, позволяет обратиться к первому узлу соответствующей ему ветви диалога. Принцип добавления новой диалоговой ветви представлен на рисунке 3.5.

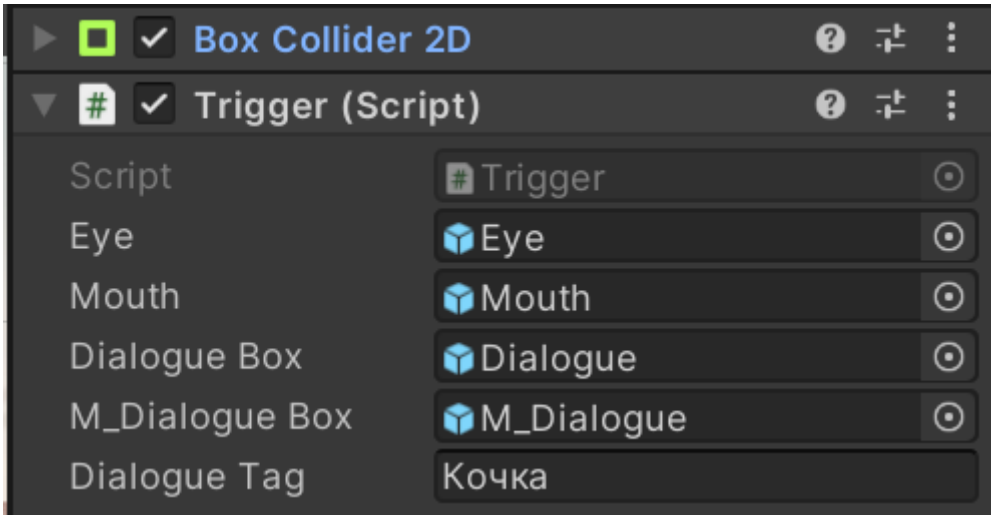

Рисунок 3.5 – Добавление новой диалоговой ветви через пользовательский интерфейс

Само формирование ветви, происходит путём внесения текстовой информации в соответствующие поля диалоговой системы. Внесённая по отдельности информация формирует элементы диалога – узлы диалоговой ветви (структурные элементы). Установление тегов в правильном порядке, позволяет объединить узлы в единую структуру – ветвь.

Более подробно принцип построения ветви представлен на рисунке 3.6.

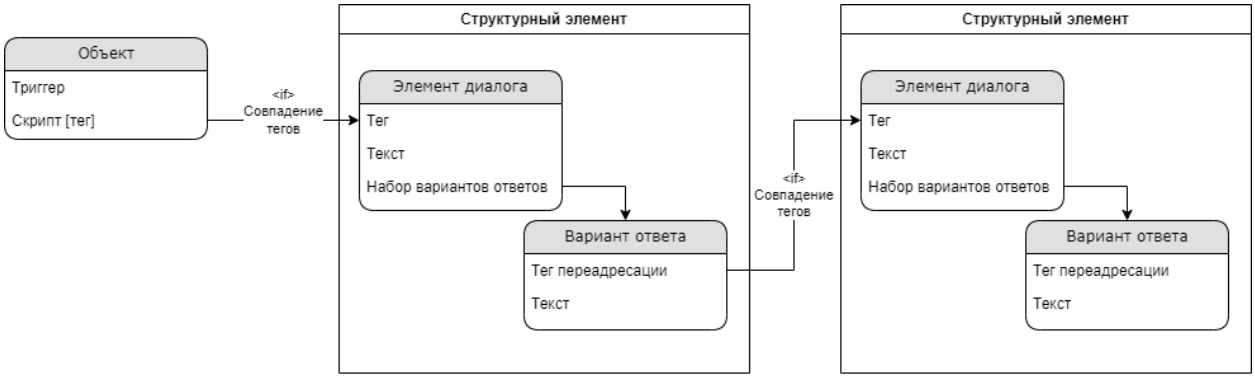

Рисунок 3.6 – Принцип построения ветви диалогового дерева

При активации интерактивного объекта, скрипт отправляет в диалоговую систему тег первого структурного элемента. Для того, чтобы данный элемент был вызван, необходимо, чтобы до этого он был добавлен в диалоговую информационную систему, а его тег совпадал с тегом, отправляемым скриптом при начале работы. Если два этих условия соблюдены, происходит вызов первого структурного объекта. Сам структурный объект состоит из элемента диалога и одного или нескольких вариантов ответов. Для создания связи между первым и последующим структурными элементами, используется схожий принцип, что и при вызове первого структурного элемента скриптом интерактивного объекта.

Для создания связи, необходимо наличие последующего структурного элемента в диалоговой информационной системе. Также, необходимо, чтобы тег переадресации одного из вариантов ответов первого структурного элемента, совпадал с тегом последующего структурного элемента. При совпадении тегов, формируется связь между структурными элементами.

По такому же принципу создаются и все последующие связи между узлами на диалоговой ветви (рисунок 3.7).

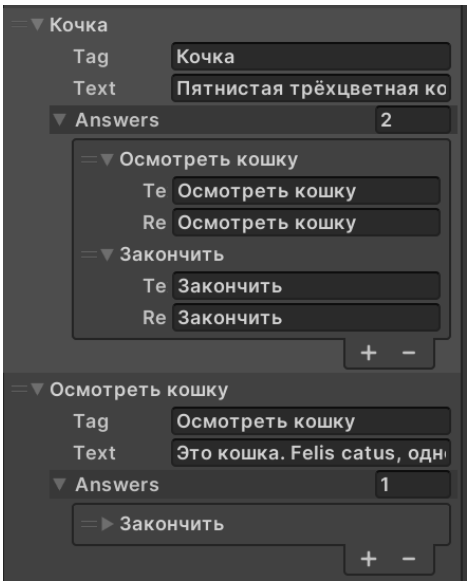

Рисунок 3.7 – Построение диалоговой ветви от корневого узла через пользовательский интерфейс

Таким образом, для пользователя-разработчика, построение связей между узлами превращается в задачу установления одинаковых тегов в два поля взаимосвязанных узлов. Теги вносятся через пользовательский интерфейс в виде текстовой информации.

Внесение текстовой информации

Внесение текстовой информации в систему, не должно задействовать работу с телом скриптов.

Тело скрипта представляется программным кодом, написание которого требует более высоких навыков владения разработкой в среде Unity.

Таким образом, наиболее простым способом внесения информации в систему, является работа с интерфейсом в среде разработки Unity. Среда разработки позволяет создавать упрощённый интерфейс, для внесения информации в информационную систему.

Так, внесение информации, не требует от пользователя системы никаких специфических навыков разработки и сохраняет низкий порог вхождения.

Для реализации пользовательского интерфейса для диалоговой информационной системы, были задействованы методы [SerializeField] и создания публичных переменных. Отображение в пользовательском интерфейсе представлено на рисунке 3.8.

Рисунок 3.8 – Пользовательский интерфейс для внесения текстовой информации

При создании публичных переменных, переменная и её значения становятся видимыми для всех остальных компонентов системы. При этом, значение публичной переменной возможно для изменения через пользовательский интерфейс. Но при этом, публичные переменные имеют больший вес, поэтому метод создания интерфейса, при котором все необходимые для внесения в интерфейс строки задаются публичными переменными, не является оптимальным по памяти.

Для решения этой задачи используется метод сериализации. Метод [SerializeField] сериализует общедоступные поля. Иными словами, при внесении поля в тип [SerializeField], поле становится общедоступным и может отображаться в пользовательском интерфейсе. Таким образом удаётся избежать дополнительного задействования памяти.

Создание выходных точек из диалоговой системы и связей с системой квестов

Под выходными точками понимается определение узлов диалоговой системы, прохождение которых приводит к остановке работы диалоговой системы на данном этапе. К выходным точкам может быть так же привязано и внесение изменений в игровом пространстве.

Так, например, при прохождении диалоговых узлов, связанных с системой квестов, необходима не только остановка работы диалоговой системы на данном этапе, но и оповещение игрока об изменении статуса квеста, что является изменением в игровом пространстве.

Для реализации создания выходных точек, задействовано создание элемента типа Script. Тело данного скрипта позволяет создавать и пополнять список выходных точек и устанавливать, при необходимости, действия, изменяющие игровое пространство.

Процесс создания такого списка так же реализован путём создания пользовательского интерфейса (рисунок 3.9).

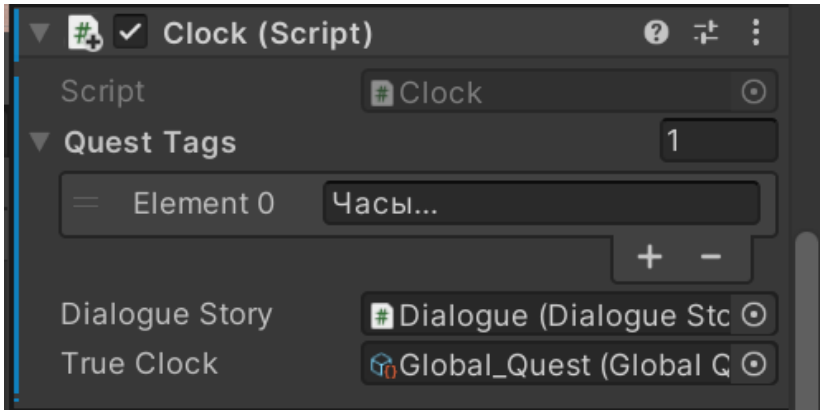

Рисунок 3.9 – Пользовательский интерфейс элемента связи диалоговой системы с системой квестов

В соответствующие поля пользовательского интерфейса вносится тег узла диалоговой системы, при прохождении которого необходимо внесение изменений в системе. Для данного тега устанавливается событие. Событие устанавливается при помощи внесения названия переменной события в глобальную систему событий (описание смотри в разделе 3.3.1). Затем, в выходной точке, указывается функция изменения значения данной переменной события с нуля на единицу, при прохождении диалогового узла с ранее указанным тегом.

Система вывода информации

Для создания системы вывода информации, задействован пользовательский интерфейс. В соответствующие слоты пользовательского интерфейса помещаются элементы системы вывода информации – необходимые тестовые фреймы.

Текстовые фреймы создаются при помощи встроенного в Unity набора инструментов. Их создание не требует от пользователя навыков программирования и завязано на выборе и создании визуальных решений для проекта.

Созданные пользователем визуальные решения связываются с системой вывода при помощи слотов пользовательского интерфейса. Каждый слот имеет

соответствующую графу названия. Выводимая на экран информация, получает свойства, установленные разработчиком при установке визуальных решений.

Для организации вывода текстовой информации, задействуется скрипт. При помощи пользовательского интерфейса возможна настройка скорости вывода текстовой информации в фреймах диалоговой системы.

3.2 Программная реализация диалоговой системы

Информационная диалоговая система реализована на языке программирования C# и представлена в виде программного кода. Программный код расположен в скриптах среды разработки, игровом движке Unity. Скрипт, как встроенный в среду разработки инструмент, позволяет установить связь между программным кодом и игровым окружением.

3.2.1 Программная реализация системы хранения информации

Система хранения информации представлена в виде конструктора. Конструктор – программный элемент, позволяющий обрабатывать поступающую информацию по установленному шаблону. Сам конструктор выполняет роль «шаблона» и представлен в виде программного кода.

Для реализации программного кода системы хранения информации в диалоговой информационной системе, был задействован метод создания структур и метод обработки событий.

Метод создания структур позволяет задать структуру каждого элемента конструктора. В программном коде, метод создания структур реализуется при помощи создания объекта типа struct. В объекте типа struct указываются атрибуты данного объекта.

Так, диалоговая ветвь представлена в виде структуры Story, и обладает следующими атрибутами:

→ String Tag – строковая переменная, в которой хранится название тега, по которому происходит обращение к ветви диалога;

→ String Text – строковая переменная, в которой хранится строка текстовой информации, выводимой в основном фрейме диалогового окна;

→ Answer[] Answers – атрибут, представленный совокупностью кнопок с вариантами ответов, относящихся к данному диалоговому окну. Каждый объект, входящий в совокупность, представлен в виде структуры варианта ответа, описываемой далее.

Реализация данной структуры на языке программирования представлена в листинге 3.1.

Листинг 3.1 – Структура Story

```
        public struct Story
                {
                    [field: SerializeField] public string Tag { get; private set; }
                    [field: SerializeField] public string Text { get; private set; }
                    [field: SerializeField] public Answer[] Answers { get; private set;
}
                }
```

Вариант ответа представлен в виде структуры Answer, содержащей следующие атрибуты:

→ String Tag – строковая переменная, содержащая тег переадресации на следующую диалоговую ветвь, к которой приводит данный вариант ответа;

→ String ReposeText – строковая переменная, содержащая текст, принадлежащий самому варианту ответа и выводимого для пользователя на кнопке с вариантом ответа.

Пример реализации в среде написания кода представлен в листинге 3.2.

Листинг 3.2 – Структура Answer

```
        public struct Answer
                {
                    [field: SerializeField] public string Text { get; private set; }
```

```
                    [field: SerializeField] public string ReposeText { get; private set;
}
                }
```

Метод обработки событий содержит функцию ChangeStory, которая выполняет задачу перехода от одной ветви диалоговой информационной системы, к другой. Функцией ChengeStory, в качестве входных данных, принимается значениеstring Tag. При отправке данного значения в функцию, происходит вызов диалоговой ветви, атрибутом Tag которой является данное строковое значение.

Более подробно принцип записи и хранения информации в диалоговой системе представлен на рисунке 3.10.

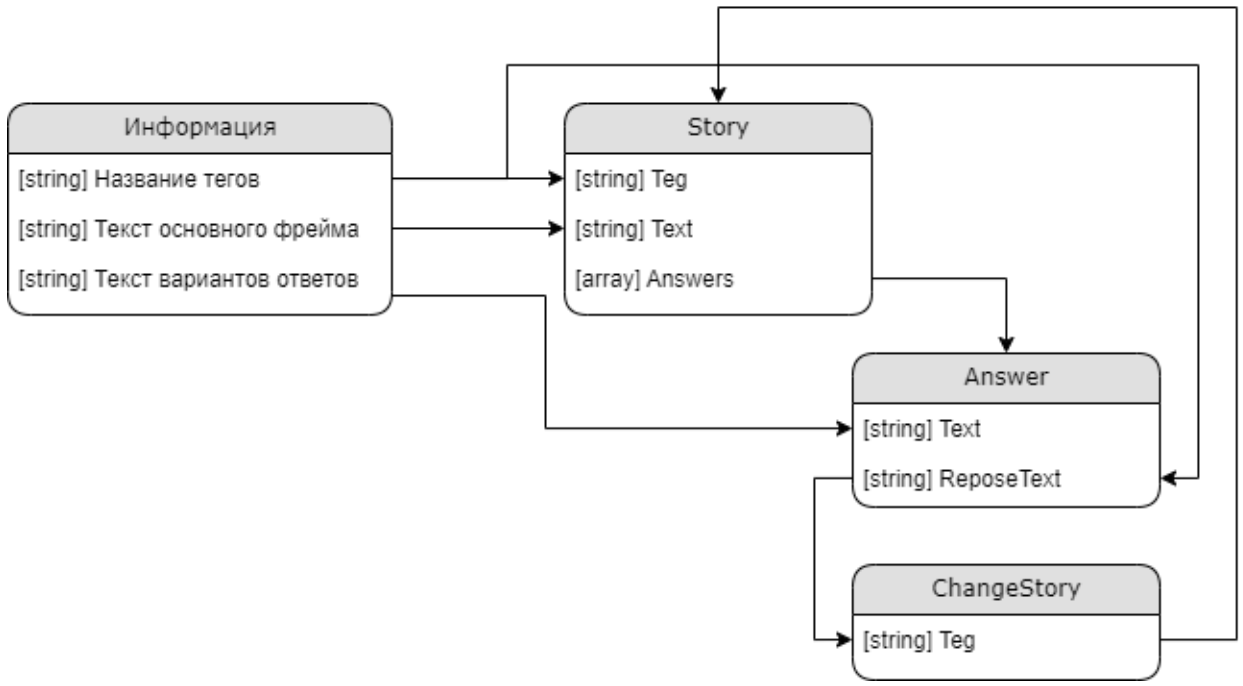

Рисунок 3.10 – Принцип записи и хранения информации в диалоговой системе.

Информация, записываемая в информационную систему, представлена в виде текстовой информации. Текстовая информация разделяется на три основных категории – текстовую информацию для основного текстового фрейма, текстовую информацию для вариантов ответа и текстовую информацию, обозначающую названия тегов для корневых узлов. Информация

для основного текстового фрейма представляет собой текст, выводимый для пользователя-игрока на экран. Этот текст имеет отношение к сюжетной составляющей игрового продукта. Информация для вариантов ответов представлена в виде текста, выводимого на кнопках с вариантами ответов. Этот текст так же доступен для пользователя. Текст названия тегов – текстовая строка, передающаяся внутри системы для обращения к узлам дерева диалогов. Текст, содержащийся в текстовой строке, является недоступным для пользователя-игрока.

