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 игры «Forgotten World» на платформе Unity

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 09.03.04.2024.308-371.ВКР

Научный руководитель,
доцент кафедры СП, к.п.н.
_____ О.Н. Иванова

Автор работы,
студент группы КЭ-433
_____ Д.С. Форсов

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-433
Форсову Дмитрию Сергеевичу,
обучающемуся по направлению
09.03.04 «Программная инженерия»

1. **Тема работы** (утверждена приказом ректора от 22.04.2024 г. № 764-13/12) Разработка игры «Forgotten World» 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/Manual/index.html (дата обращения: 05.05.2024 г.).
3.2. Документация по C#. [Электронный ресурс] URL: https://learn.microsoft.com/ru-ru/dotnet/csharp (дата обращения: 05.05.2024 г.).
3.3. Unity Asset Store. [Электронный ресурс] URL: https://assetstore.unity.com (дата обращения: 05.05.2024 г.).
3.4. Ферроне Х. Изучаем C# через разработку игр на Unity. 5-е издание. – СПб.: Питер, 2022. – 400 с.

4. **Перечень подлежащих разработке вопросов**

4.1. Произвести анализ предметной области.
4.2. Спроектировать игровое приложение.
4.3. Реализовать игровое приложение на платформе Unity.
4.4. Провести тестирование игрового приложения.

5. **Дата выдачи задания:** 29.01.2024 г.

Научный руководитель,
доцент кафедры СП, к.п.н. О.Н. Иванова

Задание принял к исполнению Д.С. Форсов

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ 4
1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 6
1.1. Предметная область проекта 6
1.2. Анализ аналогичных проектов 7
1.3. Анализ существующих решений для реализации проекта 11
2. ПРОЕКТИРОВАНИЕ СИСТЕМЫ 14
2.1. Требования к проектируемой системе 14
2.2. Концепция игры 15
2.3. Варианты использования системы 16
2.4. Архитектура разрабатываемой системы 17
2.5. Макеты интерфейса 23
3. РЕАЛИЗАЦИЯ КОМПОНЕНТОВ СИСТЕМЫ 27
3.1. Реализация конечного автомата 27
3.2. Диаграмма последовательности 35
3.3. Реализация компонента ресурсов и характеристик 36
4. ТЕСТИРОВАНИЕ 45
ЗАКЛЮЧЕНИЕ 48
ЛИТЕРАТУРА 49
ПРИЛОЖЕНИЯ 54
Приложение А. Спецификации вариантов использования 54
Приложение Б. Скриншоты итоговой версии игры 56
Приложение В. Диаграмма последовательности процесса атаки 62

ВВЕДЕНИЕ

Актуальность

Разработка компьютерных игр является одной из самых динамично развивающихся отраслей индустрии развлечений. Игровая индустрия превзошла по доходам киноиндустрию и музыкальную индустрию вместе взятые [1, 2]. Такие популярные игры, как «Grand Theft Auto V», «Red Dead Redemption 2», «Fallout 4», зарабатывают больше, чем крупнобюджетные блокбастеры из Голливуда. Прибыль «Grand Theft Auto V» составила 6 миллиардов долларов, что опережает сборы культовых фильмов Джеймса Кэмерона «Аватар» (3 миллиарда долларов), «Аватар: Путь воды» (2,2 миллиарда долларов) и «Титаник» (2,2 миллиарда долларов) [3–5].

Данный феномен обусловлен рядом факторов. Во-первых, компьютерные игры предлагают уникальную форму развлечения, которая позволяет игрокам активно взаимодействовать с виртуальным миром. Возможность влиять на окружение дает игроку особенный опыт, который не может быть полностью воссоздан в кино. Во-вторых, технологии в области компьютерных игр активно развиваются, позволяя создавать все более реалистичные и захватывающие игры с использованием виртуальной реальности и продвинутых систем искусственного интеллекта.

Стремительный рост индустрии компьютерных игр и повышенный интерес к игровым продуктам создает потребность в квалифицированных специалистах, способных разрабатывать высококачественные и инновационные игры.

Постановка задачи

Целью данной работы является разработка игры «Forgotten World». Для реализации данной цели были поставлены следующие задачи:

1) произвести анализ предметной области;
2) спроектировать игровое приложение;
3) реализовать игровое приложение;
4) провести тестирование игрового приложения.

Структура и содержание работы

Работа состоит из введения, четырех глав, заключения, списка литературы и приложений. Объем работы составляет: страниц – 62, рисунков – 33, таблиц – 5, листингов – 11, приложений – 3, использовано источников – 49.

Первая глава «Анализ предметной области» содержит описание предметной области, анализ аналогичных проектов и существующих решений для реализации проекта, на основе чего были выбраны средства реализации и выработаны требования к разрабатываемому игровому приложению.

Вторая глава «Проектирование системы» содержит описание функциональных и нефункциональных требований к разрабатываемому игровому приложению, описание вариантов использования системы, описание архитектуры разрабатываемого проекта и представление макетов интерфейса.

Третья глава «Реализация компонентов системы» содержит подробности реализации игры на платформе Unity.

Четвертая глава «Тестирование» содержит результаты функционального тестирования.

В заключении описываются основные результаты, полученные в процессе выполнения проекта, и рассматриваются дальнейшие пути развития игрового приложения.

В приложении А представлены спецификации вариантов использования системы. В приложении Б представлены скриншоты итоговой версии игрового приложения. В приложении В представлена диаграмма последовательности процесса атаки.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. Предметная область проекта

Жанр игры

Разрабатываемое игровое приложение будет сочетать в себе такие жанры, как action-adventure и survival.

Adventure – это жанр компьютерных игр, в котором игрок исследует виртуальный мир, продвигаясь по увлекательному сюжету. Игры этого жанра предлагают множество заданий, локаций и персонажей [6].

Примерами игр в жанре adventure являются такие классические игры, как «King's Quest» (1984), «The Secret of Monkey Island» (1990), а также более современные игры, вроде «The Walking Dead: The Game» (2012) и «Life is Strange» (2015).

Action – это жанр компьютерных игр, который характеризуется динамичным геймплеем, где основное внимание уделяется передвижению персонажа и сражениям с противниками. Он требует от игрока высокой степени координации между зрительными и моторными навыками, а также способности быстро реагировать на события в игре [7].

Примерами игр в жанре action являются «Half-Life 2» [8], «Batman: Arkham City» [9], «Max Payne» [10], «Red Dead Redemption» [11], «The Elder Scrolls 5: Skyrim» [12], «Portal» [13, 14].

Action-adventure – это жанр компьютерных игр, который представляет собой смешение жанров action и adventure [15].

Survival – это жанр компьютерных игр, в котором игрок выживает во враждебной среде с ограниченными ресурсами и возможностями. Игры этого жанра часто включают в себя создание предметов и постройку убежища [16].

Началом этого жанра можно считать игру «Oregon Trail» (1975), в которой игроку необходимо заботиться о группе переселенцев во время их путешествия по Орегонской тропе. Расцвет жанра пришелся на 2010-е годы, когда стали популярными такие игры, как «Minecraft» [17] и «DayZ» [18].

Статистика аналогов выбранного жанра

Среди популярных игр в выбранном жанре можно упомянуть «Grand Theft Auto V» [19], «Red Dead Redemption 2» [20], «God Of War» [21], «Tomb Rider» [22], «Assassin's Creed Valhalla» [23], «Subnautica» [24]. Некоторые из этих игр входят в топ лучших игр всех времен по версии сайта Metacritic [25]. Популярность игр в жанре action-adventure обусловлена комбинацией захватывающего сюжета и интенсивных сражений с противниками. Добавление элементов жанра survival потребует от игрока использования стратегического мышления, что, в свою очередь, повысит степень напряжения и увлекательности игры.

Потрет типичного пользователя

Портрет пользователя данной игры будет включать любителей захватывающих и насыщенных игровых сюжетов, готовых столкнуться с интенсивными сражениями.

1.2. Анализ аналогичных проектов

«Don’t Starve» – это игра в жанре action-adventure с элементами survival, разработанная и выпущенная Klei Entertainment в 2013 году. Игрок управляет персонажем по имени Уилсон, который оказывается в мире, полном опасностей и загадок, и должен выживать, собирая ресурсы и строя убежища. На рисунке 1 представлен скриншот из игры «Don’t Starve».

Преимущества:

1) уникальный визуальный стиль;
2) многообразие локаций и объектов, доступных для исследования;
3) поощряет эксперименты и творческий подход к выживанию.

Недостатки:

1) отсутствует сюжетная линия, которая могла бы заинтересовать игрока и поддерживать его мотивацию на протяжении всей игры;
2) геймплей характеризуется низкой скоростью, требуя от игрока много времени на монотонную добычу ресурсов;

3) высокий порог вхождения из-за сложности игры и малого количества обучающего контента.

Рисунок 1 – Игра «Don’t Starve»

«Don't Starve» получила положительные отзывы от критиков и пользовалась успехом среди игроков [26]. Игра была номинирована на несколько премий, включая The Webby Awards, Annie Awards, Canadian Videogame Awards и Golden Joystick Awards [27].

«The Forest» – это игра в жанре action-adventure с элементами survival horror, разработанная Endnight Games Ltd и выпущенная в раннем доступе в мае 2014 года. Окончательный выпуск игры состоялся в 2018 году. Игрок играет за выжившего после авиакатастрофы на необитаемом острове. Он должен строить убежище, собирать ресурсы, добывать еду и питьевую воду, а также бороться с каннибалами, населяющими остров. На рисунке 2 представлен скриншот из игры «The Forest».

Игра имеет следующие преимущества.

1. Игровые персонажи действуют в соответствии с продуманными алгоритмами искусственного интеллекта. Каннибалы используют тактику и работают в команде, что делает игру более сложной и интересной.

2. Игра имеет открытый мир. Игрок может свободно исследовать остров, строить свое убежище, находить еду, оружие и другие ресурсы, необходимые для выживания.

3. Игра комбинирует элементы выживания и хоррора, что создает эмоционально насыщенный и захватывающий игровой опыт.

4. Кооперативный режим. Игроки могут играть вместе и помогать друг другу в выживании.

Недостатки:

1) игра имеет проблемы с оптимизацией и может работать медленно на некоторых компьютерах;

2) ограниченное количество видов каннибалов делают геймплей скучными и однообразными.

Рисунок 2 – Игра «The Forest»

«The Forest» получила положительные отзывы от игроков и критиков [28]. К 2018 году было продано более 5 миллионов копий игры. «The

Forest» также стала достаточно популярной среди стримеров на Twitch и YouTube благодаря своей уникальной концепции. Эта игра была номинирована на Global Game Awards [29].

«Left 4 Dead» – это кооперативный шутер от первого лица, сочетающий в себе жанры action и survival horror, разработанный Turtle Rock Studios, выпущенный в 2008 году. Игроки играют за одного из четырех выживших в зомби-апокалипсисе, борясь с ордами зараженных и пытаясь добраться до места эвакуации. На рисунке 3 представлен скриншот из игры «Left 4 Dead».

Рисунок 3 – Игра «Left 4 Dead»

Игра имеет следующие преимущества.

1. Миссии в игре построены таким образом, чтобы каждая новая попытка прохождения была уникальной.

2. Игра имеет продуманный искусственный интеллект врагов. Зараженные атакуют и преследуют игроков в зависимости от их действий, что создает атмосферу напряжения и страха.

3. Наличие различных режимов: помимо основной кампании есть режимы «Выживание» и «Заражение», которые добавляют разнообразие в игру.

4. Кооперативный режим. Игроки могут играть вместе и помогать друг другу в выживании.

Недостатком является то, что в игре отсутствует сюжет. Хотя игра и строится вокруг концепции выживания в зомби-апокалипсисе, у нее нет конкретной сюжетной линии.

«Left 4 Dead» получила очень высокие оценки от критиков и имела большой успех среди игроков [30]. Игра до сих пор популярна и считается одной из самых социальных и захватывающих многопользовательских игр [31].

1.3. Анализ существующих решений для реализации проекта

Игровой движок

Unity – это популярный многоплатформенный игровой движок с открытым исходным кодом, он используется в создании игр для мобильных устройств, настольных ПК, игровых консолей и виртуальной реальности [32]. Unity имеет широкий спектр инструментов для создания 2D и 3D игр. Он поддерживает создание игр на языках C# и Bolt.

Unreal Engine – это другой популярный движок, который создан компанией Epic Games. Он используется в создании игр для ПК, игровых консолей и виртуальной реальности [33]. Unreal Engine имеет большие возможности для создания игр с высококачественной графикой. Игровой движок поддерживает создание игр на языках программирования C++ и Blueprint.

Хотя Unreal Engine является мощным инструментом для разработки игр, он может быть непригодным для начинающих команд с ограниченным бюджетом и опытом в создании игр. Отталкивающим фактором является то, что для работы с этим движком требуется мощное оборудование и знание языка программирования C++.

В отличие от Unreal Engine, Unity предоставляет более доступный и простой в использовании инструмент, который работает на языке программирования C#. Кроме того, Unity имеет обширную базу знаний [34] и большое сообщество, готовое помочь новым разработчикам в реализации проектов. Он был использован для создания множества качественных игр на различных платформах. Игровой движок Unity послужил основой для следующих игр.

1. «Hollow Knight» [35] – 2D платформер, сочетающий в себе жанры action-adventure и metroidvania, разработанный командой Team Cherry и выпущенный в 2017 году.
2. «Subnautica» [36] – игра про выживание в подводном мире, разработанная компанией Unknown Worlds Entertainment и выпущенная в 2014 году.
3. «Rust» [37] – многопользовательская игра про выживание, разработанная Facepunch Studios и выпущенная в 2013 году.
4. «Ori and the Blind Forest» [38] – action-platformer с элементами metroidvania и красивой анимированной графикой, разработанный Moon Studios и выпущенный в 2015 году.
5. «Firewatch» [39] – игра в жанре adventure с видом от первого лица, разработанная Campo Santo и выпущенная в 2016 году.

Язык программирования

C# – это современный, объектно-ориентированный язык программирования, созданный компанией Microsoft в 2001 году. Он был разработан как язык программирования для платформы .NET Framework. C# сочетает в себе простоту и выразительность языков программирования, таких как Java и C++ с возможностями более новых языков, таких как Ruby и Python [40].

C++ – компилируемый, статически типизированный язык программирования общего назначения, который объединяет в себе возможности

языка C с дополнительными средствами для более эффективного и удобного программирования, такими как классы, наследование, перегрузка операторов и шаблоны [41].

В отличие от C++, C# предоставляет более простой и эффективный процесс управления памятью благодаря механизму сборки мусора. Это позволяет разработчикам сосредоточиться на создании игровой логики и функций вместо того, чтобы тратить время на управление памятью, что способствует ускорению процесса разработки и улучшает безопасность кода. Кроме того, C# обладает понятным и читаемым синтаксисом, что упрощает его понимание и изменение.

Вывод по первому разделу

На основании проведенного анализа существующих решений для реализации проекта было принято решение использовать Unity в сочетании с языком программирования C# для разработки игры. Это связано с тем, что Unity предоставляет широкие возможности для создания игр, не имея высоких требований к аппаратному обеспечению, а C# позволяет упростить процесс управления памятью, повышая безопасность и скорость разработки. Unity также предлагает богатую библиотеку готовых компонентов и инструментов, что значительно ускоряет процесс разработки.

Помимо этого, из анализа аналогичных проектов следует, что игра должна иметь сюжет, который сможет заинтересовать игрока, четкую главную цель для поддержания мотивации на протяжении всей игровой сессии и динамичный геймплей, чтобы каждая новая попытка прохождения была уникальной. Желательно, чтобы игра имела высокий уровень оптимизации, продуманный искусственный интеллект у врагов и предоставляла игроку разнообразные варианты противников.

