МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
**«Южно-Уральский государственный университет
(национальный исследовательский университет)»**
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

ДОПУСТИТЬ К ЗАЩИТЕ

Заведующий кафедрой, д.ф.-м.н.,
профессор

_____ Л.Б. Соколинский

«___» _____ 2024 г.

Разработка компьютерной игры «Chaos Rising» на платформе Unity

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 09.03.04.2024.308-581.ВКР

Научный руководитель,
доцент кафедры СП, к.т.н.
_____ М.В. Сухов

Автор работы,
студент группы КЭ-404
_____ Я.Д. Стрельцов

Ученый секретарь
(нормоконтролер)
_____ И.Д. Володченко
«___» _____ 2024 г.

Челябинск, 2024 г.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

УТВЕРЖДАЮ
Зав. кафедрой СП
_____ Л.Б. Соколинский
29.01.2024 г.

ЗАДАНИЕ
на выполнение выпускной квалификационной работы бакалавра
студенту группы КЭ-404
Стрельцову Ярославу Денисовичу,
обучающемуся по направлению
09.03.04 «Программная инженерия»

1. **Тема работы** (утверждена приказом ректора от 22.04.2024 г. № 764-13/12)

Разработка компьютерной игры «Chaos Rising» на платформе Unity.

2. **Срок сдачи студентом законченной работы:** 03.06.2024 г.

3. **Исходные данные к работе**

3.1. Руководство Unity. [Электронный ресурс] URL: https://docs.unity3d.com/ru/530/Manual/index.html (дата обращения: 29.01.2024 г.).

3.2. Unity Learn. [Электронный ресурс] URL: https://learn.unity.com/ (дата обращения: 29.01.2024 г.).

3.3. Первые шаги в Unity. [Электронный ресурс] URL: https://unity.com/ru/learn/get-started (дата обращения: 29.01.2024 г.).

3.4. Руководство Asset Store. [Электронный ресурс] URL: https://unity.com/ru/pages/introduction-to-asset-store (дата обращения: 29.01.2024 г.).

4. **Перечень подлежащих разработке вопросов**

4.1. Провести анализ предметной области.

4.2. Провести анализ методов разработки и архитектуру компьютерной игры.

4.3. Сформулировать и описать концепцию игры.

4.4. Разработать скрипты и интегрировать их в проект.

4.5. Реализовать компьютерную игру и провести ее тестирование.

5. Дата выдачи задания: 29.01.2024 г.

Научный руководитель,
доцент кафедры СП, к.т.н. М.В. Сухов

Задание принял к исполнению Я.Д. Стрельцов

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность

В современном мире компьютерные игры играют не только роль развлечения, но и становятся важным элементом культуры и социальной активности. Благодаря постоянному развитию технологий, игровая индустрия переживает взрывной рост, привлекая миллионы игроков со всего мира.

Unity [1] является одним из наиболее популярных инструментов среди игровых разработчиков. Платформа Unity отличается гибкостью, простотой использования и возможностями для создания высококачественной графики делают ее идеальным выбором для проектов любого масштаба. Благодаря активному сообществу разработчиков и обширным возможностям платформы, разработка игры на Unity становится более эффективной и продуктивной, поэтому платформа Unity была выбрана для разработки.

Кроме того, игровая индустрия продолжает демонстрировать стабильный рост и высокие темпы развития. Популярность игр как формы развлечения и социальной активности только увеличивается, привлекая новых игроков и инвесторов.

Таким образом, разработка компьютерной игры «Chaos Rising» на платформе Unity является актуальной и перспективной задачей, отвечающей запросам современной аудитории и потребностям развивающейся игровой индустрии.

Постановка задачи

Целью выпускной квалификационной работы является разработка компьютерной игры «Chaos Rising» на платформе Unity.

Для достижения поставленной цели необходимо решить следующие задачи.

1. Провести анализ предметной области.
2. Провести анализ методов разработки и архитектуру компьютерной игры.
3. Сформулировать и описать концепцию игры.

4. Разработать скрипты и интегрировать их в проект.
5. Реализовать компьютерную игру и провести ее тестирование.

Структура и содержание работы

Работа состоит из введения, четырех глав, заключения и списка литературы. Объем работы составляет 56 страниц, объем списка литературы – 15 источников.

В первой главе «Анализ предметной области и существующих работ по тематике ВКР» делается обзор аналогичных проектов, рассматривается предметная область, а также выполняется анализ существующих решений для реализации проекта.

Вторая глава «Анализ требований к программной системе» содержит описание организации работы инди хоррора, диаграмму вариантов использования и use-case диаграммы, которые помогут описать основные требования к программной системе.

В третьей главе «Архитектура программной системы» описана диаграмма классов и компонентов приложения, которая дает представление о структуре и взаимодействии элементов системы.

В четвертой главе «Реализация системы» приведены листинги реализации компонентов системы, а также представлены подробное описание работы их алгоритмов.

В пятой главе «Тестирование системы» приведено функциональное тестирование реализованной системы подтверждающее корректность и эффективность разработанного игрового приложения.

В приложении приведены листинги всех скриптов, а также таблицы спецификаций вариантов использований.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ И СУЩЕСТВУЮЩИХ РАБОТ

1.1. Описание предметной области и анализ аналогов

Целью работы является разработка инди-хоррора на платформе Unity. Индустрия компьютерных игр постоянно расширяется и развивается, привлекая внимание миллионов игроков по всему миру. В последние годы особенно выделяется жанр хоррора, который пользуется огромной популярностью благодаря своей способности создавать напряжение.

Разработка инди-хоррор игр представляет собой опыт, объединяющий в себе элементы искусства, дизайна, программирования.

Unity предоставляет разработчикам широкий спектр инструментов для реализации их творческих идей. С его помощью можно создавать как 3D, так и 2D графику, анимации, эффекты, а также реализовывать сложные геймплейные механики и системы искусственного интеллекта для монстров и персонажей.

Таким образом, разработка инди-хоррор игры на платформе Unity открывает перед разработчиками увлекательные перспективы, позволяя создавать захватывающие и запоминающиеся игровые проекты, способные погрузить игроков в мир ужаса.

Для проведения сравнительного анализа аналогов инди-хоррор игр на платформе Unity, рассмотрим несколько популярных проектов.

1. «Five Nights at Freddy's» [2]:

– одна из самых известных инди-хоррор игр, созданная Скоттом Коутоном;

– основной акцент на механике выживания и стратегии, где игрок должен выжить в темном помещении, управляя системой видеонаблюдения и закрывая двери перед нападением аниматроников;

– использует простую, но эффективную графику и звуковое оформление для создания атмосферы ужаса.

2. «Among the Sleep» [3]:

– уникальная игра, где игрок управляет маленьким ребенком, исследующим мир с его точки зрения;

– атмосферный и непредсказуемый сюжет, создающий напряжение и ужас;

– использует красочную графику и звуковое сопровождение, чтобы погрузить игрока в захватывающий мир кошмаров и фантазий.

3. «Layers of Fear» [4]:

– психологический хоррор, в котором игрок исследует дом художника, становясь свидетелем его психического распада;

– уникальный геймплей, основанный на исследовании окружающей среды и разгадывании головоломок;

– высококачественная графика и звуковое оформление, создающие невероятно атмосферный и ужасающий опыт.

4. «Darkwood» [5]:

– survival horror с верхним видом, в котором игроки выживают в загадочном лесу, пытаясь выяснить причины странного происходящего;

– нестандартный подход к игровой механике, включая динамическую генерацию мира и уникальные способы борьбы с ужасами леса;

– затягивающий сюжет и атмосферное звуковое оформление, создающие настоящее чувство ужаса и беспокойства.

Все эти игры предлагают уникальные и захватывающие игровые опыты в жанре хоррора. Каждая из них обладает своими особенностями в геймплейной механике, атмосфере и сюжете, что делает их интересными для исследования и вдохновения при разработке собственной игры «Chaos Rising» на платформе Unity.

1.2. Анализ существующих решений для реализации проекта

Для реализации проекта «Chaos Rising» на платформе Unity существует несколько инструментов и решений, которые могут быть использованы разработчиками. Рассмотрим основные из них:

1) Unity Engine [6]:

– обладает широким набором функциональных возможностей, включая редактор для создания сцен, графический интерфейс для программирования игровой логики, инструменты для создания анимаций и визуальных эффектов;

– поддерживает разработку игр для различных платформ, таких как ПК, мобильные устройства, консоли и виртуальная реальность;

2) Asset Store [7]:

– Unity Asset Store предлагает разработчикам широкий выбор готовых ресурсов, таких как 3D и 2D модели, текстуры, анимации, звуковые эффекты и готовые решения для различных геймплейных механик;

– это позволяет существенно ускорить процесс разработки и сэкономить время и ресурсы на создании всего с нуля;

3) C# [8] и Unity Scripting [9]:

– Unity поддерживает язык программирования C#, который используется для написания игровой логики, управления объектами и взаимодействия между ними [10];

– благодаря этому разработчики могут создавать сложные и интересные геймплейные механики, реализовывать искусственный интеллект врагов и персонажей, а также создавать собственные алгоритмы поведения объектов в игре.

Вывод по первой главе

В первой главе проведен детальный анализ предметной области, который включает в себя рассмотрение примеров существующих игр и выделение их ключевых особенностей.

2. АНАЛИЗ ТРЕБОВАНИЙ К ПРОГРАММНОЙ СИСТЕМЕ

Данный раздел направлен на детальный анализ требований пользователей для определения ключевых особенностей и функциональности нашего приложения – инди хоррор игры «Chaos Rising». Целью этого анализа является обеспечение эффективного и удобного пользовательского опыта.

2.1. Функциональные требования к проектируемой системе

Система должна отвечать следующим требованиям.

1. Игрок должен иметь возможность управлять движением персонажа с помощью клавиатуры. Персонаж должен иметь возможность бегать, ходить и останавливаться в зависимости от управления персонажем игроком.

2. Противники в игре должны обладать искусственным интеллектом (ИИ), позволяющим им реагировать на действия игрока и преследовать его в соответствии с заранее определенными поведенческими шаблонами. ИИ должен быть способен адаптироваться к действиям игрока, сохраняя при этом баланс между сложностью и удовлетворением игрового процесса.

3. Игра должна предоставлять графические и звуковые эффекты, чтобы создать атмосферу ужаса и напряжения. Визуальные и звуковые эффекты должны быть взаимосвязаны с игровым процессом и действиями игрока, чтобы усилить воздействие на игрока.

4. Игра должна предоставлять возможность настройки пользовательского интерфейса, таких как разрешение экрана. Пользователи должны иметь возможность адаптировать интерфейс под свои потребности и предпочтения, чтобы обеспечить максимальный комфорт игры.