При внесении информации в диалоговую систему, происходит обработка полученной информации и внесение её в соответствующие разделы системы. Текстовая информация основного текстового фрейма и текста варианта ответов, записывается в атрибуты «text», разделов Story и Answer соответственно. Текст тегов вносится в атрибуты Teg и ReposeText. При этом, в атрибуте Teg содержится тег, к которому привязывается узел диалогового дерева, а в атрибуте ReposeText – тег, на который переадресовывает данный вариант ответа.

При необходимости смены узла, тег из атрибута ReposeText отправляется в функцию ChangeStory, принимающую строковое значение тега. При работе функции происходит переадресация на соответствующий узел диалогового дерева, при этом сменяется актуальный объект Story. Принцип реализации функции представлен в листинге 3.3.

Листинг 3.3 – Функция ChangeStory

```
        public void ChangeStory(string tag) { Debug.Log("DialogueStory: " + tag);
ChangedStory?.Invoke(_storiesDictionary[tag]); }
```

При программной реализации добавлены элементы вывода структур на экран в виде пользовательского интерфейса.

Поскольку конструктор служит для записи и хранения информации, поступающей в другие части информационной системы и представляет собой

«сортировочный» элемент системы, необходимо создать видимость конструктора для всех остальных программных элементов информационной системы. Для этого все программные элементы информационной системы объединены в единое пространство имён. Пространство имён (namespace) позволяет объединить все программные элементы, находящиеся в различных файлах, в единую систему. Для внесения конструктора в пространство имён, тело файла внесено в namespace Dialogue.

3.2.2 Программная реализация смены диалоговых ветвей и системы событий

Система смены диалоговых ветвей также включает в себя систему событий, происходящих при переходе на указанный элемент ветви.

Система реализована по принципу списка. Список включает строковые значения тегов (названия тегов) тех элементов, при переходе на которые происходит то или иное событие. Под событием может пониматься игровое событие, связанное с системой квестов (выполнение или получение игрового задания), завершение диалога (выход из диалоговой системы), или любое другое событие.

Поскольку теги всех диалоговых элементов уникальны, отсутствует необходимость многократного добавления системы в игровой проект. Система добавляется однократно, после чего в список тегов вносятся все необходимые строковые значения, вне зависимости от положения элемента в диалоговом дереве.

Система включает в себя функцию внесения в единое пространство имён Dialogue и ссылки на элементы скрипта DialogueStory. Это позволяет воспользоваться функцией смены узла в диалоговом дереве по его тегу – ChangeStory.

Для реализации системы событий в совокупности со сменой диалоговых узлов, использована функция, отслеживающая наличие текущего узла диалогового дерева в списке узлов, которым присвоены события.

Если к узлу привязано некоторое событие – происходит обращение к функции события; если на узле отсутствуют привязки к каким-либо событиям, происходит обращение к функции смены узла диалогового дерева по его тегу.

Данная система реализована при помощи функции Disable, включающей в себя оператор if, в условии которого указана проверка на наличие текущего диалогового узла в списке узлов с привязанными к ним событиями.

Принцип работы системы представлен в листинге 3.4.

Листинг 3.4 – Система работы Disable

```
private void Start()
        {
            //_dialogueStory = FindObjectOfType<DialogueStory>(true);
            _dialogueStory.ChangedStory += Disable;
        }

        private void Disable(DialogueStory.Story story)
        {
        if (_disableTags.All(disableTag => story.Tag != disableTag)) return;
        //await Task.Delay(1000);
        _dialogueStory.gameObject.SetActive(false);
        }
```

В представленном программном коде, результатом работы функций является выход из диалоговой системы. Это событие является наиболее частым в диалоговой системе, так как любая диалоговая ветвь конечна. Остальные события, привязанные к диалоговой системе, строятся подобным образом, разница заключается в теле функции Disable.

3.2.3 Программная реализация системы кнопок для смены диалоговой ветви

Система включает в себя как функции вывода информации, так и функции работы с информацией внутри диалоговой системы.

Система выполняет следующие задачи:

→ Последовательный вывод текстовой информации на кнопки игрового интерфейса;

→ Соотнесение каждой кнопки с соответствующей ей тегом переадресации на следующий узел диалогового дерева;

→ Выполнение функций переадресации и перехода к следующему узлу диалоговой ветви.

Для реализации взаимосвязи с уже созданными функциями хранения информации в диалоговой системе и функцией перехода между узлами диалогового дерева, установлена функция внесения в единое пространство имён namespace Dialogue.

Использована ссылка на скрипт, написанный в DialogueStory.

Для создания связи между выводимым текстом вариантов ответов и игровым интерфейсом, создан массив для внесения в него всех кнопок, на которых предполагается расположение текста. Из макета кнопок выделен объект текста – TMP Text. Данный объект предназначен для вывода текстовой информации.

Массив кнопок позволяет определить количество кнопок в интерфейсе и установить последовательный вывод текстовой информации на каждую кнопку, по порядку. При этом, каждая кнопка массива обладает объектом текста. Текстовый объект заполняется необходимой информацией при работе системы.

Создание массива кнопок представлено в листинге 3.5.

Листинг 3.5 – Создание массива кнопок

```
        private void Awake()
                {
                    _dialogueStory = GetComponent<DialogueStory>();
                    _dialogueStory.ChangedStory += ChangeAnswers;

                    _buttonsText = new TMP_Text[_buttons.Length];
                    _currentReplyTags = new string[_buttons.Length];

                    for(int i = 0; i < _buttons.Length; i++)
                    {
                        int button = i;
                        _buttons[i].onClick.AddListener(() => SendAnswer(button));
                        _buttonsText[i] =
_buttons[i].gameObject.GetComponentInChildren<TMP_Text>();
```

```
                }
            }
```

При обработке события начала работы с диалоговой системой происходит следующая последовательность действий:

1. Установление всех перечисленных объектов – установление связи со скриптом DialogueStrory, установление массива кнопок и установление текстовых объектов на каждой из кнопок;
2. Определение длины текстового поля в текстовом объекте кнопки;
3. Изъятие по порядковому номеру текстовой информации из ячеек массива вариантов ответов диалоговой системы;
4. Внесение изъятой текстовой информации в соответствующие текстовые объекты;
5. Параллельный перебор массива кнопок;
6. Вывод информации из ячеек массива вариантов ответов, продолжающийся до тех пор, пока не будут перебраны все непустые ячейки массива кнопок.

При заполнении ячеек массива вариантов ответа в информационной системе, предусмотрена возможность создания пустого текстового поля. Иными словами, если вариантов ответа меньше, чем предусмотрено стандартным количеством кнопок, текстовое поле неактуальных кнопок остаётся пустым. Для этого, в ячейку массива вариантов ответа записывается пустое значение.

Для смены текстов вариантов ответов, происходит обращение к прописанной функции ChangeAnswers. При обращении к этой функции, происходит изъятие из ячеек массива вариантов ответов следующих по порядку элементов. Листинг 3.6 демонстрирует программную реализацию системы.

Листинг 3.6 – Смена текстов вариантов ответов

```
    private void ChangeAnswers(DialogueStory.Story story)
        {
            for (int i = 0; i < _buttons.Length; i++)
            {
                if(story.Answers.Length <= i)
```

```
                    {
                        _buttonsText[i].text = null;
                        _buttons[i].interactable = false;
                        continue;
                    }
```

Продолжение листинга 3.6

```
                    _buttonsText[i].text = story.Answers[i].Text;
                    _currentReplyTags[i] = story.Answers[i].ReposeText;
                    _buttons[i].interactable = true;
                }
        }
```

Вывод информации из ячеек массива диалоговой системы до тех пор, пока перебираются по порядку непустые ячейки массива кнопок, позволяет соотнести количество вариантов ответов с количеством кнопок. Так, количество выведенной информации не будет недостаточным или избыточным, вне зависимости от количества кнопок в игровом интерфейсе.

3.2.4 Программная реализация системы тегов в игровом пространстве

Для установления взаимосвязи между объектами игрового окружения и диалоговой системой, необходим отдельный компонент диалоговой системы.

Помимо функций взаимосвязи, компонент также решает задачи корректного взаимодействия между пользователем-игроком и диалоговой системой. Диалоговая система должна быть доступна для игрока только в активных игровых зонах. Иными словами, диалоговая система становится доступной для игрока только если в текущий момент игрового процесса игровой персонаж находится в поле действия одного из активных объектов игрового окружения.

Данный компонент реализован в скрипте Trigger и решает следующие задачи:

→ Установление взаимосвязи между объектами игрового окружения и диалоговой системой – установление тегов из диалоговой системы на объекты игрового окружения;

→ Включение отображения диалоговой системы при вхождении игрового персонажа в поле действия активного объекта – включение отображения элементов игрового интерфейса, позволяющих начать взаимодействие с диалоговой системой.

Программная реализация компонента включает в себя внесение всех элементов в единое пространство имён. Это реализовано при помощи функции namespace Dialogue.

Также, для решения задач по включению и выключению игрового интерфейса диалоговой системы, в скрипте добавлены объекты типа GameObject. Объекты данного типа являются объектами, находящимися в игровой среде. При помощи инициализации данных объектов в скрипте, возможно управление их отображением.

Реализация системы отображения пользовательского интерфейса диалоговой системы представлена в листинге 3.7.

Листинг 3.7 – Включение и выключение игрового интерфейса диалоговой системы

```
    private void Start()
            {
                Eye.SetActive(false);
                Mouth.SetActive(false);
                DialogueBox.SetActive(false);
                M_DialogueBox.SetActive(false);

                _eyeDialogTagScript = Eye.GetComponent<GetDialogueTag>();
                _mouthDialogTagScript = Mouth.GetComponent<GetDialogueTag>();
            }
```

Для установления взаимосвязи программных компонентов системы с игровым окружением, также необходимо создание активных зон интерактивных объектов игрового окружения. Для этого на каждый интерактивный объект установлен объект типа Collider, выполняющий функции триггера. Он определяет область, в которой доступно взаимодействие с интерактивным объектом, но не создаёт физических границ для объектов.

Работа скрипта Trigger завязана на взаимодействии с коллайдером, выполняющим функцию триггера для взаимодействия с объектом игрового окружения. Это реализовано при помощи функций OnTriggerEnter2D и OnTriggerExit2D. Первая функция позволяет включать работу функций, содержащихся в теле, при столкновении коллайдера модели игрока и коллайдера области взаимодействия активного объекта игрового окружения. Вторая функция выполняет те же функции, но при прекращении взаимодействия двух коллайдеров.

Работа функции начала и конца взаимодействия коллайдеров представлена в листинге 3.8.

Листинг 3.8 – Включение и выключение игрового интерфейса диалоговой системы по триггеру

```
    private void OnTriggerEnter2D(Collider2D collision)
        {
            Eye.SetActive(true);
            Mouth.SetActive(true);

            _eyeDialogTagScript.SetCurrentDialogTag(_dialogueTag);
            _mouthDialogTagScript.SetCurrentDialogTag(_dialogueTag);
        }

        private void OnTriggerExit2D(Collider2D collision)
        {
            Eye.SetActive(false);
            Mouth.SetActive(false);
            DialogueBox.SetActive(false);
            M_DialogueBox.SetActive(false);
        }
```

При начале взаимодействия коллайдеров, отображаются элементы игрового пользовательского интерфейса диалоговой системы. При этом для пользователя-игрока становятся доступны две кнопки, отвечающие за способ исследования окружающего игрового пространства. Обе кнопки отвечают за начало работы диалоговой информационной системы, но за разные её ветви. Первая кнопка отвечает за «осмотр объектов», а вторая – за «диалог».

Для инициализации ветвей, на объекте также располагается тег, совпадающий с тегом корневого узла каждой из диалоговых ветвей. Таким

образом, устанавливается взаимосвязь между объектом и принадлежащими ему диалоговыми ветвями. Более подробно принцип установки взаимосвязи представлен на рисунке 3.11.

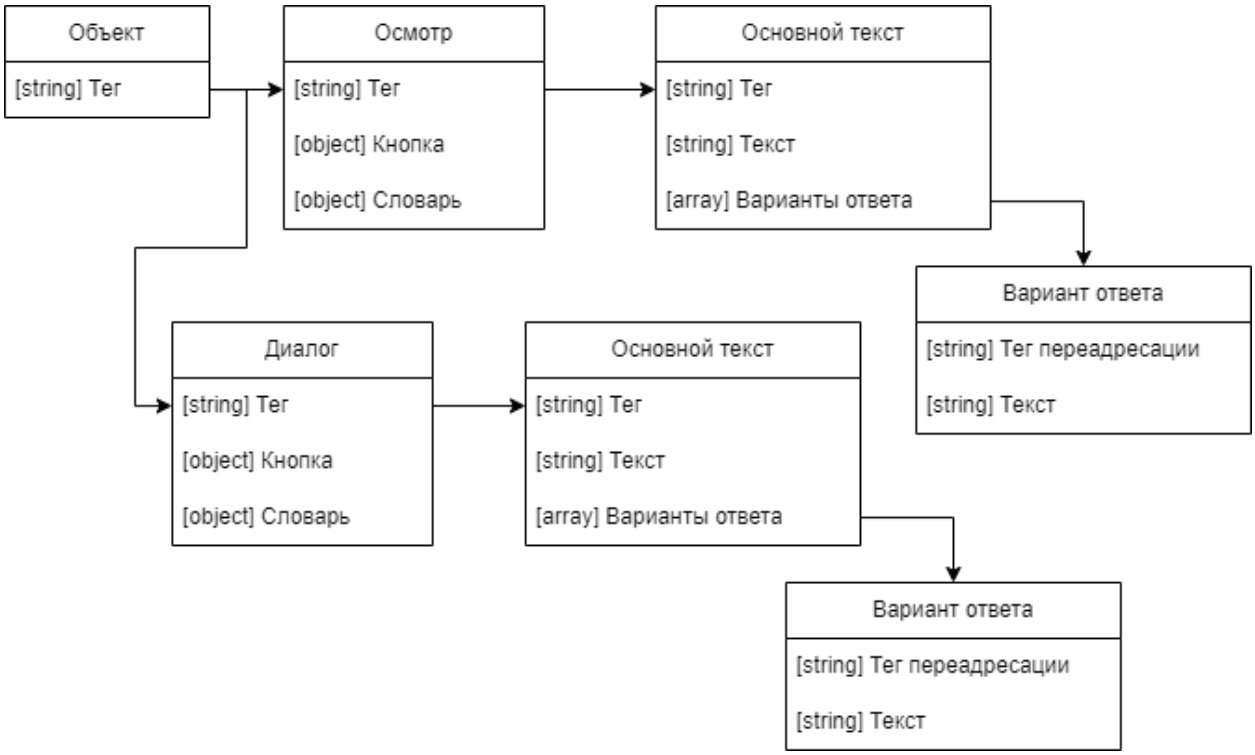

Рисунок 3.11 – Принцип связи объекта с диалоговой системой

Тег, устанавливаемый на объекте, представляется текстовой строкой, так же, как и теги, передаваемые внутри диалоговой системы, при вызове определённых диалоговых узлов. Данный тег является общим для корневого узла ветви осмотра и ветви диалога. Для корректного вызова ветви, необходимо посимвольное совпадение тега, указанного на объекте и тега, присвоенного первому диалоговому узлу ветви. Принцип вызова первого диалогового узла с объекта, схож с принципом вызова любого из узлов в диалоговой системе, за исключением необходимости передачи тега через менеджер. Менеджер выступает промежуточным звеном между диалоговой системой и объектом, перенаправляя тег с объекта и определяя его как актуальный в системе. Для каждой ветви есть свой менеджер, что позволяет разделить ветви в системе на

уровне передачи тега с объекта. Это позволяет избежать конфликта внутри системы, так как необходимо единовременное существование двух узлов с одинаковыми тегами, при том, что отправка тега в систему происходит единоразово. На схеме менеджеры представлены в виде объектов «Осмотр» и «Диалог». В системе данные менеджеры существуют в виде объектов менеджеров кнопок EyesButtonManager и MouthButtonManager.

Для включения менеджеров внутри системы, используются компоненты кнопок в среде Unity. Кнопки позволяют обрабатывать нажатия мыши на них и запускать при нажатии установленные на кнопку функции. В качестве функций, выполняемых кнопкой, установлен объект, содержащий на себе скрипт. При нажатии на кнопку, выполняется указанная функция, прописанная в скрипте.

Принцип передачи информации в системе подробно представлен на рисунке 3.12.