2. ПРОЕКТИРОВАНИЕ СИСТЕМЫ

2.1. Требования к проектируемой системе

Функциональные требования к проектируемой системе

Функциональные требования определяют функции, которые должны обеспечиваться системой [42].

Игровое приложение должно:

1) отображать главное меню при запуске;

2) предоставлять возможность настройки качества графики, выбора разрешения экрана и переключения между полноэкранным и оконным режимами;

3) показывать прогресс при загрузке игровой сцены на загрузочном экране;

4) предлагать игроку просмотреть обучение перед началом новой игры;

5) генерировать объекты игрового мира случайным образом;

6) предоставлять игроку возможность управлять передвижением игрового персонажа;

7) предоставлять игроку возможность атаковать врагов в ближнем бою;

8) предоставлять игроку возможность вставать в оборонительную стойку для увеличения защитной характеристики игрового персонажа;

9) предоставлять игроку возможность выполнить кувырок для уклонения от атак или сокращения дистанции до врагов;

10) предоставлять игроку возможность просмотреть характеристики игрового персонажа;

11) обеспечивать работу механик сытости и страха;

12) показывать экран победы, когда игрок вошел в дверь;

13) показывать игроку экран с информацией о причине смерти игрового персонажа после его гибели;

14) предоставлять возможность игроку оставить обратную связь;

15) предоставлять возможность игроку сообщить о возникшей ошибке или предложении по улучшению игры.

Нефункциональные требования к проектируемой системе

Нефункциональные требования определяют особые свойства или ограничения, накладываемые на систему [42].

Игровое приложение должно:

1) быть разработано с использованием игрового движка Unity;
2) быть разработано с использованием языка программирования C#;
3) быть совместимо с операционной системой Windows 10 и выше.

2.2. Концепция игры

Разрабатываемая игра будет являться фэнтезийным приключением с элементами выживания, где игрок будет управлять маленькой девочкой, которая оказалась в чужом измерении. Ему предстоит бороться со страхом, монстрами и голодом ради главной цели: найти дверь – единственный выход из этого измерения.

Игровой персонаж будет иметь такие характеристики как урон, защита и скорость перемещения. Ему также будут присущи такие ресурсы как здоровье, сытость и страх. Низкие уровни ресурсов будут влиять на характеристики игрового персонажа. Например, если игровой персонаж голоден, то его скорость передвижения уменьшится, а если он сильно напуган, то будет наносить меньше урона врагам.

В боях с монстрами игрок сможет использовать оружие, а также будет иметь возможность принимать оборонительную стойку для увеличения своей защитной характеристики. Для уклонения от атак или сокращения дистанции до врагов персонаж сможет выполнить кувырок.

На игровой карте будут присутствовать различные объекты, такие как еда, костры и дверь. Еда будет представлена в виде грибов, которые смогут восстанавливать сытость игрового персонажа, костры будут снижать уровень страха, а дверь будет являться главной целью игрока, войдя в которую

он побеждает. Важно отметить, что расположение грибов, дверей и монстров на карте будет случайным, что сделает каждую попытку прохождения игрового уровня уникальной.

2.3. Варианты использования системы

Диаграмма вариантов использования

Для проектирования приложения был выбран графический язык для визуализации, специфицирования, конструирования и документирования систем UML [43]. На рисунке 4 представлена диаграмма вариантов использования игрового приложения.

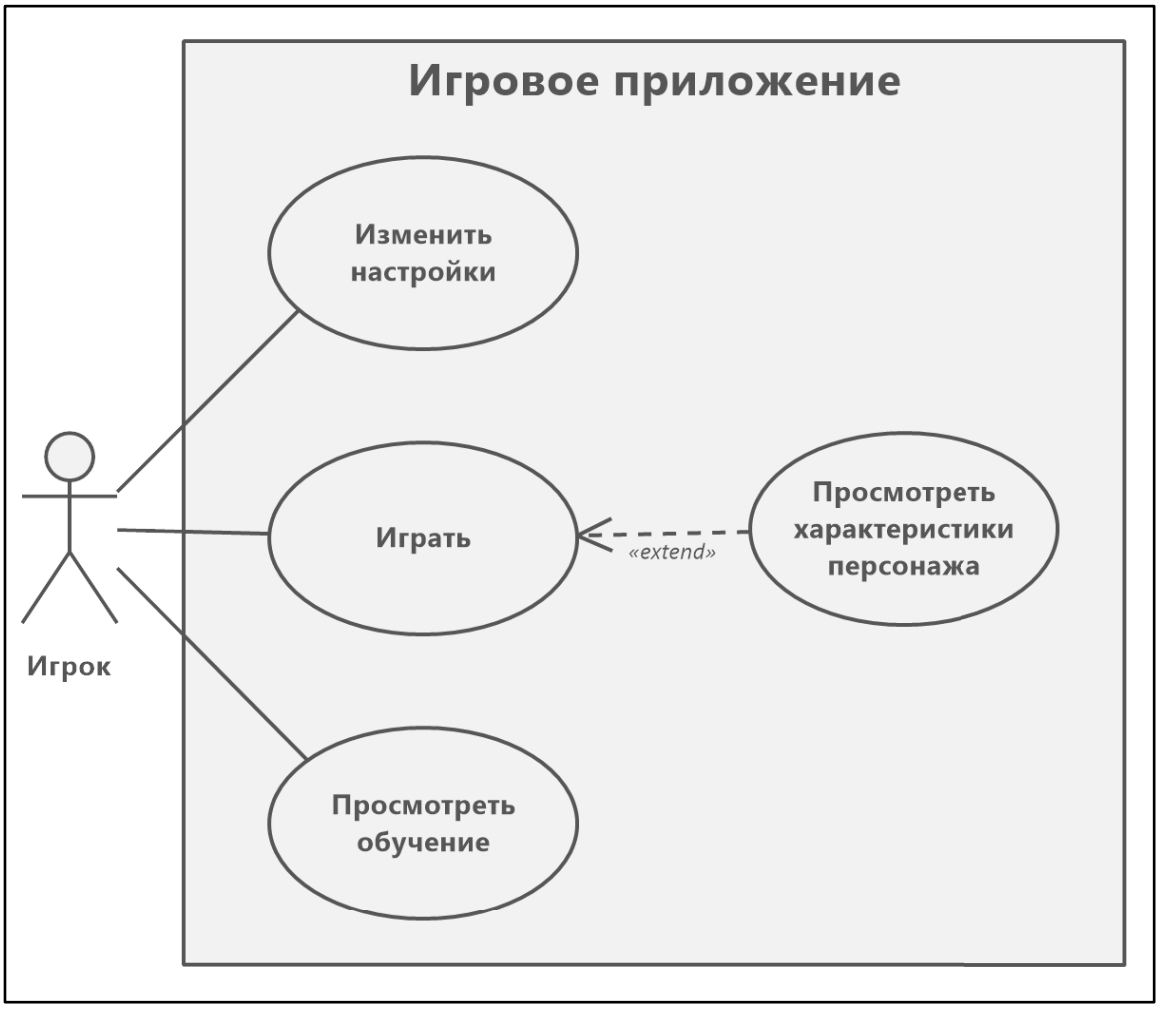

Рисунок 4 – Диаграмма вариантов использования

С игровым приложением взаимодействует единственный актор – игрок. Игрок может просмотреть обучение перед началом игры. Кроме того,

он может настраивать параметры игры перед началом или во время ее прохождения. Дополнительно, игрок может просматривать характеристики игрового персонажа. Спецификации вариантов использования приведены в приложении А.

2.4. Архитектура разрабатываемой системы

Архитектура разрабатываемого игрового приложения основана на применении паттерна проектирования Component. Данный паттерн предоставляет гибкую и модульную структуру разработки, разбивая функциональность приложения на отдельные компоненты, которые могут быть независимо написаны, модифицированы и переиспользованы [44]. Преимущества паттерна Component следующие.

1. Гибкость и расширяемость. Компоненты могут быть легко добавлены, изменены или удалены из объектов игры. Это позволяет легко создавать новые объекты с различной функциональностью путем комбинирования компонентов.
2. Повторное использование. Компоненты представляют собой независимые модули, которые могут быть повторно использованы.
3. Тестирование. Разделение функциональности на отдельные компоненты облегчает тестирование, так как каждый компонент можно тестировать отдельно, независимо от остальной логики игры.
4. Читабельность и понятность кода. Компоненты облегчают понимание архитектуры и логики игры, так как каждый компонент отвечает за конкретную функциональность. Это делает код более читабельным, понятным и поддерживаемым.

На рисунке 5 представлена диаграмма компонентов игрового приложения.

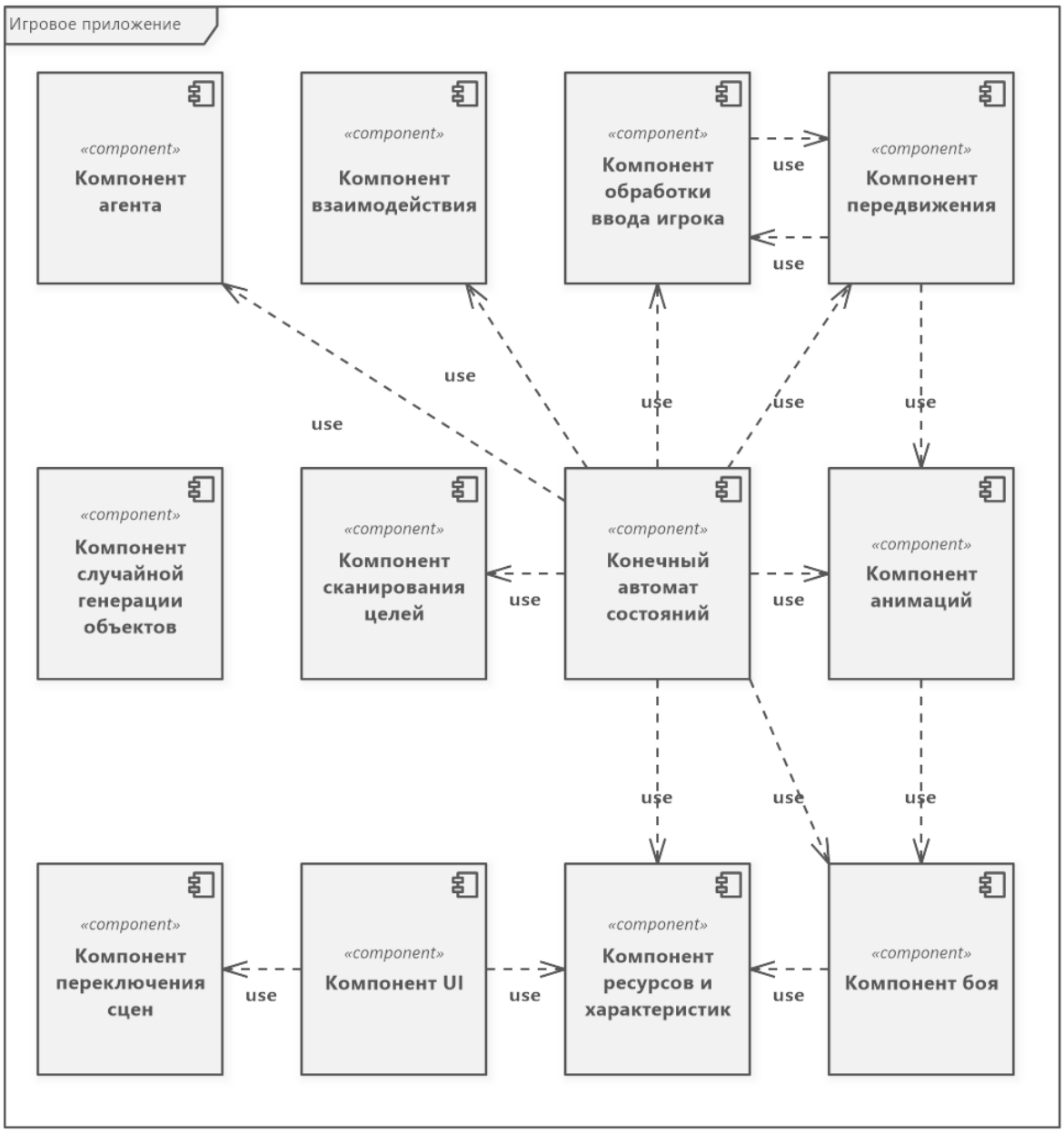

Рисунок 5 – Диаграмма компонентов

Проектируемая система состоит из следующих компонентов.

1. Компонент конечного автомата. Этот ключевой компонент отвечает за управление поведением агентов в игре, включая игрока и врагов.

2. Компонент ресурсов и характеристик. Отвечает за управление ресурсами и характеристиками, связанными с различными игровыми сущностями.

3. Компонент передвижения. Отвечает за управление передвижением агентов. Обеспечивает функциональность кувырка.

4. Компонент боя. Отвечает за реализацию боевой системы в игре. Обеспечивает функциональность оружия, атак и оборонительной стойки.

5. Компонент случайной генерации объектов. Отвечает за случайную генерацию объектов в игровом мире.

6. Компонент переключения сцен. Обеспечивает переходы между игровыми сценами.

7. Компонент анимаций. Обеспечивает функциональность для проигрывания, прерывания и синхронизации анимаций.

8. Компонент сканирования целей. Отвечает за сканирование и выбор целей для атаки и взаимодействия.

9. Компонент взаимодействия. Отвечает за взаимодействие с объектами игрового мира.

10. Компонент обработки ввода игрока. Отвечает за обработку пользовательского ввода в игре.

11. Компонент UI. Обеспечивает взаимодействие игрока с элементами пользовательского интерфейса.

12. Компонент агента. Отвечает за хранение информации об агенте. Обеспечивает удобный доступ к часто используемым компонентам агента.

Рассмотрим компонент конечного автомата подробнее. Конечный автомат – это автомат с конечным числом состояний. В любой момент времени он находится только в одном состоянии [45]. Реализация конечного автомата включает следующие составляющие.

1. State – состояние конечного автомата. Содержит список действий (action) и список переходов (transition).

2. Action – действие, которое выполняется текущим состоянием.

3. Transition – отвечает за переход между состояниями конечного автомата на основе решений о переходе (decision).

4. Decision – определяет решение о переходе из одного состояния конечного автомата в другое.

Каждая из этих составляющих будет представлена в виде отдельного класса ScriptableObject. ScriptableObject – это контейнер данных, который можно использовать для хранения больших объемов информации независимо от экземпляров класса [46]. Использование ScriptableObject позволит создавать и настраивать компоненты конечного автомата состояний прямо в инспекторе Unity без необходимости внесения изменений в код. Кроме того, такой подход способствует повторному использованию действий и решений, делая систему легко масштабируемой. Диаграммы состояний конечного автомата игрока и врага представлены на рисунках 6 и 7 соответственно.

