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 «Полет амфибии» на платформе Unity

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 02.03.02.2024.308-547.ВКР

Научный руководитель,
доцент кафедры СП, к.ф.-м.н.
_____ И.И. Клебанов

Автор работы,
студент группы КЭ-402
_____ М.В. Лежников

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-402
Лежникову Михаилу Валерьевичу,
обучающемуся по направлению
02.03.02 «Фундаментальная информатика и информационные технологии»

1. **Тема работы** (утверждена приказом ректора от 22.04.2024 г. № 764-13/12)
Разработка компьютерной игры «Полет амфибии» на платформе Unity.
2. **Срок сдачи студентом законченной работы:** 03.06.2024 г.
3. **Исходные данные к работе**
3.1. Unity. Developer. [Электронный ресурс] URL: https://unity.com/ru (дата обращения: 24.02.2024 г.).
3.2. Unity. Game Dev Courses. [Электронный ресурс] URL: https://unity.com/learn/swordsandshovels (дата обращения: 24.02.2024 г.).
4. **Перечень подлежащих разработке вопросов**
4.1. Провести анализ предметной области и обзор аналогов.
4.2. Спроектировать приложение.
4.3. Реализовать и протестировать приложение.
5. **Дата выдачи задания:** 29.01.2024 г.

Научный руководитель,
доцент кафедры СП, к.ф.-м.н. И.И. Клебанов

Задание принял к исполнению М.В. Лежников

ГЛОССАРИЙ

1. *Инди-разработчик* – специалист, который занимается самостоятельной разработкой игр или входит в состав небольшой команды разработчиков, выпускающей компьютерные и мобильные игры без финансовой поддержки крупных компаний [1].

2. *Платформер* – жанр компьютерных игр, в которых основу игрового процесса составляют прыжки по платформам, сбор предметов, необходимых для победы над врагами или завершения уровня [2].

3. *Спрайт* – это графический объект в компьютерной графике, который представляет собой небольшое изображение или анимацию, используемую для отображения персонажей, объектов, эффектов и других элементов в видеоиграх или других интерактивных приложениях. [3].

4. *Скрипт* – это небольшая программа, которая содержит последовательность действий, созданных для автоматического выполнения определенной задачи. Скрипты обычно используются для автоматизации повторяющихся операций или для добавления дополнительной функциональности к программам [4].

5. *Геймплей* – игровой процесс с точки зрения игрока. Геймплей включает в себя разные содержательные аспекты компьютерной игры, в том числе технические, такие как внутриигровая механика, совокупность определенных методов взаимодействия игры с игроком и др. [5].

6. *Скриншот* – это картинка, на которой изображено все то, что находилось на экране компьютера в момент нажатия клавиши PrintScreen [6].

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность

В современном мире игры стали неотъемлемой частью культуры и развлечений, предоставляя игрокам возможность погрузиться в уникальные миры, испытать захватывающие приключения и прочувствовать разнообразные эмоции. Одним из наиболее популярных жанров игр являются платформеры, где основной акцент сделан на управлении персонажем, преодолении препятствий и решении головоломок. Платформеры остаются актуальными благодаря своей доступности и разнообразию игровых механик, что делает их привлекательными для широкой аудитории. Разработка игры в этом жанре позволяет реализовать креативные идеи и создать увлекательный игровой процесс, что подчеркивает актуальность данной работы.

Постановка задачи

Целью выпускной квалификационной работы является разработка компьютерной игры «Полет амфибии» на платформе Unity. Для достижения поставленной цели необходимо решить следующие задачи:

1) провести анализ предметной области и обзор аналогов;
2) спроектировать приложение;
3) реализовать и протестировать приложение.

Структура и содержание работы

Работа состоит из глоссария, введения, четырех глав, заключения и списка литературы. Объем работы составляет 34 страницы, объем списка литературы – 15 источников.

В глоссарии представлены определения ключевых терминов, связанных с разработкой игр на платформе Unity. Глоссарий помогает понимать специфическую терминологию и концепции, используемые в процессе создания игры.

В первой главе описывается жанр игры, особенности, свойственные этому жанру, а также рассмотрены схожие проекты, определены их сильные

и слабые стороны. Этот анализ позволяет выявить ключевые элементы, которые необходимо учесть при разработке собственной игры.

Вторая глава посвящена проектированию системы. В ней приведен эскизный проект для более четкого понимания концепции игры, определены варианты использования системы и составлены макеты интерфейса. Особое внимание уделено описанию архитектуры приложения и выбору технологий, которые обеспечат эффективную реализацию проекта.

В третьей главе описана реализация различных аспектов игры, средств разработки, их плюсов и минусов. Здесь подробно рассматриваются методы и техники, используемые для создания игровых механик, а также даются примеры кода и пояснения к ним.

В четвертой главе описано тестирование приложения. Приводятся результаты функционального тестирования, анализируются ошибки и даются рекомендации по их исправлению. Также рассматриваются аспекты производительности и стабильности игры.

Таким образом, данная работа представляет собой комплексное исследование и практическое применение навыков разработки игр на платформе Unity. В процессе выполнения работы были решены ключевые задачи, начиная от анализа предметной области и заканчивая тестированием готового продукта. Работа включает в себя теоретические и практические аспекты, что делает ее полезной как для разработчиков, так и для исследователей в области игровых технологий.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. Описание предметной области

Предметная область проекта – платформер, является частью игровой индустрии, которая стремительно развивается и пользуется большой популярностью у игроков всех возрастов. Платформеры относятся к жанру платформенных игр, в которых игрок управляет персонажем, перемещая его по различным платформам, преодолевая препятствия, собирая предметы и борясь с врагами.