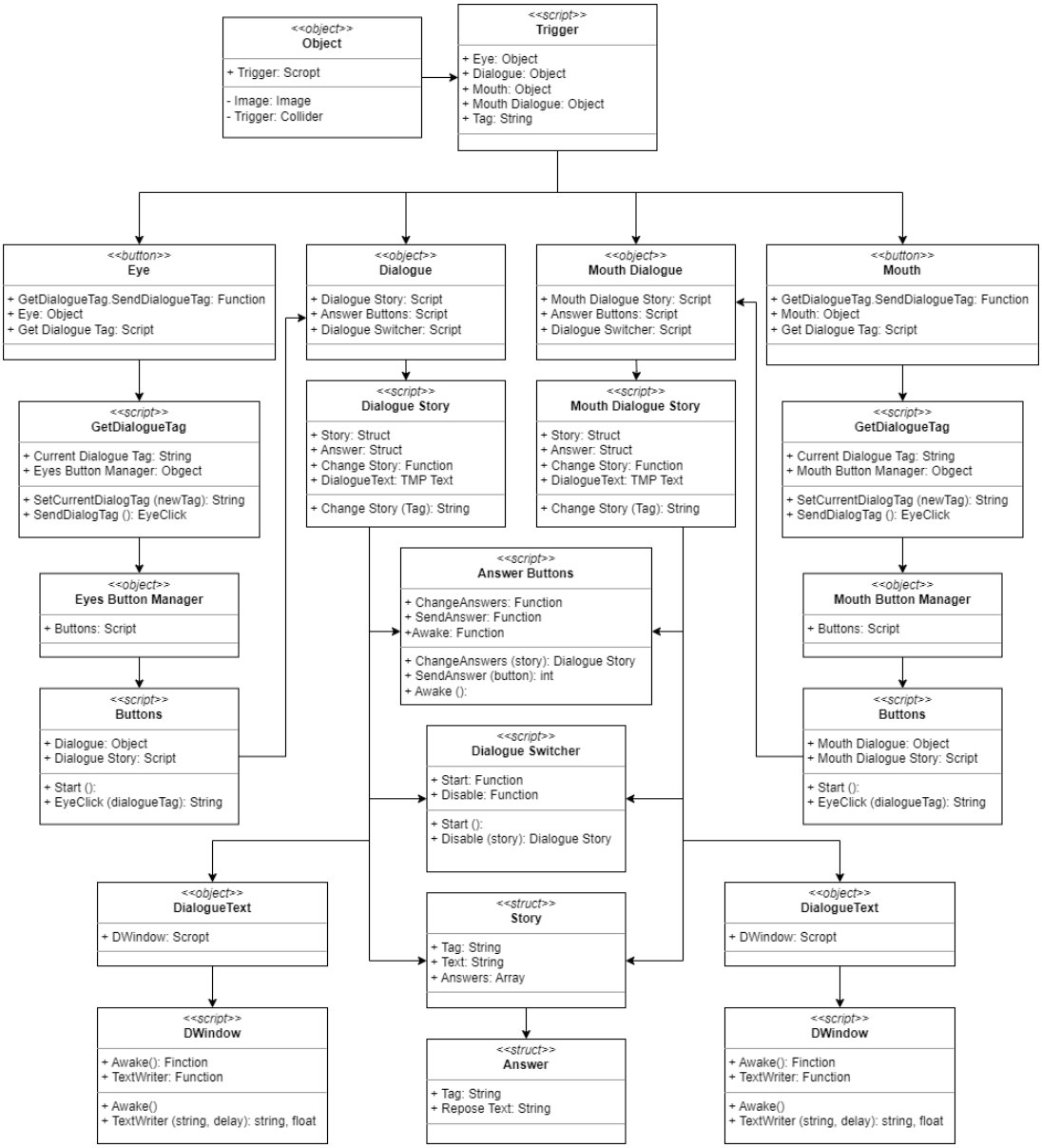

Рисунок 3.12 – Принцип передачи информации в диалоговой системе

На рисунке также отображены программные функции, отвечающие за выполнение каждой задачи.

На интерактивном объекте Object в игровой среде располагается скрипт Trigger. Он отвечает за получение и передачу первого тега в диалоговой системе и вызов первой диалоговой ветви. При этом, с началом работы скрипта, происходит начало отображения элементов игрового интерфейса диалоговой системы – отображаются кнопки «осмотра» и «диалога», представленные в виде объектов Eye и Mouth. Также, для корректной передачи тега внутри системы, Trigger содержит указатели и на хранилища (словари) каждой из ветвей

«осмотра» и «диалога». Эти указатели представлены в виде объектов Dialogue и Mouth Dialogue. При нажатии на какую-либо из кнопок «осмотра» или «диалога», происходит отправка тега в диалоговую систему. Это реализовано через менеджер кнопок. Для установки связи между менеджером кнопок и самими кнопками, используется скрипт Get Dialogue Tag. В скрипте содержатся указатель на необходимый менеджер кнопок и функции, позволяющие отправить передаваемый тег в систему как актуальный. Менеджеры кнопок в системе представлены в виде объектов Eyes Button Manager и Mouth Button Manager. Менеджеры кнопок содержат скрипт Buttons, связывающий действие нажатие кнопки с функциями отправки тега в диалоговую систему и начала отображения игрового интерфейса диалоговых окон. Игровой интерфейс диалоговых окон содержится на объектах Dialogue и Mouth Dialogue. Каждый из этих объектов, также содержит скрипт, отвечающий за работу словаря, содержащегося на данной диалоговой ветви. Словарь представляет собой тексты диалогов, связанные между собой при помощи системы тегов. Словари представлены скриптами Dialogue Story и Mouth Dialogue Story, для каждой из ветвей соответственно. Каждый словарь содержится на своей структурной ветви, но, несмотря на это, принцип работы у них одинаковый. В сущности, каждый словарь работает на основании одного и того же скрипта Dialogue Story, но для удобства пользователя-разработчика, между словарями ветвей создано разделение. Разделение служит для создания более интуитивно понятного интерфейса для пользователя-разработчика.

Далее, принцип передачи информации внутри системы полностью одинаковый для обеих диалоговых ветвей. Скрипты Dialogue Story и Mouth Dialogue Story (Dialogue Story), связаны со скриптами Answer Buttons и Dialogue Switcher. Также, они содержат структуру Story, связанную со структурой Answer.

Связка структур Story и Answer служит структурным элементом для диалоговой системы. В них происходит распределение текстовой информации внутри системы. Скрипт Dialogue Switcher позволяет создавать выходные точки из диалоговой системы и обрабатывать иные события, ключами для которых

служит прохождение определённых узлов диалоговой ветви. Скрипт Answer Buttons отвечает за корректный вывод текстов вариантов ответов в пользовательском интерфейсе.

Также, на объектах Dialogue Story и Mouth Dialogue Story, содержатся объекты текстовых фреймов Dialogue Text. На каждом объекте текстового фрейма расположен объект скрипта DWindow. Данный скрипт отвечает за вывод текстовой информации.

3.2.5 Программная реализация вывода текстовой информации

Для вывода текстовой информации вариантов ответов в диалоговой системе, задействуются функции, расположенные в скрипте AnswerButtons. Для вывода текстовой информации основных текстовых фреймов, задействуется отдельный скрипт.

Скрипт содержит функции отображения актуальной текстовой информации и функции вывода текста. Текст выводится при помощи «побуквенной печати» - каждый символ текстовой строки выводится отдельно, с небольшой задержкой. Это позволяет создать визуально приятное отображение для функций вывода текста.

Функции отображения актуальной текстовой информации представлены в листинге 3.9.

Листинг 3.9 – Отображение актуальной текстовой информации

```
    private void Awake()
            {
                _text = GetComponent<TMP_Text>();
                _dialogueStory = FindObjectOfType<DialogueStory>();
                _dialogueStory.ChangedStory += ChangeAnswers;
            }
```

Для вывода текстовой информации задействуется текстовый фрейм, содержащийся в пользовательском интерфейсе диалоговой системы. Значение текстовой информации, содержащейся в фрейме, приравнивается к значению

текстовой информации, хранящейся в актуальной ячейке словаря диалоговой системы.

При этом, задействуется функция ChangeAnswers. Функция представлена в листинге 3.10.

Листинг 3.10 – Функция ChangeAnswers

```
        private        void        ChangeAnswers(DialogueStory.Story        story)        =>
StartCoroutine(TextWriter(story.Text, delay));
```

Функция получения актуальной текстовой информации, задействует функцию, отвечающую за организацию посимвольного вывода текстовой информации TextWriter. Функция использует методы Coroutine, что позволяет создавать задержку, перед выводом следующего символа.

При передаче актуальной текстовой информации, происходит передача строки символов. Строка символов обрабатывается функций TextWriter.

Программная реализация функции TextWriter, реализующей посимвольный вывод текста, представлена в листинге 3.11.

Листинг 3.11 – Функция TextWriter

```
        IEnumerator TextWriter(string str, float delay)
                {
                  _text.text = "";
                    foreach (var symbol in str)
                    {
                        print(symbol);

                        _text.text += symbol;

                        yield return new WaitForSeconds(delay);
                    }
                }
```

При обработке значения текстовой строки, происходит «разбиение» строки на отдельные символы. Значение строки передаётся в функцию TextWriter, где происходит запись значения строки в переменную text типа _text – объект TMP, текстовый фрейм интерфейса.

При передаче информации, происходит отделение текущего первого символа и передача его в новую переменную. Таким образом, новая переменная постепенно получает всё значение строки, по символам, с небольшой задержкой при передаче каждого символа.

В результате работы скрипта, текст выводится посимвольно на экран, в пользовательском интерфейсе диалоговой системы.

3.3 Программная реализация системы квестов

Программная реализация системы квестов также осуществляется на языке программирования C#, в среде разработки игрового движка Unity. Для реализации системы квестов задействуются методы создания новых элементов в среде разработки.

3.3.1 Программная реализация глобальной системы событий

Основной функцией системы квестов является фиксирование текущего состояния внутренних и внешних систем и сравнение данного состояния с ожидаемым. Для полноценной реализации данной механики, необходимо создать видимость всех систем для системы квестов. Это достижимо путём создания глобальной системы событий.

«Глобальная» система подразумевает видимость структур для всех остальных систем.

Создание глобальной системы могло быть реализовано при помощи создания скрипта и внесения всех остальных скриптов в единое пространство имён. В таком случае, все скрипты могли бы быть представлены в виде единой системы, каждый из компонентов которой представлен в виде самостоятельного файла с кодом. Но при таком методе реализации возможно было бы возникновение конфликта между различными компонентами на более позднем этапе разработки.

Глобальная система событий может быть представлена как компонент системы квестов и как один из компонентов, объединяющий систему квестов с диалоговой системой.

Для создания глобальной системы был задействован принцип «переключателя». В таком случае, при достижении ожидаемого состояния внутренними или внешними системами, происходит изменение ключевых значений в глобальной системе. Под ключевыми значения понимается значение «ключа». Ключами же в глобальной системе служат переменные, отображающие совпадение или несовпадение текущих состояний систем с ожидаемым.

Для реализации глобальной системы, в среде разработки Unity был создан объект нового типа GlobalQuest. Для создания объекта такого типа, был задействован метод создания классов ScriptableObject. Это позволяет создать объект, обладающий свойствами, прописанными в теле скрипта. Вместе с этим был задействован метод [CreateAssetMenu], позволяющий добавить объект данного типа в общее меню среды разработки Unity.

Программная реализация функции GlobalQuest представлена в листинге 3.12.

Листинг 3.12 – Функция GlobalQuest

```
[CreateAssetMenu]
public class GlobalQuest : ScriptableObject
{
    public int Quest_Event;
    public int ButtleTrue;
    public int TrueClock;
    public int StartQuest;
    public int GetButtle;
}
```

В теле скрипта содержатся переменные типа int, отображающие совпадение или несовпадение текущих состояний систем с ожидаемыми. Состояние, когда переменная равна нулю, отображает несовпадение данных

состояний. Когда состояние рассматриваемой системы достигает ожидаемого, значение переменной заменяется на единицу.

При необходимости создать новое событие, в список переменных может быть добавлена новая переменна. Это происходит при помощи внесения названия данной переменной в общий список переменных событий. При этом, в системе квестов добавляется ключ, выполняющий функцию изменения значения переменной на единицу. Пользовательский интерфейс компонента представлен на рисунке 3.13.

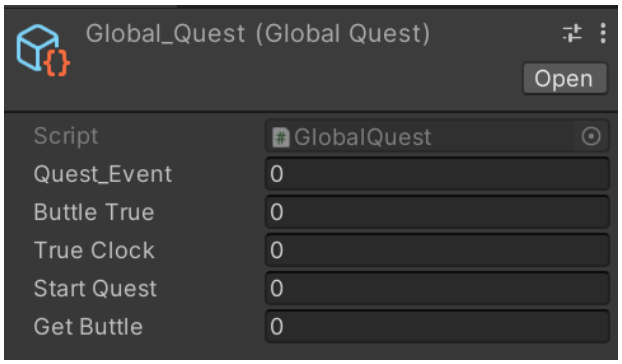

Рисунок 3.13 – Пользовательский интерфейс глобальной системы событий

Функции глобальной системы событий выполняет объект типа GlobalQuest названный Global_Quest. На объекте, при его создании, располагается скрипт GlobalQuest, а сам объект обладает заданными в скрипте свойствами. Поскольку скрипт не содержит функций, задающих физические значения для объекта, то объект не имеет физической оболочки и выполняет только функции хранилища. При этом, значения всех переменных событий сразу выводятся в пользовательский интерфейс. Через пользовательский интерфейс, при тестовом запуске проекта, возможно отслеживание значений переменных событий и, как следствие, оценка корректности работы всей системы квестов.

Поскольку система квестов связана с диалоговой системой, зачастую, ключ может быть представлен в виде совокупности тега диалоговой системы и функции события, происходящего в случае получения данного тега.

Более подробно принцип работы глобальной системы событий представлен на рисунке 3.14.

Рисунок 3.14 – Принцип работы глобальной системы событий

Глобальная система содержит в себе список переменных событий типа int. Изначально данные переменные имею нулевое значение. Система квестов связана с диалоговой системой и принимает значения тегов, передаваемых внутри диалоговой системы. При передаче тега, к которому привязано событие из системы квестов, срабатывает ключ. При срабатывании ключа в системе квестов, происходит изменение значения переменной события из глобальной системы с нуля на единицу.

Изменение значения переменной служит сигналом для изменения состояния внутренних и/или внешних систем, в зависимости от установленного, желаемого результата работы.

3.3.2 Программная реализация ключей системы событий

Под ключами системы событий понимается единая система, связующая глобальную систему событий с диалоговой системой. В данной системе располагаются ключи, которые служат для принятия тегов из диалоговой системы и обработки функций, выполняемых при прохождении узлов диалоговой системы с данными тегами.

Для реализации ключей используются механики, схожие с механиками создания выходных точек из диалоговой системы.

Для связи ключа с узлом диалоговой системы используются теги. Обращение к узлу происходит при помощи тега, так же, как и обращение к нему внутри диалоговой системы. Если ключ содержит тег определённого узла ветви диалога, то при прохождении данного узла, происходит и обращение к ключу в системе квестов.

При обращении к ключу, происходит работа функции, привязанной к данному тегу. В большинстве случаев, функция включает в себя только изменение значения переменной события в глобальной системе событий, но могут быть установлены и другие значения для функций. Для изменения значения переменной события, переменная события добавляется в список переменных функции, через использование глобального объекта GlobalQuest.

Большинство ключей имеет схожую программную реализацию. Пример программной реализации ключа представлен в листинге 3.13. Листинг содержит программную реализацию ключа, отвечающего за одно из внутриигровых событий. Данное внутриигровое событие имеет название TrueClock и отвечает за изменение состояние диалоговой системы и системы квестов.

Листинг 3.13 – Ключ для глобальной системы событий

```
    private void Disable(DialogueStory.Story story)
            {
                if (_QuestTags.All(disableTag => story.Tag != disableTag)) return;
                //await Task.Delay(1000);
                _dialogueStory.gameObject.SetActive(false);
                Debug.Log("CLOCK");

                _TrueClock.TrueClock = 1;
            }
```

Перед началом работы ключа, происходит определение совпадение тега узла. При совпадении текущего тега с тегом узла с привязанным ключом, запускается работа тела скрипта. В теле происходит обращение к переменной

события TrueClock через объявление переменной в переменных кода _TrueClock. При этом, происходит присвоение переменной значения равного единице.

Также, в тело скрипта добавлена функция вывода информации в консоль среды разработки для отслеживания корректной работы функций при тестовом запуске приложения в эмуляторе.