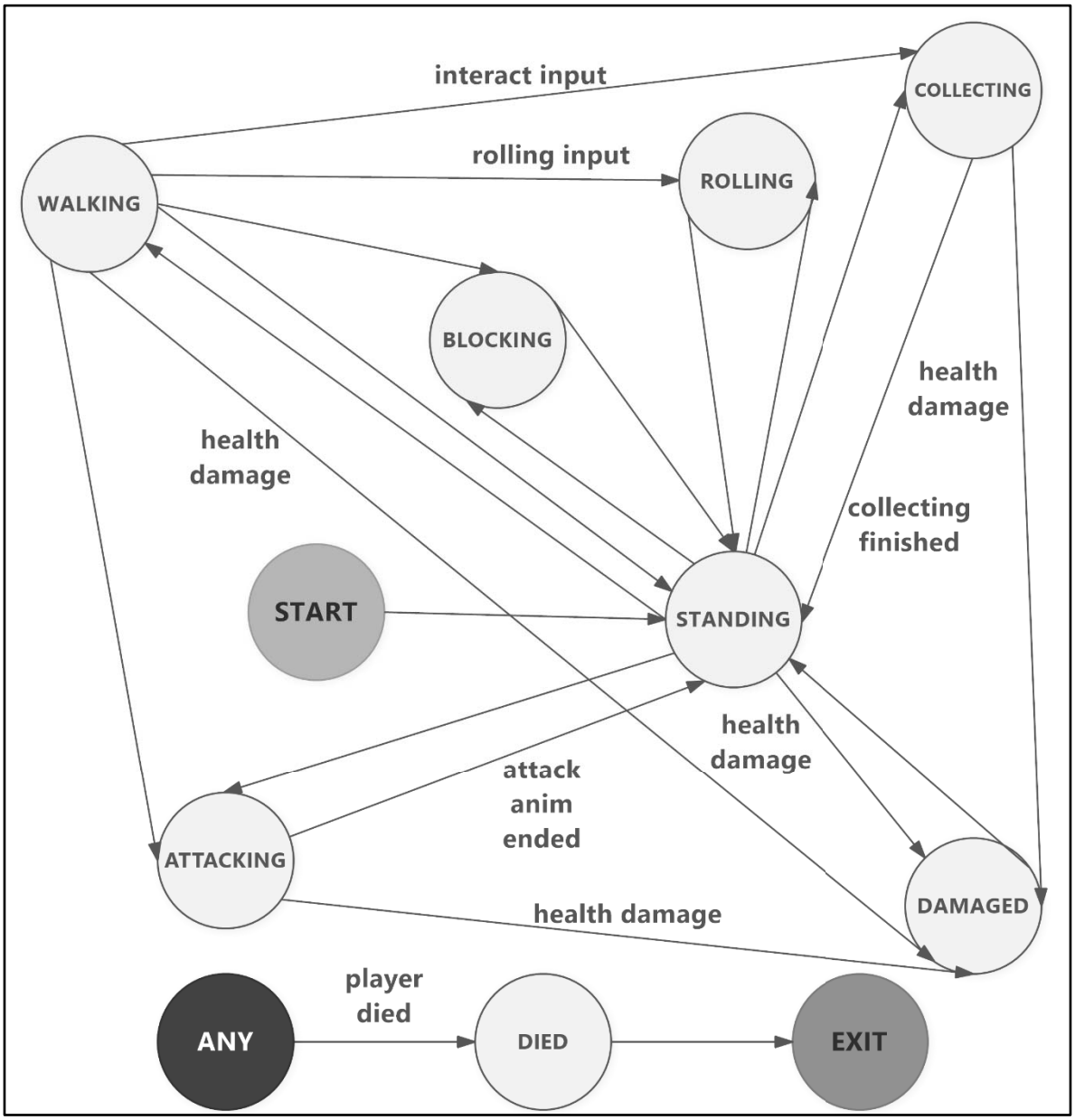

Рисунок 6 – Диаграмма состояний конечного автомата игрока

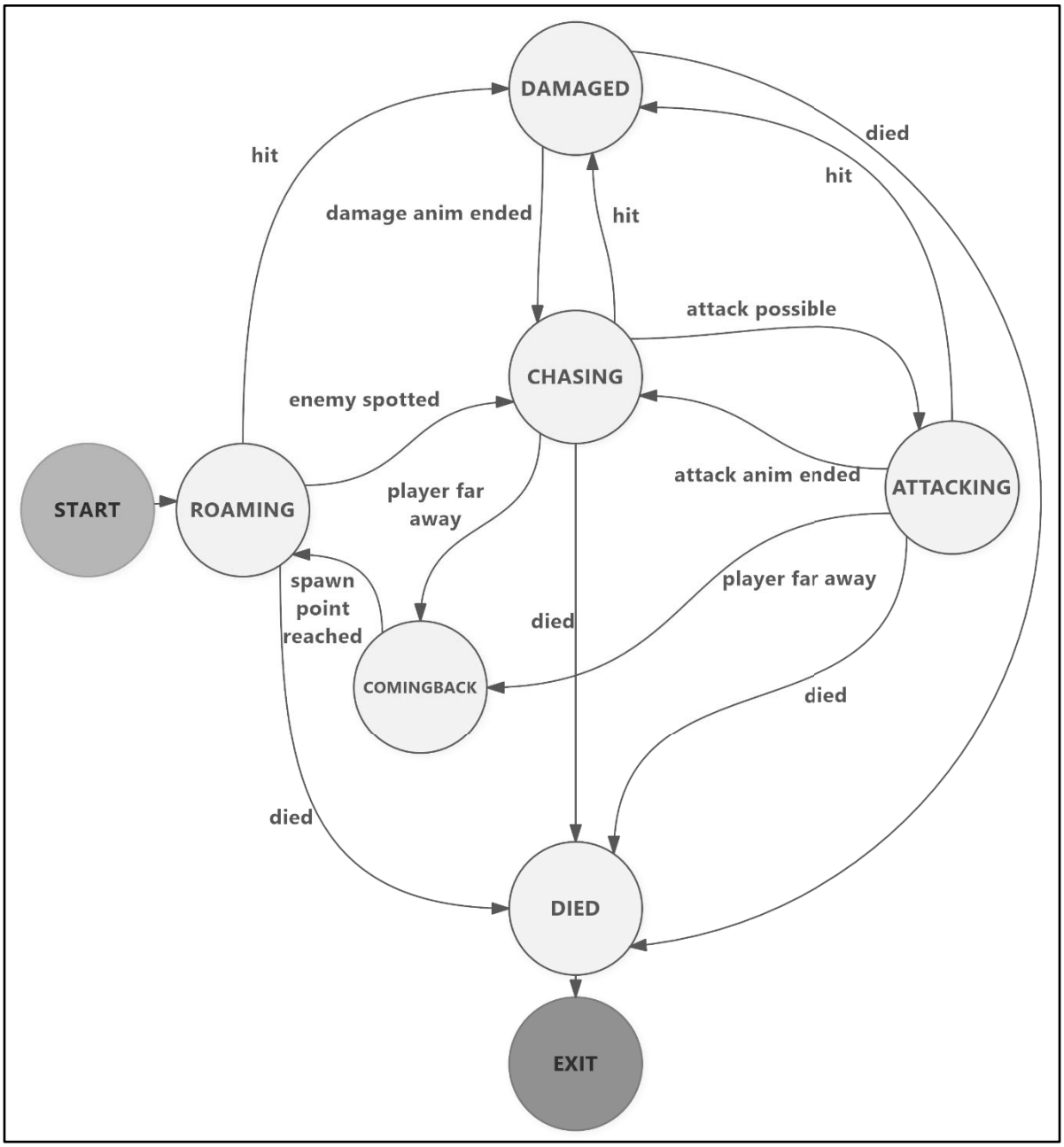

Рисунок 7 – Диаграмма состояний конечного автомата врага

ScriptableObject не имеет собственного экземпляра и не может хранить информацию, связанную с конкретным игровым объектом. Для решения этой проблемы используется паттерн Blackboard. Blackboard – шаблон проектирования, основанный на идее централизованного общего пространства памяти, называемой blackboard [47]. Использование данного паттерна позволяет действиям и решениям сохранять необходимые данные для конкретного GameObject и использовать их в работе.

Рассмотрим компонент ресурсов и характеристик подробнее. Его функциональность включает в себя хранение, добавление и изменение ресурсов и характеристик, связанных с различными игровыми сущностями. Рассмотрим основные составляющие компонента ресурсов и характеристик.

Характеристика представляет собой атрибут игрового объекта, который обычно остается постоянным или изменяется редко. Примерами могут служить такие атрибуты, как урон, время перезарядки, дальность стрельбы и прочие.

Модификатор характеристики позволяет осуществлять обратимые изменения в характеристиках игровых объектов. Он определяет величину изменения, тип модификации и порядок применения модификатора, а также предоставляет возможность указать длительность его действия в игровом мире.

Ресурс представляет собой атрибут игрового объекта, который изменяется чаще характеристик. Он может расходоваться и восстанавливаться во время игрового процесса. Примерами могут служить здоровье, голод, жажда и прочие.

Модификатор ресурса позволяет изменять значение ресурса с помощью заданного метода изменения (на определенное число или процент от максимального значения).

Ресурсный тик представляет собой механизм автоматического изменения значения ресурса через определенные временные интервалы с возможностью установки ограничений на количество таких изменений или их общее время выполнения.

Эффект позволяет одновременно применять обратимые изменения к ресурсам и характеристикам на определенный период времени. Он может быть положительным (бафф), отрицательным (дебафф) или нейтральным.

2.5. Макеты интерфейса

«Главное меню» – стартовый экран, показываемый игроку после запуска игрового приложения. Экран содержит кнопки «Новая игра», «Продолжить», «Настройки», «Обратная связь», «Сообщить» и «Выход». Макет экрана представлен на рисунке 8.

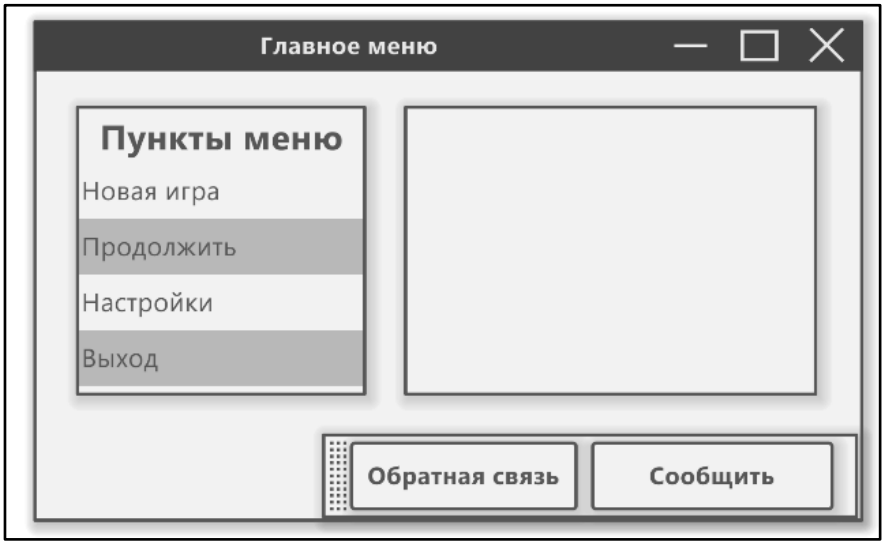

Рисунок 8 – Макет экрана «Главное меню»

«Настройки» – экран игрового приложения, позволяющий игроку настроить параметры игры по своим предпочтениям. Экран содержит выпадающие списки для смены разрешения экрана и качества графики, переключатель полноэкранного режима и кнопку «Применить» для применения настроек. Макет экрана представлен на рисунке 9.

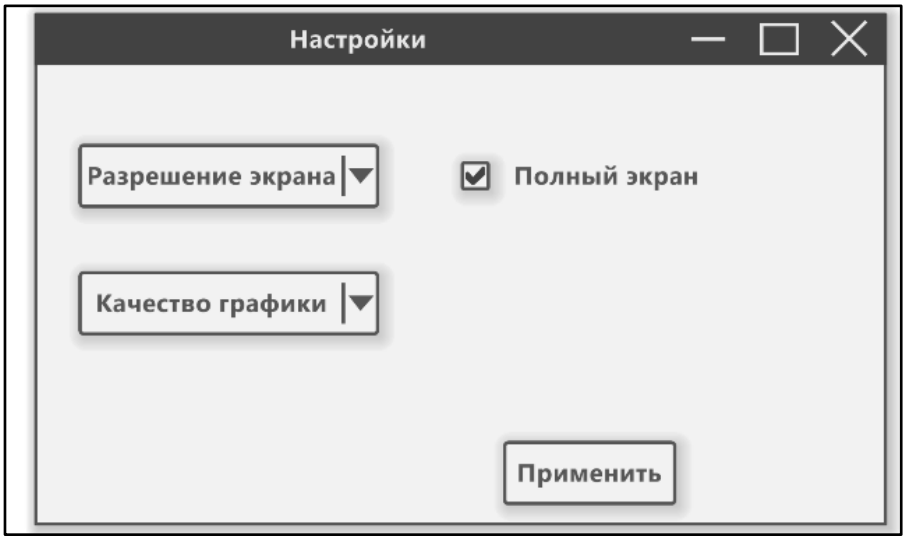

Рисунок 9 – Макет экрана «Настройки»

«Пауза» – экран игрового приложения, позволяющий игроку приостановить игровой процесс. Содержит кнопки «Продолжить», «Настройки», «Главное меню», «Обратная связь», «Сообщить» и «Выход». Макет экрана представлен на рисунке 10.

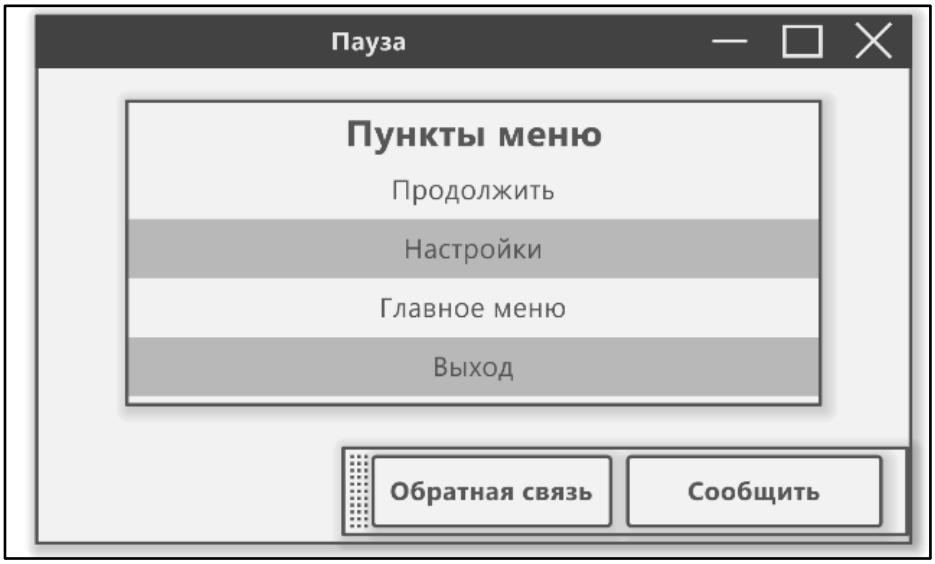

Рисунок 10 – Макет экрана «Пауза»

«Загрузка сцены» – экран игрового приложения, который показывается игроку во время загрузки игровой сцены и отображает текущий прогресс загрузки благодаря полосе прогресса. После окончания загрузки сцены появляется кнопка «Продолжить», при нажатии на которую, игрок попадает на игровую сцену. Макет экрана представлен на рисунке 11.

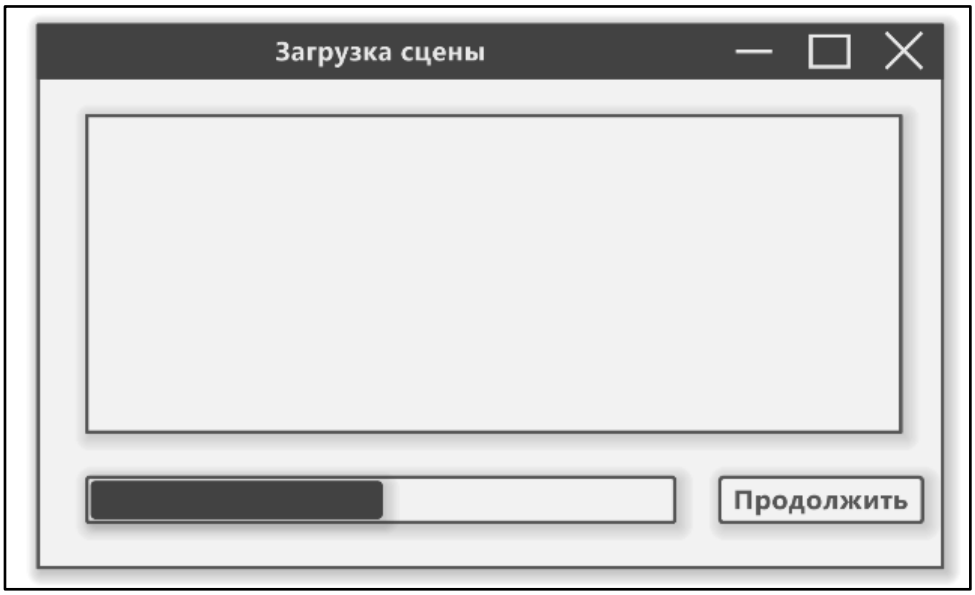

Рисунок 11 – Макет экрана «Загрузка сцены»

«Обратная связь» – экран игрового приложения, позволяющий получить обратную связь от игрока путем ответов на вопросы. Ответы на вопросы представлены в виде шкалы от 1 до 10. Для отправки обратной связи предусмотрена кнопка «Отправить». Макет экрана представлен на рисунке 12.