Основными характеристиками платформеров являются:

1) способность персонажа перемещаться во всех направлениях: персонаж может двигаться вправо, влево, вверх и вниз, что позволяет ему свободно исследовать игровой мир и взаимодействовать с различными элементами окружения;

2) присутствие платформ и препятствий: игровой мир состоит из различных платформ и препятствий, которые персонаж должен преодолевать, используя свои навыки и способности;

3) наличие врагов: враги представляют собой угрозу для персонажа, и игроку необходимо побеждать их, чтобы продвигаться дальше по уровню;

4) сбор предметов: в процессе игры персонаж может собирать различные предметы, которые улучшают его характеристики или предоставляют дополнительные возможности, такие как увеличение скорости, дополнительные жизни или новые способности.

Платформеры на Unity могут иметь различные сюжетные линии, уровни сложности, стили графики и звуковое оформление. Они могут быть как одиночными, так и многопользовательскими, что расширяет возможности игры.

Разработка платформера на Unity требует использования различных инструментов и технологий, таких как Unity Engine, языки программирования, физические системы, анимации, спрайты, звуковые эффекты и другие [7]. Создание игры также может потребовать использования сторонних

ресурсов и платформ, таких как Unity Asset Store, которые предоставляют готовые ассеты и инструменты для разработки игры. Преимущества платформеров представлены ниже.

1. Простота управления: интуитивно понятное управление позволяет игрокам быстро освоиться и начать играть.
2. Разнообразие уровней: возможность создавать множество уникальных уровней с различными задачами и препятствиями.
3. Высокая реиграбельность: игроки могут возвращаться к игре, чтобы улучшить свои результаты или пройти уровни разными способами.
4. Креативные возможности: разработчики могут использовать свою креативность для создания уникальных миров, персонажей и механик.

В целом, предметная область проекта является интересной и разнообразной частью игровой индустрии, которая привлекает множество игроков и разработчиков со всего мира. Разработка платформера на Unity представляет собой увлекательный процесс, требующий знаний и навыков в области программирования, дизайна и анимации, а также творческого подхода к созданию уникального игрового опыта [8].

1.2. Сравнительный анализ аналогов

На рынке существует множество платформеров, как независимых, так и от больших разработчиков.

Super Mario Bros [9]

Super Mario Bros – это легендарная игра, разработанная и выпущенная компанией Nintendo [10] в 1985 году. В этой игре игроки управляют персонажем по имени Марио, который отправляется спасать принцессу Пич из замка, находясь на пути полного разнообразных препятствий и врагов. Пример игрового процесса из игры представлен на рисунке 1.

Рисунок 1 – Игровой процесс «Super Mario Bros»

Игра имеет следующие преимущества.

1. Инновации: Super Mario Bros внесла множество инноваций в игровую индустрию, включая прокачку персонажа, изменение размера, скрытые уровни и т.д.

2. Дизайн уровней: уровни игры характеризуются умелым дизайном, балансом сложности и разнообразием задач.

3. Простота управления: игра предлагает простое управление, которое легко понять и освоить, что делает ее доступной для всех.

4. Звуковое сопровождение: иконичная музыка и звуковые эффекты «Super Mario Bros» стали знаковыми для игровой индустрии.

Игра имеет следующие недостатки.

1. Сложность: для некоторых игроков Super Mario Bros может оказаться слишком сложной из ‑за требующих точности прыжков и реакции задач.

2. Нет сохранения: отсутствие системы сохранения прогресса означает, что игрокам приходится начинать с самого начала после поражения.

3. Super Mario Bros остается культовой игрой, оказавшей значительное влияние на развитие видеоигр и заслуживающей признание как классика жанра платформеров.

Super Meat Boy [11, 12]

Super Meat Boy – это инди-платформер, разработанный командой Team Meat и выпущенный в 2010 году. В игре игроки управляют мясным персонажем по имени Мит Бой, который спасает свою возлюбленную, Бинди, от злобного доктора Фетуса через серию захватывающих и часто экстремально сложных уровней. Пример из игры представлен на рисунке 2.