2.2. Нефункциональные требования к проектируемой системе

Система должна отвечать следующим требованиям.

1. Игра должна обеспечивать стабильную производительность на различных конфигурациях компьютеров. Минимальные требования к аппаратному обеспечению должны быть определены для обеспечения приемлемой игровой производительности на целевой платформе.

2. Игра должна обеспечивать высокую степень надежности работы, минимизируя количество ошибок и сбоев в работе. Приложение должно быть способно обрабатывать экстремальные сценарии использования.

3. Интерфейс игры должен быть интуитивно понятным и легким в использовании даже для новых игроков. Игра должна предоставлять возможность настройки интерфейса для удовлетворения различных предпочтений пользователей.

4. Игра должна адаптироваться под различные типы экранов и разрешения, чтобы обеспечить одинаковое качество игрового опыта на различных устройствах.

5. Игра должна обеспечивать адекватное управление персонажем и другими игровыми элементами с помощью клавиатуры и компьютерной мыши.

6. Графические эффекты должны иметь высокое разрешение, а звуковые эффекты должны иметь высокое качество для создания атмосферного игрового мира.

2.3. Описание функциональности игры

На основании описанных в пункте 2.1 функциональных требований, была разработана диаграмма вариантов использования инди хоррора «Chaos Rising». UML диаграмма [11] показана на рисунке 1.

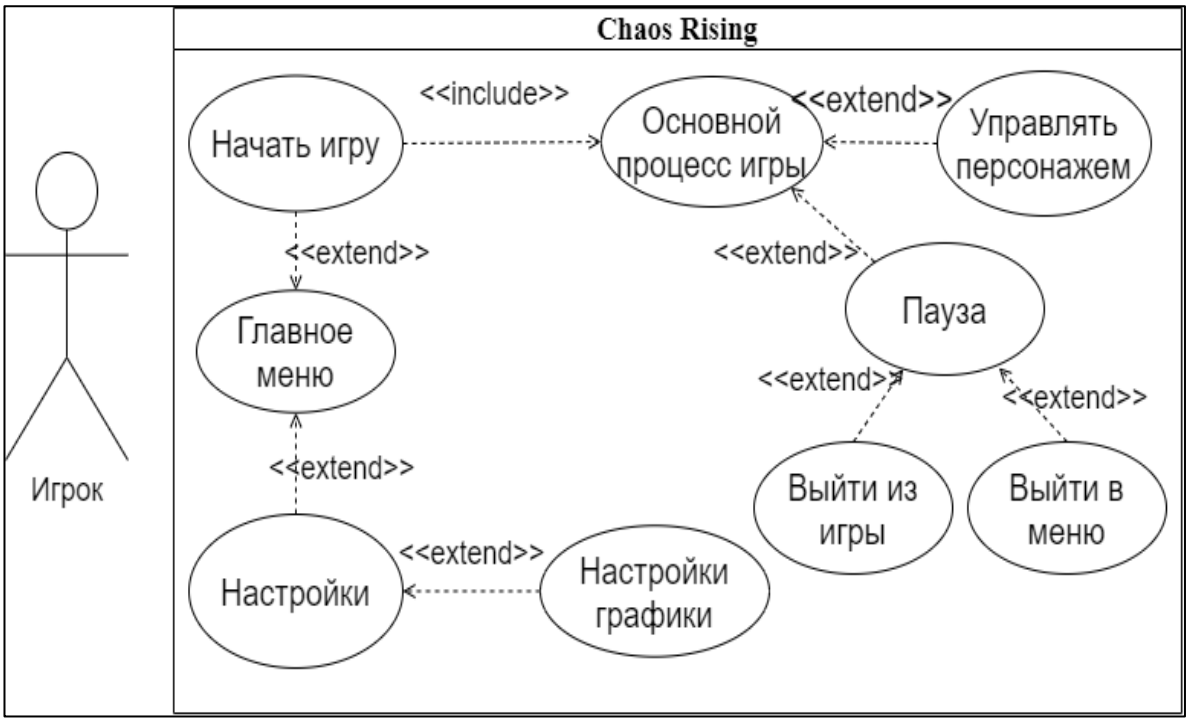

Рисунок 1 – UML диаграмма вариантов использования

Игрок – это ключевой участник игрового процесса, который контролирует действия игрового персонажа и взаимодействует с игровым миром. Его целью является успешное завершение игровых задач и преодоление препятствий, стоящих на его пути.

Монстр – это объект с искусственным интеллектом, который служит в качестве противника игрока в игровом мире. Он создает угрозу для игрока, препятствуя достижению игровых целей.

Далее приведено краткое описание вариантов использования.

1. Начать игру. Игрок начинает новую игровую сессию.

2. Основной процесс игры. Основная часть игрового процесса, где игрок взаимодействует с игровым миром и выполняет основные задачи.

3. Пауза. Игрок временно останавливает игровой процесс.

4. Управлять персонажем. Игрок контролирует движение и действия игрового персонажа, используя управление клавиатурой и мышью.

5. Настройки. Игрок настраивает параметры игры по своему усмотрению.

6. Настройки графики. Игрок может регулировать графические параметры игры, такие как разрешение экрана, уровень детализации, для достижения оптимального визуального опыта.

7. Качество графики. Игрок настраивает качество графики в соответствии с его конфигурацией компьютера.

8. Выход из игры. При нажатии меню паузы, игрок может закончить игровой процесс и выйти на рабочий стол.

9. Выход в главное меню. При нажатии меню паузы игрок может выйти в главное меню.

В приложении А представлены спецификации основных вариантов использования для инди-хоррора «Chaos Rising». Внутри приведено подробное описание ключевых действий, доступных игроку во время игрового процесса, и включает в себя информацию о главных актерах и их взаимодействии с игровой системой. Анализ этих вариантов использования позволяет пользователям лучше понять функциональные возможности игры и ознакомиться с тем, какие действия им предстоит выполнить в процессе игры. Благодаря этому игроки смогут эффективнее ориентироваться в игровом мире.

Вывод по второй главе

В данном разделе выявлены и описаны основные функциональные и нефункциональные требования к создаваемой системе. Определены ключевые аспекты, которые необходимо реализовать для достижения целевой функциональности игры.

3. АРХИТЕКТУРА ПРОГРАММНОЙ СИСТЕМЫ

3.1 Общее описание архитектуры системы

Игровой процесс разделен на несколько сцен, каждая из которых выполняет определенные функции и имеет свою логику работы.

Главное меню – представляет собой интерфейс, в котором игрок может выбрать различные действия, такие как: настройка игры, выход из игры и продолжение игры. Эта сцена содержит три кнопки: «Продолжить игру» «Настройки» и «Выйти из игры». Настройки включают в себя параметры качества графики, разрешение и режим полноэкранного режима.

Вступительная кат-сцена – в этой сцене игрок встречается с изображениями и субтитрами, которые вводят его в сюжет игры. Эта кат-сцена служит для создания атмосферы игры и предварительного ознакомления с историей.

Основная игра – представляет собой геймплейную сцену, где игрок управляет главным героем, исследуя лес, собирая записки и убегая от монстра. В этой сцене также присутствует меню паузы, которое позволяет игроку остановить игровой процесс и настроить параметры игры.

Финальная кат-сцена – в этой сцене игрок также встречается с изображениями и субтитрами, которые завершает игру и представляет развитие сюжета. В этой сцене игроку показывается завершение истории.

Архитектура системы игры «Chaos Rising» базируется на модели «сценарий-игра». Каждая сцена представляет определенный игровой этап и содержит соответствующую логику и ресурсы. Используется модульная структура, позволяющая легко добавлять новые сцены и функциональность в игру.

Каждая сцена взаимодействует с главным контроллером игры, который управляет переходами между сценами и обменом информацией между ними. Вся логика игры разделена на модули, что обеспечивает ее гибкость и масштабируемость.

Архитектура также включает в себя систему управления ресурсами, обеспечивающую эффективное использование аппаратных ресурсов и оптимизацию производительности игры.

Архитектура программной системы игры «Chaos Rising» представляет собой модульную и гибкую структуру, способную эффективно управлять ресурсами и обеспечивать высокую производительность и качество игрового опыта. Разделение игры на сцены позволяет эффективно организовать игровой процесс и обеспечить логичное развитие сюжета.

3.2. Описание компонентов, составляющих систему

На рисунке 2 представлена диаграмма компонентов игрового приложения.

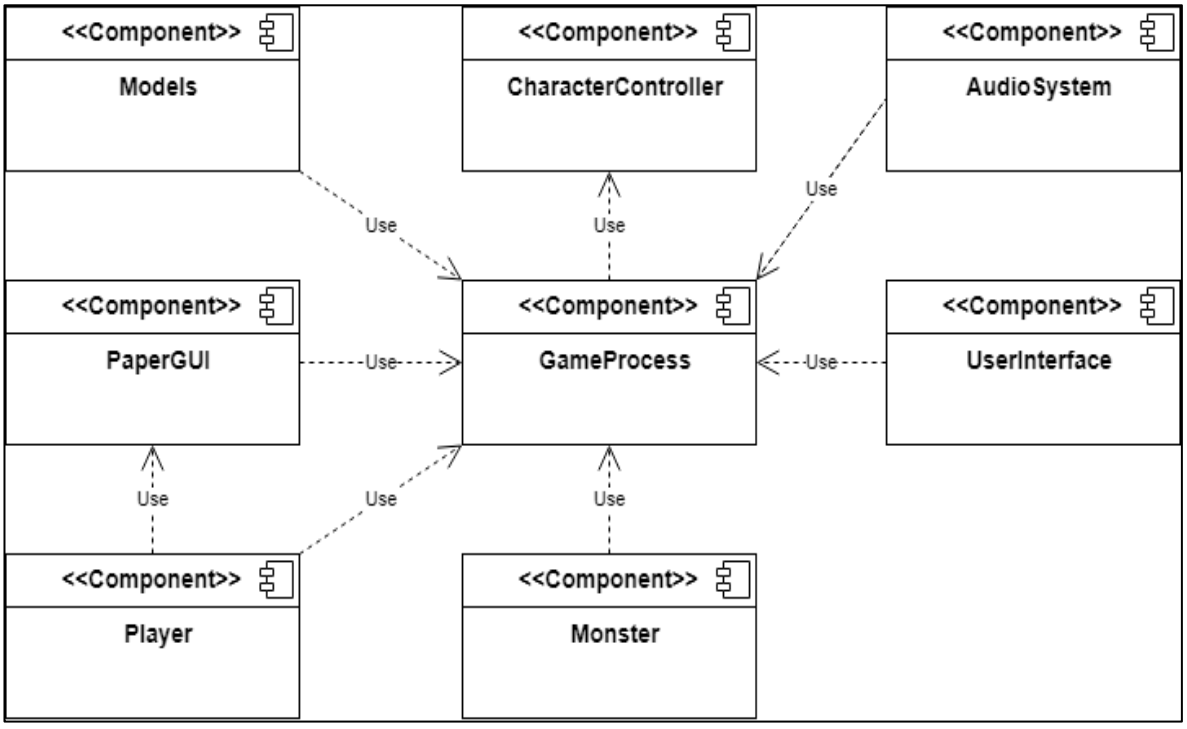

Рисунок 2 – Диаграмма компонентов

Данные компоненты содержат в себе классы, описывающие поведение, свойства, параметры и зависимости игровых объектов.