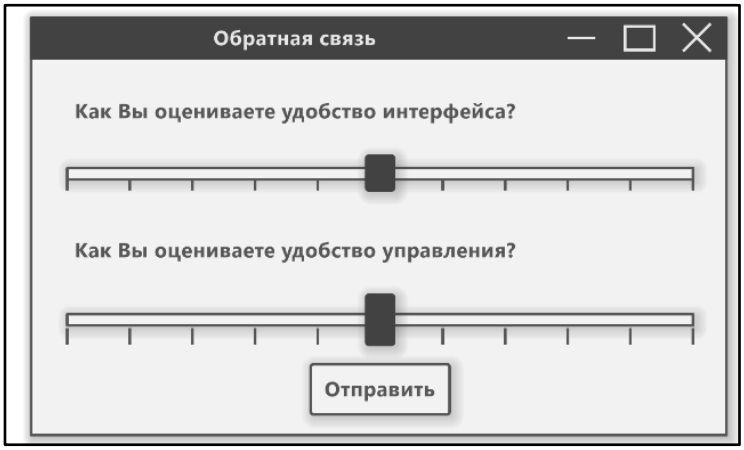

Рисунок 12 – Макет экрана «Обратная связь»

«Сообщить» – экран игрового приложения, позволяющий игроку сообщить о возникшей ошибке или предложении по улучшению игры. Содержит выпадающее меню для выбора типа сообщения (баг, предложение), поля для ввода темы и содержания сообщения. Дополнительно, присутствуют поля для указания имени и электронной почты игрока, которые позволяют установить обратную связь с ним. Макет экрана представлен на рисунке 13.

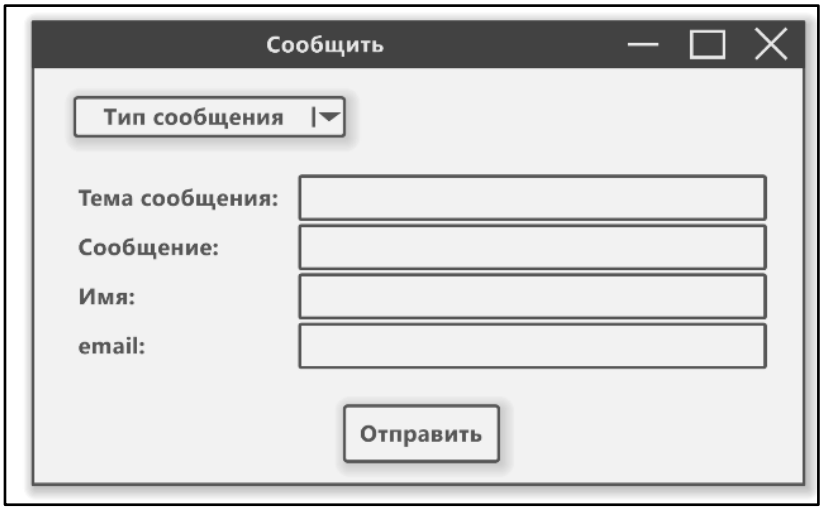

Рисунок 13 – Макет экрана «Сообщить»

На рисунке 14 представлены всплывающее окно «Характеристики» и диалоговое окно «Обучение». Окно «Характеристики» содержит в себе значения скорости, защиты, урона и другой информации о игровом персонаже. «Обучение» представляет собой всплывающее окно, которое появляется перед игроком с вопросом о желании увидеть обучение. Оно содержит кнопки выбора: «Да», «Нет».

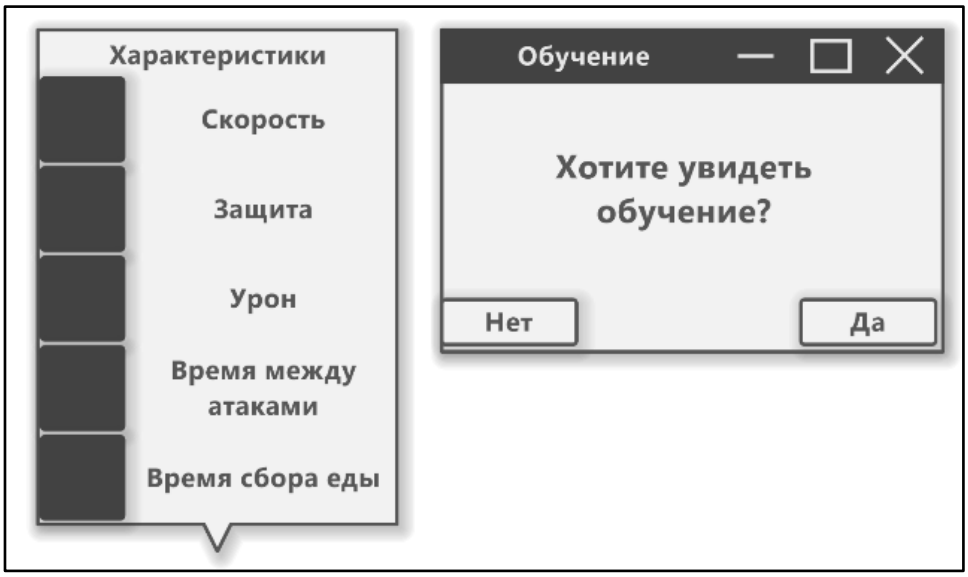

Рисунок 14 – Макет всплывающего окна «Характеристики» (слева) и макет диалогового окна «Обучение» (справа)

Вывод по второму разделу

В разделе были представлены функциональные и нефункциональные требования к проектируемой системе, а также была расписана концепция разрабатываемой игры. Для проектирования системы использовался графический язык UML. В качестве иллюстрации взаимодействия игрока с игровым приложением была представлена диаграмма вариантов использования. В процессе разработки архитектуры игры был выбран и применен паттерн проектирования Component для эффективного разделения ответственности между модулями системы. Для описания архитектуры была представлена диаграмма компонентов игрового приложения и диаграммы состояний конечных автоматов игрока и врага. Помимо этого, были представлены макеты интерфейса разрабатываемого игрового приложения.

3. РЕАЛИЗАЦИЯ КОМПОНЕНТОВ СИСТЕМЫ

Средства реализации

Для реализации игрового приложения использованы:

1) игровой движок Unity версии 2021.3 LTS;
2) интегрированная среда разработки Visual Studio Community 2022;
3) язык программирования C# версии 9.

Важную роль в разработке игры сыграл художник, Павел Кондратенков [48], который занимался созданием визуальной части проекта. Его вклад включал следующие аспекты.

1. Создание детализированных 3D-моделей персонажей, окружения и предметов, которые стали основой визуального оформления игры.
2. Создание различных игровых локаций и сцен, которые придали игре уникальную атмосферу.
3. Рисование различных иконок и изображений, которые используются в интерфейсе игры, помогая сделать его более привлекательным для пользователя.
4. Подбор анимаций для персонажей, которые включают в себя движения, действия и эффекты, добавляющие реалистичность и динамичность игровому процессу.

3.1. Реализация конечного автомата

На рисунке 15 представлена диаграмма классов конечного автомата. Диаграмма наглядно демонстрирует структуру и взаимосвязи между основными классами, используемыми для реализации конечного автомата в игровом приложении. Рассмотрим данные классы подробнее.

`BaseStateMachine` – основной класс конечного автомата, является наследником `MonoBehaviour`. `MonoBehaviour` – базовый класс игрового движка Unity, позволяющий присоединять классы-наследники к игровым объектам через инспектор [49].

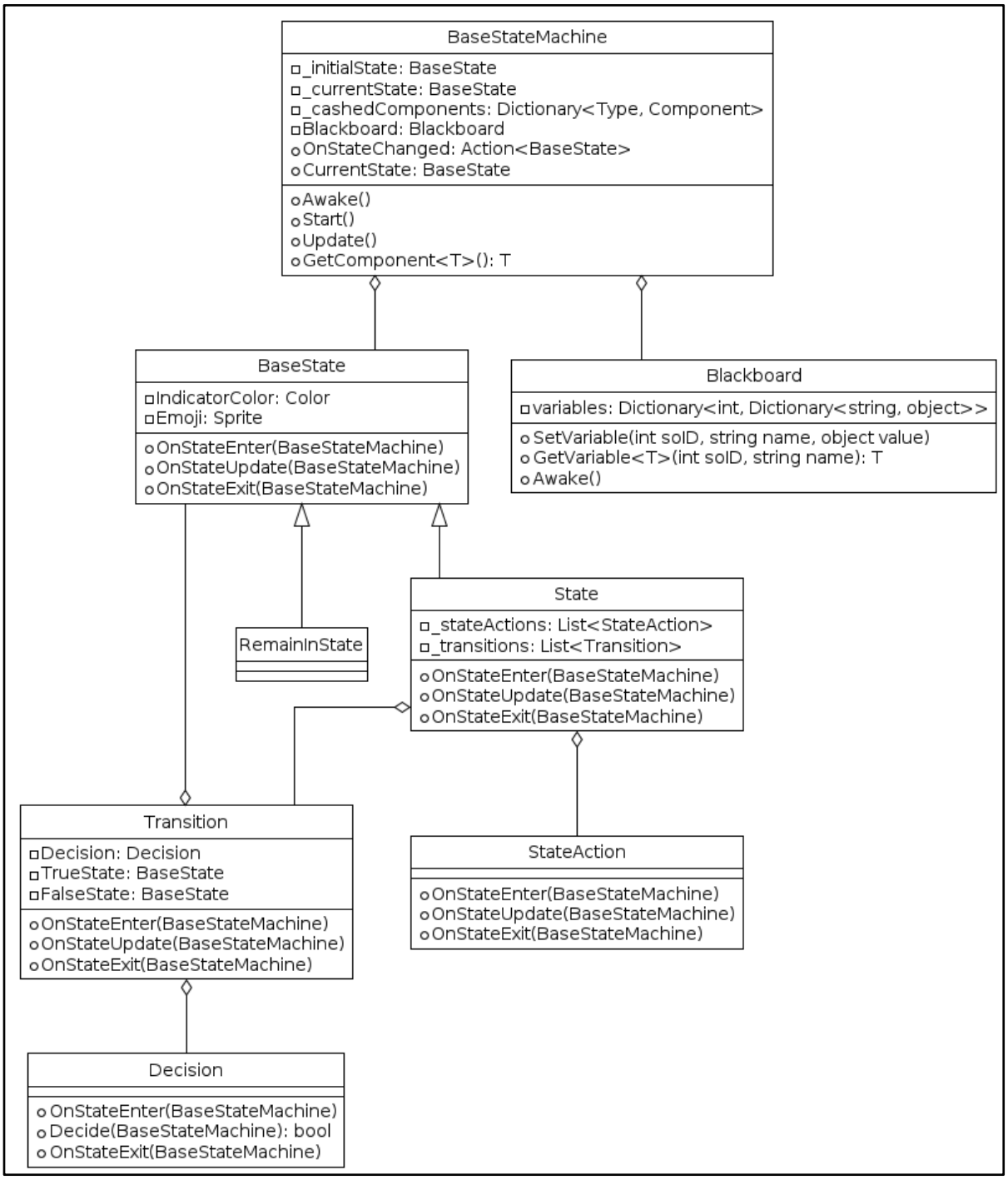

Рисунок 15 – Диаграмма классов конечного автомата

В игровом движке Unity метод `Update()` предоставляется классом `MonoBehaviour` и вызывается движком на каждом кадре игры. Свойство `CurrentState` представляет собой текущее состояние конечного автомата. В классе `BaseStateMachine` в методе `Update()` вызывается метод `OnStateUpdate()` у текущего состояния, что позволяет обновлять логику действий и переходов этого состояния каждый кадр игры. При изменении

состояния, у предыдущего состояния вызывается метод `OnStateExit()`, у нового состояния вызывается метод `OnStateEnter()`. Реализация класса `BaseStateMachine` представлена в листинге 1.

Листинг 1 – Реализация класса `BaseStateMachine`

```
using System;
using System.Collections.Generic;
using UnityEngine;
/// <summary>
/// Конечный автомат
/// </summary>
[RequireComponent(typeof(Blackboard))]
public class BaseStateMachine : MonoBehaviour
{
    public event Action<BaseState> OnStateChanged;
    [SerializeField] private BaseState _initialState;
    private BaseState _currentState;
    private Dictionary<Type, Component> _cashedComponents;
    public Blackboard Blackboard { get; private set; }
    public virtual BaseState CurrentState
    {
        get => _currentState;
        set
        {
            if (value is not RemainInState)
            {
                _currentState?.OnStateExit(this);
                _currentState = value;
                _currentState.OnStateEnter(this);
                OnStateChanged?.Invoke(_currentState);
            }
        }
    }
    protected virtual void Awake()
    {
        _cashedComponents = new Dictionary<Type, Component>();
        Blackboard = GetComponent<Blackboard>();
    }
    private void Start()
    {
        CurrentState = _initialState;
    }
    private void Update()
    {
        CurrentState.OnStateUpdate(this);
    }
}
```

`BaseState` – базовый класс состояния конечного автомата, является наследником `ScriptableObject`. Содержит виртуальные методы, каждый из которых принимает объект класса `BaseStateMachine` в качестве контекста текущего состояния конечного автомата. Реализация класса `BaseState` представлена в листинге 2.