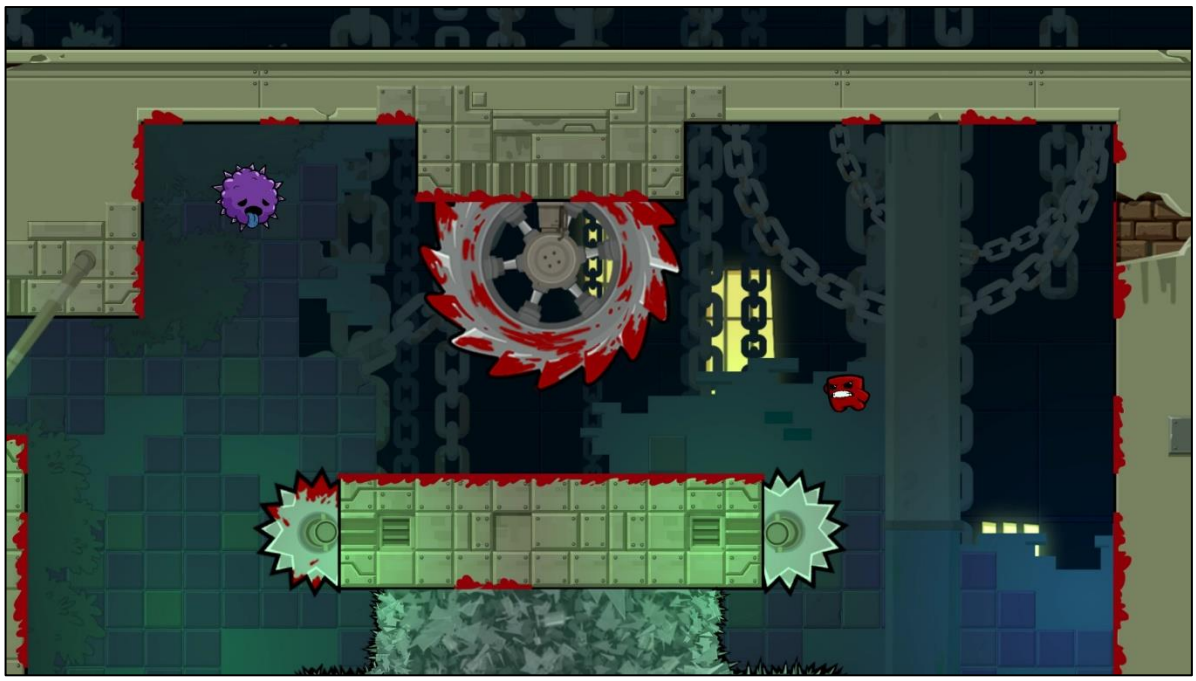

Рисунок 2 – Игровой процесс «Super Meat Boy»

Игра имеет следующие преимущества.

1. Вызов и сложность: Super Meat Boy славится своей высокой сложностью и требовательностью. Она предлагает игрокам настоящий вызов, поощряя мастерство и улучшение навыков.

2. Быстрые перезапуски: быстрые и мгновенные перезапуски уровней позволяют игрокам моментально попробовать снова, не теряя времени на загрузку или ожидание.

3. Аутентичность: игра представляет собой яркий пример инди-игры с уникальным стилем, звуками и анимацией.

4. Коллекционные элементы: Super Meat Boy содержит множество скрытых уровней и персонажей, стимулируя исследование.

Игра имеет следующие недостатки.

1. Высокая сложность: для некоторых игроков игра может оказаться слишком сложной и вызывающей чувство фрустрации.

2. Минималистичный сюжет: игра сконцентрирована на геймплее, поэтому сюжетные элементы ограничены.

3. Super Meat Boy – это игра для тех, кто ищет сложный вызов в жанре платформеров. Ее аутентичный стиль и яркая сложность сделали ее любимцем среди поклонников жанра.

Celeste [13]

Celeste – это инди-платформер, разработанный командой Maddy Makes Games [14] и выпущенный в 2018 году. В этой игре игроки берут на себя роль героини по имени Мадлен, которая путешествует по горной вершине Celeste Mountain, преодолевая сложные платформенные испытания и внутренние барьеры. Пример из игры представлен на рисунке 3.

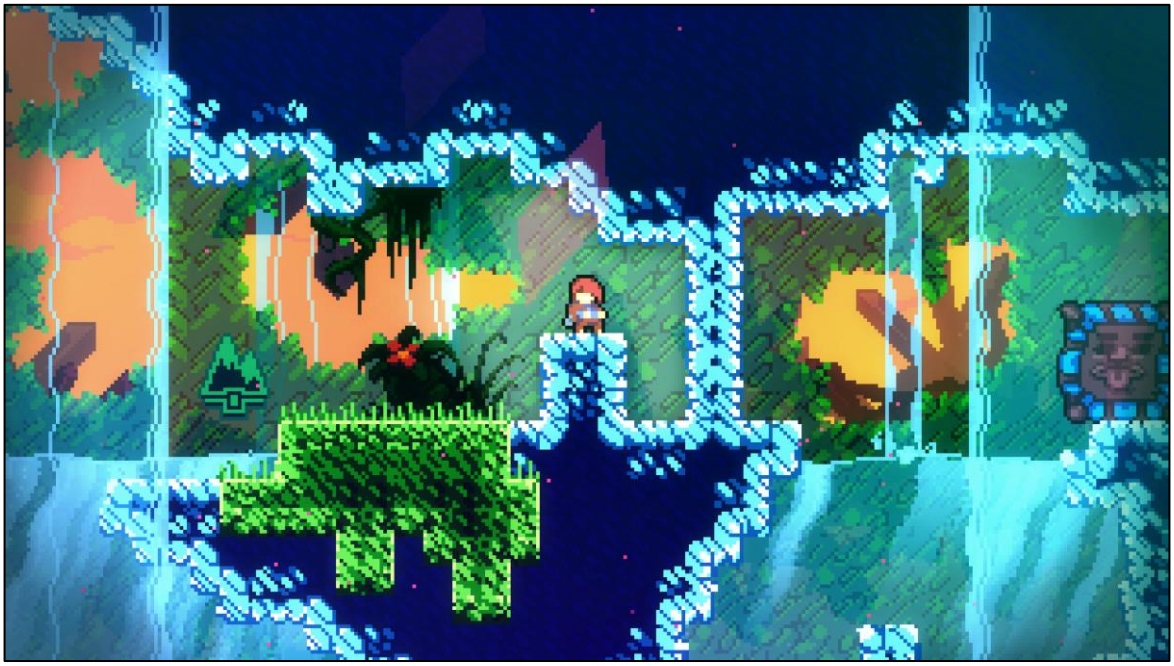

Рисунок 3 – Игровой процесс «Celeste»

Игра имеет следующие преимущества.

1. Сильный сюжет: Celeste обладает глубоким сюжетом, касающимся тематики депрессии и самопознания, что делает игру более эмоциональной и значимой.

2. Дизайн уровней: уровни игры являются прекрасно спроектированными, предлагая игрокам разнообразные головоломки и вызовы.

3. Точное управление: игра предлагает точное и отзывчивое управление, что особенно важно для точных прыжков и движений.

4. Стиль и атмосфера: Celeste обладает великолепной пиксельной графикой и уникальной атмосферой.

Игра имеет следующие недостатки.