3.3.3 Программная реализация общей системы событий

В общей системе событий происходит обработка значений, находящихся в глобальной системе событий. При различных комбинациях данных значений, возможны различные состояния внутренних и внешних систем, изменение состояний игрового пространства.

Для системы квестов, общая система событий служит менеджером, в котором указаны все рассматриваемые состояния глобальной системы событий и соответствующие данным состояниям, состояния внешних систем.

Общая система событий также может быть рассмотрена как совокупность основной системы событий и нескольких, менее объёмных систем, выполняющих второстепенные функции.

Основная система событий включает в себя события, обработка которых имеет первостепенное значение для работы игрового проекта. Корректная обработка данных событий определяет возможность прохождения игроком всего игрового сюжета. Второстепенные системы событий строятся по аналогичному с основными системами принципу, но их работа решает более мелкие задачи, влияющие на игровые события, но не определяющие работоспособность всего игрового приложения.

Принцип построения систем одинаковый, как для основной системы событий, так и для второстепенных. Принцип построения основывается на отслеживании значений переменных событий в глобальной системе событий. Для этого используется внесения переменной типа GlobalQuest в список переменных в каждой такой системе. Затем, в теле скрипта происходит

обращение к определённой переменной события и определения значений функций для обоих её состояний. Также, рассматривается и совокупность состояний нескольких переменных событий, если определённое состояние системы зависит сразу от нескольких игровых событий.

Проверка состояний или их совокупности происходит при помощи операторов if, в теле которых происходит определение состояний или совокупностей.

Пример проверки состояний представлен в листинге 3.14.

Листинг 3.14 – Проверка состояний

```
        if (_globalQuest.Quest_Event == 0 &&
                        _globalQuest.ButtleTrue == 0 &&
                        _globalQuest.TrueClock == 0 &&
                        _globalQuest.StartQuest == 0 &&
                        _globalQuest.GetButtle == 0 )
                    {
                        Box1.SetActive(true);
                        Box2.SetActive(false);
                        Box3.SetActive(false);
                        Box4.SetActive(false);
                        ClockBox.SetActive(false);
                        ObjectBox1.SetActive(true);
                        ObjectBox2.SetActive(false);
                    }
```

Для добавления нового события и его обработки, необходимо внесение переменной события в список переменных глобальной системы событий. После этого, необходимо добавление данной переменной в систему событий (основную или второстепенную). При этом, обращение к переменной события происходит внутри системы, при обращении к переменной глобальных событий GlobalQuest. Далее, в системе происходит определение функций, соответствующих состояниям данной переменной событий.

Стоит заметить, что для второстепенных систем событий может также быть создана и своя «глобальная» система событий. Для этого, в среде разработки Unity добавляется объект типа GlobalQuest, но имеющий отличное название от Global_Quest (данный объект служит глобальной системой событий для основной системы событий). При этом, в среде кода второстепенной системы

событий, указывается ссылка на другой объект, который служит для данной системы «глобальным».

Разделение системы событий на основную и второстепенные позволяет не только распределить задачи внутри системы, но и избежать повреждений всей игровой системы при внесении незначительных изменений в систему событий.

В таком случае, основная система событий служит «скелетом» для программной реализации системы квестов, а второстепенные системы, вносящие меньшее значение в работу всего продукта, определяются как «сопутствующие» или «дополнительные». Даже в случае их некорректной работы или при полном их удалении, повреждается лишь менее значительная часть проекта, но не нарушается его полная работоспособность.

3.4 Программная реализация второстепенных информационных систем

Программная реализация второстепенных информационных систем может быть разделена на три основные категории:

→ Программная реализация систем управления;

→ Программная реализация систем меню;

→ Программная реализация второстепенных внутриигровых систем.

Каждая группа систем включает в себя набор скриптов, функций и встроенных систем, реализующих соответствующую задачу.

3.4.1 Программная реализация систем управления

Системы управления выполняют функции управления игровым персонажем, функции перехода между сценами, а также частично пересекается с функциями второстепенных внутриигровых систем.

Системы реализованы внутри скриптов. Каждый из скриптов выполняет свой спектр функций и может быть связан с другими скриптами, выполняющими другие функции схожего спектра задач.

Управление игровым персонажем

Скрипт управления игровым персонажем располагается на объекте шаблона игрового персонажа в среде разработки Unity. Скрипт отвечает за функции передвижения модели персонажа в игровом пространстве. Расположение скрипта на шаблоне игрового персонажа представлено на рисунке 3.15.

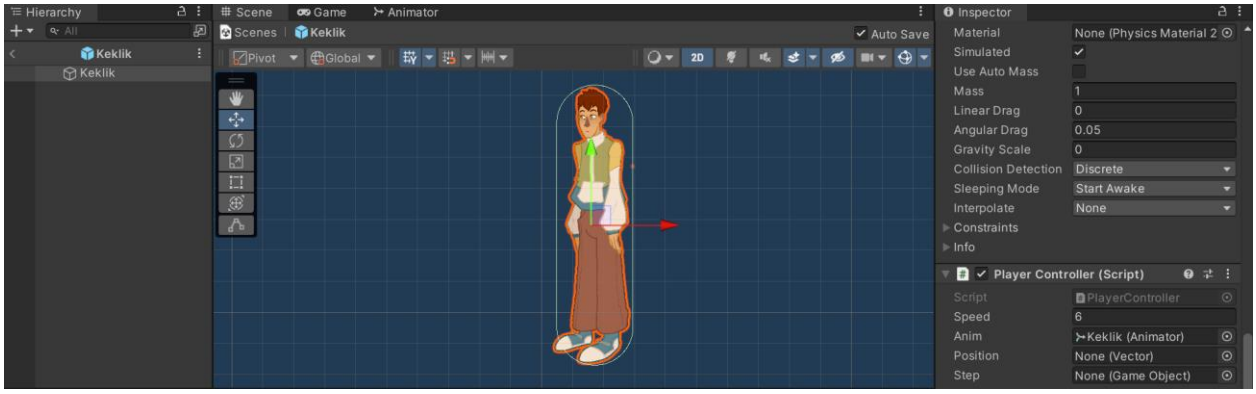

Рисунок 3.15 – Шаблон модели игрового персонажа

При передвижении персонажа в игровом пространстве, также воспроизводится звук шагов персонажа, при этом звук шагов на различных сценах может быть разным. Эта система затрагивает второстепенные внутриигровые системы.

Также, система управления персонажем, частично пересекается и с системой перехода между сценами, описанной далее. Данное пересечение систем необходимо для реализации корректного отображения модели персонажа на новой игровой сцене.

Модель персонажа содержит анимации, которые проигрываются при движении модели персонажа в игровом пространстве. Для этого в шаблон модели персонажа добавлен аниматор. В аниматоре указаны все необходимые анимации персонажа, а также настроены переходы между анимациями.

С точки зрения программного кода, система управления персонажем выполняет функции изменения значения координат, на которых располагается

модель персонажа, при нажатии пользователем-игроком на соответствующие управлению клавиши клавиатуры.

Управление персонажем реализовано при помощи скрипта PlayerController. В скрипте содержатся функции изменения координат модели персонажа, а также функции воспроизведения анимаций и поворота спрайтов модели.

Для изменения координат модели персонажа задействуются встроенные функции обработки событий нажатий на клавиши. К ним добавлена переменная, определяющая скорость изменения координат – скорость движения персонажа. Функция передвижения модели представлена в листинге 3.15.

Листинг 3.15 – Функции изменения координат модели игрового персонажа

```
    void Update()
        {
            direction.x = Input.GetAxisRaw("Horizontal");
            //direction.y = Input.GetAxisRaw("Vertical");
        }

        private void FixedUpdate()
        {
            direction.x = Input.GetAxisRaw("Horizontal");
            rb.MovePosition(rb.position + direction * speed * Time.fixedDeltaTime);
```

В функции не задействуется изменение координат по оси Y, так как в игровом пространстве данная функция не требуется.

Функции воспроизведения анимаций содержат обращения к аниматору, расположенному на модели игрового персонажа. При обращении к аниматору, указываются функции начала воспроизведения анимации и название необходимого перехода анимации (название перехода указано в аниматоре). Если происходит изменение координат модели персонажа, то анимация (и переход к ней), становятся активными: anim.SetBool("isWalk", true). В противном случае, происходит переход на анимацию «статики», воспроизводимую по умолчанию.

Воспроизведение анимаций соотносится с функциями изменения координат модели персонажа. Представление в среде разработкиотображено в листинге 3.16.

Листинг 3.16 – Анимации модели персонажа с функциями изменения координат

```
if (direction.x == 0)
        {
            anim.SetBool("isWalk", false);
            step.SetActive(false);
        }
        else
        {
            anim.SetBool("isWalk", true);
            step.SetActive(true);
        }
```

Для реализации поворота спрайта модели, реализована функция Flip(). Реализация функции в среде разработки кода представлена в листинге 3.17.

Листинг 3.17 – Функция Flip

```
void Flip()
    {
        left = !left;
        Vector3 Scaler = transform.localScale;
        Scaler.x *= -1;
        transform.localScale = Scaler;
    }
```

В теле функции управления персонажем, происходит проверка на значение изменение координат. Если координаты изменяются в положительную сторону – поворот спрайта не происходит; если в отрицательную – происходит. Для выполнения поворота спрайта, функция Flip вызывается в теле функции управления игровым персонажем.

Переход между сценами

Переход между сценами осуществляется при достижения игроком края текущей игровой сцены и перемещении модели игрового персонажа в область смены сцен.

Для корректной смены сцен, необходимо создание ряда функций, отвечающих как за техническую реализацию перехода, так и за визуальные эффекты.

Техническая реализация перехода может быть разделена на два типа функций – смена игровой сцены и перенос координат модели игрового персонажа.

Область смены сцен в среде разработки представлена в виде коллайдера, расположенного на пустом объекте и выполняющего функции триггера. На объекте области расположен скрипт, выполняющий функции смены сцены (рисунок 3.16).

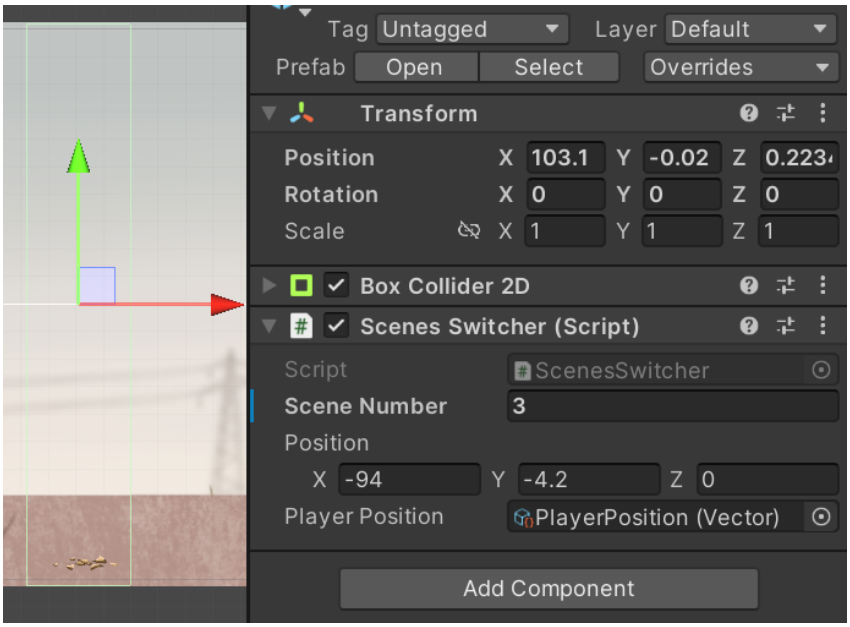

Рисунок 3.16 – Область смены игровой сцены

Скрипт смены сцены представлен в теле скрипта ScenesSwitcher. В теле скрипта содержится две функции, первая из которых отвечает за реализацию функций скрипта при столкновении коллайдеров, а вторая – за изменение номера сцены на указанное значение.

Функция смены сцены реализована через обращение к номеру необходимой сцены. Номер сцены указывается через пользовательский интерфейс среды разработки. Функция принимает целочисленное значение номера сцены, на которую будет осуществлён переход. Номер сцены

обрабатывается встроенной функций LoadScene доступной при использовании библиотеки UnityEngine.SceneManagement. Функция смены сцен представлена в листинге 3.18.

Листинг 3.18 – Функция смены сцен

```
    public void ChangeScene(int scene)
        {
            SceneManager.LoadScene(scene);
            PlayerPosition.initialValue = position;
        }
```

Обращение к функции смене сцены происходит в теле первой функции, отвечающей за работу смены сцен при столкновении коллайдеров области смены сцен и модели игрового персонажа. В теле функции происходит отправка актуального для данного перехода значения сцены в функцию смены сцены. Работа функции при столкновении коллайдеров обеспечивается при помощи встроенной функции OnTriggerEnter2D. Данная функция позволяет запустить работу тела функции только при столкновении коллайдеров (листинг 3.19).

Листинг 3.19 – Функция смены сцен по триггеру

```
    private void OnTriggerEnter2D(Collider2D collision)
        {
            ChangeScene(SceneNumber);
        }
```

Функция переноса координат модели игрового персонажа задействует как скрипт перехода между сценами, так и скрипт управления игровым персонажем.

Для получения координат положения модели игрового персонажа, актуальных для новой сцены, необходима переменная, в которой значения координат будут сохранены вне зависимости от текущей сцены. Такая переменная реализована путём создания глобальной переменной нового типа в среде разработки. Создание переменной реализовано при помощи создания скрипта Vector. Скрипт внесён в класс ScriptableObject. Это позволяет создавать в среде разработки объекты, обладающие свойствами, указанными в теле

скрипта. При этом, к скрипту применены функция [CreateAssetMenu], что позволяет создавать объекты данного типа через меню в среде разработки.

Данный скрипт предназначен для сохранения трёх значений координат. Для этого, в теле скрипта содержится создание переменной типа Vector3, сохраняющей эти значения. Программная реализация представлена в листинге 3.20.

Листинг 3.20 – Функция Vector

```
[CreateAssetMenu]
public class Vector : ScriptableObject
{
    public Vector3 initialValue;
}
```

В среде разработки Unity был создан объект PlayerPosition, обладающий свойствами Vector и сохраняющий значение координат. Таким образом, при внесении данного объекта в качестве переменной в среду кода PlayerController и среду кода ScenesSwitcher, возможно разделение хранимых значений координат от самих объектов, но сохранение возможности передачи данных между глобальной переменной и переменными кода.

В скрипте управления игровым персонажем, добавлена переменная position типа Vector. После этого, в методе Start добавлена функция загрузки координат появления модели персонажа в игровой среде. Принцип работы сохранения координат с привязкой к системе управления персонажем представлен в листинге 3.21.

Листинг 3.21 – Функция сохранения координат в системе управления персонажем

```
void Start()
    {
        transform.position = position.initialValue;
        anim = GetComponent<Animator>();
        rb = GetComponent<Rigidbody2D>();
    }
```

Начальное положение появления персонажа на сцене устанавливается при помощи отдельного объекта. Значение координат задаётся при осуществлении смены игровых сцен.

Для этого, в код смены игровых сцен добавлена переменная position типа Vector. Значение данной переменной задаётся через пользовательский интерфейс, а обращение к значениям происходит только при переходе между сценами. Программная реализация представлена в листинге 3.22.

Листинг 3.22 – Добавление переменной position типа Vector

```
public void ChangeScene(int scene)
    {
        SceneManager.LoadScene(scene);
        PlayerPosition.initialValue = position;
    }
```

Для создания визуальных эффектов перехода был создан эффект затемнения экрана при начале перехода между сценами, и уменьшения затемнения до полного исчезновения при появлении новой игровой сцены.

Эффект создан при помощи анимации. Анимация включает в себя два ключевых кадра – отсутствие затемнения и полное затемнение. В случае исчезновения затемнения экрана, анимация имеет обратный порядок.

Анимация проигрывается при помощи отдельного элемента. Разделение элементов создано для создания независимых переходов и анимаций на различных игровых сценах. При таком методе разработки, анимация на выделенной сцене может быть настроена независимо от работы функций перехода между сценами.

3.4.2 Программная реализация систем меню

Системы меню выполняют функции работы главного меню и внутриигрового меню. Внутриигровое меню частично включает функции, схожие с функциями главного меню.

Главное меню включает следующие функции:

→ Начало игры;

→ Продолжение игры с момента сохранения (сохранение происходи автоматически на каждом этапе игрового процесса);

→ Выход из игры.

Также, в главное меню включены функции анимаций интерактивных элементов меню. Внешний вид игрового меню представлен на рисунке 3.17.