Далее рассмотрена диаграмма более подробно и приведено описание компонентов, составляющих систему.

`GameProcess` (Игровой процесс) – компонент, который управляет основной логикой игры. Принимает входные данные от пользователя и обрабатывает их. Взаимодействует с другими компонентами для управления игровым миром.

`CharacterController` (Управление персонажем) – отвечает за управление движением и действиями игрового персонажа. Принимает команды от игрока и передает их в игровой процесс для выполнения.

`Models` (Модели) – содержит модели персонажей, объектов и окружения в игре. Обеспечивает визуальное представление игрового мира.

`AudioSystem` (Аудио система) – отвечает за воспроизведение звуковых эффектов и музыки в игре. Создает атмосферу и улучшает иммерсию игрового опыта.

`UserInterface` (Пользовательский интерфейс) – содержит элементы интерфейса, такие как кнопки, текстовые поля и т. д. Обеспечивает взаимодействие игрока с игровым миром и настройками игры.

`PaperGUI` (Графический интерфейс для записок) – отображает интерфейс для сбора игровых записок. Взаимодействует с игровым процессом для обработки событий, связанных с записками.

`Player` (Игрок) – представляет игрового персонажа, которым управляет игрок. Отвечает за взаимодействие игрока с игровым миром и сбор записок.

`Monster` (Монстр) – представляет враждебного персонажа в игре. Реализует логику поведения монстра и его взаимодействие с игровым миром.

Эти компоненты взаимодействуют между собой, чтобы обеспечить правильное функционирование игры «Chaos Rising».

3.3. Диаграмма деятельности

Диаграмма деятельности – это визуальное представление последовательности действий или процессов, которые происходят в системе. На рисунке 3 представлена диаграмма деятельности основных игровых процессов и взаимодействий игрока с системой.

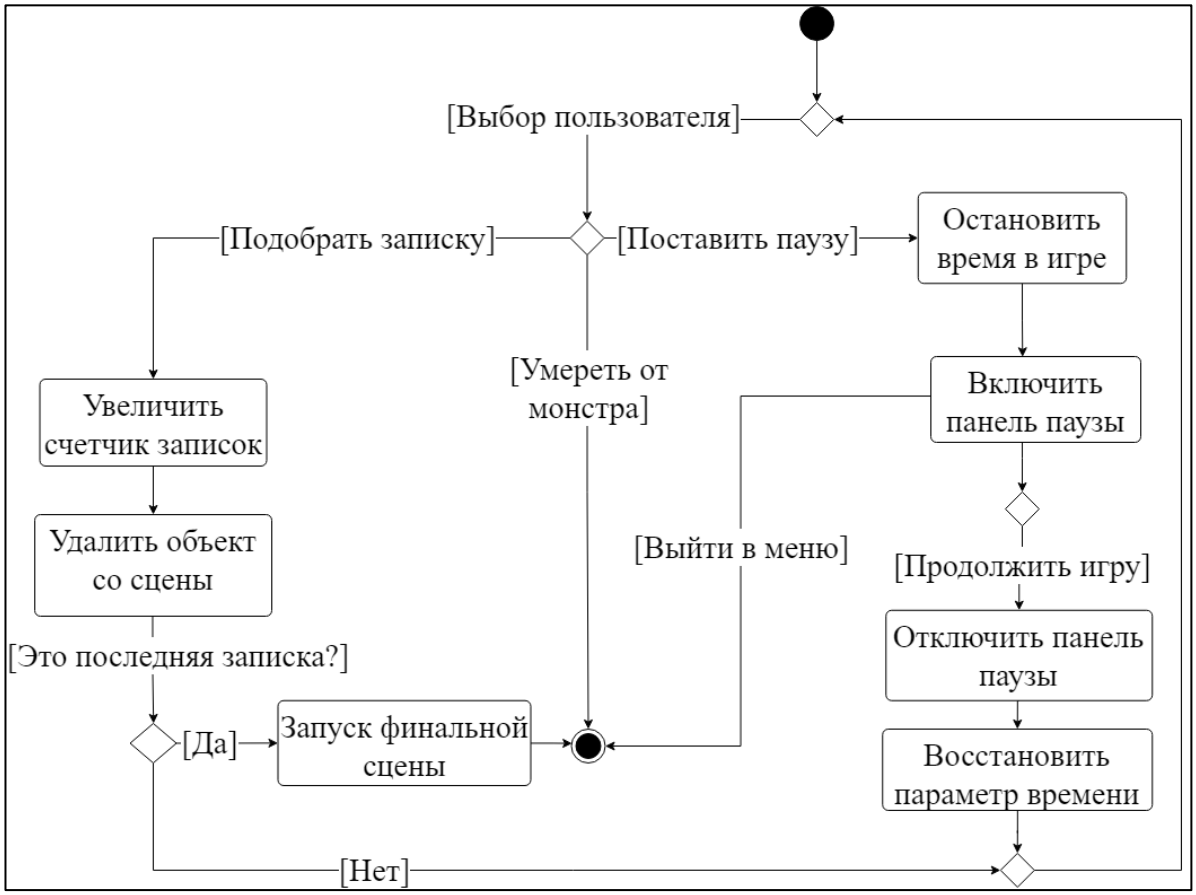

Рисунок 3 – Диаграмма деятельности

Рассмотрим более подробно основную механику подбора записок.

1. Начало. Игрок начинает игру и появляется на игровой карте.

2. Игровая карта. Игрок перемещается по игровой карте, находя и собирая записки.

3. Поиск записок. Игрок исследует окружение, чтобы найти расставленные по карте записки.

4. Подъем записки. Когда игрок находит записку, он подбирает ее, что приводит к увеличению счетчика собранных записок.

5. Уничтожение записки. После подъема записки она уничтожается со сцены, чтобы не мешать дальнейшему прохождению игры.

6. Проверка счетчика. Проверяется количество собранных записок. Если они все собраны, происходит переход к следующему этапу.

7. Финальная кат-сцена. После сбора всех записок запускается финальная кат-сцена, которая завершает игру.

8. Конец. Игра завершается после проигрывания финальной кат-сцены.

Процесс сбора записок в игре «Chaos Rising» играет важную роль не только в развитии сюжета, но и в формировании игрового опыта игрока. Кроме того, механика сбора записок стимулирует игрока к исследованию игровой среды, поиску скрытых уголков. Это создает уникальный и захватывающий игровой опыт, в котором каждый найденный предмет является частью большой сюжетной загадки, которая показывает, как игроку найти выход из загадочного леса. Финальная кат-сцена, запускаемая после сбора всех записок, служит не только завершением игрового процесса, но и представляет собой кульминацию всей игры, окончательное разрешение вопроса по поиску выхода.

Вывод по третьей главе

Глава предоставляет детальное описание архитектуры программной системы, включая диаграммы классов и компонентов. Обоснована структура и взаимодействие между основными элементами системы, что позволяет обеспечить модульность и масштабируемость проекта.

4. РЕАЛИЗАЦИЯ СИСТЕМЫ

4.1. Анимационные клипы персонажа

Для того чтобы улучшить игровой опыт и сделать управляемого персонажа более реалистичным, была разработана анимация, которая передает ощущение реальности, а также помогает игроку лучше понимать, как его действия влияют на движения персонажа в игровом пространстве. Схема разработанной анимации представлена на рисунке 4.

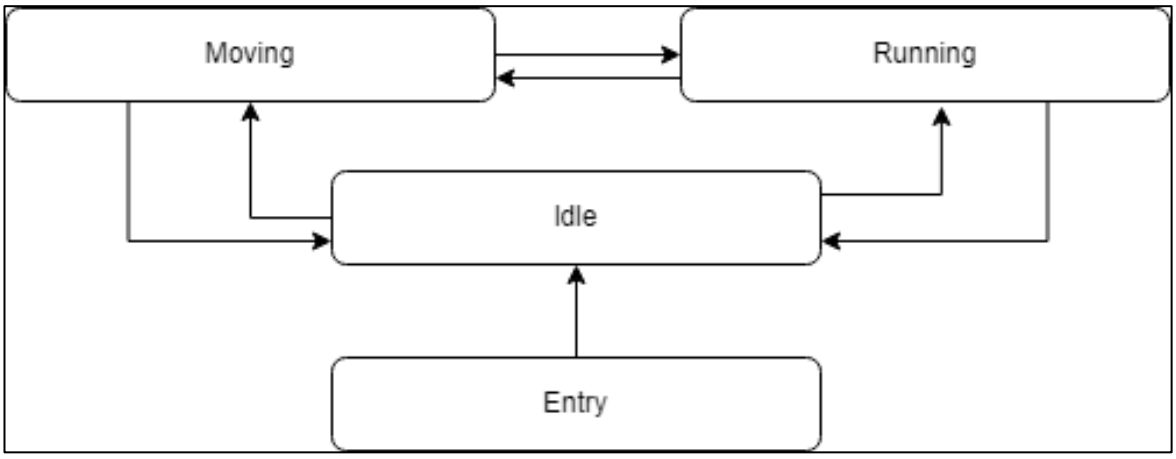

Рисунок 4 – Схема анимации персонажа

На рисунке представлены несколько анимационных клипов, которые связаны между собой. Каждая анимация имеет свое значение типа `int`, по умолчанию персонаж стоит и имеет значение `int = 0`. Когда персонаж начинает движение, переменная `int` становится равна 1, когда персонаж бежит переменная становится равной 2.

Для того, чтобы анимационные клипы работали, переменную `int`, следует менять из скрипта, который отвечает за передвижение персонажа. Листинг скрипта представлен в листинге 1 приложения Б.

Этот код отвечает за управление перемещением игрока в игре с использованием клавиатуры и компьютерной мыши, а также за анимацию персонажа в зависимости от его действий.

Ниже приведен разбор кода более подробно.

1. В начале кода объявляются переменные для управления скоростью перемещения, чувствительностью мыши, гравитацией и другими параметрами игрока.