1. Сложность: для некоторых игроков сложность уровней может оказаться слишком высокой, что может вызвать фрустрацию.

2. Celeste – это не только отличный платформер с прекрасным дизайном уровней, но и глубокая и эмоциональная история, которая делает ее особенной среди игр жанра.

Анализируя аналоги игровых приложений, можно сделать выводы: приложение должно иметь простое, но интуитивно понятно управление для привлечения широкой аудитории, предоставлять игрокам сложность в прохождении учитывая баланс, и сделать уникальную механику в игровом приложении, чтобы она выделялась среди других игр.

2. ПРОЕКТИРОВАНИЕ

2.1. Концепция игры

Amphibian Ascension – это адреналиновая игра-платформер, вдохновленная элементами Super Meat Boy, но с уникальными механиками и сложными уровнями. Игроки воплощаются в роли храброго персонажа, который совершает дерзкие прыжки через непроходимые препятствия и стены, чтобы достичь цели.

Основные механики представлены ниже.

1. Здоровье: игроки имеют ограниченное количество здоровья, которое теряется при столкновении с опасностями. Это добавляет элемент риска и стратегии к прыжкам и прохождению уровней.
2. Двойной прыжок: персонаж может совершать двойные прыжки, что позволяет ему покрывать большие расстояния и обходить опасные зоны.
3. Отпрыгивание от стен: стены на уровне будут служить как платформы, от которых персонаж может оттолкнуться, чтобы достичь недоступных мест и высоких платформ.

Геймплей

Игра ориентирована на сложные уровни с креативными и оригинальными платформенными задачами. Главная уникальная механика – это механика прыжка, которая зависит от того, как долго игрок удерживает кнопку. Это добавляет глубину управления и позволяет разнообразить способы преодоления препятствий.

Цель игры

Основная цель игры – преодолеть сложные уровни, сохраняя минимальное здоровье. Игрокам предстоит точно измерять время прыжков, правильно выбирать момент для использования двойного прыжка и аккуратно отпрыгивать от стен, чтобы минимизировать риски.

Атмосфера

Визуальный стиль будет оформлен в ярких цветах и заметно различающихся уровнях, создавая визуальную разнообразность. Музыка и звуковое сопровождение будут дополнять напряженность игрового процесса.

2.2. Макеты пользовательского интерфейса

Были созданы макеты пользовательского интерфейса. Они представляют собой примерное представление главного меню и внутриигрового интерфейса. При запуске игры игрок видит главное меню, содержащее название игры и 2 кнопки: «Играть» и «Выход» (рисунок 4).

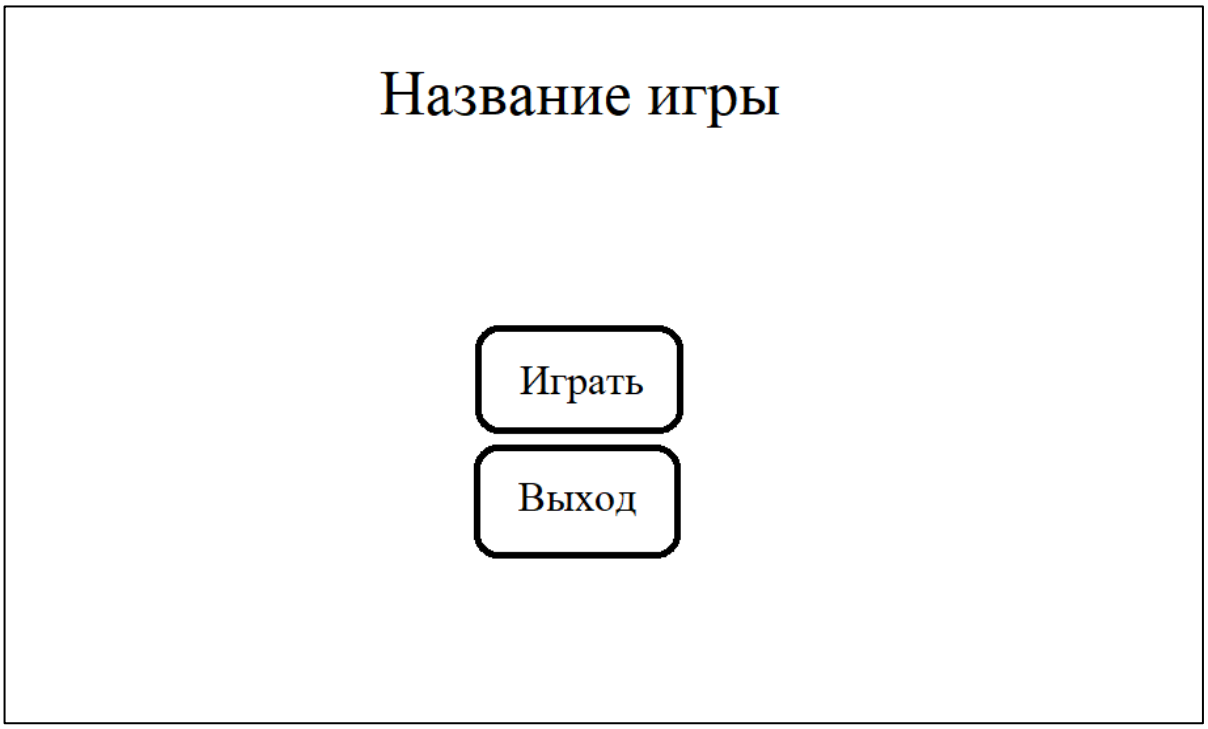

Рисунок 4 – Макет главного меню

Во время игры игрок также должен будет видеть свой уровень здоровья. Он будет отображаться в левом верхнем углу экрана.