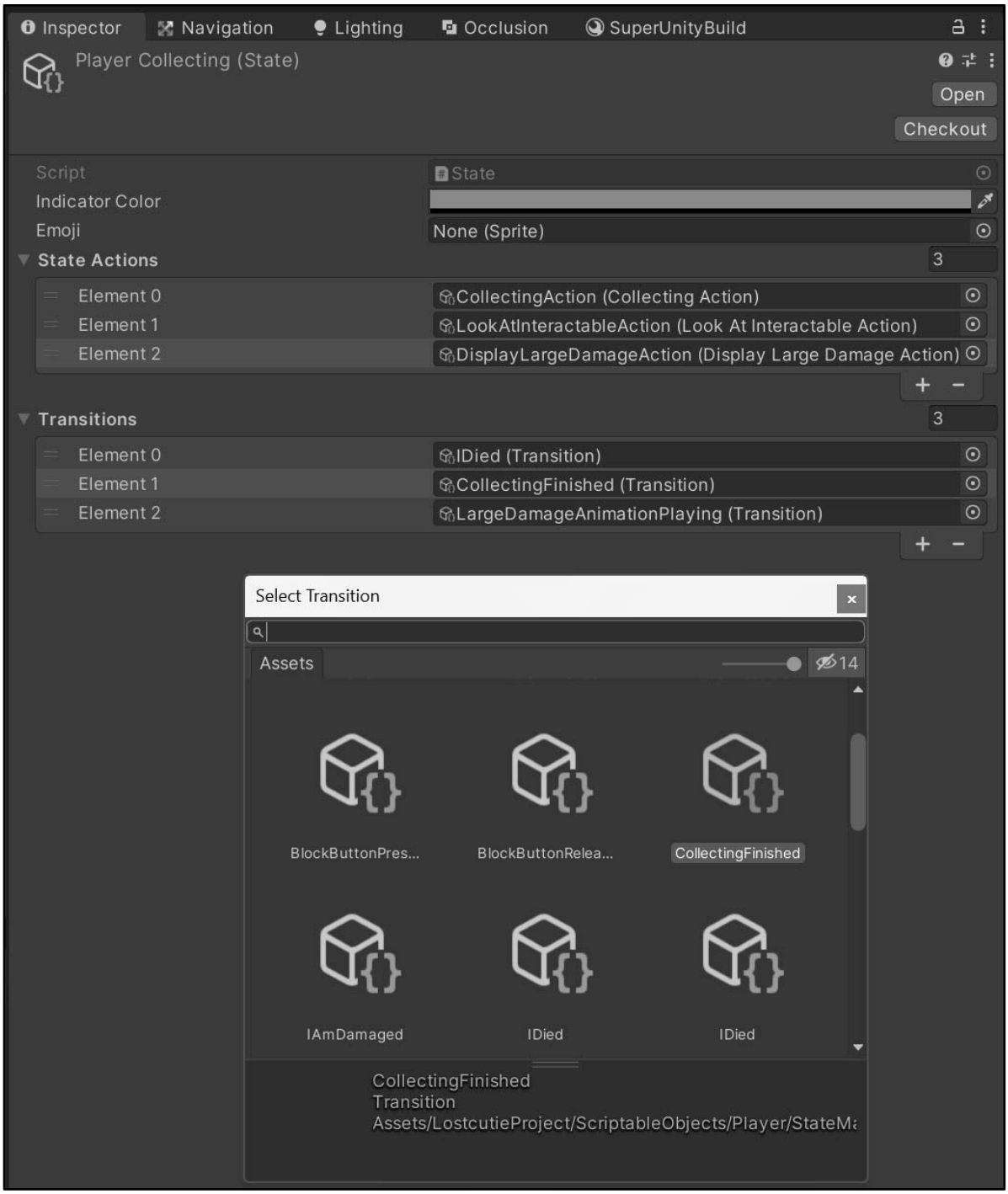

Рисунок 16 – Добавление перехода для состояния Player Collecting

Листинг 2 – Реализация класса `BaseState`

```
using UnityEngine;
/// <summary>
/// Базововое состояние конечного автомата
/// </summary>
public class BaseState : ScriptableObject
{
    public virtual void OnStateEnter(BaseStateMachine machine) { }
    public virtual void OnStateUpdate(BaseStateMachine machine) { }
    public virtual void OnStateExit(BaseStateMachine machine) { }
}
```

`State` – класс состояния конечного автомата, который наследуется от класса `BaseState`. В данном классе выполняются действия и переходы, назначенные текущему состоянию. Благодаря использованию `ScriptableObject` имеется возможность назначать действия и переходы для состояния прямо через инспектор Unity (рисунок 16). Более того, атрибут `CreateAssetMenu` позволяет создавать `ScriptableObject` через меню создания объекта. Например, строка кода `CreateAssetMenu(menuName = "FSM/State")` добавляет в контекстное меню создания объекта пункт FSM, в котором можно найти кнопку создания нового состояния (рисунок 17). Реализация класса `State` представлена в листинге 3.

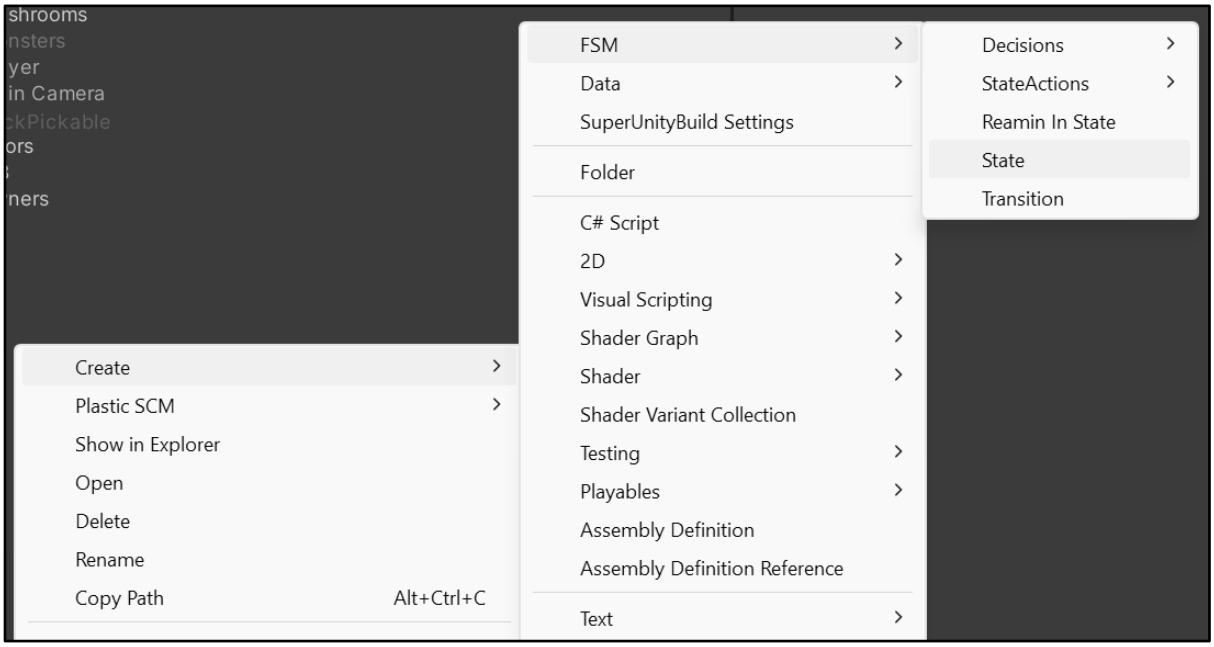

Рисунок 17 – Создание объекта `State` через меню инспектора

Листинг 3 – Реализация класса `State`

```
using System.Collections.Generic;
using UnityEngine;
[CreateAssetMenu(menuName = "FSM/State")]
public sealed class State : BaseState
{
    [SerializeField] private List<StateAction> _stateActions = new
List<StateAction>();
    [SerializeField] private List<Transition> _transitions = new
List<Transition>();
    public override void OnStateEnter(BaseStateMachine machine)
    {
        foreach (var action in _stateActions)
        {
            action.OnStateEnter(machine);
```

```
        }
    }
    public override void OnStateUpdate(BaseStateMachine machine)
    {
        foreach (var action in _stateActions)
        {
            action.OnStateUpdate(machine);
        }
        BaseState stateBeforeTranstion = machine.CurrentState;
        foreach (var transition in _transitions)
        {
            transition.OnStateUpdate(machine);
            if (machine.CurrentState != stateBeforeTranstion) break;
        }
    }
    public override void OnStateExit(BaseStateMachine machine)
    {
        foreach (var action in _stateActions)
        {
            action.OnStateExit(machine);
        }
    }
}
```

`Transition` – класс, отвечающий за переход из одного состояния конечного автомата в другое на основе решения о переходе `Decision`. Наследуется от класса `ScriptableObject`. Поле `TrueState` представляет состояние, в которое произойдет переход, если решение `Decision` вернет значение `true`. Поле `FalseState` представляет состояние, в которое произойдет переход, если решение `Decision` вернет значение `false`. Настраивать переход также можно через инспектор Unity (рисунок 18). Реализация класса `Transition` представлена в листинге 4.

Листинг 4 – Реализация класса `Transition`

```
using UnityEngine;
/// <summary>
/// Отвечает за переход между состояниями конечного автомата на основе
решений о переходе
/// </summary>
[CreateAssetMenu(menuName = "FSM/Transition")]
public sealed class Transition : ScriptableObject
{
    [SerializeField] private Decision Decision;
    [SerializeField] private BaseState TrueState;
    [SerializeField] private BaseState FalseState;
    public void OnStateEnter(BaseStateMachine machine)
    {
        Decision.OnStateEnter(machine);
    }
    public void OnStateUpdate(BaseStateMachine machine)
    {
        if (Decision.Decide(machine))
        {
```

```
            machine.CurrentState = TrueState;
        }
        else
        {
            machine.CurrentState = FalseState;
        }
    }
    public void OnStateExit(BaseStateMachine machine)
    {
        Decision.OnStateExit(machine);
    }
}
```

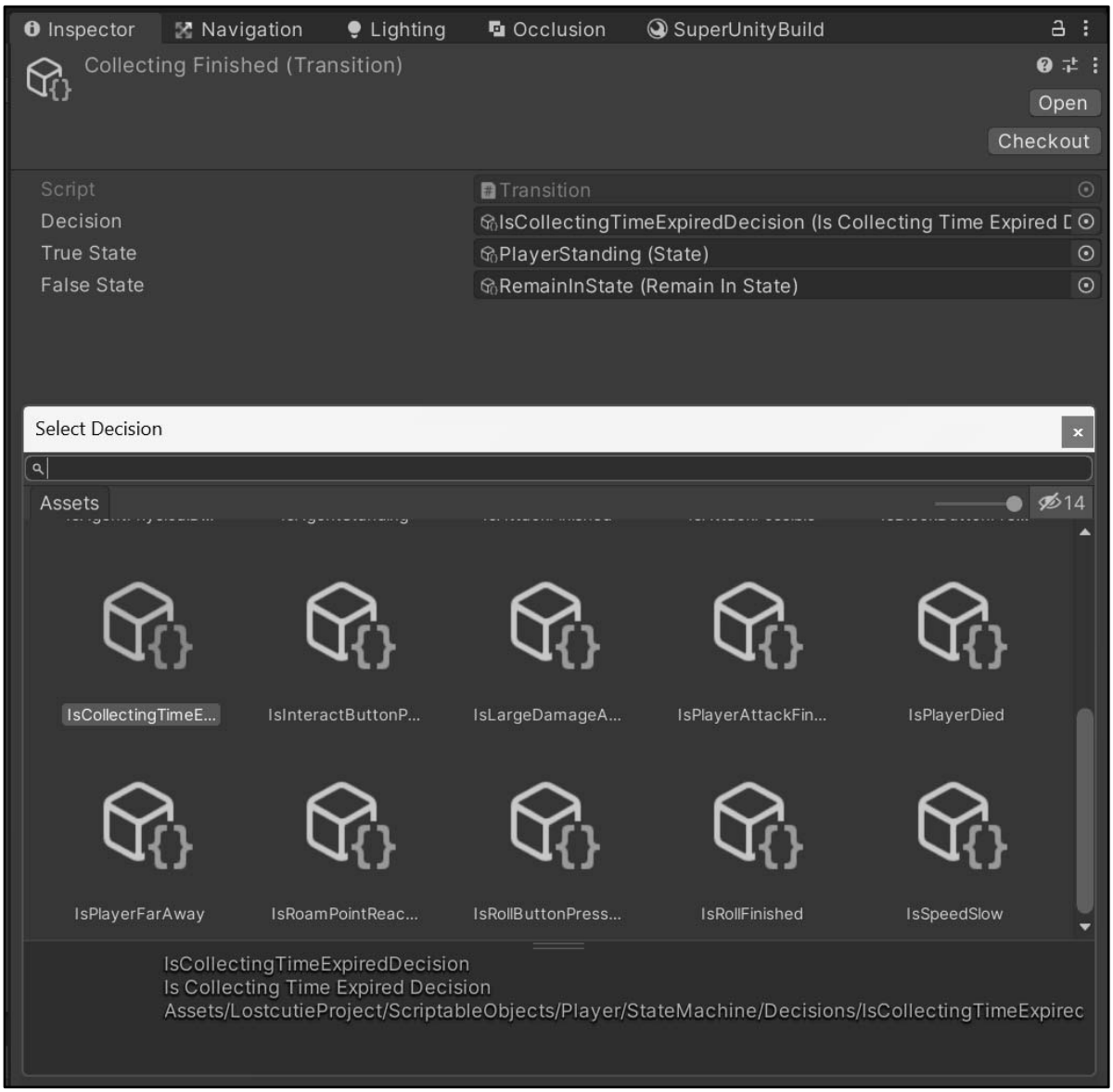

Рисунок 18 – Настройка перехода Collecting Finished

`RemainInState` – класс, который удобно использовать, когда не требуется осуществлять переход в новое состояние. Пример использования можно увидеть на рисунке 18, где поле `FalseState` задано этим классом. В

данном примере, пока сбор ресурса не закончился, игрок должен оставаться в состоянии PlayerCollecting.

Decision – абстрактный класс, используемый Transition для принятия решения о переходе из одного состояния конечного автомата в другое. Наследуется от класса ScriptableObject. Данный класс имеет метод Decide(), который вызывается переходом Transition для принятия решения о переходе между состояниями. Реализация класса Decision представлена в листинге 5.

Листинг 5 – Реализация класса Decision

```
using UnityEngine;
/// <summary>
/// Определяет решение о переходе из одного состояния конечного автомата в
другое
/// </summary>
public abstract class Decision : ScriptableObject
{
    public virtual void OnStateEnter(BaseStateMachine machine) { }
    public abstract bool Decide(BaseStateMachine machine);
    public virtual void OnStateExit(BaseStateMachine machine) { }
}
```

Blackboard – класс, используемый действиями и решениями для хранения необходимых данных, присущих конкретному игровому объекту. Является наследником класса MonoBehaviour. Обращение к конкретной переменной происходит через ее строковое название и идентификатор ScriptableObject. Класс содержит обобщенный метод GetVariable<T>(), который позволяет получать значения переменных без явного приведения к нужному типу. Реализация класса Blackboard представлена в листинге 6.

Листинг 6 – Реализация класса Blackboard

```
using System.Collections.Generic;
using UnityEngine;
/// <summary>
/// Отвечает за хранение переменных, которые используются компонентами
конечного автомата
/// </summary>
public class Blackboard : MonoBehaviour
{
    // int: id ScriptableObject
    public Dictionary<int, Dictionary<string, object>> variables;
    protected virtual void Awake()
    {
```

```
        variables = new Dictionary<int, Dictionary<string, object>>();
    }
 public void SetVariable(int scriptableObjectID, string name, object value)
    {
        if (!variables.ContainsKey(scriptableObjectID))
        {
          variables[scriptableObjectID] = new Dictionary<string, object>();
        }
        variables[scriptableObjectID][name] = value;
    }
    public T GetVariable<T>(int scriptableObjectID, string name)
    {
        if (variables.TryGetValue(scriptableObjectID, out var data) &&
data.TryGetValue(name, out var value))
        {
            return (T)value;
        }
        return default;
    }
}
```

3.2. Диаграмма последовательности

Для демонстрации работы конечного автомата, рассмотрим диаграмму последовательности процесса атаки врага игроком (рисунок 13 в приложении В).

На диаграмме отмечены области состояний конечного автомата игрока в виде фреймов `StandingState` и `AttackingState`.

Процесс начинается, когда игрок нажимает кнопку атаки. Информация о нажатии кнопки записывается классом `PlayerInputHandler`. Затем игровой движок вызывает метод `Update()` у класса `BaseStateMachine`, который обновляет текущее состояние игрока `StandingState` путем вызова метода `OnStateUpdate()`.

Во время обновления текущего состояния, происходит проверка на возможность перехода в состояние `AttackingState` при помощи `AttackTransition`. Для определения возможности атаки, `AttackTransition` обращается к решению `AttackDecision`.

`AttackDecision` получает информацию от класса `PlayerInputHandler` о нажатии кнопки атаки. Затем `AttackDecision` обращается к классу `WeaponManager`, чтобы узнать, имеется ли у игрока оружие. Далее, `AttackDecision` получает информацию о готовности атаки от класса

Combat. После получения всей необходимой информации решение AttackDecision сообщает переходу AttackTransition о том, что все условия для перехода выполнены.