2. В методе `Start()` инициализируются компоненты, такие как `Character-Controller` и `Animator`, а также устанавливается начальное состояние курсора.

3. В методе `Update()` получаются вводы от клавиатуры и мыши для управления движением игрока и поворотом камеры.

4. Затем определяется, в каком состоянии находится игрок (идет, бежит или стоит), и в зависимости от этого устанавливается соответствующая анимация персонажа.

5. Проверяется, находится ли персонаж на земле, и в зависимости от этого определяется вертикальная скорость игрока (падение).

6. Происходит перемещение игрока в пространстве с помощью метода `Move() CharacterController` на основе ввода от клавиатуры.

7. Поворот игрока и камеры осуществляется на основе ввода с мыши, а также ограничивается углом обзора.

Этот код позволяет игроку свободно перемещаться по игровому миру, а также отображает соответствующие анимации персонажа в зависимости от его действий.

4.2. Звуки передвижения персонажа и фоновые звуки

Звуковое сопровождение игрового процесса играет важную роль в создании атмосферы и углублении вовлеченности игрока в игровой мир. И чтобы управляемый персонаж не казался бесшумным, были реализованы звуки ходьбы, которые соответствуют движению персонажа, для улучшения реализма и атмосферы игры [12], а также была взята и добавлена фоновая музыка из фонотеки YouTube [13], на которую не распространяются авторске права.

Скрипт, отвечающий за активацию звуков ходьбы управляемого персонажа представлен в листинге 2 приложения. Б.

Этот скрипт отвечает за воспроизведение звуковых эффектов при движении игрока в игре. Ниже приведен разбор кода более подробно:

1) звуковые ресурсы:

– в начале скрипта определяется массив аудиоклипов (звуковых файлов), которые будут использоваться для воспроизведения звуков при движении игрока, звуовые эфекты ходьбы взяты с сайта «ZvukiPro» [14];

2) настройка времени между звуками:

– в скрипте определена переменная `timebetweensounds`, которая указывает время между воспроизведением звуков при движении игрока;

3) методы воспроизведения звуков:

– методы `movesound()` и `runsound()` отвечают за воспроизведение звуков при ходьбе и беге соответственно;

– они вызываются с определенной частотой с помощью функции `invokerepeating()`, которая позволяет запускать методы с заданным интервалом времени;

4) логика воспроизведения звуков:

– в методе `update()` определяется, происходит ли движение игрока (ходьба или бег), и в зависимости от этого устанавливаются значения флагов ismoving и isrunning;

– если игрок движется (или бежит), соответствующий флаг устанавливается в true, что позволяет в дальнейшем воспроизводить звуки;

5) воспроизведение звуков:

– в методах `movesound()` и `runsound()` проверяется текущее состояние флагов ismoving и isrunning;

– если флаг установлен в true, проигрывается звуковой эффект из массива аудиоклипов.

Этот скрипт позволяет реализовать звуковое сопровождение для движения игрока в игре, делая движение более реалистичным.

4.3. Геймплей и игровые механики

В игре «Chaos Rising» игрок управляет персонажем, который оказывается запертым в заброшенном лесу, где его преследует зловещий монстр. Основная цель игры - собрать все записки, разбросанные по лесу, чтобы раскрыть тайны происходящего и выбраться из этого кошмара.

Геймплей сосредоточен на выживании и исследовании, создавая атмосферу напряженности и страха. Игроку предстоит исследовать заброшенный лес, скрываться от монстра и собирать записки, которые являются ключевыми элементами для раскрытия истории и достижения победы.

1. Исследование леса. Игрок начинает игру в центре леса, он должен исследовать окружающую местность, чтобы найти десять записок. Лес представляет собой запутанный лабиринт, где каждый поворот может скрывать опасность.

2. Уклонение от монстра. В ходе исследования леса игрок должен остерегаться монстра, который бродит по лесу в поисках него. Если монстр замечает персонажа, он начинает его преследовать. Игроку приходится бежать и прятаться, чтобы избежать его внимания.

3. Сбор записок. Главная цель игры - найти и собрать все записки. Они могут быть спрятаны в различных уголках леса, в темных укрытиях.

4. Финал и развязка. После того, как игрок соберет все записки, начнется финальная сцена, которая раскроет все тайны и завершит игру.

Геймплей в «Chaos Rising» создает напряженную атмосферу, где игроку приходится балансировать между поиском записок и уклонением от монстра, чтобы выжить и раскрыть все секреты этого загадочного леса.

Основная игровая механика, сбор записок, на которой построена цель игры. Первая часть механики, за которую отвечает скрипт «GUIPapers», представлена в листинге 3 приложения Б.

Этот скрипт отвечает за отображение интерфейса игрока, связанного с коллекционированием записок в игре, а также за обработку победы при сборе всех записок. Он управляет отображением количества собранных записок на экране, а также запускает переход к финальной кат-сцене и завершению игры после сбора всех записок. Ниже приведен подробный разбор кода.

В коде используются переменные, описанные ниже.

1. Переменная `papers`: количество собранных записок. Эта переменная отслеживает, сколько записок игрок уже нашел.

2. Переменная `timeDown`: время до исчезновения надписи о собранных записках. Используется для временного отображения информации о собранных записках.

3. Переменная `timer`: таймер для отслеживания времени исчезновения надписи. Он постепенно уменьшается и скрывает надпись по истечении времени.

4. Переменная `_visible`: флаг для отображения количества собранных записок. Определяет, должна ли информация о собранных записках быть видимой на экране.

5. Переменная `win`: флаг для отображения победного сообщения. Становится истинным, когда игрок собирает все записки.

6. Переменная `waitingForMainMenu`: флаг, указывающий, что начата задержка перед переходом в главное меню. Используется для предотвращения повторного запуска корутины.

В коде используются методы описанные ниже.

1. Метод `start()`: инициализация переменных при запуске игры. Устанавливает начальные значения переменных.

2. Метод `update()`: обновление состояния игры каждый кадр. Здесь обрабатываются таймер, проверка количества собранных записок и запуск победной процедуры. Таймер уменьшается каждый кадр. Когда он дости-

гает нуля, скрывается надпись о собранных записках. Если собрано 10 записок, устанавливается флаг победы и запускается корутина для задержки перед переходом в главное меню.

3. Метод `waitAndLoadMainMenu()`: корутина для задержки перед переходом в главное меню. Задержка в 3 секунды перед запуском сцены главного меню.

4. Метод `loadMainMenu()`: функция для загрузки сцены главного меню и разблокировки курсора. Загружает сцену `LastScene` и делает курсор видимым.

5. Метод `onGUI()`: метод для отображения графического интерфейса, включая количество собранных записок и победное сообщение. Он устанавливает размер шрифта и цвет для всех надписей. Если `_visible` истинно, отображается количество собранных записок. Если `win` истинно, отображается сообщение о победе.

Следующий скрипт отвечает за уничтожение объекта записки, когда игрок подбирает ее и за увеличение счетчика собранных записок. Он также управляет взаимодействием игрока с записками, проверяя, когда игрок находится в зоне триггера и добавляя собранные записки к общему счетчику. Листинг 4 представлен в приложении Б.

Ниже приведен подробный разбор кода, за который отвечает скрипт «Destroy Papers».

В коде используются переменные, описанные ниже.

1. Переменная `papers`: объект, представляющий записки. Содержит ссылку на объект записок в игре.

2. Переменная `player`: объект игрока. Содержит ссылку на объект игрока в игре.

3. Переменная `trigger`: флаг, указывающий, что игрок находится в зоне триггера. Используется для отслеживания состояния игрока в зоне триггера.

4. Переменная `gp`: ссылка на компонент `GUIPapers`. Обеспечивает доступ к компоненту, управляющему интерфейсом и счетчиком записок.

В коде используются методы описанные ниже.

1. Метод `awake()`: инициализация переменных при пробуждении объекта. Находит объекты записок, игрока и компонента `GUIPapers`.

2. Метод `update()`: обновление состояния каждый кадр. Если игрок находится в зоне триггера, увеличивается количество собранных записок, устанавливается таймер и флаг видимости надписи, после чего уничтожается объект записки.

3. Метод `onTriggerEnter(Collider other)`: устанавливает флаг `trigger`, если игрок вошел в зону триггера. Используется для определения, когда игрок пересекает границу триггера.

4. Метод `onTriggerExit(Collider other)`: сбрасывает флаг `trigger`, если игрок вышел из зоны триггера. Используется для определения, когда игрок покидает зону триггера.

Код работает по следующей логике.

1. При входе игрока в зону триггера устанавливается флаг `trigger`.

2. Если игрок находится в зоне триггера, увеличивается количество собранных записок, устанавливается таймер и флаг видимости надписи, а затем уничтожается объект записки. Это позволяет игроку видеть информацию о собранной записке на экране в течение определенного времени.

3. При выходе игрока из зоны триггера флаг `trigger` сбрасывается. Это предотвращает повторное добавление той же записки в счетчик.

Когда игрок подбирает записку, запускается скрипт DestroyPapers, который увеличивает счетчик записок в скрипте GUIPapers и уничтожает объект записки. Этот процесс также включает временное отображение информации о собранной записке на экране, чтобы игрок мог видеть прогресс. Если все записки собраны, запускается победное сообщение и переход на следующую сцену. Это обеспечивает завершение игрового процесса и переход к следующему этапу игры или завершению.

4.4. ИИ противника

В игре «Chaos Rising» монстр играет важную роль, создавая атмосферу ужаса и напряжения для игрока. Монстр преследует игрока, вынуждая его проявлять осторожность и стратегическое мышление. Для реализации поведения монстра используется скрипт `AI_Ray`, который управляет его передвижением и реакциями на действия игрока. Ниже описаны основные аспекты реализации ИИ монстра. Скрипт представлен в листинге 5 приложения Б.

1. Инициализация компонентов. В начале скрипта определяются основные переменные и компоненты, необходимые для работы ИИ монстра. Это включает в себя нахождение игрока на сцене и инициализацию компонента `NavMeshAgent`, который отвечает за передвижение монстра по сцене.

2. Состояния монстра. Монстр может находиться в двух состояниях: патрулирование и преследование. В зависимости от расстояния до игрока, монстр переключается между этими состояниями. Если игрок находится в радиусе преследования, монстр начинает погоню.

3. Патрулирование. В режиме патрулирования монстр передвигается между заданными точками на карте. Когда монстр достигает последней точки патрулирования, он начинает новый цикл обхода, если не достигнуто максимальное количество обходов.

4. Преследование. В режиме преследования монстр увеличивает скорость и начинает погоню за игроком. Этот режим активируется, когда игрок входит в радиус преследования, и прекращается, если игрок выходит за пределы этого радиуса.