Во время игрового процесса игрок может поставить игру на паузу, что предоставляет ему возможность временно остановить процесс игры для осуществления различных действий. Находясь на паузе, игрок может принять решение продолжить играть, чтобы вернуться игровому процессу, переза-

пустить уровень, если столкнулся с трудностями или хочет попробовать новые стратегии; выйти в главное меню или выйти из игры, если нужно завершить игровой сеанс. Этот механизм позволяет игрокам легко управлять игровым процессом и наслаждаться игрой в соответствии с их предпочтениями и потребностями. Разработанный макет представлен на рисунке 5.

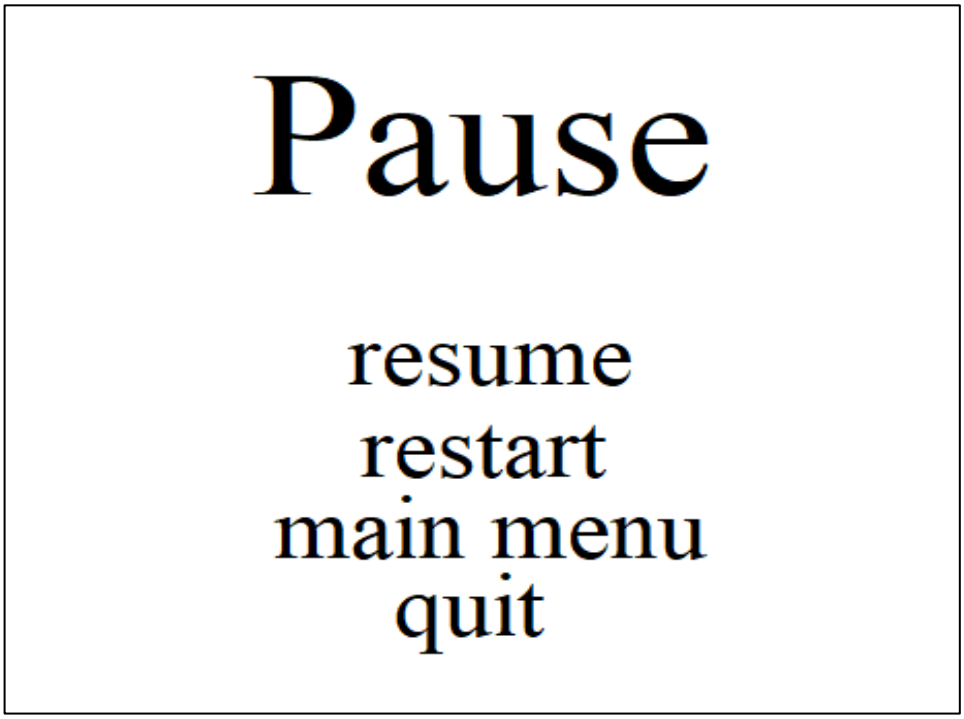

Рисунок 5 – Макет экрана паузы

2.3. Функциональные требования и нефункциональные требования

Функциональные требования – это требования, которые определяют действия, которые должна выполнять система, без учета ограничений, связанных с ее реализацией, то есть определяют поведение системы в процессе обработки информации. Разрабатываемое приложение должно удовлетворять следующим функциональным требованиям.

1. Пользователь должен иметь возможность управлять персонажем.
2. Возможность осуществлять перемещение по горизонтальной оси.
3. Возможность осуществлять прыжок.
4. Игровой персонаж должен получать урон.
5. В приложении должно быть реализовано взаимодействие ландшафтом.

Нефункциональные требования – это требования, которые не определяют поведение системы, но описывают атрибуты системы или атрибуты системного окружения. Разрабатываемое приложение должно удовлетворять следующим нефункциональным требованиям.

1. Приложение должно работать на операционной системе Windows 10 и новее.

2. Приложение должно быть реализовано с использованием платформы Unity.

2.4. Диаграмма вариантов использования и диаграмма компонентов системы

Была разработана диаграмма вариантов использования, в которой представлены игровые возможности. Разработанная диаграмма изображена на рисунке 6.