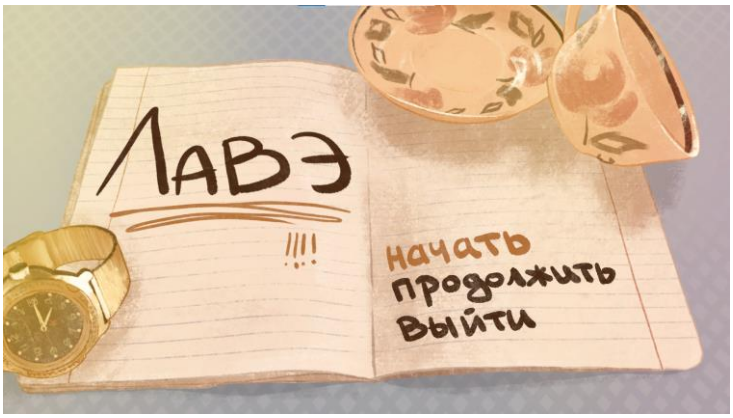

Рисунок 3.17 – Пользовательский интерфейс главного меню

Внутриигровое меню содержит такие функции, как:

→ Возвращение к игровому процессу;

→ Выход в главное меню (с автоматическим сохранением);

→ Выход из игры.

Выход из игры в обоих случаях выполняет одинаковую функцию. Внешний вид внутриигрового меню представлен на рисунке 3.18.

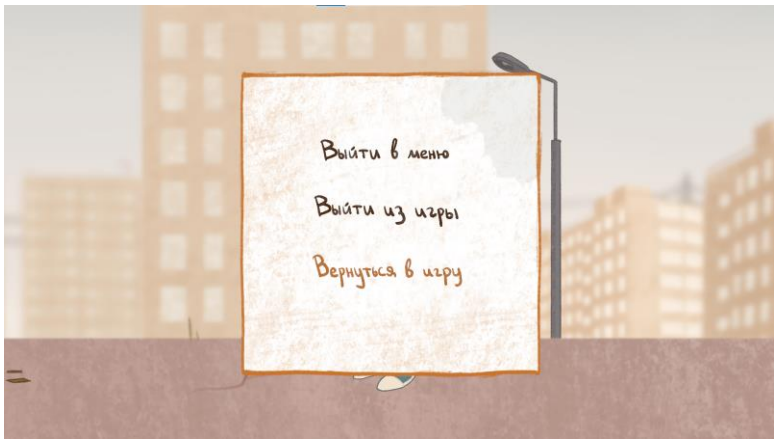

Рисунок 3.18 – Пользовательский интерфейс внутриигрового меню

Для каждой функции создан отдельный скрипт. Для настройки работы скриптов, для главного и внутриигрового меню создан менеджер.

Начало игры в главном меню

Функции начала игрового процесса включат в себя не только переход к игровым сценам, но и сброс игрового прогресса к начальным настройкам. Для разработки этих функций в скрипт начала игры включены ранее описанные функции перехода между сценами и добавлены функции сброса игрового прогресса. Сброс игрового прогресса происходит при помощи обращения к глобальным переменным системы событий игрового приложения.

Поскольку глобальная система событий содержит в себе переменные событий, выполняющие роль «переключателей» и фиксации определённых состояний игровых систем, сброс игрового прогресса возможен при помощи приравнивания значений всех переменных событий к нулю. Программная реализация процесса представлена в листинге 3.23. В представленном процессе происходит обращение ко всем внутриигровым переменным событий и присвоение им нулевого значения. Переменные событий носят актуальные для разрабатываемого игрового проекта названия.

Листинг 3.23 – Процесс сброса игрового прогресса

```
public void NewGameClick()
    {
        ChangeScene(SceneNumber);

        QuestSystem.Quest_Event = 0;
        QuestSystem.ButtleTrue = 0;
        QuestSystem.GetButtle = 0;
        QuestSystem.StartQuest = 0;
        QuestSystem.TrueClock = 0;
        _treasure.TreasureObject = 0;
    }
```

Установка начальных значений для всех переменных событий внесена в функцию NewGameClick. Скрипт, в котором расположена функция,

располагается на объекте менеджера главного меню. При нажатии на соответствующую кнопку («начать»), происходит обращение к функции NewGameClick. В теле функции содержится обращение к функции смены сцены. При нажатии на кнопку происходит переход к новой игровой сцене и присвоение всем переменным событий нулевого значения.

При запуске игрового процесса, необходима передача корректных координат для модели игрового персонажа. В данном случае, начало игрового процесса может быть рассмотрено как переход между сценами.

Для этого, на сцене, следующей за которой является первая игровая сцена, добавлена функция отправки корректных координат для модели игрового персонажа. Передача координат происходит по такому же принципу, что и передача при смене игровых сцен.

Продолжение игры в главном меню

Продолжение игрового процесса отличается от начала игры сохранением игрового прогресса. Сохранение игрового прогресса происходит автоматически при помощи глобальных переменных событий. При продолжении игрового процесса игроку не требуется просмотр вводных игровых сцен. При этом кнопка продолжения игры доступна только при наличии игрового прогресса, что не позволяет игроку пропустить вводные сцены при первом запуске игры. Такая механика позволяет избежать нарушения линии повествования в игровом сюжете.

Для создания включения и выключения кнопки продолжения игрового процесса при наличии или отсутствии игрового прогресса соответственно, используется функция проверки наличия игрового прогресса при помощи обращения к переменным событий. Если среди переменных событий имеются переменные с ненулевым значением – в игре имеется прогресс и, как следствие, игра может быть продолжена.

Кнопка «продолжить», отвечающая за продолжение игры, работает по схожему принципу с кнопкой начала игры. Различие заключается в запускаемом при нажатии скрипте.

При продолжении игры не происходит сброса игрового прогресса. Как следствие, скрипт не содержит функций обращения к переменным событий. При нажатии на кнопку происходит переход к последней сцене, на которой игрок завершил игровой процесс.

Выход из игры в главном меню

При выходе из игры необходимо полное завершение работы программы. При этом сохраняются только значения глобальных переменных.

Для реализации выхода из игры и завершении работы программы, используются встроенные функции Unity. При нажатии на кнопку выхода, происходит обращение к функции ExitGameClik. В теле функции содержится встроенная функция Application.Quit(), позволяющая завершить текущий процесс работы программы.

Реализация кода представлена в листинге 3.24.

Листинг 3.24 – Работа функции выхода из игры

```
public void ExitGameClik()
    {
        Application.Quit();
        Debug.Log("Exit");
    }
```

В функцию добавлен элемент вывода информации в консоль среды разработки. Этот элемент позволяет отследить работу функции при тестовом запуске приложения в эмуляторе среды разработки Unity.

Внутриигровое меню

Внутриигровое меню реализовано в виде отдельного окна, открытие которого происходит при нажатии пользователем-игроком на клавишу «Escape».

Окно внутриигрового меню реализовано путём добавления в общую иерархию объектов GameMenu и GameMenuManager.

Первый объект представлен в виде объекта типа Canvas. Этот объект позволяет создавать отдельное окно в игровом пространстве. В окре размещены объекты интерфейса внутриигрового меню.

Второй объект служит менеджером для внутриигрового меню и выполняет функции координации работы его кнопок. При нажатии на определённые кнопки внутриигрового меню, происходит обращение к менеджеру и скриптам, расположенным на объекте менеджера. Сам менеджер представлен в виде пустого объекта, не обладающего физическими свойствами.

Для реализации открытия игрового меню при нажатии кнопки, используется обращение к менеджеру игрового меню и скрипту PauseMenu. Данный скрипт обрабатывает нажатия на кнопки пользователем-игроком. При нажатии на клавишу Esc, происходит обращение к телу функции. Программная реализация функции представлена в листинге 3.25.

Листинг 3.25 – Отображение интерфейса внутриигрового меню

```
    void FixedUpdate()
        {
            if (Input.GetKeyDown(KeyCode.Escape))
            {
                GameMenu.SetActive(true);
            }
        }
```

Функция обработки нажатия на клавишу вписана в тело функции FixedUpdate. Это позволяет обрабатывать событие нажатия на клавишу с частотой кадров. Иными словами, проверка на нажатие происходит с каждым новым кадром игры.

Функция имеет обращение к объекту GameObject, представленного в виде объекта окна внутриигрового меню. При нажатии на кнопку и выполнении функции, окно становится активным, выводится соответствующая графическая информация.

При открытии внутриигрового меню, отображаются кнопки действий. При нажатии на каждую кнопку, происходит обращение к менеджеру

внутриигрового меню. Менеджер содержит скрипт PauseMenu_ButtonManager, который обрабатывает все функции нажатия на кнопки.

Возвращение к игровому процессу во внутриигровом меню

Возвращение к игровому процессу служит функцией закрытия внутриигрового меню и возвращению к моменту, на котором игровой процесс был приостановлен.

При нажатии на кнопку «вернуться к игре», происходит обращение к функции BackButtonClick(), содержащейся в скрипте PauseMenu_ButtonManager менеджера кнопок внутриигрового меню.

Программная реализация функции возращения в игру представлена в листинге 3.26.

Листинг 3.26 – Функция возвращения в игру

```
public void BackButtonClick()
    {
        GameMenu.SetActive(false);
    }
```

Функция включает в себя обращение к объекту типа GameObject. Данный объект представлен в проекте в виде окна внутриигрового меню. При выполнении функции, окно меню перестаёт быть активным.

Выход из игры внутриигрового меню

При нажатии на кнопку «выход из игры», происходит обращение к менеджеру внутриигрового меню и вызов функции ExitButtonClik(). Функция работает по тому же принципу, что и функция выхода из игры в основном меню. При выполнении команды используется встроенная функция Application.Quit().

Программная реализация функции представлена в листинге 3.27.

Листинг 3.27 – Выход из игры во внутриигровом меню

```
public void ExitButtonClik()
    {
        Application.Quit();
        Debug.Log("Exit");
    }
```

Выход в главное меню в меню игры

Выход в главное меню из меню игры позволяет игроку вернуться на начальный экран игровой программы.

При выполнении команды, происходит обращение к менеджеру и функции MenuButtonClick().

При выполнении функции, осуществляется переход на первую сцену. Для этого задействуется функция смены сцен. Функция имеет аналогичный принцип работы со сменой сцен при игровом процессе

Функция выхода в главное меню представлена в листинге 3.28.

Листинг 3.28 – Функция выхода в главное меню

```
public void MenuButtonClick()
    {
        ChangeScene(SceneNumber);

    }
```

При вызове функции происходит отправка значения первой (нулевой в списке иерархии) сцены в функцию перехода между сценами. Значение сцены вносится через пользовательский интерфейс.

3.4.3 Программная реализация внутриигровых информационных систем

Под внутриигровыми информационными системами понимаются системы, выполняющие функции воспроизведения звуковых эффектов как при определённых игровых событиях, так и постоянных звуковых эффектов на сценах.

Звуковые эффекты в игровом приложении могут воспроизводиться постоянно (как, например, фоновая музыка игры или фоновые звуки), так и при определённых игровых событиях (как, например, звуки шагов главного героя).

Поскольку ряд функций воспроизведения звуковых эффектов связан с работой других функций, работа внутриигровых информационных систем может

быть представлена как система разрозненных программных элементов, каждый из которых выполняет свои, локальные задачи.

При этом, стоит заметить, что многие из функций имеют схожий принцип работы, несмотря на выполнение различных функций.

Среди всех функций, могут быть выделены несколько, наиболее крупных:

- Система воспроизведения фоновых звуков;
- Система звуков игрового персонажа.

Первая система является общей для всех игровых сцен. Она служит для установки фонового звука на каждой сцене. При этом, фоновый звук может изменяться в зависимости от сцены.

Вторая система отвечает за воспроизведение звуковых эффектов только при определённых условиях – если модель игрового персонажа находится в движении. При этом, на разных сценах звук эффекта может быть разным.

Система воспроизведения фоновых звуков

Поскольку система воспроизведения фоновых звуков является общей для всех игровых сцен, ей модель была вынесена в шаблоны. На каждую сцену был добавлен шаблон системы. При этом, в шаблон системы помещаются различные звуковые файлы, что позволяет изменять фоновую музыку при изменении сцен.

Системой воспроизведения звука служит пустой объект с добавленным элементом Audio Source. Данный компонент позволяет проигрывать звуковые эффекты, а также производить их настройку.

Система звуков игрового персонажа

Система звуков игрового персонажа включена в скрипт управления игровым персонажем. Это необходимо для настройки ситуативного воспроизведения звуковых эффектов.

Система звуков вынесена из модели игрока для упрощения системы смены звуковых эффектов на сценах. При этом, скрипт управления игровым персонажем PlayerController содержит ссылку на данный объект.

В скрипте PlayerController система звуков представлена в виде объекта типа GameObject.

В системе изменения координат положения модели игрового персонажа, в разделе воспроизведения анимаций, включены функции активации и деактивации объекта звуковой системы (листинг 3.29).

Листинг 3.29 – Функция воспроизведения звуковых эффектов на модели персонажа

```
if (direction.x == 0)
        {
            anim.SetBool("isWalk", false);
            step.SetActive(false);
        }
        else
        {
            anim.SetBool("isWalk", true);
            step.SetActive(true);
        }
```

Если изменение координат не происходит – модель персонажа находится в состоянии покоя, анимация ходьбы не проигрывается и объект step, представляющий звуковую систему, не активен. Если происходит изменение координат – модель персонажа находится в движении. В таком случае проигрывается анимация и объект звуковой системы становится активным.

3.4.4 Программная реализация систем геймплея

Для реализации полноценного функционала игрового приложения, необходимо наполнение игрового проекта разнообразным контентом и различными механиками геймплея.

Для данного игрового проекта, помимо основного, сюжетного квеста, разработаны менее объёмные внутриигровые задания. Задания выполнены в жанре «головоломок».

Головоломка для пользователя-игрока представлена в виде совокупности текстовой информации, выводимой на определённой ветви диалогового дерева, и графической реализации, основанной на работе программы.

С точки зрения программной реализации, графическая оболочка головоломки представлена в виде набора интерактивных областей, расположенных на последовательности игровых сцен. Решением головоломки является определение верной последовательности интерактивных областей на сценах. При верном определении области на текущей сцене, происходит переход на последующую сцену. При неверном выборе – происходит возвращение к начальному положению данного задания.

Кнопки на сценах имеют сюжетно обусловленную графическую оболочку и представлены в виде набора дверей. Реализация представлена на рисунке 3.19.

Рисунок 3.19 – Графическая оболочка головоломки

Поскольку интерактивные области распложены в игровой среде и являются элементами игрового окружения, для обработки взаимодействия с ними необходима система обработки положения и нажатия мыши.

Для наиболее корректной реализации данной системы, был вынесен единичный элемент интерактивной области и определён в качестве шаблона. Остальные интерактивные области, расположенные в игровом пространстве, являются копиями разработанного шаблона.

В качестве шаблона интерактивной области выделен объект с графической оболочкой двери.

На установленный шаблон добавлен скрипт, выполняющий функции обработки нажатий мыши и перехода на установленную сцену (рисунок 3.20).

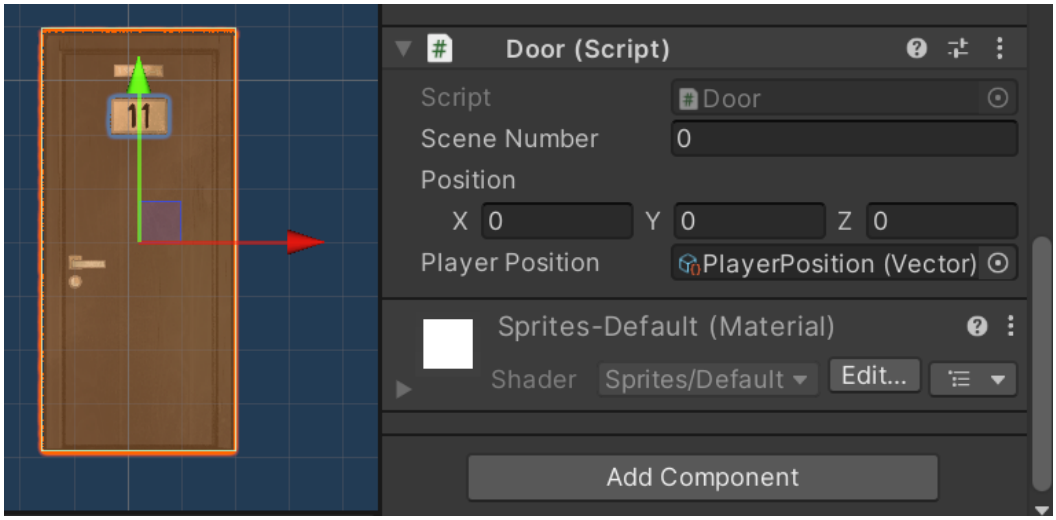

Рисунок 3.20 – Шаблон интерактивной области

Все интерактивные области выполняют функцию перехода на новую игровую сцену. В зависимости от верности выбранной области, переход может происходить или на последующую сцену, или на первую сцену игрового задания. Правильность выбора обуславливается текстовым заданием.