5. Отображение радиуса преследования. Для удобства настройки в редакторе Unity скрипт включает функцию `OnDrawGizmosSelected`, которая отображает радиус преследования монстра.

4.5. Анимация монстра

Для создания анимаций использовался онлайн-сервис Mixamo [15], который предлагает большой выбор готовых анимаций и инструментов для их настройки.

Регистрация и загрузка модели

Для начала работы с Mixamo необходимо зарегистрироваться на сайте и войти в свою учетную запись, после входа на сайт загружается 3D модель монстра. Mixamo поддерживает различные форматы файлов, включая FBX и OBJ.

Автоматическое риггирование

Если модель монстра не имеет скелета, Mixamo предлагает функцию автоматического риггирования, загруженная модель проходит процесс риггирования, в ходе которого к модели добавляются кости, необходимые для создания анимации. На рисунке 5 представлен пример процесса риггирования.

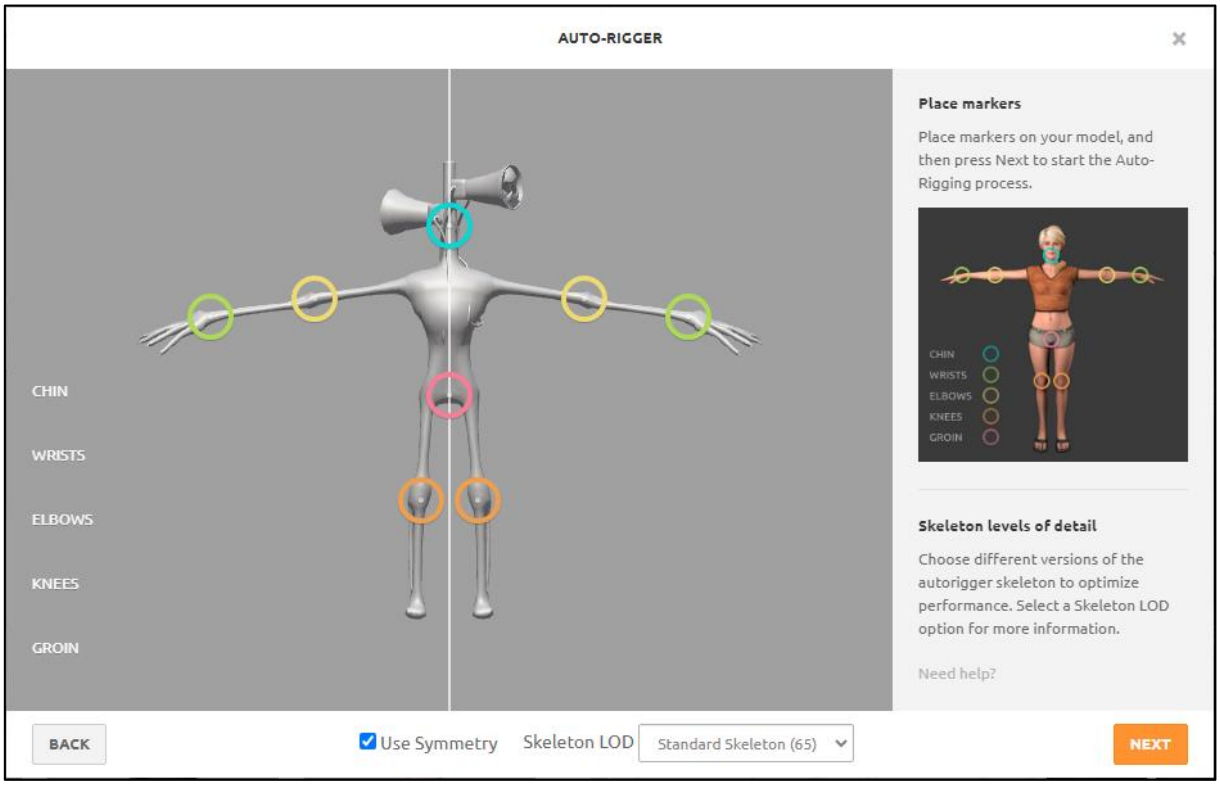

Рисунок 5 – Пример риггирования модели

Выбор и применение анимаций

Mixamo предоставляет обширную библиотеку готовых анимаций. Для монстра были выбраны анимации, соответствующие его поведению в игре.

Настройка анимаций

Анимации были оптимизированы для того, чтобы обеспечить реалистичное и плавное движение монстра в игре.

Экспорт и интеграция в Unity

После завершения настройки анимаций они были экспортированы в формате FBX, в Unity импортированная анимация были добавлены к аниматору монстра.

4.6. Управление персонажем

Управление является одной из ключевых составляющих любого игрового проекта, обеспечивая взаимодействие игрока с игровым миром и его персонажем. В инди-хорроре «Rising Chaos» управление было разработано с учетом интуиции и простоты, что позволяет игроку полностью погрузиться в атмосферу игры и сконцентрироваться на прохождении. Ниже приведен подробный разбор управления.

1. За передвижение отвечают клавиши WASD. Стандартное управление движением персонажа. Клавиша W отвечает за движение вперед, A – влево, S – назад и D – вправо. Такое управление позволяет игроку легко ориентироваться в игровом пространстве и быстро реагировать на изменения окружающей среды.

2. За обзор отвечает манипулятор мышь. Управление обзором осуществляется с помощью манипулятора мыши. Повороты головы персонажа и обзор вокруг него регулируются движениями манипулятора, что позволяет игроку осматривать окружение в 360 градусов. Это особенно важно в жанре хоррор, где своевременное обнаружение угрозы может спасти жизнь персонажа.

3. За бег отвечает клавиша Shift. Для ускорения движения персонажа используется клавиша Shift. Бег позволяет быстро перемещаться по локации и убегать от врагов, что добавляет динамики и повышает уровень напряжения в критических ситуациях.

Скрипт представленный в листинге 1 приложения Б, уже был расписан в подглаве «Анимационные клипы персонажа» с точки зрения анимации, теперь следует его подробнее разобрать с точки зрения управления.

1. Переменные и компоненты. В начале скрипта определяются основные переменные и компоненты, необходимые для управления персонажем. Это включает в себя скорость перемещения, чувствительность манипулятора мышь, гравитацию, углы поворота и компоненты `CharacterController` и `Animator`.

2. Обновление состояния персонажа. В методе `Update` обрабатываются пользовательские вводы и обновляется состояние персонажа. Это включает перемещение, анимации и обработку физики.

3. Перемещение и анимация. При обновлении состояния персонажа обрабатываются вводы с клавиатуры для перемещения вперед, назад и в стороны. В зависимости от состояния вводов и нажатия клавиши Shift, персонаж либо бежит, либо идет. Эти состояния отображаются с помощью анимаций.

4. Обработка физики. Для обработки физики используется компонент `CharacterController`, который обеспечивает корректное перемещение персонажа с учетом гравитации.

5. Поворот камеры. Для поворота камеры используется ввод с мыши, который управляет углом поворота персонажа и камеры. Углы поворота ограничиваются заданными значениями, чтобы избежать чрезмерного вращения.

4.7. Интерфейс

Интерфейс пользователя играет ключевую роль в создании удобного и интуитивного взаимодействия с игрой «Chaos Rising». Основные компоненты интерфейса включают: сцену меню, меню настроек и меню паузы.

Главное меню

Главное меню – это первая сцена, с которой взаимодействует игрок при запуске игры. Оно включает несколько кнопок, каждая из которых выполняет определенную функцию.

1. Настройки – кнопка, открывающая меню настроек, где игрок может изменить параметры графики, разрешение экрана и включить или выключить полноэкранный режим.

2. Продолжить игру – кнопка, позволяющая игроку вернуться к последней сохраненной игровой сессии.

3. Выйти из игры – кнопка, завершающая работу игры.

Главное меню реализовано простым скриптом, он представлен в листинге 1.

Листинг 1 – Скрипт главного меню

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

public class MainMenu : MonoBehaviour
{
    public void LoadLevel()
    {
        SceneManager.LoadScene("CatScene");
    }