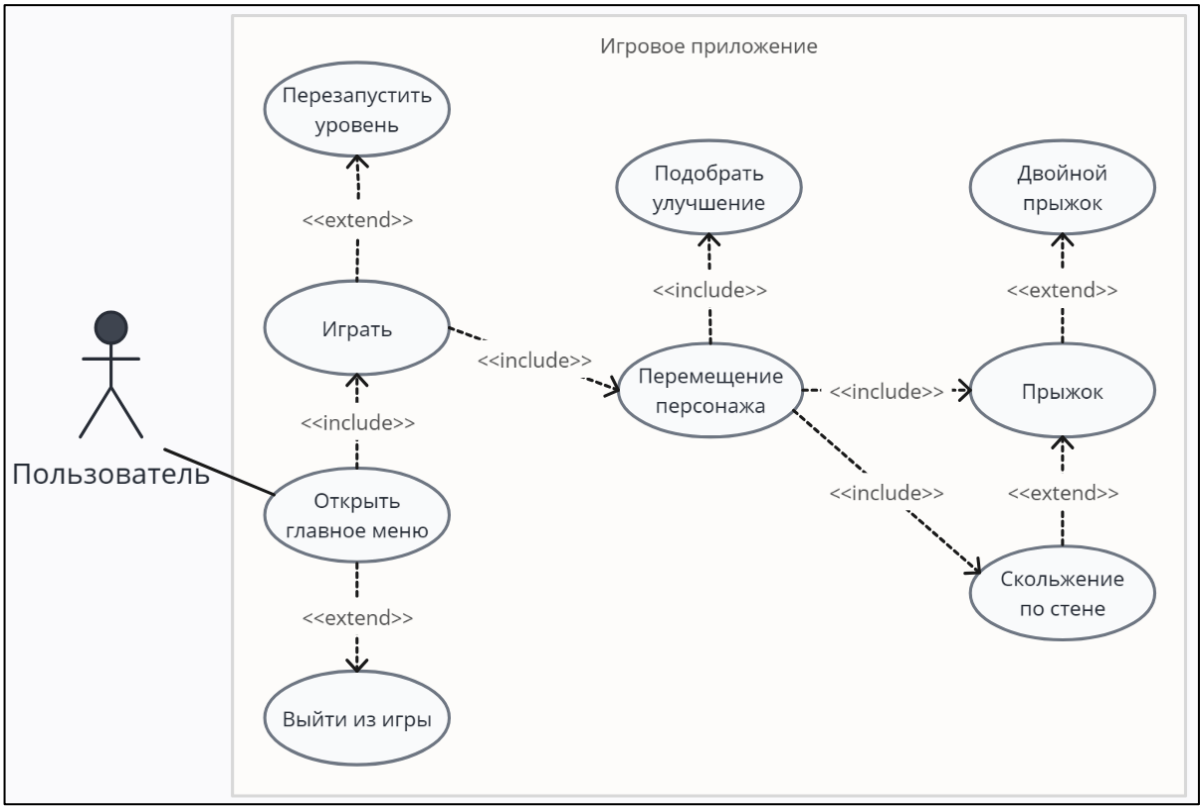

Рисунок 6 – Диаграмма вариантов использования

Диаграмма охватывает ключевые действия, которые пользователь может выполнять, а также связи между этими действиями.

Основные элементы диаграммы представлены ниже.

1. Пользователь: представлен в виде стилизованного человечка слева на диаграмме, пользователь взаимодействует с игровым приложением.

2. Играть: основной вариант использования, включающий в себя несколько вариантов, пользователь выбирает этот вариант для начала игры, включает перемещение персонажа.

3. Открыть главное меню: позволяет пользователю получить доступ к основным опциям игры, включает опцию «Играть», расширяет возможность выйти из игры.

4. Перемещение персонажа: важный вариант использования, включающий в себя несколько вариантов: включает прыжок, включает подбор улучшений, включает скольжение по стене.

5. Прыжок: включает возможность прыжка персонажа, расширяет возможность двойного прыжка.

6. Скольжение по стене: расширяет возможность перемещения персонажа.

7. Подобрать улучшение: пользователь может подобрать различные улучшения, которые влияют на способности персонажа, включает возможность перемещения персонажа.

8. Двойной прыжок: расширяет возможность прыжка.

9. Перезапустить уровень: расширяет возможность игры, предоставляя пользователю опцию начать уровень заново.

10. Выйти из игры: расширяет возможность открытия главного меню, предоставляя пользователю опцию завершить игру.

Была разработана диаграмма компонентов, которая показывает разбиение системы на структурные компоненты и связи показана на рисунке 7.

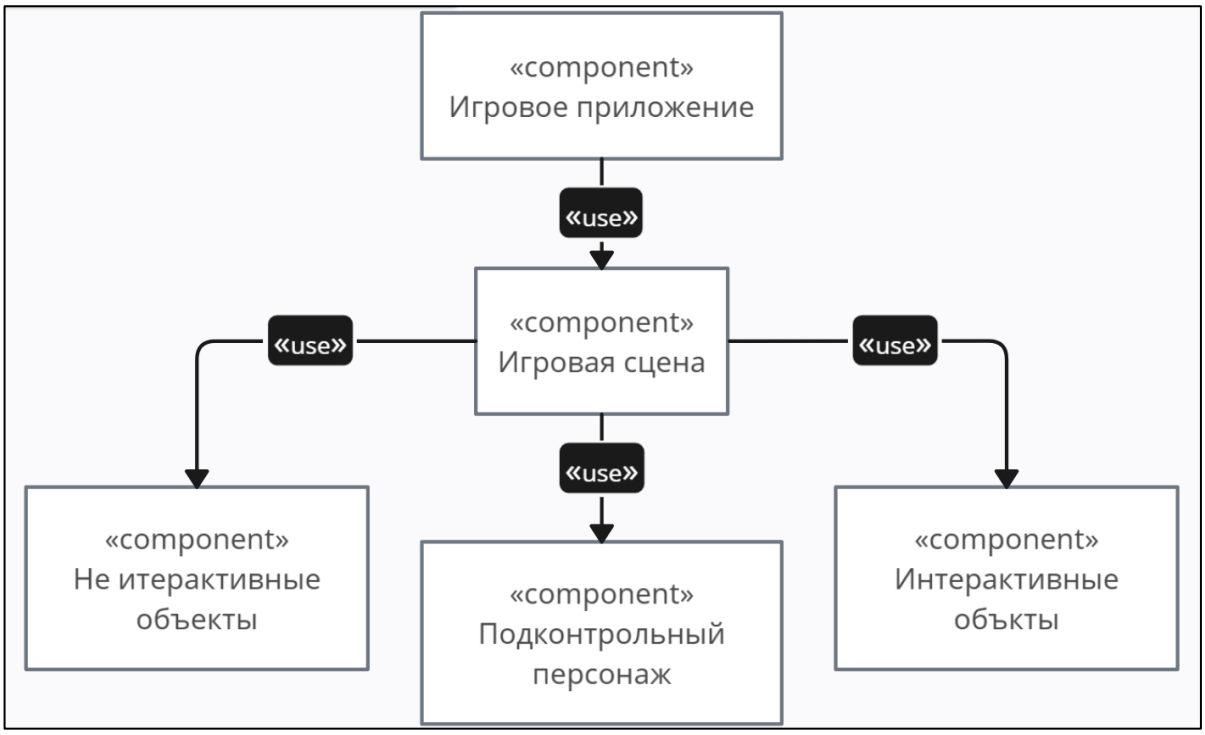

Рисунок 7 – Диаграмма компонентов системы

Описание компонентов, показанных на рисунке 7, представлено ниже.