AttackTransition изменяет текущее состояние на AttackingState с помощью метода ChangeState() класса BaseStateMachine вызывает метод OnStateEnter() у AttackingState. Затем AttackingState выполняет действие AttackAction. AttackAction инициирует атаку, вызывая метод Attack() класса Combat.

Класс Combat запускает анимацию атаки через класс AnimationController. В определенный кадр анимации Animation-Controller вызывает метод Contact() у класса Combat. Затем класс Combat, основываясь на полученной информации от класса WeaponManager о характеристиках текущего оружия игрока, наносит урон врагу с помощью метода MakeDamage(). В конце анимации атаки, AnimationController сообщает классу Combat о завершении анимации, и Combat запускает таймер перезарядки атаки.

На следующем кадре игры BaseStateMachine обновляет состояние AttackingState, которое проверяет возможность перехода в состояние StandingState через AttackEndedTransition. Для определения возможности перехода, AttackEndedTransition обращается к решению AttackEndedDecision. AttackEndedDecision, основываясь на информации от класса Combat, сообщает AttackEndedTransition о выполнении всех условий для перехода. В результате, AttackEndedTransition изменяет текущее состояние обратно на StandingState. Таким образом, процесс атаки завершается.

3.3. Реализация компонента ресурсов и характеристик

Для иллюстрации взаимосвязей в системе ресурсов и характеристик была разработана диаграмма классов, представленная на рисунке 19.

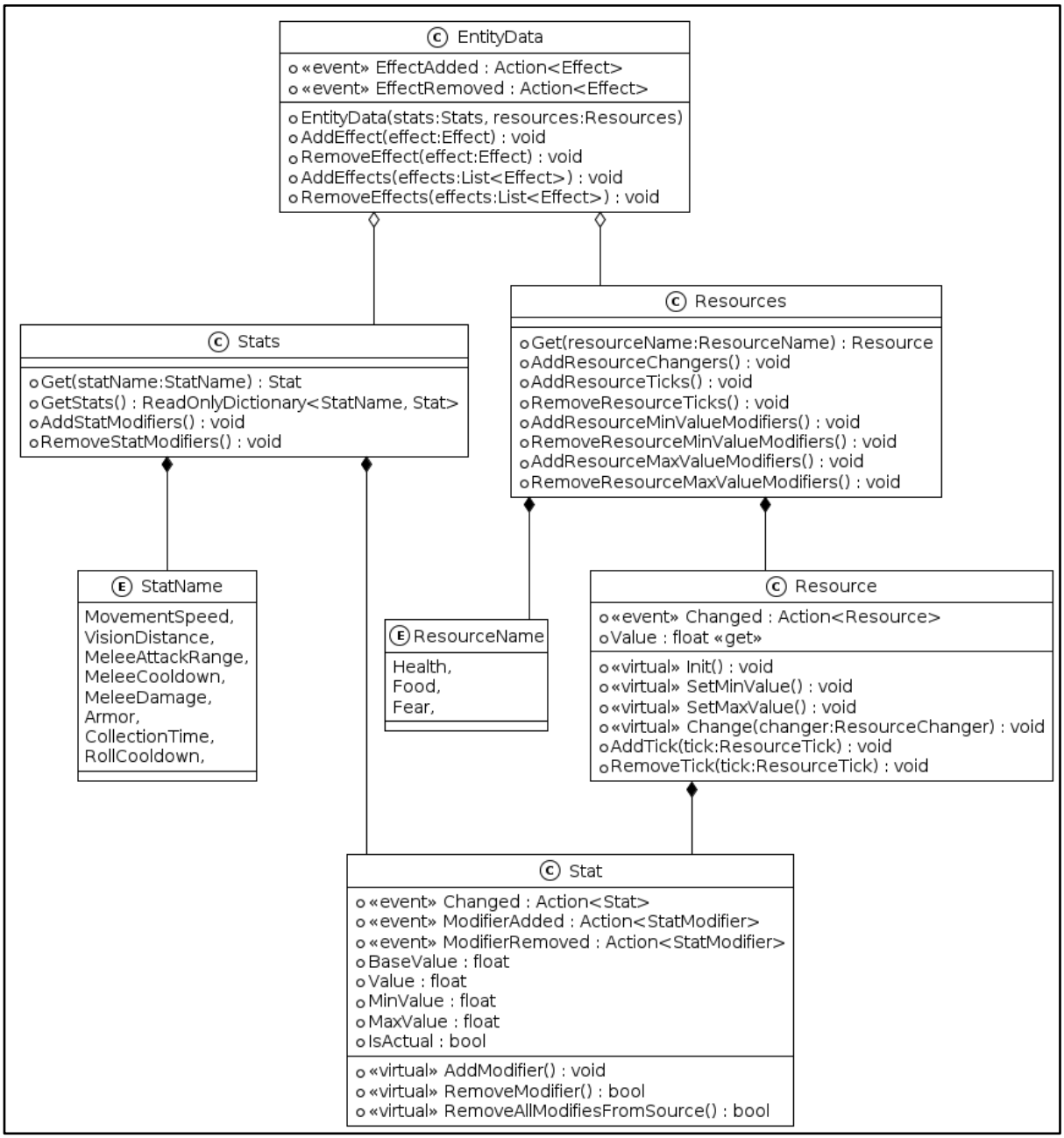

Рисунок 19 – Диаграмма классов компонента ресурсов и характеристик

`Stat` – класс, представляющий собой характеристику игрового объекта. Свойство `BaseValue` представляет базовое значение характеристики до применения модификаторов. Методы `AddModifier()` и `RemoveModifier()` позволяют добавить и удалить модификатор характеристики соответственно. Метод `RemoveAllModifiersFromSource()` предоставляет возможность удалить все модификаторы, связанные с указанным источни-

ком. Метод `CalculateFinalValue()` вычисляет финальное значение характеристики с учетом всех добавленных модификаторов. В свойстве `Value` используется паттерн «Ленивая инициализация» («Lazy initialization») для оптимизации вычислений текущего значения характеристики. Данный паттерн позволяет отложить вычисление значения свойства до момента его фактического использования.

Кеширование финального значения помогает избежать повторных вычислений значения характеристики, если оно уже было определено и остается актуальным. События `Changed`, `ModifierAdded`, `ModifierRemoved` используются для оповещения подписчиков об изменениях в характеристике. Эти решения способствуют оптимизации производительности при работе с большим количеством характеристик в игре. Реализация класса `Stat` приведена в листинге 7.

Листинг 7 – Реализация класса `Stat`

```
namespace StatSystem.Data
{
    public class Stat
        protected float _value;
        public float Value
        {
            get
            {
                if (!IsActual)
                {
                    _value = CalculateFinalValue();
                    IsActual = true;
                    _value = Math.Clamp(_value, MinValue, MaxValue);
                }
                return _value;
            }
        }
[field: SerializeField] public float MinValue { get; protected set; } = 0;
[field: SerializeField] public float MaxValue { get; protected set; } =
100;
        protected bool isActual = false;
        public bool IsActual
        {
            get => isActual;
            set
            {
                isActual = value;
                if (!value)
                {
                    Changed?.Invoke(this);
                }
            }
        }
}
```

`Resource` – класс, представляющий собой ресурс игрового объекта. Он обеспечивает управление значением ресурса в пределах заданных минимальных и максимальных значений. Эти границы определяются объектом класса `Stat`, что позволяет применять модификаторы характеристик к минимальному и максимальному значению ресурса. Методы `Increase()` и `Decrease()` позволяют увеличивать или уменьшать значение ресурса соответственно. Методы `AddTick()` и `RemoveTick()` предназначены для управления добавлением и удалением тиков, которые представляют собой периодические изменения значения ресурса в заданные временные интервалы. Эти изменения обрабатываются асинхронно, что позволяет выполнить их в фоновом режиме, не блокируя основной поток выполнения игрового приложения.

`Stats` – класс, отвечающий за хранение и доступ к характеристикам. Его реализация основана на структуре данных в виде словаря, где ключами являются элементы перечисления `StatName`, а значениями – объекты класса `Stat`, представляющие сами характеристики. Это обеспечивает эффективный доступ к характеристикам по их имени как в инспекторе Unity, так и из программного кода. Для доступа к конкретной характеристике предусмотрен метод `Get()`.

`Resources` – класс, отвечающий за хранение ресурсов и доступ к ним. Его основой является структура данных в виде словаря, где ключами выступают элементы перечисления `ResourceName`, а значениями являются объекты класса `Resource`, представляющие собой сами ресурсы. Это обеспечивает эффективный доступ к ресурсам по их имени как в инспекторе Unity, так и в программном коде. Для получения доступа к конкретному ресурсу служит метод `Get()`.

`EntityData` – класс, представляющий собой хранилище данных о сущности в контексте игрового мира, агрегирующий информацию о ее характеристиках и ресурсах. Методы `AddEffect()` и `RemoveEffect()` позволяют соответственно добавлять и удалять эффекты.

В контексте компонента ресурсов и характеристик эффекты играют важную роль, поэтому для более наглядного представления эффекта приведена отдельная диаграмма классов (рисунок 20).

`StatModifier` – класс, представляющий собой модификатор характеристики. Поле `Type` определяет тип изменения, который может быть абсолютным числом (`Flat`), процентным добавлением (`PercentAdd`) или процентным умножением (`PercentMult`). Поле `Order` определяет порядок применения модификатора, что имеет значение при применении нескольких модификаторов. Поле `Source` представляет объект, который вызвал применение этого модификатора. В качестве такого объекта может выступать способность персонажа, предмет экипировки и прочее. Реализация класса `StatModifier` представлена в листинге 8.

Листинг 8 – Реализация класса `StatModifier`

```
using System;
using UnityEngine;
namespace StatSystem.Data
{
    [Serializable]
    public class StatModifier
    {
[field: SerializeField] public float Value { get; private set; }
[field: SerializeField] public StatModifierType Type { get; private set; }
[field: SerializeField] public int Order { get; private set; }
[field: SerializeField] public object Source { get; private set; }
[field: SerializeField, Min(0)] public int TimeInSeconds { get; private
set; }
public StatModifier(float value, StatModifierType type, int order, object
source, int timeInSeconds = 0)
        {
 if (timeInSeconds < 0) throw new Exception("timeInSeconds must be >=0!");
            Value = value;
            Type = type;
            Order = order;
            Source = source;
            TimeInSeconds = timeInSeconds;
        }
public StatModifier(float value, StatModifierType type, int order) :
this(value, type, order, null) { }
public StatModifier(float value, StatModifierType type, object source) :
this(value, type, (int)type, source) { }
public StatModifier(float value, StatModifierType type) : this(value, type,
(int)type) { }
    }
    public enum StatModifierType
    {
        [InspectorName("Изменить на число")] Flat = 100,
        [InspectorName("Изменить на процент")] PercentAdd = 200,
        [InspectorName("Изменить взяв процент")] PercentMult = 300
    }
```

```
}
```

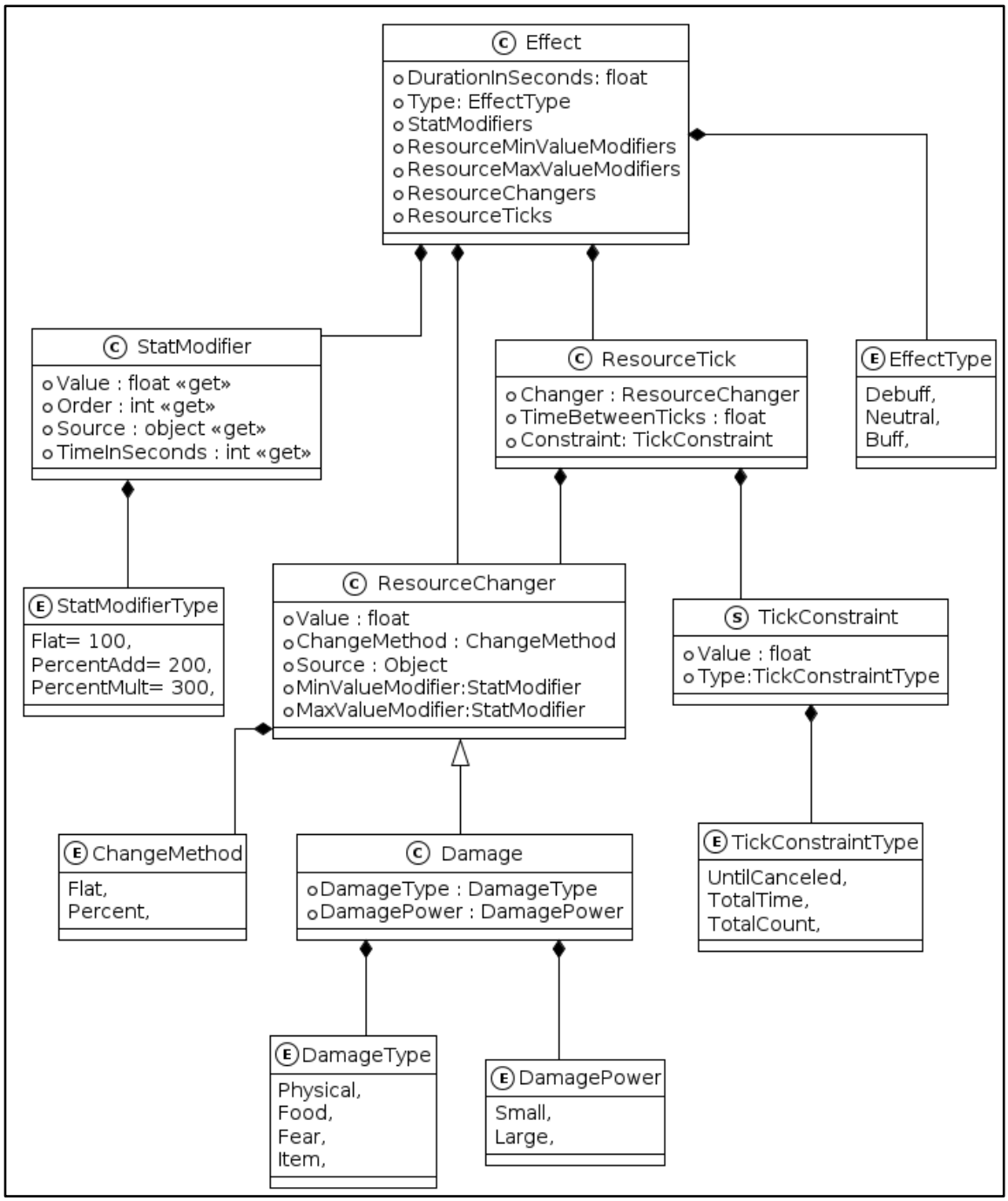

Рисунок 20 – Диаграмма классов эффекта

`ResourceChanger` – класс, представляющий изменение ресурса. В поле `Value` определяется величина изменения, а в поле `ChangeMethod` указывается метод изменения ресурса: он может быть абсолютным при исполь-

зовании метода Flat или выражаться в процентах от максимального значения при выборе метода Percent. Реализация класса ResourceChanger приведена в листинге 9.

Листинг 9 – Реализация класса ResourceChanger

```
using System;
using UnityEngine;
namespace StatSystem.Data
{
    [Serializable]
    public class ResourceChanger
    {
[field: SerializeField] public float Value { get; private set; }
[field: SerializeField] public ChangeMethod ChangeMethod { get; private
set; }
public UnityEngine.Object Source { get; private set; }
public ResourceChanger(float value, ChangeMethod changeMethod,
UnityEngine.Object source, StatModifier minValueModifier = null,
StatModifier maxValueModifier = null)
        {
            Value = value;
            ChangeMethod = changeMethod;
            Source = source;
        }
    }
    public enum ChangeMethod
    {
[InspectorName("Изменить на число")] Flat, // прибавить или убавить
[InspectorName("Изменить на процент от максимального")] Percent // умножить
или разделить
    }
}
```

Damage – класс, представляющий урон в игре и являющийся наследником класса ResourceChanger. Он содержит поля DamageType и DamagePower, определяющие тип и силу урона соответственно. Реализация класса Damage приведена в листинге 10.