    public void ExitGame()
    {
        Application.Quit();
    }
}
```

Главное меню организовано таким образом, чтобы игрок мог быстро и легко получить доступ к основным функциям игры.

Меню настроек

Меню настроек предоставляет игроку возможность настроить параметры игры в соответствии с его предпочтениями и возможностями компьютера. Основные настройки включают элементы, представленные ниже.

1. Качество графики – игрок может выбрать уровень графики (низкий, средний, высокий), чтобы настроить визуальные эффекты в зависимости от производительности его системы.

2. Разрешение экрана – позволяет выбрать разрешение экрана для оптимального отображения игры.

3. Полноэкранный режим – включает или выключает полноэкранный режим.

Меню настроек позволяет игроку оптимизировать игровой опыт, обеспечивая наилучшее соотношение качества графики и производительности.

Скрипт настроек представлен в листинге 6 приложения Б, разбор работы кода представлен ниже:

1) `start()`:

– инициализирует выпадающее меню разрешений экрана с текущими доступными разрешениями;

– устанавливает текущий индекс разрешения экрана;

– загружает сохраненные настройки (разрешение, качество, полноэкранный режим);

2) `setFullscreen(bool isFullscreen)`: устанавливает полноэкранный режим;

3) `setResolution(int resolutionIndex)`: устанавливает разрешение экрана на основе выбранного индекса из выпадающего меню;

4) `setQuality(int qualityIndex)`: устанавливает уровень качества графики;

5) `saveSettings()`: сохраняет текущие настройки (качество графики, разрешение экрана, полноэкранный режим) в `PlayerPrefs`;

6) `loadSettings(int currentResolutionIndex)`:

– загружает сохраненные настройки из `PlayerPrefs` и применяет их;

– если настройки не найдены, устанавливаются значения по умолчанию.

Меню паузы

Меню паузы активируется нажатием клавиши escape, когда игрок временно останавливает игровой процесс. Оно обеспечивает возможность:

1) возобновить игру – кнопка для возвращения к игровому процессу;

2) выйти в главное меню – кнопка для выхода из текущей игры и возврата в главное меню;

3) выйти из игры – кнопка для выхода из текущей игры и возврат на рабочий стол.

Меню паузы позволяет игроку сделать перерыв в игре, при этом сохраняя контроль над возможностью быстро вернуться в игру.

Листинг 7 представлен в приложении Б, ниже представлен подробный разбор кода:

1) `start()`: находит все источники звука (`AudioSource`) в сцене и сохраняет их в массив;

2) `update()`: проверяет, была ли нажата клавиша "Escape". Если да, то в зависимости от текущего состояния игры (пауза или игра) вызывает методы `Resume()` или `Pause()`;

3) `resume()`:

– возобновляет игру: скрывает меню паузы, устанавливает нормальную скорость времени, прячет курсор и блокирует его;

– восстанавливает видимость всех текстовых блоков, которые не являются частью меню паузы;

– возобновляет воспроизведение всех звуковых источников;

4) `pause()`:

– останавливает игру: показывает меню паузы, останавливает время, делает курсор видимым и разблокированным;

– скрывает все текстовые блоки, которые не являются частью меню паузы;

– приостанавливает воспроизведение всех звуковых источников;

5) `isChildOfPauseMenu(GameObject obj)`: проверяет, является ли переданный объект или его родитель частью меню паузы;

6) `loadMenu()`: загружает главное меню, восстанавливая нормальную скорость времени;

7) `quitGame()`: завершает игру.

Кат-сцены

Кат-сцены играют важную роль в повествовании игры, позволяя представить сюжетные элементы и погрузить игрока в игровую атмосферу. В нашем проекте используется две кат-сцены: вводная и финальная. Обе кат-сцены управляются похожим кодом и служат для установления начала и завершения игрового процесса. Рассмотрим их подробнее.

Вводная кат-сцена

Вводная кат-сцена запускается после меню и перед началом основной игры. Она состоит из серии изображений, сопровождаемых субтитрами, которые вводят игрока в сюжетную линию. Кат-сцена задает тон игры и объясняет начальные условия, с которыми столкнется игрок.

Финальная кат-сцена

Финальная кат-сцена запускается после того, как игрок собрал все записки. Она также состоит из серии изображений, которые завершают историю игры и подводят итог приключениям главного героя. После окончания финальной кат--сцены игрок возвращается в главное меню.

Скрипт, управляющий показом изображений в кат-сценах и переходом между сценами показан в листинге 8 приложения Б.

Отображение интерфейса в игре

В процессе игры интерфейс также отображает важную информацию для игрока, такую как количество собранных записок. Этот элемент интерфейса реализован через GUI-элементы и обновляется в реальном времени, обеспечивая игроку необходимую обратную связь.

Интерфейс «Chaos Rising» разработан таким образом, чтобы обеспечить максимальную удобство и интуитивность, позволяя игроку сосредоточиться на игровом процессе и взаимодействии с миром игры.

4.8. Создание игрового уровня

Создание игрового уровня является важной частью разработки, так как от него зависит атмосфера и общее восприятие игры игроком. В реализуемом проекте игровой уровень представляет собой лес, в котором главный герой ищет записки и скрывается от монстра. Для достижения нужной атмосферы и детализации были использованы различные элементы окружения и объекты. Рассмотрим основные шаги, выполненные при создании игрового уровня.

Разработка террейна

Первоначально был создан террейн (ландшафт) с помощью встроенного инструмента Unity для работы с террейном. Этот инструмент позволил нам задать нужную форму рельефа, добавить возвышенности и углубления, что сделало местность более реалистичной.

Создание и текстурирование дома

В центре игрового уровня расположен дом, собранный из кубов Unity. На кубы были наложены текстуры, чтобы придать им вид деревянных и кирпичных поверхностей.

Расстановка объектов окружения

Окружение играет важную роль в создании атмосферы и погружения в игру. Правильная расстановка объектов существенно влияет на геймплей и общее восприятие игрового мира. Ниже описан процесс создания и размещения ключевых объектов на террейне для формирования атмосферы хоррора.

1. Лавочки и тотемы были размещены в различных местах уровня, чтобы создать ощущение старого леса с элементами цивилизации. Пример такого тотема приведен на рисунке 6.

Рисунок 6 – Тотем

2. Расставленные по уровню могильные плиты усиливают атмосферу таинственности и тревоги, что способствует вовлечению игрока. Могильные плиты приведены на рисунке 7.

Рисунок 7 – Могильные плиты

3. Деревья были добавлены с помощью кисти для рисования растительности в Unity. Это позволило быстро и эффективно заполнить лес большим количеством деревьев разных типов и размеров, что придало уровню густоту и естественность. Пример деревьев приведен на рисунке 8.

Рисунок 8 – Деревья

Ограничение игровой зоны

Чтобы игрок не покидал пределы игрового уровня, территория была окружена забором. Забор служит не только физическим ограничением, но и визуальным элементом, который органично вписывается в общий стиль уровня. Рисунок 9 показывает, как это выглядит в игровом проекте.

Рисунок 9 – Забор ограничивающий игровую зону

Вывод по четвертой главе

В этом разделе описаны этапы реализации компонентов системы. Приведены листинги ключевых скриптов, таких как: управление персонажем, ИИ противника, а также интерфейсные элементы.

5. ТЕСТИРОВАНИЕ СИСТЕМЫ

В этой главе представлены результаты функциональных тестов, проведенных на реализованном проекте «Chaos Rising» на платформе Unity. В таблице 1 приведены основные сценарии тестирования и их результаты.

Таблица 1 – Результаты функционального тестирования

№	Название теста	Шаги	Ожидаемый результат	Тест пройден?
1	Тест на корректное перемещение игрока	1. Запустить игру 2. Убедиться что игрок может перемещаться	Игрок должен корректно перемещаться по игровому миру в зависимости от ввода пользователя.	Да
2	Тест на анимацию передвижения игрока	1. Запустить игру. 2. Убедиться, что при движении игрока воспроизводятся соответствующие анимации (ходьба).	При каждом действии игрока (ходьба) должна воспроизводиться соответствующая анимация.	Да
3	Тест на воспроизведение звуков при движении	1. Запустить игру. 2. Убедиться, что при движении игрока вперед, назад, влево и вправо воспроизводятся соответствующие звуковые эффекты.	При каждом движении игрока должен воспроизводиться соответствующий звуковой эффект.	Да
4	Тест на воспроизведение звуков при беге	1. Запустить игру. 2. Убедиться, что при беге игрока воспроизводятся соответствующие звуковые эффекты.	При беге игрока должен воспроизводиться соответствующий звуковой эффект	Да
5	Тест на проверку видимости игрока	1. Убедиться, что метод `checkSight()` правильно определяет видимость игрока, если игрок находится в поле зрения монстра. 2. Создать сценарий, где игрок находится в поле зрения монстра, и проверить, что он начинает преследование.	Монстр должен начать преследование	Да

Окончание таблицы 1

№	Название теста	Шаги	Ожидаемый результат	Тест пройден?
6	Тест на поведение монстра	Убедиться, что ИИ монстра правильно реагирует на окружение (патрулирует карту, когда игрока не видно, атакует, когда видно, обход препятствий)	ИИ монстра правильно реагирует на окружающую обстановку	Да
7	Тест на убийство игрока монстром	1. Убедиться, что монстр атакует игрока при достижении им определенной близости. 2. Переместить игрока вблизи монстра и проверить, что монстр переходит в состояние агрессии и игровая камера меняется на камеру смерти	Монстр убивает игрока	Да
8	Тест на правильное переключение разрешения экрана	1. Переставить разрешение экрана 2. Убедиться что разрешение меняется	Разрешение меняется	Да
9	Тест на корректное срабатывание паузы	1. Открыть клавишей ESC меню паузы 2. Убедиться что время и звук в игре полностью остановлены	Пауза работает корректно	Да
10	Тест на корректное переключение сцен, во всех скриптах	Убедиться что все скрипты, имеющие функцию смены сцены работают корректно	Сцены меняются корректно	Да
11	Тест на сбор записок	1. Поднять записку 2. Убедиться что при поднятии записки, она собирается	Записка удалилась со сцены	Да
12	Тест на счетчик сбора записок	1. Поднять записку 2. При поднятии записки, интерфейс игры должен оповестить игрока о кол	Интерфейс оповестил о количестве поднятых записок	Да
13	Тест на корректное изменение качества графики	1. Изменить качество графики в настройках 2. Убедиться что качество изменилось	Качество графики меняется в соответствии настройкам	Да

Вывод по пятой главе

Тестирование корректного перемещения игрока показало, что игрок может свободно и правильно перемещаться по игровому миру в зависимости от ввода пользователя. Анимации движения, такие как ходьба и бег, воспроизводятся корректно в соответствии с действиями игрока. Воспроизведение звуковых эффектов при движении игрока также функционирует правильно: звуковые эффекты воспроизводятся для разных направлений движения (вперед, назад, влево, вправо), а в ходе бега игрока – соответствующие звуковые эффекты, что добавляет реалистичности игровому процессу.

Работа ИИ монстра была проверена на правильность реакции на окружающую среду и действия игрока. Монстр патрулирует карту, атакует игрока при его обнаружении, преследует его в случае видимости и обходит препятствия. Эти функции работают корректно, что подтверждает высокое качество реализации ИИ в игре.

Интерфейс и взаимодействие с ним также прошли тестирование. Меню паузы корректно останавливает игру, блокирует время и звук, обеспечивая полное погружение в игровую паузу. Разрешения экрана переключаются правильно, и все изменения отображаются корректно на экране, что важно для комфортного игрового процесса на разных устройствах.

Переходы между сценами были проверены на корректность выполнения. Тесты подтвердили, что все скрипты, отвечающие за смену сцен, работают правильно и без ошибок. Переходы между сценами осуществляются плавно, что важно для поддержания непрерывного игрового опыта.

Проведенные тесты подтвердили, что система соответствует всем заявленным требованиям и стандартам. Все функциональные и нефункциональные элементы работают корректно, обеспечивая плавный и интерактивный игровой процесс. Таким образом, можно сделать вывод, что проект «Chaos Rising» готов к дальнейшему этапу разработки и может быть представлен пользователям для получения обратной связи и проведения финального тестирования.

ЗАКЛЮЧЕНИЕ

В ходе данной выпускной квалификационной работы был разработан и реализован проект компьютерной игры под названием «Chaos Rising» на платформе Unity. Проект охватывает различные аспекты создания игры, начиная с анализа требований и заканчивая реализацией основных игровых механик. В ходе выполнения выпускной квалификационной работы были решены следующие задачи.

1. Проведен анализ предметной области.
2. Проведен анализ методов разработки и архитектуры компьютерной игры.
3. Сформулирована и описана концепция игры.
4. Разработаны скрипты и интегрированы в проект.
5. Реализована компьютерная игра и проведено тестирование.

Варианты дальнейшего развития проекта включают:

1) улучшение графики и анимаций; добавление новых игровых механик;
2) оптимизацию производительности;
3) расширение игрового мира и добавление новых уровней.

Эти улучшения помогут сделать игру более интересной и увлекательной для игроков.

В результате выполнения выпускной квалификационной работы были решены все поставленные задачи, таким образом, цель данной работы достигнута.

ЛИТЕРАТУРА

1. SkillFactory Media. [Электронный ресурс] URL: https://blog.skillfactory.ru/glossary/unity/ (дата обращения: 07.02.2024 г.).
2. Five Nights at Freddy's. [Электронный ресурс] URL: https://store.steampowered.com/app/319510/Five_Nights_at_Freddys/?l=russian (дата обращения: 07.02.2024 г.).
3. Among the Sleep. [Электронный ресурс] URL: https://store.steampowered.com/app/250620/Among_the_Sleep__Enhanced_Edition/ (дата обращения: 07.02.2024 г.).
4. Layers Of Fear. [Электронный ресурс] URL: https://store.steampowered.com/app/1946700/Layers_of_Fear/ (дата обращения: 07.02.2024 г.).
5. Darkwood. [Электронный ресурс] URL: https://store.steampowered.com/app/274520/Darkwood/ (дата обращения: 07.02.2024 г.).
6. Unity. [Электронный ресурс] URL: http://unity3d.com/ru/unity (дата обращения: 07.02.2024 г.).
7. Unity Asset Store. [Электронный ресурс] URL: https://www.assetstore.unity3d.com/ (дата обращения: 07.02.2024 г.).
8. Документация по языку C#. [Электронный ресурс] URL: https://learn.microsoft.com/ru-ru/dotnet/csharp/ (дата обращения: 07.02.2024 г.).
9. Unity Scripting. [Электронный ресурс] URL: https://docs.unity3d.com/Manual/ScriptingSection.html (дата обращения: 07.02.2024 г.).
10. Hocking J. Unity in Action: Multiplatform Game Development in C#. – New York: Manning Publications, 2015. – 352 p.
11. Ларман К. Применение UML 2.0 и шаблонов проектирования. Введение в объектно-ориентированный анализ, проектирование и итеративную разработку. 3-е изд. – М.: «Вильямс», 2013. – 736 с.

12. Calabrese D. Unity 2D Game Development. – Birmingham: Packt Publishing, 2014. – 126 p.

13. StudioYouTube. [Электронный ресурс] URL: https://youtube.com/audiolibrary (дата обращения: 07.02.2024 г.).

14. ZvukiPro. [Электронный ресурс] URL: https://zvukipro.com/zvukiludei/51-zvuki-shagov-cheloveka.html (дата обращения: 07.02.2024 г.).

15. Mixamo. [Электронный ресурс] URL: https://www.mixamo.com/#/ (дата обращения: 07.02.2024 г.).

ПРИЛОЖЕНИЯ

Приложение А. Таблицы спецификаций вариантов использования

Спецификация вариантов использования (ВИ) системы приведена в таблицах 1–5.

Таблица 1 – Спецификация ВИ «Начать игру»

Прецедент: Начать игру
ID: 1
Краткое описание: Игрок начинает новую игровую сессию.
Главные актеры: Игрок
Второстепенные актеры: Нет
Предусловия: Нет
Основной поток: 1. Вариант использования начинается, когда игрок выбирает опцию «Начать игру» из главного меню. 2. Игровой мир загружается.
Постусловия: Игровое приложение переходит в режим основного игрового процесса.
Альтернативные потоки: Нет

Таблица 2 – Спецификация ВИ «Основной процесс игры»

Прецедент: Основной процесс игры
ID: 2
Краткое описание: Игрок взаимодействует с основным игровым контентом.
Главные актеры: Игрок
Второстепенные актеры: Противники
Предусловия: Запущено новая игровая сессия или загружено сохранение.
Основной поток: 1. Вариант использования начинается, когда игрок находится в игровом мире. 2. Игрок взаимодействует с окружающим миром.
Постусловия: 1. Игровое приложение продолжает работать в обычном режиме игрового процесса.
Альтернативные потоки: Нет

Таблица 3 – Спецификация ВИ «Пауза»

Прецедент: Пауза
ID: 3
Краткое описание: Игрок приостанавливает игровой процесс.
Главные актеры: Игрок
Второстепенные актеры: Противники
Предусловия: Игровой процесс активен.
Основной поток: 1. Вариант использования начинается, когда игрок нажимает соответствующую клавишу или активирует опцию в меню. 2. Игровой процесс приостанавливается.
Постусловия: 1. Игровое приложение переходит в режим паузы.
Альтернативные потоки: Нет

Таблица 4 – Спецификация ВИ «Управлять персонажем»

Прецедент: Управлять персонажем
ID: 4
Краткое описание: Игрок управляет движением и действиями игрового персонажа.
Главные актеры: Игрок
Второстепенные актеры: Противники
Предусловия: Игровой процесс активен.
Основной поток: 1. Вариант использования начинается, когда игрок использует клавиши для управления персонажем. 2. Персонаж перемещается или выполняет другие действия в соответствии с командами игрока.
Постусловия: 1. Персонаж продолжает взаимодействовать с окружающим миром в соответствии с действиями игрока.
Альтернативные потоки: Нет

Таблица 5 – Спецификация ВИ «Настройки»

Прецедент: Настройки
ID: 5
Краткое описание: Игрок настраивает параметры игры по своему усмотрению.
Главные актеры: Игрок
Второстепенные актеры: Противник
Предусловия: Игровое приложение активно.
Основной поток: 1. Вариант использования начинается, когда игрок открывает меню настроек. 2. Игрок выбирает категорию настроек (например, качество графика, разрешение экрана. 3. Игрок вносит необходимые изменения в параметры. 4. Игрок сохраняет изменения.
Постусловия: 1. Игровые настройки обновлены согласно выбранным игроком параметрам.
Альтернативные потоки: Нет

Приложение Б. Листинги скриптов используемых в игре

Листинг 1 – Скрипт передвижения персонажа