1. Компонент игрового приложения реализует базовую функциональность Unity.

2. Компонент игровых сцен – содержит набор классов, реализующих Главные экраны меню, меню паузы, основной игровой сценой.

3. Компонент подконтрольного персонажа – ответственен за управление персонажем.

4. Компонент интерактивных объектов – содержит набор классов, реализующих сущности, с которыми может взаимодействовать.

5. Компонент не интерактивных объектов – содержит набор объектов, с которыми подконтрольный персонаж не может взаимодействовать.

3.РЕАЛИЗАЦИЯ

3.1. Программные средства реализации

Для реализации системы был выбран игровой движок Unity версии 2023.1. Unity – это один из самых популярных и мощных игровых движков в индустрии, предоставляющий разработчикам возможность создавать как двухмерные, так и трехмерные игры. Вот несколько ключевых характеристик Unity [15].

1. Мощный визуальный редактор. Unity предоставляет интуитивно понятный визуальный интерфейс для создания игрового контента без необходимости писать много кода.

2. Обширная библиотека ресурсов. Unity поставляется с богатой библиотекой ресурсов, таких как модели, текстуры, анимации и звуковые эффекты, что ускоряет процесс разработки игр.

3. Высокая производительность. Движок Unity оптимизирован для работы с графикой и физикой, обеспечивая высокую производительность игр даже на мобильных устройствах.

4. Обширное сообщество и поддержка. Unity имеет огромное сообщество разработчиков, что обеспечивает доступ к обучающим материалам, форумам поддержки и решению проблем.

5. Множество платформ. Unity поддерживает разработку игр для различных платформ, включая Windows, macOS, Android, iOS, консоли, виртуальную реальность и многое другое.

6. Интегрированный редактор. Unity поставляется с мощным интегрированным редактором, который облегчает создание игровых объектов, управление анимацией, разработку игровых уровней и многое другое.

7. Физический движок. Unity предоставляет встроенный физический движок для создания реалистичного поведения объектов в игре, что облегчает создание физических головоломок и симуляцию поведения персонажей.

Интерфейс Unity также интуитивен и позволяет разработчикам создавать и редактировать игровые объекты, анимации, а также проектировать игровые уровни. Unity предлагает гибкость как для начинающих, так и для опытных разработчиков, и является одним из наиболее популярных выборов для создания качественных игр.

3.2. Реализация интерфейса

Главное меню реализовано при помощи отдельной сцены MainMenu. Оно состоит из названия игры и двух кнопок: «Играть» и «Выйти».

Скриншот главного меню изображен на рисунке 8.

Рисунок 8 – Главное меню

Реализация главного меню представлена в листинге 1.

Листинг 1 – Код реализации главного меню

```
public class MainMenu : MonoBehaviour
{
    public void LoadFirstLevel()
    {
        SceneManager.LoadScene(1);
    public void Quit()
        Application.Quit();
```

Объект `HealthBar` отображает здоровье игрока. Реализация объекта `HealthBar` показана в листинге 2.

Листинг 2 – Код реализации `HealthBar`

```
public class HealthBar : MonoBehaviour
{
    [SerializeField] private Health playerHealth;
    [SerializeField] private Image totalHealthBar;
    [SerializeField] private Image currentHealth;
    private void Start()
    {
        totalHealthBar.fillAmount = playerHealth.currentHealth / 10;
    }
    private void Update()
    {
        currentHealth.fillAmount = playerHealth.currentHealth / 10;
    }
}
```

3.3. Реализация спрайтов

Вся графическая составляющая игры была реализована с помощью спрайтов. Спрайты персонажа и противников были взяты в некоммерческих целях из открытого доступа. Ниже приведены изображения основных спрайтов, используемых в игре.

Рисунок 9 содержит изображение спрайта «Главный персонаж».

Рисунок 9 – «Главный персонаж»

Рисунок 10 содержит изображение спрайта «Прыжок от стены».

Рисунок 10 – «Прыжок от стены»

Рисунок 11 содержит изображение спрайта «Двойного прыжка».

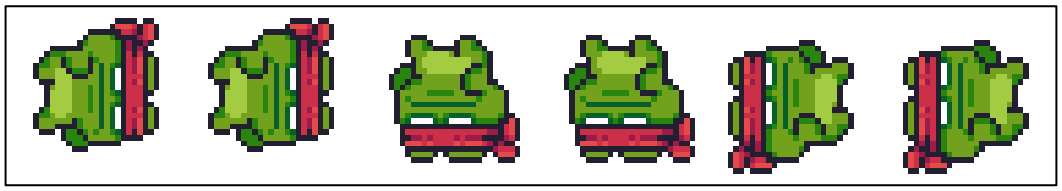

Рисунок 11 – «Двойной прыжок»

Рисунок 12 содержит изображение спрайта «Получение урона».

Рисунок 12 – «Получение урона»

Использование спрайтов в компьютерной графике и разработке игр имеет множество преимуществ. Ниже приведены основные преимущества использования спрайтов.

1. Простота и эффективность: спрайты позволяют легко управлять графическим контентом игры, обеспечивая эффективное использование памяти и ресурсов.

2. Гибкость: спрайты можно легко анимировать и комбинировать для создания разнообразных визуальных эффектов.

3. Доступность: существуют многочисленные библиотеки и ресурсы для бесплатного использования спрайтов, что упрощает процесс разработки и позволяет сосредоточиться на геймплее.