Также, все интерактивные области имеют одинаковую функцию обработки события нажатия мыши.

Работа данных функций реализована в скрипте Door, прикреплённом к шаблону интерактивной области.

Для обработки нажатий мыши, задействован класс IPointerClickHandler. Работа класса определяется в установленной функции OnPointerClick(PointerEventData eventData). Для корректной работы данной функции, необходимо наличие объекта типа Collider, выполняющего функции триггера, на интерактивном объекте. Также, для обработки событий, необходимо задействовать функции объекта Event System. Данный объект является встроенным объектом в среде разработки Unity. Для корректной работы функции, необходимо наличие данного объекта на игровой сцене.

Также, для отслеживания положения курсора мыши на экране, необходимо добавление компонента Physics 2D Raycaster на камеру текущей сцены. Данный

компонент позволяет отслеживать камере положение курсора на экране и фиксировать совпадение его положение с положением интерактивной области.

Функция обработки нажатия на интерактивную область представлена в листинге 3.30.

Листинг 3.30 - Функция обработки нажатия на интерактивную область

```
public void OnPointerClick(PointerEventData eventData)
    {
        ChangeScene(SceneNumber);
        PlayerPosition.initialValue = position;
        Debug.Log("Click");
    }
```

При нажатии на интерактивную область, происходит переход на новую сцену. Переход на новую сцену работает по тому же принципу, что и переход между сценами при стандартном передвижении персонажа в игровом пространстве.

При переносе игрока на новую сцену, необходимо задание новых координат для его положения. Это осуществляется по тому же принципу, что и при стандартном переходе между сценами, с применением глобальной переменной хранения координат.

Для удобства отслеживания работы функции, в тело добавлен вывод текстовой информации в консоль при тестовом запуске в эмуляторе Unity.

Аналогичные методы создания интерактивных объектов применены и к другим игровым объектам. Данные методы позволяют расширить принцип изучения игрового пространства и внести дополнительные поля для исследования.

3.5 Итоги разработки

Разработаны все необходимые информационные системы.

Основные информационные системы разработаны с учётом необходимости низкого порога вхождения при дальнейшей эксплуатации. Для этого при разработке были добавлены элементы пользовательского интерфейса, позволяющие вносить необходимые изменения с систему без задействования программного кода.

Второстепенные информационные системы позволяют создать разнообразный геймплей для игрового проекта. При их разработке были задействованы различные методы взаимодействия пользователя с игровым пространством и различные методы изучения игрового мира.

ГЛАВА 4. Тестирование информационных систем

4.1 Внедрение информационных систем

Поскольку основными адаптивными информационными системами, разработанными в ходе выполнения работы являются диалоговая информационная система и система квестов, в разделах внедрения и тестирования информационных систем, рассматриваются только данные системы. Остальные системы разработаны для конкретного игрового продукта и не подразумевают многократного использования в других игровых проектах. При необходимости, второстепенные информационные системы также могут служить основой в других информационных продуктах, но их разработка не подразумевает высокого уровня адаптивности и простой установки в других проектах.

4.1.1 Внедрение диалоговой информационной системы

Внедрение диалоговой информационной системы рассматривается от создания интерактивного объекта в игровом пространстве, до создания первого узла диалоговой ветви. Отдельно рассматривается процесс привязки события к диалоговому узлу.

Добавление информационной системы в проект

Первым шагом при внедрении информационной системы в игровой проект или проект геймификации прикладных образовательных материалов, является добавление пакета систем в общую иерархию проекта.

Добавление пакета осуществляется путём скачивания пакета с ресурса или получения иными путями и размещение его в одной из папок проекта. Папка может быть создана пользователем-разработчиком с желаемым, удобным названием.

Пакет добавляется в иерархию в виде набора объектов типа Prefab (шаблонов) и скриптов.

В большинстве случаев, скрипты уже установлены на объектах, за исключением случаев, когда скрипт подразумевает ситуативное включение.

Сразу после добавления пакета в проект, на сцену проекта необходимо добавить объекты диалоговой системы и их менеджеры: Dialogue_Eye, Dialogue_Mouth, EyesButtonManager и MouthButtonManager соответственно (рисунок 4.1).

Рисунок 4.1 – Добавление систем в проект

При добавлении систем на сцену разработки игрового проекта, объекты могут быть внесены в «папки» созданные при помощи расположения пустых объектов в иерархии. В представленном примере, менеджеры размещены в качестве дочерних объектов в «папке» ButtonsManagers.

Настройка объектов информационной системы

Для корректной работы информационной системы, необходима первоначальная настройка добавленных объектов. Поскольку скрипты, расположенные на объектах, имеют ссылки на другие объекты информационной системы, необходимо добавление объектов в соответствующие строки скриптов. Это производится при помощи пользовательского интерфейса. Настройка каждого объекта рассматривается отдельно. Общий принцип работы системы и принцип расположения скриптов также представлен на рисунке 3.12 в главе 3.

Объект Dialogue_Eye представлен в виде совокупности объектов Dialogue и Eye. Первый объект отвечает за хранение словаря ветви «осмотра» объектов. Второй объект является кнопкой в пользовательском интерфейсе диалоговой системы (рисунок 4.2).

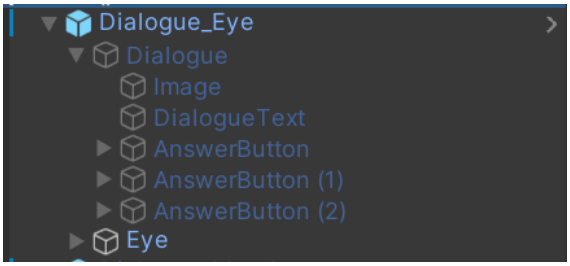

Рисунок 4.2 – Структура элемента Dialogue_Eye

Элемент Dialogue содержит ряд дочерних элементов: DialogueText и набор элементов AnswerButton. DialogueText представляется объектом текстового фрейма в пользовательском интерфейса, а набор объектов AnswerButton – набором кнопок для вариантов ответов.

На объекте Dialogue находится три скрипта Dialogue Story, Answer Buttons и Dialogue Switcher.

Dialogue Story не требует добавления объектов, но задействуется для внесения текстовой информации.

Answer Buttons требует добавления в массив всех кнопок вариантов ответов. В представленном примере реализации имеется три кнопки, но при наличии иного количества, массив поддерживает добавление меньшего или большего числа объектов.

Dialogue Switcher предназначен для создания выходных точек. Скрипт требует внесения тега конечного узла и объекта Dialogue, на котором расположен скрипт Dialogue Story (рисунок 4.3).

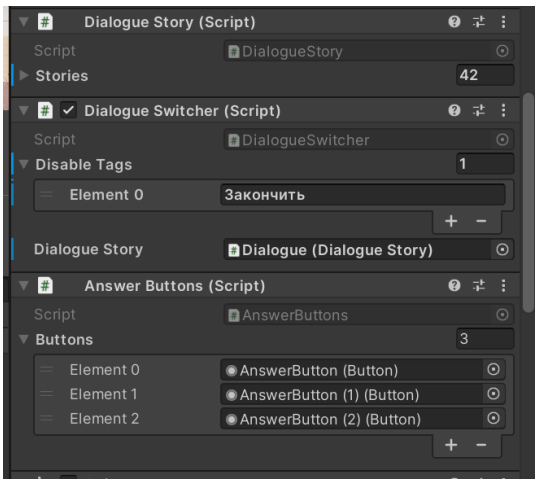

Рисунок 4.3 – Настройки компонента Dialogue

Объект DialogueText содержит скрипт DWindow. Скрипт запрашивает значение для переменной Delay, определяющей скорость посимвольного вывода текстовой информации. Пользовательский интерфейс функции представлен на рисунке 4.4.

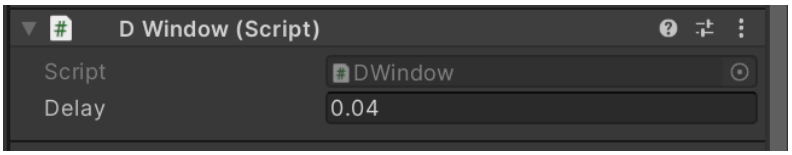

Рисунок 4.4 – Настройки компонента DialogueText

Объект Eye содержит два параметра, требующих настройки – настройка работы скрипта и настройка работы функции кнопки.

Объект содержит скрипт GetDialogueTag. Скрипт запрашивает добавление объекта менеджера EyesButtonManager. Настройка функций менеджера рассматривается отдельно.

Для настройки функций работы кнопки, необходимо обращение к встроенной функции OnClick() через пользовательский интерфейс. В функцию необходимо добавление объекта, содержащего исполняемый скрипт, объекта Eye. В исполняемых функциях необходим выбор функции GetDialogueTag.SendDialogueTag (рисунок 4.5).

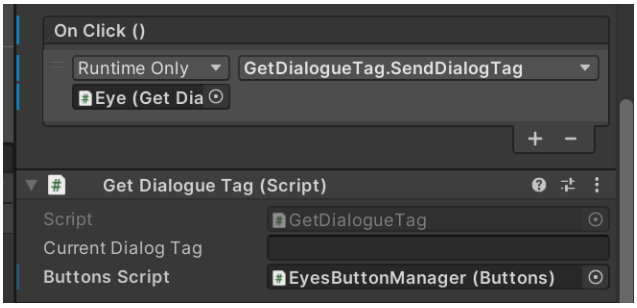

Рисунок 4.5 – Настройки элемента Eye

Далее, необходима настройка менеджера EyesButtonManager. Менеджер содержит скрипт Buttons. Скрипт запрашивает значения для полей Dialogue Box

и Dialogue Story. В каждое из данных полей добавляется объект Dialogue. В первом случае, объект определяется как объект интерфейса в игровом пространстве. Во втором случае – как объект, содержащий скрипт Dialogue Story (рисунок 4.6).

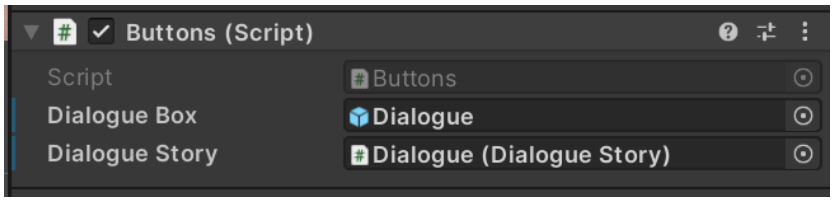

Рисунок 4.6 – Настройки компонента EyesButtonManager

Аналогичные настройки производятся для ветви Mouth. Внутреннее устройство ветви аналогично и задействует аналогичные функции и методы.

Структура компонента Dialogue_Mouth аналогична структуре компонента Dialogue_Eye. Компонент Dialogue_Mouth представлен в виде совокупности компонентов M_Dialogue и Mouth. Первый компонент выполняет функции пользовательского интерфейса диалоговой системы и функции хранения словаря ветви «диалога», а второй компонент представлен в виде кнопки «начала диалога».

Компонент M_Dialogue также содержит компонент DialogueText, на котором расположен скрипт DWindow. Скрипт требует тех же настроек, как и в случае с ветвью Eye. Также, компонент M_Dialogue содержит дочерние элементы набора кнопок, аналогичный набору кнопок в объекте Dialogue.

Компонент M_Dialogue содержит три скрипта Dialogue Story, Answer Buttons и Dialogue Switcher. Скрипты совпадают со скриптами на объекте Dialogue и требует аналогичных настроек, только в данном случае, все компоненты изымаются из иерархии компонента Dialogue_Mouth. Структура представлена на рисунке 4.7.

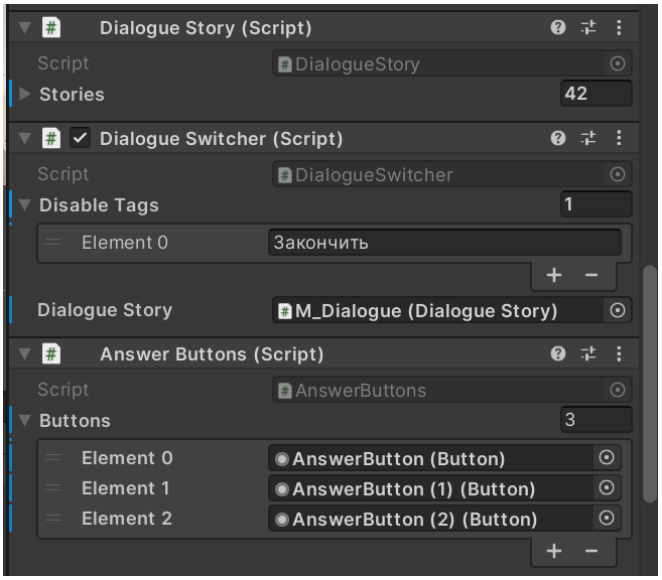

Рисунок 4.7 – Настройки компонента M_Dialogue

Настройка компонента Mouth также аналогична настройке компонента Eye, но обращение происходит к компонентам иерархии Dialogue_Mouth и к компоненту менеджера MouthButtonManager.

В графу Button Script скрипта GetDialogueTag помещается объект MouthButtonManager.

При настройках функций кнопки, в качестве объекта-носителя устанавливается объект Mouth, а в качестве исполняемой функции выбирается та же функция GetDialogueTag.SendDialogueTag (рисунок 4.8).

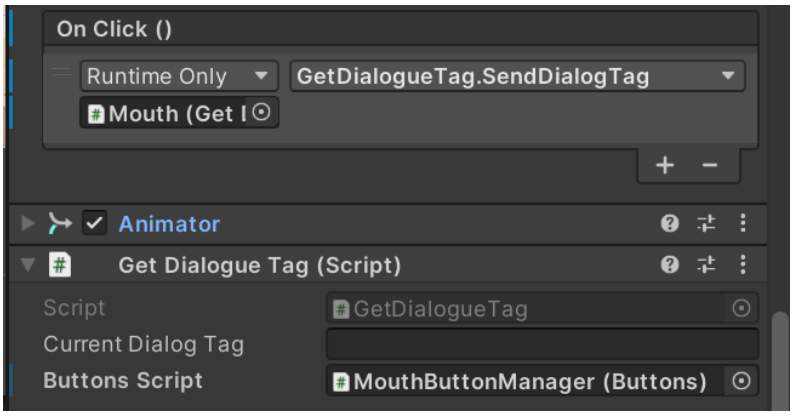

Рисунок 4.8 – Настройки компонента Mouth

Для настройки менеджера MouthButtonManager, в скрипт Buttons, графы Dialogue Box и Dialogue Story устанавливаются объект M_Dialogue. В первом случае объект считывается как объект интерфейса, а во втором – как носитель скрипта, выполняющего функции словаря данной диалоговой ветви (рисунок 4.9).

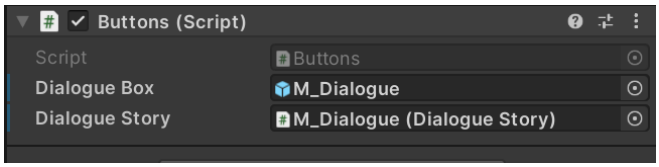

Рисунок 4.9 – Настройки компонента MouthButtonManager

Создание интерактивного объекта

Процесс внедрение диалоговой системы начинается с создания в игровом пространстве интерактивного объекта. Сам объект может быть представлен в виде спрайта или любого другого объекта.

Для создания интерактивной области на объекте, необходимо добавление компонента типа Collider на объект. При этом, коллайдер должен выполнять функции триггера. Пример коллайдера и его настроек представлен на рисунке 4.10.

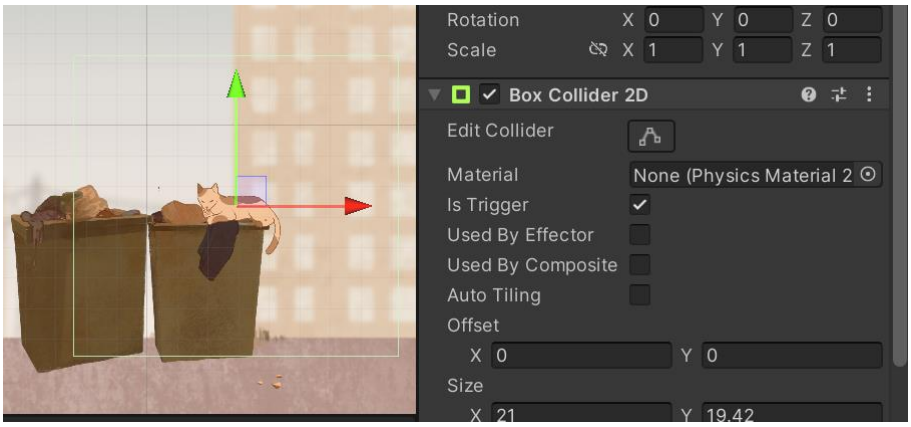

Рисунок 4.10 – Интерактивный объект с добавленным компонентом коллайдера

Добавленный коллайдер необходимо настроить по размерам интерактивной области. Интерактивная область подразумевает ту область, в которой пользователю-игроку будет доступно взаимодействие с объектом. Если

объект в игровом пространстве имеет небольшие размеры по отношению к модели игрового персонажа, необходима установка большего, по сравнению с интерактивным объектом, коллайдера.