```
public class PlayerMovement : MonoBehaviour
{
    public float movespeed;
    public float sensitivity;
    public float gravity;
    private float mouseX;
    private float mouseY;
    private float vertical;
    private float horizontal;
    public Vector2 clampangle;
    private Vector3 Velocity;
    private Vector2 angle;
    public Transform cameraTransform;
    private CharacterController charactercontroller;
    private Animator animator;
    private void Start()
    {
        charactercontroller = GetComponent<CharacterController>();
        animator = GetComponent<Animator>();
        Cursor.lockState = CursorLockMode.Locked;
        Cursor.visible = false;
    }
    private void Update()
    {
        vertical = Input.GetAxis("Vertical");
        horizontal = Input.GetAxis("Horizontal");
        Vector3 playerMovementInput = new Vector3(horizontal, 0.0f, verti-
cal);
        Vector3 moveVector = transform.TransformDirection(playerMovementIn-
put);
        if (Input.GetAxis("Vertical") > 0  Input.GetAxis("Vertical") < 0
Input.GetAxis("Horizontal") > 0 || Input.GetAxis("Horizontal") < 0)
        {
            if (Input.GetKey(KeyCode.LeftShift) && Input.GetAxis("Verti-
cal") > 0)
            {
                movespeed = 6;
                //бежит
                animator.SetInteger("switch", 2);
            }
            else
            {
                movespeed = 2f;
                //идет
                animator.SetInteger("switch", 1);
            }
        }
      else
        {
            animator.SetInteger("switch", 0);
        }
        if (charactercontroller.isGrounded)
        {
            Velocity.y = -if;
        }
        else
        {
            Velocity.y -= gravity * Time.deltaTime;}
        charactercontroller.Move(moveVector * movespeed * Time.deltaTime);
```

```
        charactercontroller.Move(Velocity * Time.deltaTime);
        mouseX = Input.GetAxis("Mouse X");
        mouseY = Input.GetAxis("Mouse Y");
        angle.x -= mouseY * sensitivity;
        angle.y -= mouseX * -sensitivity;
        angle.x = Mathf.Clamp(angle.x, -clampangle.x, clampangle.y);
        Quaternion rotation = Quaternion.Euler(angle.x, angle.y, 0.0f);
        Quaternion rotationTwo = Quaternion.Euler(0.0f, angle.y, 0.0f);
        transform.rotation = rotationTwo;
        cameraTransform.rotation = rotation;
    }
}
```

Листинг 2 – Скрипт активации звуков ходьбы персонажа

```
public class MoveSounds : MonoBehaviour
{
    private AudioSource source;
    public AudioClip[] clips;
    private Material material;
    private bool ismoving = false;
    private bool isrunning = false;
    private int currentmaterial = 0;
    public float timebetweensounds = 1f;
    private void Start()
    {
        InvokeRepeating("MoveSound", 0, timebetweensounds);
        InvokeRepeating("RunSound", 0, timebetweensounds /2f);
        source = GetComponent<AudioSource>();
    }
    void Update()
    {
        RaycastHit hit;
        if (Input.GetAxis("Vertical") > 0  Input.GetAxis("Vertical") < 0
Input.GetAxis("Horizontal") > 0 || Input.GetAxis("Horizontal") < 0)
        {
            if (Input.GetKey(KeyCode.LeftShift) && Input.GetAxis("Verti-
cal") > 0)
            {
                ismoving = false;
                isrunning = true;
            }
            else
            {
                ismoving = true;
                isrunning = false;
            }
        }
        else
        {
            ismoving = false;
            isrunning = false;
        }
    }
    private void MoveSound()
    {
        if (ismoving)
        {
            source.PlayOneShot(clips[Random.Range(0, clips.Length)]);
        }
```

```
    }
    private void RunSound()
    {
        if (isrunning)
        {
            source.PlayOneShot(clips[Random.Range(0, clips.Length)]);
        }
```

Листинг 3 – Скрипт коллекционирования записок

```
using UnityEngine;
using UnityEngine.SceneManagement;
using System.Collections;
public class GUIPapers : MonoBehaviour
{
    public int Papers;
    public int TimeDown;
    private float Timer;
    public bool _visible;
    private bool win;
    private bool waitingForMainMenu = false;
    void Start()
    {
        Papers = 0;
        _visible = false;
    }
    void Update()
    {
        if (TimeDown != 0)
        {
            Timer = (float)TimeDown;
            TimeDown = 0;
        }
        if (Timer > 0)
            Timer -= Time.deltaTime;
        if (Timer < 0)
            Timer = 0;

        if (Timer == 0)
        {
            _visible = false;
        }
        if (Papers == 10)
        {
            win = true;
            _visible = false;
            if (!waitingForMainMenu)
            {
                StartCoroutine(WaitAndLoadMainMenu());
            }
        }
    }
    IEnumerator WaitAndLoadMainMenu()
    {
        waitingForMainMenu = true;
        yield return new WaitForSeconds(3f);
        LoadMainMenu();
    }
    void LoadMainMenu()
    {
```