Листинг 10 – Реализация класса Damage

```
using System;
using UnityEngine;
namespace StatSystem.Data
{
    [Serializable]
    public class Damage : ResourceChanger
    {
[field: SerializeField] public DamageType DamageType { get; private set; }
[field: SerializeField] public DamagePower DamagePower { get; private set;
}
public Damage(float value, ChangeMethod changeMethod, UnityEngine.Object
source, DamageType damageType, DamagePower damagePower = DamagePower.Small)
            : base(value, changeMethod, source)
        {
            DamageType = damageType;
```

```
            DamagePower = damagePower;
        }
public Damage(ResourceChanger resourceModifier, DamageType damageType,
DamagePower damagePower = DamagePower.Small)
: this(resourceModifier.Value, resourceModifier.ChangeMethod,
resourceModifier.Source, damageType, damagePower) { }
    }
    public enum DamageType
    {
        Physical,
        Food,
        Fear,
        Item
    }
    public enum DamagePower
    {
        Small,
        Large
    }
}
```

`ResourceTick` – это класс, который представляет собой периодическое изменение значения ресурса. Поле `Changer` агрегирует объект класса `ResourceChanger`, который будет применяться к ресурсу каждые `TimeBetweenTicks` секунд. Поле `Constraint` определяет ограничение на продолжительность применения изменений к ресурсу. Это ограничение может быть до отмены (`UntilCanceled`), по истечении определенного времени (`TotalTime`) или после определенного количества применений (`TotalCount`). Реализация класса `ResourceTick` приведена в листинге 11.

Листинг 11 – Реализация класса `ResourceTick`

```
using UnityEngine;
namespace StatSystem.Data
{
    [System.Serializable]
    public class ResourceTick
    {
[field: SerializeField] public ResourceChanger Changer { get; protected
set; }
[field: SerializeField] public float TimeBetweenTicks { get; protected set;
}
[field: SerializeField] public TickConstraint Constraint { get; protected
set; }
    [System.Serializable]
    public struct TickConstraint
    {
        public TickConstraintType Type;
        public float Value;
        public TickConstraint(TickConstraintType type) : this(type, 0) { }
        public TickConstraint(TickConstraintType type, float value)
        {
            Type = type;
            if (Type == TickConstraintType.TotalCount)
```

```
            {
                value = (int)value;
            }
            Value = value;
        }
    }
    }
}
```

`Effect` – класс, который объединяет модификаторы характеристик и ресурсов, что обеспечивает удобное настраивание эффектов в инспекторе Unity для различных игровых элементов. Например, для игрового предмета «Зелье силы» можно создать эффект, который увеличивает силу атаки, максимальное здоровье и скорость перемещения на определенный промежуток времени, при этом снижая ресурс «Страх» на 5 единиц каждые 2 секунды. Также, используя этот класс, можно эффективно управлять эффектами в системе инвентаря, боевой системе, системе оружия и других системах, что повышает скорость и эффективность разработки.

Вывод по третьему разделу

В разделе была описана реализация компонентов игрового приложения с учетом функциональных и нефункциональных требований к разрабатываемой системе. Для демонстрации работы компонента «Конечный автомат» была представлена и детально описана диаграмма последовательности процесса атаки врага игроком. Кроме того, в данном разделе были представлены средства реализации проекта. Скриншоты итогового игрового приложения представлены в приложении Б.

4. ТЕСТИРОВАНИЕ

Для тестирования игрового приложения применялось функциональное тестирование в соответствии с функциональными требованиями, сформулированными во втором разделе.

Функциональное тестирование – основной вид тестирования, проверяющий соответствие системы функциональным требованиям, предъявленным к ней на самом первом этапе проектирования программного продукта [42]. Результаты функционального тестирования представлены в таблице 1.

Таблица 1 – Результаты функционального тестирования

№	Название теста	Действия	Результат	Тест пройден?
1.	Отображение главного меню при запуске.	Запустить игровое приложение.	При запуске игрового приложение отобразилось главное меню.	Да
2.	Предоставление возможности настройки параметров игры на экране «Настройки».	В главном меню выбрать пункт «Настройки». Изменить разрешение экрана.	Разрешение экрана игрового приложения поменялось.	Да
3.	Отображение прогресса при загрузке игровой сцены на загрузочном экране.	В главном меню выбрать пункт «Новая игра». Отказаться от просмотра обучения.	На загрузочном экране отображается прогресс загрузки игровой сцены.	Да
4.	Предлагать игроку просмотреть обучение перед началом новой игры.	В главном меню выбрать пункт «Новая игра».	Перед началом новой игры отобразилось окно с предложением пройти обучение.	Да
5.	Генерация объектов игрового мира случайным образом.	Зайти на игровую сцену.	На игровой сцене сгенерированы монстры и грибы случайным образом.	Да

Продолжение таблицы 1

№	Название теста	Действия	Результат	Тест пройден?
6.	Предоставление игроку возможности управления передвижением игрового персонажа.	Зайти на игровую сцену. Зажать кнопку передвижения игрового персонажа и поводить мышью.	При перемещении мыши с зажатой кнопкой передвижения игровой персонаж передвигается.	Да
7.	Предоставление игроку возможности атаковать врагов в ближнем бою.	Зайти на игровую сцену. Подойти к монстру и посредством нажатия кнопки атаки ликвидировать его.	Монстр ликвидирован посредством нажатия кнопки атаки.	Да
8.	Предоставление игроку возможности вставать в оборонительную стойку для увеличения защитной характеристики игрового персонажа.	Зайти на игровую сцену. Подойти к монстру и посредством нажатия кнопки оборонительной стойки увеличить защитную характеристику игрового персонажа.	Защитная характеристика игрового персонажа увеличена посредством нажатия кнопки оборонительной стойки.	Да
9.	Предоставление игроку возможности выполнить кувырок для уклонения от атак или сокращения дистанции до врагов.	Зайти на игровую сцену. Подойти к монстру и посредством нажатия кнопки кувырка увернуться от его атаки.	Игровой персонаж увернулся от атаки монстра путем нажатия кнопки кувырка.	Да
10.	Предоставление игроку возможности просмотреть характеристики игрового персонажа.	Зайти на игровую сцену. Нажать на кнопку просмотра характеристик в правом нижнем углу.	По нажатию кнопки успешно просмотрены характеристики игрового персонажа.	Да
11.	Обеспечение работы механик сытости и страха.	Зайти на игровую сцену. Убедиться в том, что сытость уменьшается, а страх увеличивается со временем. Убедиться в том, что нахождение рядом с кострами уменьшает страх, а употребление грибов увеличивает сытость.	Сытость уменьшается, а страх увеличивается со временем. Нахождение рядом с кострами уменьшает страх, а употребление грибов увеличивает сытость.	Да
12.	Отображение экрана победы, когда игрок вошел в дверь.	Зайти на игровую сцену. Дойти до двери и нажать кнопку взаимодействия, чтобы войти в нее.	При входе в дверь отображается экран победы.	Да

Окончание таблицы 1

№	Название теста	Действия	Результат	Тест пройден?
13.	Отображение экрана с информацией о причине смерти игрового персонажа после его гибели.	Зайти на игровую сцену. Довести персонажа до смерти от голода.	После гибели игрового персонажа отображается экран с информацией о том, что персонаж умер от голода.	Да
14.	Предоставление возможности игроку оставлять обратную связь.	В главном меню нажать кнопку «Обратная связь». После заполнения формы, нажать кнопку отправить.	После нажатия кнопки «Отправить» ниже формы обратной связи, игроку отображается окно с подтверждением о том, что обратная связь успешно отправлена.	Да
15.	Предоставление возможности игроку сообщать о возникшей ошибке или предложении по улучшению игры.	В главном меню нажать кнопку «Сообщить». После заполнения формы, нажать кнопку отправить.	После нажатия кнопки «Отправить» ниже формы «Сообщить», игроку отображается окно с подтверждением о том, что сообщение успешно отправлено.	Да

Вывод по четвертому разделу

В процессе тестирования игрового приложения были созданы и проведены тесты на соответствие функциональным и нефункциональным требованиям игрового приложения. Все проведенные тесты были пройдены успешно.

ЗАКЛЮЧЕНИЕ

В данной работе была разработана игра «Forgotten World». Основные результаты следующие.

1. Произведен анализ предметной области, в ходе которого были изучены существующие аналоги и методы реализации игр подобного жанра. Это позволило определить основные направления разработки и выбрать наиболее подходящие средства реализации.
2. Спроектировано игровое приложение с учетом всех требований и особенностей, выявленных в процессе анализа. Была разработана архитектура системы и описаны основные компоненты.
3. Реализовано игровое приложение.
4. Проведено функциональное тестирование игрового приложения. Все проведенные тесты были успешно пройдены.

Дальнейшим направлением развития данного проекта будет добавление битвы с боссом, разработка и интеграция системы диалогов, квестов, инвентаря, магазина и сохранений для обогащения игрового опыта игрока и расширения игрового мира.

Важным аспектом дальнейшего развития будет поддержка и развитие сообщества игроков. Введение системы обратной связи и возможность оставлять отзывы и предложения позволит оперативно реагировать на потребности пользователей и вносить необходимые улучшения в приложение. Планируется регулярное добавление игрового контента, что позволит поддерживать интерес к игре на длительный период времени.

Разработка подобных элементов не только увеличит интерес игроков, но и позволит создать уникальный продукт, способный занять достойное место на рынке компьютерных игр.

ЛИТЕРАТУРА

1. The Gaming Industry Is Now Bigger Than Movies And Music Combined. [Электронный ресурс] URL: https://www.thc-pod.com/episode/the-gaming-industry-is-now-bigger-than-movies-and-music-combined (дата обращения: 05.05.2024 г.).

2. Highest-grossing games vs highest-grossing movies. [Электронный ресурс] URL: https://www.theocelot.co.uk/highest-grossing-games-vs-highest-grossing-movies (дата обращения: 05.05.2024 г.).

3. How much has Avatar 2 made so far? [Электронный ресурс] URL: https://www.digitalspy.com/movies/a42293101/avatar-2-money-box-office-so-far/ (дата обращения: 05.05.2024 г.).

4. «Avatar 2» is already the 10th highest-grossing movie ever made. [Электронный ресурс] URL: https://www.cnbc.com/2023/01/06/avatar-2-is-already-the-10th-highest-grossing-movie-ever-made-.html (дата обращения: 05.05.2024 г.).

5. This violent videogame has made more money than any movie ever. [Электронный ресурс] URL: https://www.marketwatch.com/story/this-violent-videogame-has-made-more-money-than-any-movie-ever-2018-04-06 (дата обращения: 05.05.2024 г.).

6. Adventure video games – Codex Gamicus – Humanity’s collective gaming knowledge at your fingertips. [Электронный ресурс] URL: https://gamicus.fandom.com/wiki/Adventure_video_games (дата обращения: 05.05.2024 г.).

7. Action video games – Codex Gamicus – Humanity’s collective gaming knowledge at your fingertips. [Электронный ресурс] URL: https://gamicus.fandom.com/wiki/Action_video_games (дата обращения: 05.05.2024 г.).

8. Half-Life 2 for PC Reviews – Metacritic. [Электронный ресурс] URL: https://www.metacritic.com/game/pc/half-life-2 (дата обращения: 05.05.2024 г.).

9. Batman Arkham City – Reviews, Articles, People, Trailers and more at Metacritic - Metacritic. [Электронный ресурс] URL: https://www.metacritic.com/search/all/Batman%20Arkham%20City/results (дата обращения: 05.05.2024 г.).

10. Max Payne for PC Reviews – Metacritic. [Электронный ресурс] URL: https://www.metacritic.com/game/pc/max-payne (дата обращения: 05.05.2024 г.).

11. Red Dead Redemption for PlayStation 3 Reviews – Metacritic. [Электронный ресурс] URL: https://www.metacritic.com/game/playstation-3/red-dead-redemption (дата обращения: 05.05.2024 г.).

12. The Elder Scrolls V: Skyrim for PC Reviews – Metacritic. [Электронный ресурс] URL: https://www.metacritic.com/game/pc/the-elder-scrolls-v-skyrim (дата обращения: 05.05.2024 г.).

13. Лучшие игры Экшен – топ игр всех времен в жанре Экшен, список популярных. [Электронный ресурс] URL: https://www.igromania.ru/games/all/action/ (дата обращения: 05.05.2024 г.).

14. Portal for PC Reviews – Metacritic. [Электронный ресурс] URL: https://www.metacritic.com/game/pc/portal (дата обращения: 05.05.2024 г.).

15. Action-adventure game | Ultimate Pop Culture Wiki | Fandom. [Электронный ресурс] URL: https://ultimatepopculture.fandom.com/wiki/Action-adventure_game (дата обращения: 05.05.2024 г.).

16. Survival game | Ultimate Pop Culture Wiki | Fandom. [Электронный ресурс] URL: https://ultimatepopculture.fandom.com/wiki/Survival_game (дата обращения: 05.05.2024 г.).

17. Minecraft for PC Reviews – Metacritic. [Электронный ресурс] URL: https://www.metacritic.com/game/pc/minecraft (дата обращения: 05.05.2024 г.).

18. DayZ for PlayStation 4 Reviews – Metacritic. [Электронный ресурс] URL: https://www.metacritic.com/game/playstation-4/dayz (дата обращения: 05.05.2024 г.).

19. Grand Theft Auto V for PC Reviews – Metacritic. [Электронный ресурс] URL: https://www.metacritic.com/game/pc/grand-theft-auto-v (дата обращения: 05.05.2024 г.).

20. Red Dead Redemption 2 for PlayStation 4 Reviews – Metacritic. [Электронный ресурс] URL: https://www.metacritic.com/game/playstation-4/red-dead-redemption-2 (дата обращения: 05.05.2024 г.).

21. God of War for PlayStation 4 Reviews – Metacritic. [Электронный ресурс] URL: https://www.metacritic.com/game/playstation-4/god-of-war (дата обращения: 05.05.2024 г.).

22. Tomb Raider for PC Reviews – Metacritic. [Электронный ресурс] URL: https://www.metacritic.com/game/pc/tomb-raider (дата обращения: 05.05.2024 г.).

23. Assassin's Creed Valhalla for PlayStation 5 Reviews – Metacritic. [Электронный ресурс] URL: https://www.metacritic.com/game/playstation-5/assassins-creed-valhalla (дата обращения: 05.05.2024 г.).

24. Subnautica for Xbox One Reviews – Metacritic. [Электронный ресурс] URL: https://www.metacritic.com/game/xbox-one/subnautica (дата обращения: 05.05.2024 г.).

25. Best PC Video Games of All Time – Metacritic. [Электронный ресурс] URL: https://www.metacritic.com/browse/games/score/metascore/all/pc/filtered (дата обращения: 05.05.2024 г.).

26. Don't Starve. // Metacritic [Электронный ресурс] URL: https://www.metacritic.com/game/pc/dont-starve (дата обращения: 05.05.2024 г.).

27. Don't Starve (Video Game 2013) – Awards – IMDb. [Электронный ресурс] URL: https://www.imdb.com/title/tt3052500/awards/#ev0035875 (дата обращения: 05.05.2024 г.).

28. The Forest (2014). // Metacritic [Электронный ресурс] URL: https://www.metacritic.com/game/pc/the-forest-2014 (дата обращения: 05.05.2024 г.).

29. The Forest Global Game Awards Nominee 2014. [Электронный ресурс] URL: https://www.game-debate.com/awards/2014/nominee/the-forest (дата обращения: 05.05.2024 г.).

30. Left 4 Dead. // Metacritic [Электронный ресурс] URL: https://www.metacritic.com/game/pc/left-4-dead (дата обращения: 22.05.2023 г.).