Далее, в иерархию компонентов интерактивного объекта, необходимо добавить компонент скрипта Trigger. Данный скрипт выполняет функции соотнесения интерактивного объекта с первым узлом диалоговой ветви.

Для добавления скрипта, необходимо перетащить его объект из пакета информационных систем на список компонентов интерактивного объекта. Также, добавление может быть произведено при помощи кнопки добавления, расположенной в конце иерархии компонентов. Объект с добавленным скриптом представлен на рисунке 4.11.

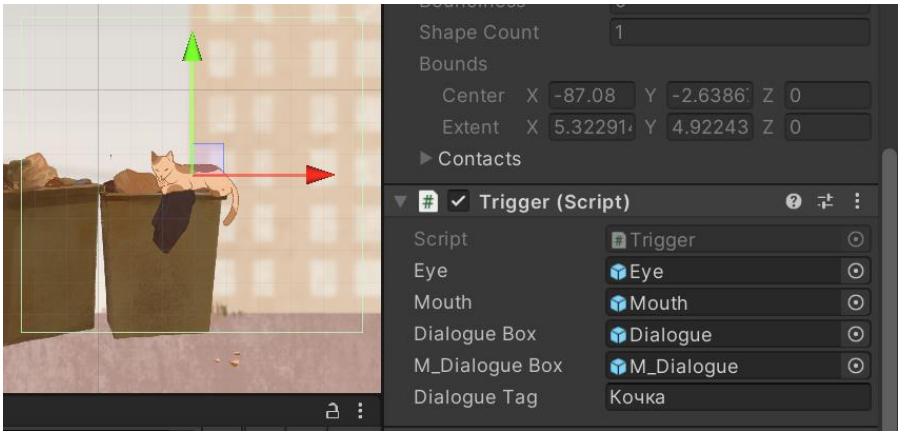

Рисунок 4.11 – Интерактивный объект с добавленным скриптом

Для корректной работы скрипта, необходимо добавление соответствующих объектов и тега. Объекты представляются в виде элементов, добавленных ранее в среду разработки. Для каждого объекта выделено своё поле. Для добавления объекта, необходимо перетащить его из списка элементов проекта в соответствующее поле триггера. Настройка всех компонентов произведена заранее (описание настройки компонентов указано выше).

В поле Eye необходимо внести компонент Eye из иерархии Dialogue_Eye. Из той же иерархии добавляется объект Dialogue в поле Dialogue Box. Из

иерархии компонентов Dialogue_Mouth добавляются два компонента – Mouth и M_Dialogue в графы Mouth и M_Dialogue Box соответственно.

Последнее поле предназначено для внесения первого тега. По данному тегу будет происходить обращение к диалоговой ветви. Тот же тег будет задействован для создания первого узла диалоговой ветви.

Поле заполняется текстовой строкой, наиболее удобной для пользователя. Текстовая строка может иметь любую длину и содержать любые символы.

Создание диалоговой ветви

Для создания диалоговой ветви, необходимо задействование ранее настроенных компонентов диалоговой системы и тега, внесённого в поле тега интерактивного объекта.

Диалоговые ветви создаются при помощи внесения информации в соответствующие поля компонентов Dialogue и M_Dialogue. Оба компонента содержат скрипт DialogueStory. Для создания диалоговой ветви, необходимо последовательное внесение текстовой информации в массив Stories.

Информация вносится в оба раздела Dialogue и M_Dialogue, но пользовательская текстовая информация может различаться.

При внесении информации, создаётся пустой компонент – узел диалоговой ветви (рисунок 4.12).

Рисунок 4.12 – Пустой узел диалоговой ветви

При создании первого узла, необходимо внесение тега, указанного на в скрипте интерактивного объекта, в поле Tag созданного узла. При внесении тега,

необходимо учитывать посимвольное сравнение тегов, при передаче информации внутри системы, поэтому теги должны полностью совпадать, включая написание.

В поле Text узла вносится текстовая информация, выводимая в основном текстовом фрейме пользовательского интерфейса. Данный текст будет доступен для пользователя и, как правило, обусловлен сюжетом игры.

Далее, необходимо добавление вариантов ответов, доступных пользователю при прохождении данного диалогового узла. Для добавления вариантов ответов, необходимо внесение численного значения количества вариантов ответов в графу Answers. Также, возможно последовательное добавление вариантов ответов через символ «+» списка Answers. В первом случае, в листе вариантов ответов появится необходимое количество графов для внесения информации. Во втором случае – графы будут появляться последовательно. Структура представлена на рисунке 4.13.

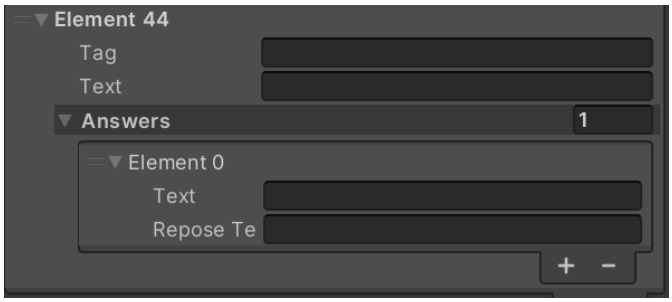

Рисунок 4.13 – Узел диалоговой ветви с одним графом варианта ответа

При добавлении вариантов ответов, необходимо учитывать доступное количество вариантов ответов, предусмотренное пользовательским интерфейсом. Если пользовательский интерфейс поддерживает до N вариантов ответов, текущий узел диалоговой ветви может содержать от 0 до N вариантов ответов. При количестве вариантов ответов, большего N, система может работать некорректно.

Далее, необходимо заполнение добавленных графов вариантов ответов. Для этого, текстовая информация, доступная для пользователя, помещается в

строку Repose Text, а тег переадресации, ведущий к следующему узлу диалогового дерева, помещается в строку Text. Тег переадресации недоступен для пользователя и является внутренней информацией системы. Тег может быть представлен в виде текстовой строки, удобной для пользователя-разработчика, но не несущей информации для пользователя-игрока. Принцип заполнения отображён на рисунке 4.14.

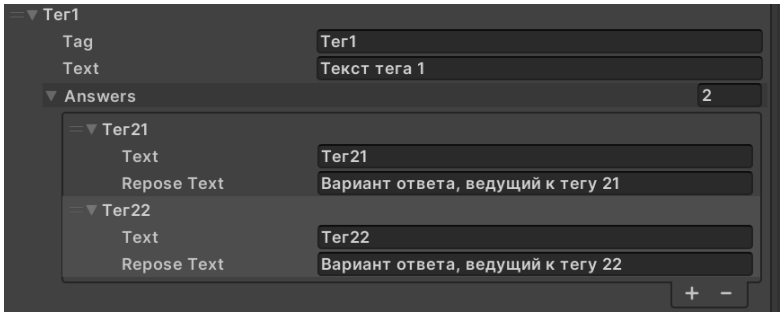

Рисунок 4.14 – Вид заполненного узла диалоговой ветви

Так как информация вносится в обе диалоговых ветви Dialogue и M_Dialogue, стоит отметить, что первый узел обеих ветвей имеет одинаковый тег с интерактивным объектом. Текстовая информация, предназначенная для пользователя-игрока и вносимая в узлы может различаться, но первый тег должен совпадать для обеспечения корректной работы системы. Также, может различаться и дальнейший ход и развитие ветвей.

После заполнения первого узла диалоговой ветви, необходимо создание последующих узлов (при их наличии) или выходных точек из диалоговой системы.

Далее рассмотрено создание последующих узлов диалоговой ветви. Создание выходных точек диалоговой системы рассмотрено отдельно.

Для создания последующего узла диалоговой ветви, необходим перенос тега переадресации с каждого варианта ответа в строку тега нового диалогового узла. Для добавления нового узла задействуется функция «+» в списке узлов или изменение числа узлов в графе Stories на большее (рисунок 4.15).

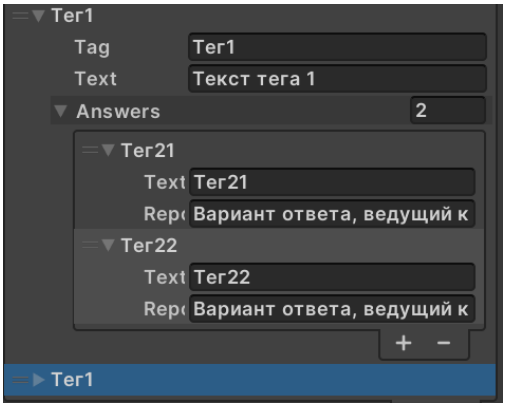

Рисунок 4.15 – Добавление последующего узла

При добавлении последующего узла, автоматически сохраняются настройки предыдущего узла диалоговой ветви. Для настройки нового узла, необходимо изменение значения графы Tag. Для корректной работы системы, необходимо указание того же тега, который был указан на соответствующем варианте ответа. Вариант ответа, на котором тег переадресации совпадает с тегом узла, является переходным к данному узлу.

Новые узлы также могут иметь набор вариантов ответов. При добавлении вариантов ответов необходимо соблюдение тех же условий добавления, как и при создании первого диалогового узла.

Варианты ответа могут иметь тег переадресации, ведущий на ранее созданные узлы. При этом, узлы могут находиться как в текущей диалоговой ветви, так и в параллельных ветвях. Пример структуры представлен на рисунке 4.16.

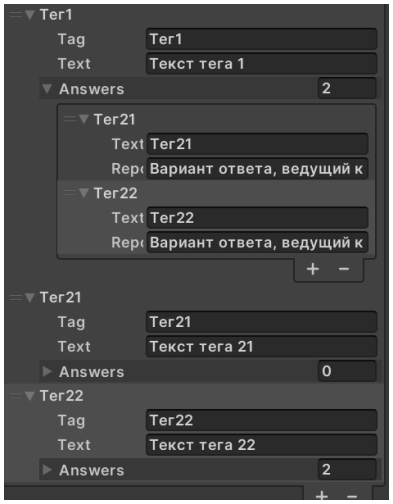

Рисунок 4.16 – Соотнесение вариантов ответов и последующих узлов

Последующие узлы диалоговой ветви заполняются аналогичным образом. При этом стоит учитывать такие аспекты, как:

→ Посимвольное совпадение тега переадресации варианта ответа с тегом последующего узла (на который совершается переходи при выборе данного варианта ответа пользователем-игроком);

→ Наличие последующего или выходного узла для каждого варианта ответа.

Создание выходного узла

При создании выходного узла из диалоговой системы, задействуется скрипт Dialogue Switcher. Данный скрипт запрашивает ввод строки тега, который является выходным узлом для диалоговой ветви.

Для ряда диалоговых ветвей, выходной узел может быть общим. При использовании одного выходного узла для нескольких диалоговых ветвей, необходимо чтобы каждый предшествующий узел содержал вариант ответа, тегом переадресации которого является тег выходного узла.

При создании выходного узла, необходимо создание диалогового узла по определённому принципу. Для данного узла не требуется заполнение поля Text, так как при переходе к данному узлу происходит завершение работы диалоговой системы на данный момент. Пример заполнения узла представлен на рисунке 4.17.

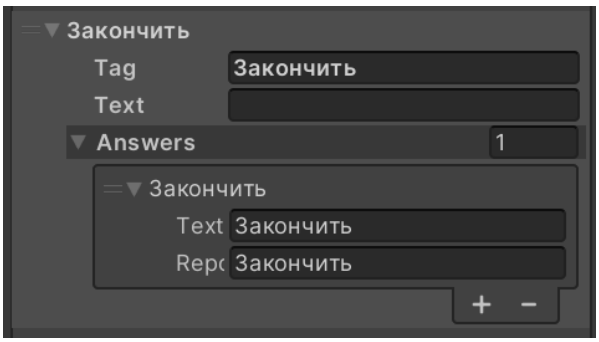

Рисунок 4.17 – Выходной узел в ветви диалога

При этом, необходимо наличие ссылки на самого себя внутри узла. Ссылка создаётся при помощи внесения в варианты ответа данного узла единственного варианта ответа, тегом переадресации которого является тег этого же узла.

При использовании выходного узла, необходимо наличие скрипта Dialogue Switcher на объекте Dialogue или M_Dialogue.

В скрипте указывается ссылка на Story, при этом стоит учитывать, в какой из иерархий Eye или Mouth находится данный выходной узел. Для каждой глобальной ветви, выходной узел уникальный и имеет указатель на свой параметр Story.

В скрипт выходного узла необходимо внесение того же тега, который стоит на выходном узле в диалоговой ветви (рисунок 4.18).

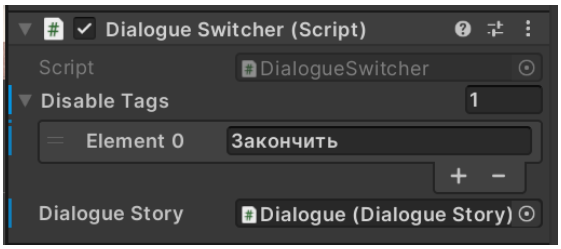

Рисунок 4.18 – Расположение тега на скрипте выходного узла

При Обращении к данному узлу, происходит завершение работы диалоговой системы на данном игровом этапе и выключение пользовательского интерфейса.

4.1.2 Внедрение системы квестов

Для установки системы квестов, необходимо добавление нескольких обязательных элементов информационной системы.

К обязательным компонентам системы относятся компоненты глобальной системы событий и компоненты установления связи между диалоговой информационной системой и информационной системой квестов.

Глобальная система событий представляется в виде объекта типа Prefab (шаблон) и может быть добавлена в проект путём добавления из пакета

информационной системы. Для добавления объекта, необходимо внесение его в папку иерархии проекта.

Для корректной работы глобальной системы событий, необходимо также добавление скрипта, определяющего создание объекта глобальной системы событий в среде разработки Unity. Глобальный объект и скрипт, задающий его функции, представлены на рисунке 4.19.

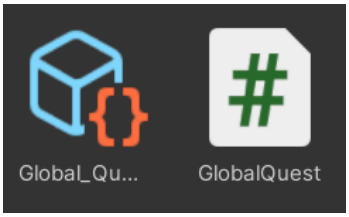

Рисунок 4.19 – Объект глобальной системы событий в иерархии проекта

Объект глобальной системы событий не требует добавления объекта на сцену проекта среды разработки.

Для корректной работы системы, необходимо добавление актуальных переменных событий в список переменных объекта. Для этого необходимо внесение названий переменных в тело объекта. При создании переменных событий, необходима установка начального, нулевого значения для каждой переменной, для корректной работы системы.

При изменении игрового проекта, возможно изменение, добавление или удаление переменных событий (рисунок 4.20).

Рисунок 4.20 – Список переменных событий в глобальной системе

После добавление глобальной системы событий, необходимо внесение элементов, позволяющих установить связь между узлами диалоговой системы и системой квестов.

Для этого, в среду разработки добавляется компонент ключа. Компонент ключа представлен в виде скрипта. Для корректной работы ключа, необходимо внесение скрипта в иерархию объектов Dialogue или M_Dialogue, в зависимости от того, к узлу какой из глобальных ветвей будет привязан текущий ключ.

Один ключ может быть привязан сразу к нескольким событиям. Если ключ уникален, то система требует отдельного добавления данного ключа в иерархию. Если один и тот же результат внутри игрового процесса достигается при прохождении различных диалоговых узлов, данные диалоговые узлы могут быть добавлены в один ключ. Для этого необходимо внесение тегов всех узлов в список тегов узла. Для добавления нескольких тегов возможно использование кнопки «+» в списке тегов узла или изменение числа тегов на большее (в таком случае, произойдёт добавление сразу нескольких строк для добавления тега). Каждая строка тега заполняется текстом, при этом текст тега в строке должен посимвольно совпадать со строкой тега на диалоговом узле. Интерфейс ключа представлен на рисунке 4.21.