```
        SceneManager.LoadScene("LastScene");
        Cursor.lockState = CursorLockMode.None;
        Cursor.visible = true;
    }
    void OnGUI()
    {
        GUI.skin.label.fontSize = 25;
        GUI.skin.label.normal.textColor = Color.red;
        if (_visible)
        {
            GUI.Label(new Rect(Screen.width / 2, Screen.height / 3 - 50,
180, 30), "Papers       /10");
            GUI.Label(new Rect(Screen.width / 2 + 100, Screen.height / 3 -
50, 180, 30), Papers.ToString());
        }
        if (win)
        {
            GUI.Label(new Rect(Screen.width / 2 - 100, Screen.height / 3 -
100, 200, 100), "Last note picked up", new GUIStyle()
            {
                alignment = TextAnchor.MiddleCenter,
                fontSize = 50,
                normal = { textColor = Color.red }
            });
        }
    }
}
```

Листинг 4 – Скрипт счетчика и уничтожения записок

```
using UnityEngine;
public class DestroyPapers : MonoBehaviour
{
    public GameObject Papers;
    public GameObject player;
    private bool trigger;
    private GUIPapers gp;
    void Awake()
    {
        Papers = GameObject.Find("Papers");
        player = GameObject.FindGameObjectWithTag("Player");
        gp = GameObject.Find("GUIPapers").GetComponent<GUIPapers>();
    }
    void Update()
    {
        if (trigger)
        {
            if (gp != null)
            {
                gp.Papers++;
                gp.TimeDown = 3;
                gp._visible = true;
            }
            Destroy(gameObject);
        }
    }
    public void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Player"))
        {trigger = true;}
```

```
    }
    public void OnTriggerExit(Collider other)
    {
        if (other.CompareTag("Player"))
        {trigger = false;}
    }
}
```

Листинг 5 – Скрипт искусственного интеллекта монстра

```
using UnityEngine;
using UnityEngine.AI;
public class AI_Ray : MonoBehaviour
{
    private Transform player;
    private NavMeshAgent navMeshAgent;
    public Transform[] patrolPoints;
    private int currentPatrolIndex;
    private int patrolLoopCount = 0;
    public int maxPatrolLoops = 1;
    public float chaseRadius = 15f;
    // Скорость в разных состояниях
    public float patrolSpeed = 2f;
    public float chaseSpeed = 5f;
    private enum State { Patrolling, Chasing }
    private State currentState = State.Patrolling;
    void Start()
    {
        player = GameObject.FindGameObjectWithTag("Player")?.transform;
        navMeshAgent = GetComponent<NavMeshAgent>();
        if (patrolPoints.Length > 0)
        {
            currentPatrolIndex = 0;
            navMeshAgent.SetDestination(patrolPoints[currentPatrolIn-
dex].position);
        }
    }
    void Update()
    {
        if (player == null) return;
        float distanceToPlayer = Vector3.Distance(transform.position,
player.position);

        switch (currentState)
        {
            case State.Patrolling:
                Patrolling();
                if (distanceToPlayer <= chaseRadius)
                {
                    currentState = State.Chasing;
                }
                break;
            case State.Chasing:
                Chasing();
                if (distanceToPlayer > chaseRadius)
                {
                    currentState = State.Patrolling;
                    navMeshAgent.SetDestination(patrolPoints[currentPatro-
lIndex].position);
                }
```

```
                break;
            }
        }
        void Patrolling()
        {
            navMeshAgent.speed = patrolSpeed;
            if (!navMeshAgent.pathPending && navMeshAgent.remainingDistance <
0.5f)
            {
                currentPatrolIndex = (currentPatrolIndex + 1) %
patrolPoints.Length;
                if (currentPatrolIndex == 0)
                {
                    patrolLoopCount++;
                    if (patrolLoopCount >= maxPatrolLoops)
                    {
                        currentState = State.Chasing;
                        navMeshAgent.SetDestination(player.position);
                        return;
                    }
                }
                navMeshAgent.SetDestination(patrolPoints[currentPatrolIn-
dex].position);
            }
        }
        void Chasing()
        {
            navMeshAgent.speed = chaseSpeed;
            navMeshAgent.SetDestination(player.position);
        }
        void OnDrawGizmosSelected()
        {
            Gizmos.color = Color.yellow;
            Gizmos.DrawWireSphere(transform.position, chaseRadius);
        }
}
```

Листинг 6 – Скрипт меню настроек

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

public class Settings : MonoBehaviour
{
    public AudioMixer audioMixer;
    public Dropdown resolutionDropdown;
    public Dropdown qualityDropdown;
    public Slider volumeSlider;
    float currentVolume;
    Resolution[] resolutions;
    void Start()
    {
        resolutionDropdown.ClearOptions();
        List<string> options = new List<string>();
        resolutions = Screen.resolutions;
        int currentResolutionIndex = 0;
```

```
        for (int i = 0; i < resolutions.Length; i++)
        {
            string option = resolutions[i].width + "x" + resolu-
tions[i].height + " " + resolutions[i].refreshRate + "Hz";
            options.Add(option);
            if (resolutions[i].width == Screen.currentResolution.width
                  && resolutions[i].height == Screen.currentResolu-
tion.height)
                currentResolutionIndex = i;
        }
        resolutionDropdown.AddOptions(options);
        resolutionDropdown.RefreshShownValue();
        LoadSettings(currentResolutionIndex);
    }
    public void SetVolume(float volume)
    {
        audioMixer.SetFloat("Volume", volume);
        currentVolume = volume;
    }
    public void SetFullscreen(bool isFullscreen)
    {
        Screen.fullScreen = isFullscreen;
    }
    public void SetResolution(int resolutionIndex)
    {
        Resolution resolution = resolutions[resolutionIndex];
        Screen.SetResolution(resolution.width,
                  resolution.height, Screen.fullScreen);
    }

    public void SetQuality(int qualityIndex)
    {

        QualitySettings.SetQualityLevel(qualityIndex);

    }
    public void SaveSettings()
    {
        PlayerPrefs.SetInt("QualitySettingPreference",
                  qualityDropdown.value);
        PlayerPrefs.SetInt("ResolutionPreference",
                  resolutionDropdown.value);
        PlayerPrefs.SetInt("FullscreenPreference",
                  System.Convert.ToInt32(Screen.fullScreen));
        PlayerPrefs.SetFloat("VolumePreference",
                  currentVolume);
    }

    public void LoadSettings(int currentResolutionIndex)
    {
        if (PlayerPrefs.HasKey("QualitySettingPreference"))
            qualityDropdown.value =
                        PlayerPrefs.GetInt("QualitySettingPreference");
        else
            qualityDropdown.value = 3;
        if (PlayerPrefs.HasKey("ResolutionPreference"))
            resolutionDropdown.value =
                        PlayerPrefs.GetInt("ResolutionPreference");
        else
            resolutionDropdown.value = currentResolutionIndex;
        if (PlayerPrefs.HasKey("FullscreenPreference"))
```

```
            Screen.fullScreen =
            System.Convert.ToBoolean(PlayerPrefs.GetInt("FullscreenPrefer-
ence"));
        else
            Screen.fullScreen = true;
        if (PlayerPrefs.HasKey("VolumePreference"))
            volumeSlider.value =
                        PlayerPrefs.GetFloat("VolumePreference");
        else
            volumeSlider.value =
                        PlayerPrefs.GetFloat("VolumePreference");
    }
}
```

Листинг 7 – Скрипт меню паузы

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI;
public class Menu_Pause : MonoBehaviour
{
    public static bool GameIsPaused = false;
    public GameObject pauseMenuUI;
    private AudioSource[] allAudioSources;
    private void Start()
    {
        allAudioSources = FindObjectsOfType<AudioSource>();
    }
    void Update()
    {
        if (Input.GetKeyDown(KeyCode.Escape))
        {
            if (GameIsPaused)
            {
                Resume();
            }
            else
            {
                Pause();
            }
        }
    }
    public void Resume()
    {
        pauseMenuUI.SetActive(false);
        Time.timeScale = 1f;
        GameIsPaused = false;
        Cursor.lockState = CursorLockMode.Locked;
        Cursor.visible = false;
        Text[] allTexts = FindObjectsOfType<Text>();
        foreach (Text text in allTexts)
        {
            if (!IsChildOfPauseMenu(text.gameObject))
            {
                text.enabled = true;
            }
        }
        foreach (AudioSource audioSource in allAudioSources)
```

```
        {
            audioSource.UnPause();
        }
    }
    public void Pause()
    {
        pauseMenuUI.SetActive(true);
        Time.timeScale = 0f;
        GameIsPaused = true;
        Cursor.lockState = CursorLockMode.None;
        Cursor.visible = true;
        Text[] allTexts = FindObjectsOfType<Text>();
        foreach (Text text in allTexts)
        {
            if (!IsChildOfPauseMenu(text.gameObject))
            {
                text.enabled = false;
            }
        }
        foreach (AudioSource audioSource in allAudioSources)
        {
            audioSource.Pause();
        }
    }
    private bool IsChildOfPauseMenu(GameObject obj)
    {
        Transform parent = obj.transform.parent;
        while (parent != null)
        {
            if (parent.gameObject == pauseMenuUI)
            {
                return true;
            }
            parent = parent.parent;
        }
        return false;
    }
    public void LoadMenu()
    {
        Debug.Log("Load");
        Time.timeScale = 1f;
        SceneManager.LoadScene("MainMenu");
    }
    public void QuitGame()
    {
        Debug.Log("Quit");
        Application.Quit();
    }
}
```

Листинг 8 – Скрипт кат-сцены

```
public class LastScene : MonoBehaviour
{
    public Image[] images;
    private int currentIndex = 0;
    void Start()
    {
        for (int i = 1; i < images.Length; i++)
        {
```

```
            images[i].gameObject.SetActive(false);
        }
    }
    void Update()
    {
        if (Input.GetKeyDown(KeyCode.Space))
        {
            if (currentIndex < images.Length - 1)
            {
                images[currentIndex].gameObject.SetActive(false);
                currentIndex++;
                images[currentIndex].gameObject.SetActive(true);
            }
            else
            {SceneManager.LoadScene("MainMenu");}
        }
    }
}using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI;
public class LastScene : MonoBehaviour
{
    public Image[] images;
    private int currentIndex = 0;
    void Start()
    {
        for (int i = 1; i < images.Length; i++)
        {
            images[i].gameObject.SetActive(false);
        }
    }
    void Update()
    {
        if (Input.GetKeyDown(KeyCode.Space))
        {
            if (currentIndex < images.Length - 1)
            {images[currentIndex].gameObject.SetActive(false);
                currentIndex++;
                images[currentIndex].gameObject.SetActive(true);}
            else
            {SceneManager.LoadScene("MainMenu");}
        }
    }
}
```