31. Left 4 Dead Review – IGN. [Электронный ресурс] URL: https://www.ign.com/articles/2008/11/17/left-4-dead-review (дата обращения: 05.05.2024 г.).

32. Unity, платформа разработки в реальном времени | Платформа для 3D-, 2D-, VR- и AR-контента. // Unity [Электронный ресурс] URL: https://unity.com (дата обращения: 05.05.2024 г.).

33. Unreal Engine | The most powerful real-time 3D creation tool. // Unreal Engine [Электронный ресурс] URL: https://www.unrealengine.com/en-US (дата обращения: 05.05.2024 г.).

34. Technologies U. Unity – Manual: Unity User Manual 2021.3 (LTS). [Электронный ресурс] URL: https://docs.unity3d.com/Manual/index.html (дата обращения: 05.05.2024 г.).

35. Hollow Knight. [Электронный ресурс] URL: https://www.hollowknight.com/ (дата обращения: 05.05.2024 г.).

36. Homepage – Subnautica. [Электронный ресурс] URL: https://unknownworlds.com/subnautica/ (дата обращения: 05.05.2024 г.).

37. Rust – Explore, Build and Survive. [Электронный ресурс] URL: https://rust.facepunch.com/ (дата обращения: 05.05.2024 г.).

38. Ori – The Will of the Wisps. [Электронный ресурс] URL: https://www.orithegame.com/ (дата обращения: 05.05.2024 г.).

39. Campo Santo – Firewatch. [Электронный ресурс] URL: https://www.firewatchgame.com/ (дата обращения: 05.05.2024 г.).

40. Документация по C#. Начало работы, руководства, справочные материалы. | Microsoft Learn. [Электронный ресурс] URL: https://learn.microsoft.com/ru-ru/dotnet/csharp/ (дата обращения: 05.05.2024 г.).

41. Microsoft C/C++ Documentation | Microsoft Learn. [Электронный ресурс] URL: https://learn.microsoft.com/en-us/cpp/?view=msvc-170 (дата обращения: 05.05.2024 г.).

42. Bourque P., Fairley R. SWEBOK. // IEEE Computer society-e изд., 2014. – 96 с.

43. Booch G., Rumbaugh J., Jacobson I. The Unified Modeling Language User Guide. // Addison-Wesley, 2005. – 504 с.

44. Nystrom R. Game Programming Patterns. // Genever Benning, 2014. – 353 с.

45. Гома Х. UML. Проектирование систем реального времени, распределенных и параллельных приложений. // Litres, 2022. – 702 с.

46. Unity – Manual: ScriptableObject. [Электронный ресурс] URL: https://docs.unity3d.com/Manual/class-ScriptableObject.html (дата обращения: 05.05.2024 г.).

47. Gamil M. 106 System Design Patterns for Interview Preparation. // Mostafa Gamil, 2023. – 222 с.

48. Pavel Kerbabyk. // artstation [Электронный ресурс] URL: https://kerbabyk.artstation.com/ (дата обращения: 05.05.2024 г.).

49. Unity – Scripting API: MonoBehaviour. [Электронный ресурс] URL: https://docs.unity3d.com/ScriptReference/MonoBehaviour.html (дата обращения: 05.05.2024 г.).

ПРИЛОЖЕНИЯ

Приложение А. Спецификации вариантов использования

Спецификации вариантов использования (ВИ) системы приведены в таблицах 1–4.

Таблица 1 – Спецификация ВИ «Просмотреть обучение»

Прецедент: Просмотреть обучение
ID: 1
Краткое описание: Просмотр обучения перед началом новой игры.
Главные актеры: Игрок
Второстепенные актеры: Нет
Предусловия: игрок нажал кнопку «Новая игра»
Основной поток: 1. Прецедент начинается, когда игрок нажимает кнопку «Новая игра». 2. В диалоговом окне «Хотите просмотреть обучение?» игрок нажимает кнопку «Да» 3. Игрок переключает слайды обучения с помощью нажатия на кнопку «Стрелка вперед». 3. На последнем слайде игрок нажимает кнопку «Продолжить» и попадает в игровой мир.
Постусловия: Игрок просмотрел обучение.
Альтернативные потоки: I. Отказ от просмотра обучения. 1. Игрок попадает в игровой мир без просмотра обучения.

Таблица 2 – Спецификация ВИ «Изменить настройки»

Прецедент: Изменить настройки
ID: 2
Краткое описание: Изменение настроек игры.
Главные актеры: Игрок
Второстепенные актеры: Нет
Предусловия: запущено игровое приложение
Основной поток: 1. Прецедент начинается, когда игрок нажимает кнопку «Настройки». 2. Система отображает окно с настройками игры. 3. Игрок изменяет настройки игры по своему усмотрению. 4. После завершения изменений, игрок нажимает кнопку «Применить». 5. Игровое приложение применяет измененные настройки.
Постусловия: Игрок успешно изменил настройки в соответствии с выбранными параметрами.
Альтернативные потоки: отсутствуют

Таблица 3 – Спецификация ВИ «Просмотреть характеристики персонажа»

Прецедент: Просмотреть характеристики персонажа
ID: 3
Краткое описание: Просмотр характеристик игрового персонажа.
Главные актеры: Игрок
Второстепенные актеры: Нет
Предусловия: запущен игровой уровень
Основной поток: 1. Прецедент начинается, когда игрок нажимает кнопку просмотра характеристик. 2. Система отображает окно с характеристиками персонажа. 3. Игрок просматривает характеристики персонажа.
Постусловия: Игрок успешно просмотрел характеристики персонажа
Альтернативные потоки: отсутствуют

Таблица 4 – Спецификация ВИ «Играть»

Прецедент: Играть
ID: 4
Краткое описание: Основное действие игрока.
Главные актеры: Игрок
Второстепенные актеры: Нет
Предусловия: загрузился игровой уровень
Основной поток: 1.Прецедент начинается, когда завершается прецедент 1. 2. Игровое приложение загружает главную сцену игры. 3. Игрок нажимает кнопку «Продолжить» на загрузочном экране и попадает на игровой уровень.
Постусловия: Игрок успешно попал на игровой уровень и может играть.
Альтернативные потоки: отсутствуют

Приложение Б. Скриншоты итоговой версии игры

На рисунках 1–12 представлены скриншоты итоговой версии игрового приложения.

Рисунок 1 – Главное меню

Рисунок 2 – Меню настроек

Рисунок 3 – Диалог «Хотите увидеть обучение?»

Рисунок 4 – Слайд экрана обучения

Рисунок 5 – Экран загрузки

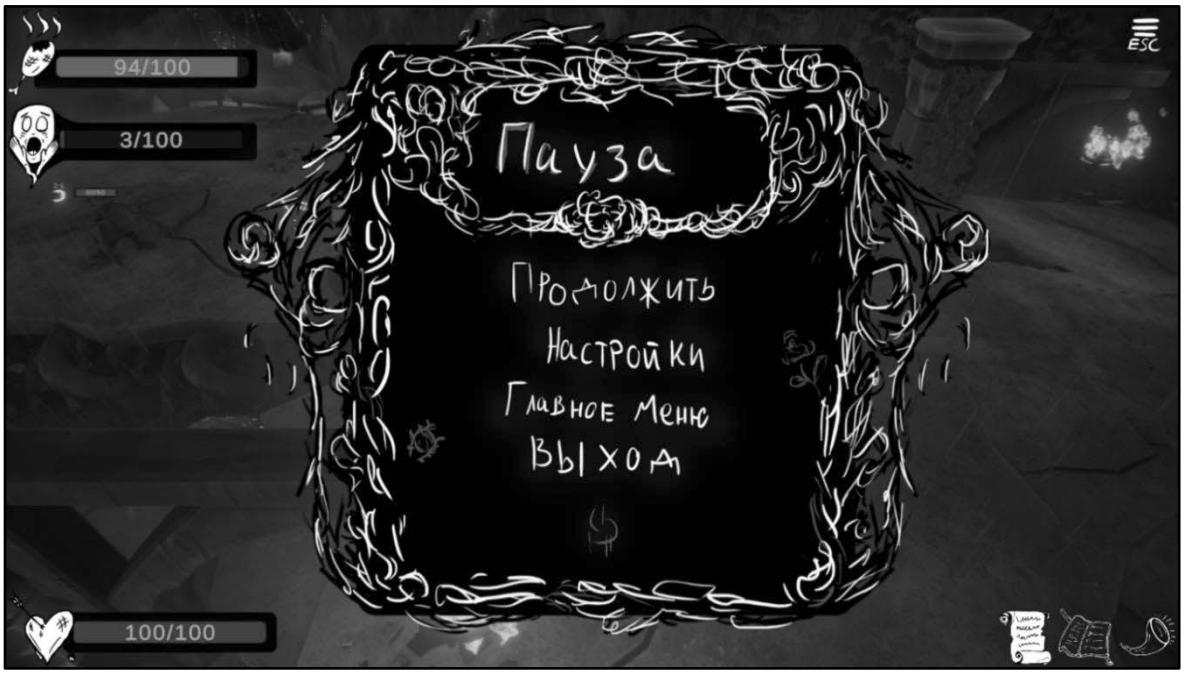

Рисунок 6 – Меню настроек

Рисунок 7 – Всплывающее окно «Характеристики»

Рисунок 8 – Бой с монстром

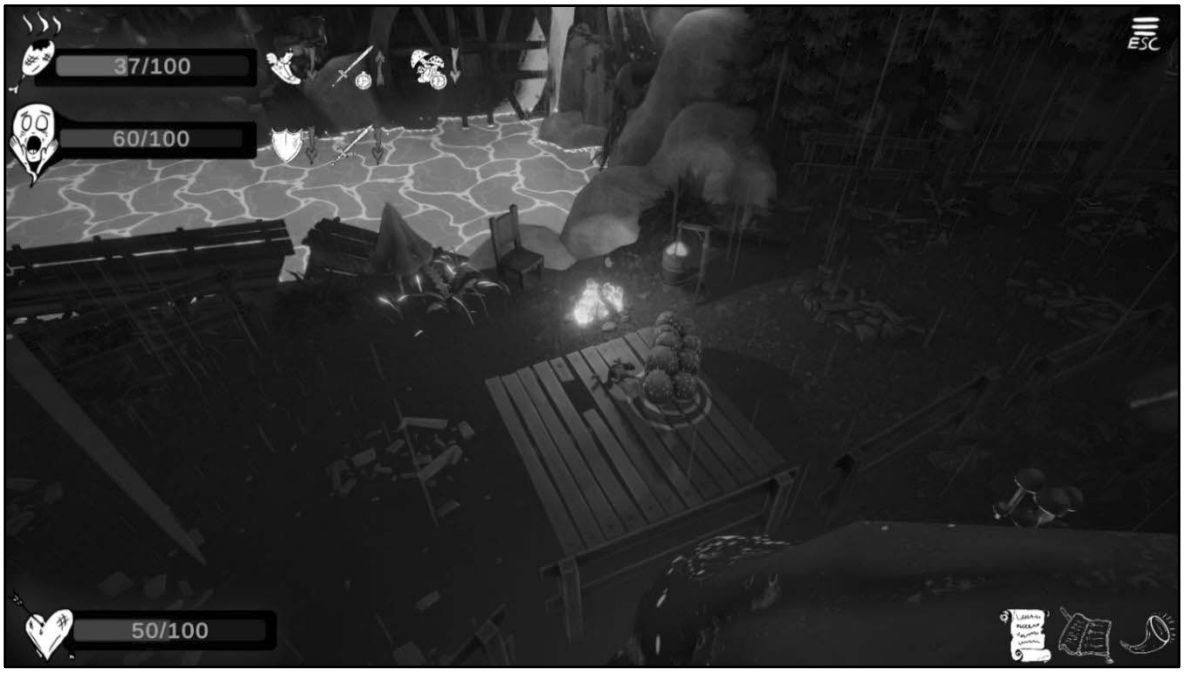

Рисунок 9 – Восстановление сытости и страха путем поедания гриба у костра

Рисунок 10 – Экран смерти от голода

Рисунок 11 – Случайно сгенерированная дверь

Рисунок 12 – Экран победы

Приложение В. Диаграмма последовательности процесса атаки

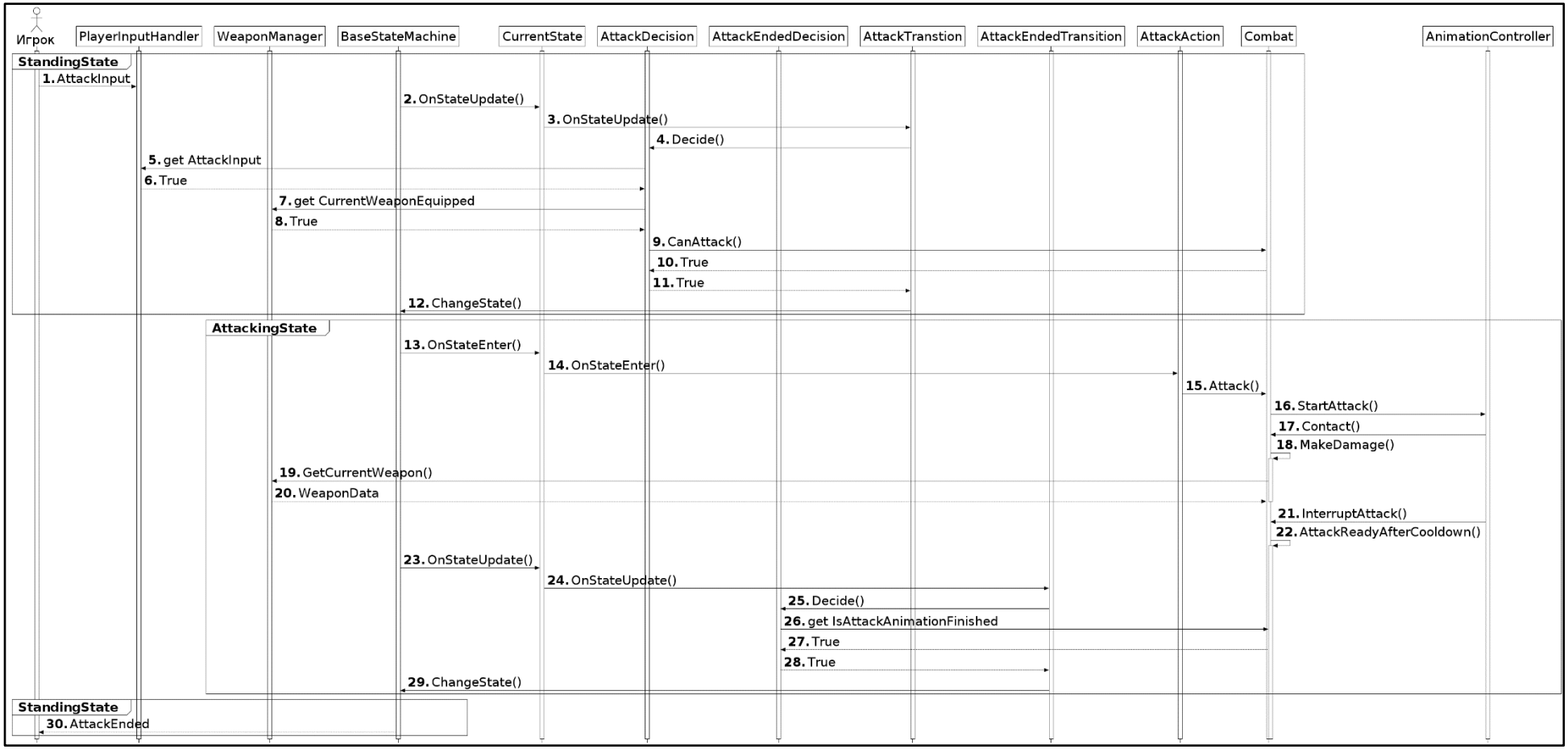

Рисунок 13 – Диаграмма последовательности процесса атаки