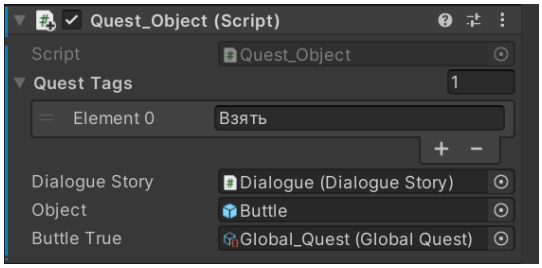

Рисунок 4.21 – Ключ в иерархии объектов Dialogue и M_Dialogue

При добавлении ключа, принцип работы с ним сход с принципом работы с выходными узлами диалоговой системы.

Необходимо добавление тега узла диалоговой системы, при прохождении которого происходит событие из системы квестов. При этом, в ключе

указывается имя переменной события, для изменения значения данной переменной на единицу и фиксации события в системе.

Скрипт ключа требует внесения двух объектов – словаря и объекта глобальной системы событий. Первый объект представляется в виде объекта Dialogue или M_Dialogue, в зависимости от иерархии ключа. Второй объект представляется объектом Global_Quest, находящимся в иерархии проекта. Принцип заполнения ключа представлен на рисунке 4.22.

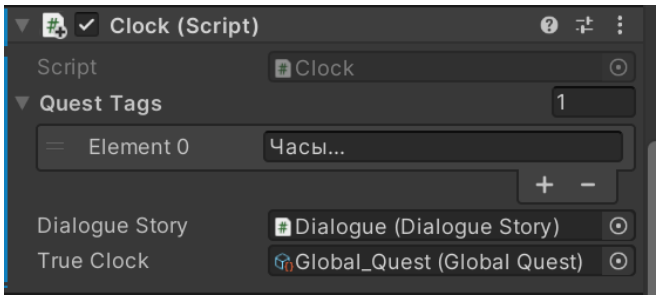

Рисунок 4.22 – Заполненный ключ

При добавлении ключа, возможно изменение его структуры и определение побочных задач и функций, выполняемых ключом при прохождении диалогового узла с указанным тегом.

Поскольку диалоговая информационная система служит системой вывода пользовательской информации для системы квестов, система квестов не требует внесения отдельных компонентов для реализации данных задач. Остальные компоненты информационной системы событий являются второстепенными и могут быть добавлены из пакета при необходимости использования в разрабатываемом игровом проекте или продукте геймификации прикладных образовательных материалов. Эти компоненты имеют меньшую адаптивность и предназначены для применения в конкретном игровом проекте, но на их основе могут быть разработаны схожие компоненты для других проектов.

4.2 Тестирование информационных систем

Для оценки работоспособности разработанных информационных систем, на их основе был разработан самостоятельный игровой проект.

Игровой проект выполнен в жанре «квест». Основной геймплей завязан на исследовании игрового окружения и прохождении игрового сюжета.

Интерактивные элементы игрового окружения реализованы при помощи разработанной диалоговой системы. В данном случае, диалоговая система выводит не только информацию диалогов, но и служит для вывода информации об игровом окружении.

Для оценки работоспособности систем, в игровом проекте добавлен основной игровой квест и множество диалоговых ветвей, служащих как описаниями, так и диалогами с неигровыми персонажами.

При прохождении диалоговой ветви, происходит корректный переход между узлами диалоговой ветви и корректный выход из диалоговой системы при завершении диалогов или осмотров игрового окружения.

Также, при прохождении узлов, на которых расположены игровые события, происходит корректное отображение фиксации игрового события внутри системы.

Для тестирования работоспособности пакета информационных систем, пакет был предоставлен компании-заказчику. Компания-заказчик произвела оценку работоспособности системы по отношению к проектам геймификации прикладных образовательных материалов. Для этого, тексты прикладных образовательных материалов, были добавлены в диалоговую информационную систему в соответствующем для работы виде. Для проверки работоспособности системы квестов, были созданы новые переменные событий, а на узлы диалоговой системы установлены ключи. При тестовом запуске системы, была получена полная работоспособность пакета информационных систем.

Компания-заказчик оценила работу пакета информационных систем как соответствующую требованиям и ожиданиям.

Для тестирования работоспособности игрового проекта, игра была размещена на электронном ресурсе itch.io. При этом был проведён мониторинг статистики игрового проекта. Также, была получена обратная связь от пользователей-игроков при помощи комментариев, оставленных на площадке. Игроками были обнаружены незначительные недочёты в работе игрового приложения. Недочёты были устранены в кротчайшие сроки после их выявления.

Игроками были выявлены проблемы оптимизации разрешения элементов интерфейса на некоторых сценах игрового приложения. Для устранения проблем, связанных с разрешением, были настроены параметры изменения элементов интерфейса при изменении разрешения всего экрана, установлены якоря для расположения элементов на экранах сцен.

После внесения изменений и устранении проблем, проект был обновлён на платформе. Проект поддерживает период мониторинга обратной связи с целью выявления ранее необнаруженных проблем.

Игровой проект имеет положительные отзывы и высокий рейтинг.

Заключение

Итогом работы является информационный пакет для геймификации прикладных образовательных материалов и самостоятельное игровое приложение, основанное на разработанном информационном пакете.

Проведена аналитика предметной области. Рассмотрены применимые для разработки инструменты и выполнен их сравнительный анализ на основании установленных задач. Установлены наиболее подходящие инструменты. Выполнен теоретический обзор на затрагиваемые в ходе работы темы.

Выполнено проектирование информационных систем с описанием теоретического решения поставленных задач. Проведены оценки преимуществ и недостатков методов проектирования, применимых для реализуемых информационных систем. Проведена структуризация систем и установление предполагаемых методов коммуникации между ними.

Выполнена программная реализация информационных систем на основе теоретического проектирования. Выбраны наиболее оптимальные методы решений, доступные при выбранных инструментах. Разработан информационный пакет, включающий все необходимые для разработки информационные системы, предназначенные для геймификации прикладных образовательных материалов. Разработан самостоятельный игровой проект на базе информационного пакета с добавлением индивидуальных игровых систем.

Проведено тестирование разработанных информационных систем и анализ полученных результатов. Получена обратная связь от пользователей информационного пакета и игрового приложения.

Список литературы

1. Яблоков К. В. Исторические компьютерные игры как способ моделирования исторической информации // История и математика: Анализ и моделирование социально-исторических процессов / Ред. Малков С. Ю., Гринин Л. Е., Коротаев А. В. — М.: КомКнига/УРСС, 2007. С. 263—303 (Дата обращения: 04.02.2024)
2. Popkin, Helen (2010-06-01). "FarmVille invades the real world". MSNBC. (Дата обращения: 06.04.2024)
3. Геймификация в образовании. Наука и жизнь. — 2018. № 12. (Дата обращения: 17.03.2024)
4. Дейкстра Э. Дисциплина программирования = A discipline of programming. — 1-е изд. — М.: Мир, 1978. — 275 с (Дата обращения: 18.03.2024)
5. Роберт У. Себеста. Основные концепции языков программирования / Пер. с англ. — 5-е изд. — М.: Вильямс, 2001. — 672 с. — ISBN 5-8459-0192-8 (рус.) ISBN 0-201-75295-6 (англ.). (Дата обращения: 18.04.2024)
6. Рассмотрение различных жанров компьютерных игр. Режим доступа: https://gb.ru/blog/zhanry-kompjuternyh-igr/
7. Описание и сравнительная характеристика игровых движков Unity и Unreal. Режим доступа: https://gamecreating.ru/unity-vs-unreal-kakoj-dvizhok-vybrat-nachinayushhemu-razrabotchiku/
8. Руководство Unity. Встроенные библиотеки в среде разработки Unity. Режим доступа: https://docs.unity3d.com/ru/530/Manual/PresetLibraries.html
9. Руководство Unity. Единое пространство имён в среде разработки Unity. Режим доступа: https://docs.unity3d.com/ru/2018.4/Manual/Namespaces.html
10. Сериализация классов и имён в среде разработки Unity. Режим доступа: https://otus.ru/nest/post/2561/
11. Руководство Unity. Работа с компонентом для создания и настройки анимаций Animator в среде разработки Unity. Режим доступа: https://docs.unity3d.com/ScriptReference/Animator.html

12. Дональд Э. Кнут. Глава 2.3. Деревья // Искусство программирования = The Art of Computer Programming. — 3-е изд. — М.: Вильямс, 2002. — Т. 1. Основные алгоритмы. — 720 с. — ISBN 5-8459-0080-8 (рус.) ISBN 0-201-89683-4 (англ.). (Дата обращения: 24.02.2024)
13. Дональд Э. Кнут. Глава 2.3. Деревья // Искусство программирования = The Art of Computer Programming. — 3-е изд. — М.: Вильямс, 2002. — Т. 1. Основные алгоритмы. — 720 с. — ISBN 5-8459-0080-8 (рус.) ISBN 0-201-89683-4 (англ.). (Дата обращения: 14.04.2024)
14. Геймификация прикладных образовательных материалов с применением технологий системы квестов в игровом пространстве. Режим доступа: https://science-education.ru/ru/article/view?id=26865

Приложение А. Техническое задание

1. Наименование и область применения

Разработка программного обеспечения игрового проекта «ЛАВЭ».

Применимо как пакет для геймификации прикладных образовательных материалов и как самостоятельное игровое приложение.

2. Основания для разработки

Основанием для разработки является задание для дипломного проекта, утвержденное кафедрой «Информатика и информационные технологии» Московского политехнического университета.

3. Исполнитель

Исполнителем является студент кафедры «Информатика и информационные технологии» Московского политехнического университета группы 201-726 Курносова Арсения Валерьевна

4. Назначение разработки

4.1. Назначение системы

Геймификация прикладных образовательных материалов;

Самостоятельное игровое приложение.

4.2. Цели создания системы

Создания пакета для геймификации прикладных образовательных материалов и разработка самостоятельного игрового приложения.

5. Технические требования к системе

5.1. Задачи, подлежащие решению

Разработка диалоговой информационной системы;

Разработка системы квестов;

Разработка второстепенных информационных систем для создания полноценного игрового приложения и наполнения его разнообразными геймплейными решениями.

5.2. Требования к функциональным характеристикам

Наиболее высокий уровень адаптивности основных информационных систем для дальнейшего использования в качестве информационного пакета для геймификации прикладных образовательных материалов;

Выполнение второстепенными информационными системами задач по управлению игровым проектом и наполнению его различными геймплейными решениями.

5.3. Требования к входным и выходным данным

В качестве входных данных, основными информационными системами принимаются данные пользователя-разработчика. Данные относятся к прикладным материалам, необходимым для геймификации;

В качестве выходных данных, принимается работа основных информационных систем, представленная в виде готового гейплейного решения, содержащего информацию из прикладных образовательных материалов.

5.4. Требования к составу программных компонентов

Программные компоненты разделены на основные и второстепенные. Основные программные компоненты формируют информационный пакет для геймификации прикладных образовательных материалов.

Второстепенные информационные системы формируют гейплейное наполнение самостоятельного игрового приложения, основанного на основных информационных системах.

5.5. Требования к компонентам программного продукта

Программный продукт представлен в виде двух основных компонентов – пакета информационных систем, предназначенных для геймификации прикладных образовательных материалов, и самостоятельного, полноценного игрового продукта, основанного на информационном пакете и служащего демонстрационной моделью работоспособности пакета.

5.6. Требования к составу и параметрам технических средств

5.6.1. Требования к составу программного обеспечения

Основные информационные системы – диалоговая информационная система и система квестов;

Второстепенные информационные системы – системы для обеспечения полноценной работы игрового приложения и его различного геймплейного наполнения.

5.6.2. Требования к аппаратному обеспечению

Информационный пакет предназначен для дальнейшего использования в среде разработки Unity. Игровое приложение предназначено для установки и запуска на персональном компьютере с операционной системой Windows.

5.7. Условия эксплуатации

Пакет для геймификации предназначен для дальнейшего использования другими разработчиками в среде разработки Unity.

Игровое приложение предполагает самостоятельное использование свободным игроком.

6. Стадии и этапы разработки

Название этапа	Даты реализации этапа
Получение технического задания	04.09.2023
Аналитика	23.09.2023 – 24.10.2023
Теоретическое проектирование	20.10.2023 – 12.11.2023
Согласование	14.10.2023 – 12.11.2023
Практическая разработка	04.02.2024 – 28.04.2024
Тестирование	28.04.2024 – 06.06.2024
Подготовка отчётной документации	06.06.2024 – 12.05.2024

7. Техническая документация, предоставляемая по окончанию работы

Документация, предоставляемая по окончанию работы представлена в виде отчёта, соответствующем требованиям ВКР.

Приложение Б. Презентация проекта

Федеральное государственное автономное образовательное учреждение высшего образования «Московский политехнический университет»

Разработка программного обеспечения игрового проекта «ЛАВЭ»

Автор: Курносова Арсения Валерьевна
Студент группы 201-726
Научный руководитель: Семенов С.М.
Кандидат технических наук, ст. преподаватель каф. ИиИТ

Введение

- Геймификация прикладных образовательных материалов
- Проектирование адаптивных, универсальных информационных систем
- Проект, предназначенный для образовательной организации

Актуальность

- Геймификация - основная тенденция развития современного дистанционного электронного образования
- Компьютерные игры – один из наиболее популярных способов организации досуга

Цель

Разработка адаптивной ИС для дальнейшего применения в геймификации прикладных образовательных материалов, разработка игрового приложения.

Задачи

- Разработка основных информационных систем (диалоговая система и система квестов)
- Разработка второстепенных информационных систем

Вариативная диалоговая система

Вариативная диалоговая система

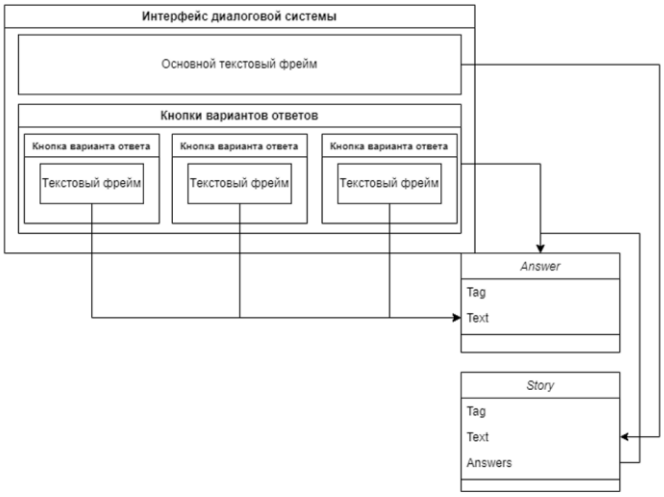

Рисунок 3.3 – Система хранения информации в диалоговой системе

Вариативная диалоговая система

Рисунок 3.6 – Принцип построения ветви диалогового дерева

Система квестов

Система квестов

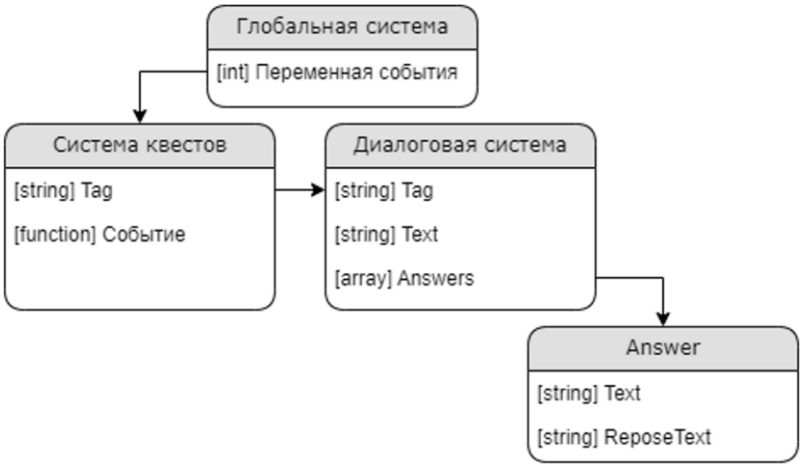

Рисунок 3.14 – Принцип работы глобальной системы событий

Второстепенные информационные системы

МЕНЮ и СИСТЕМА

- Системы основного меню
- Системы внутриигрового меню
- Система сохранения игры

УПРАВЛЕНИЕ и СЦЕНЫ

- Система управления персонажем
- Система переходов между сценами

ИНФОРМАЦИЯ

- Системы вывода информации
- Системы звуков

Отзывы

- ВКР **принято** компанией-заказчиком **в полном объёме**
- Получены положительные пользовательские отзывы на игровой проект, **производится техническая поддержка** проекта

Вывод

- Разработаны **основные** игровые системы, выполняющие основные функции геймплея
- Разработаны **второстепенные** игровые функции, отвечающие за общее наполнение игрового – демонстрационного приложения

- Разработан **полный пакет адаптивных информационных систем** для создания геймифицированных образовательных материалов

Приложение В. Техническое задание компании-заказчика