Интеграция спрайтов в игровой движок Unity является важным этапом при разработке игр. Ниже приведены шаги процесса интеграции спрайтов в Unity.

1. Импорт спрайтов: для начала необходимо импортировать все необходимые спрайты в проект Unity. Это делается путем перетаскивания файлов спрайтов в папку Assets в Unity.

2. Создание анимаций: используя инструмент Animation в Unity, можно создавать анимации для различных состояний персонажа (например, бег, прыжок, получение урона). Для этого нужно добавить спрайты в Animation Clip и настроить их последовательность.

3. Настройка анимаций: с помощью Animator Controller можно настроить переходы между различными анимациями, определяя условия для смены анимаций, такие как нажатие клавиш или столкновение с объектами.

4. Интеграция с кодом: анимации могут быть связаны с кодом, чтобы определенные действия персонажа (например, прыжок или получение урона) вызывали соответствующие анимации. Это достигается через скрипты, которые управляют состоянием анимаций.

Использование спрайтов в разработке платформера на Unity предоставляет множество преимуществ, таких как простота, гибкость и доступность. Это позволяет создавать качественные визуальные эффекты и анимации, улучшая общий игровой процесс.

3.4. Управление персонажем

Перемещение главного персонажа игры осуществляется при помощи клавиш WASD, прыжок персонажа осуществляется с помощью клавиши пробел. Это стандартный способ управления в большинстве игр, обеспечивающий удобство и интуитивное понимание для игрока. В данной секции рассмотрены ключевые аспекты управления персонажем.

Передвижение персонажа осуществляется с помощью класса, который отвечает за физическое поведение объекта в 2D пространстве. Код передвижения игрока представлен в листинге 3.

Листинг 3 – Код реализации передвижения игрока

```
private void FixedUpdate()
{
    if (!isRoomChange && !isWallJumping && !inDamage)
    {
        rb.velocity = new Vector2(horizontal * speed, rb.velocity.y);
}
```

Этот метод проверяет, не находится ли персонаж в состоянии изменения комнаты, прыжка от стены или получения урона. Если ни одно из этих условий не выполняется, то скорость персонажа устанавливается на основе текущего горизонтального ввода и заданной скорости.

Реализация прыжка и двойного прыжка представлена в листинге 4. Эти функции обеспечивают основные возможности перемещения персонажа в вертикальной плоскости, что является важным элементом игрового процесса.

Листинг 4 – Реализация `Jump` и `DoubleJump`

```
private void Jump()
{
    // Обновление таймера для прыжка от стены
    if (isWallSliding)
    {
        isWallJumping = false;
        wallJumpingDirection = -transform.localScale.x;
        wallJumpingCooldownCounter = wallJumpingCooldown;
        CancelInvoke(nameof(StopWallJumping));
    else
    if (Input.GetKeyDown(KeyCode.Space))
    {
        if (IsGrounded())
            rb.velocity = new Vector2(rb.velocity.x, jumpPower);
            doubleJump = !doubleJump;
            if (!doubleJump)
            {
                animator.SetBool("IsDoubleJump", true);
            }
            SoundManager.Instance.PlaySound(jumpSound);
        else if (canWallJump && wallJumpingCooldownCounter > 0f)
            isWallJumping = true;
            rb.velocity = new Vector2(wallJumpingDirection * wallJumping-
Power.x, wallJumpingPower.y);
            wallJumpingCooldownCounter = 0f;
            if (transform.localScale.x != wallJumpingDirection)
                isFacingRight = !isFacingRight;
                Vector3 localScale = transform.localScale;
                localScale.x *= -1f;
                transform.localScale = localScale;
            doubleJump = !doubleJump;
            if (!doubleJump)
                animator.SetBool("IsDoubleJump", true);
                SoundManager.Instance.PlaySound(jumpSound);
                Invoke(nameof(StopWallJumping), wallJumpingDuration);
        else if (canDoubleJump && doubleJump)
            rb.velocity = new Vector2(rb.velocity.x, jumpPower);
            doubleJump = !doubleJump;
            if (!doubleJump)
            {
                animator.SetBool("IsDoubleJump", true);
            SoundManager.Instance.PlaySound(jumpSound);
    if (Input.GetKeyUp(KeyCode.Space) && rb.velocity.y > 0)
        rb.velocity = new Vector2(rb.velocity.x, rb.velocity.y * 0.5f);
}
```

В этом методе рассмотрены различные сценарии прыжка: обычный прыжок, прыжок от стены и двойной прыжок. Это делает игровой процесс более динамичным и интересным.

Скольжение по стене позволяет персонажу замедлять падение, что особенно полезно в ситуациях, когда необходимо точно приземлиться на узкую платформу или избежать препятствий. Реализация скольжения по стене представлена в листинге 5.

Листинг 5 – Реализация скольжения по стене

```
private void WallSlide()
{
    if (IsWall() && !IsGrounded() && horizontal != 0f)
    {
         isWallSliding = true;
         rb.velocity = new Vector2(rb.velocity.x, Mathf.Clamp(rb.veloc-
ity.y, -wallSlidingSpeed, float.MaxValue));
    }
    else
    {
        isWallSliding = false;
   }
}
```

Метод `WallSlide` проверяет, находится ли персонаж рядом со стеной, не на земле ли он и есть ли горизонтальное движение. Если все условия выполняются, активируется режим скольжения по стене.

Когда персонаж получает урон, его отбрасывает назад, что придает игре реалистичности и усложняет игровой процесс. Реализация отбрасывания при получении урона представлена в листинге 6.

Листинг 6 – Реализация отбрасывания при получении урона

```
public void MakeDamageMove(Transform from)
{
    Vector2 directionVector = new Vector2(transform.position.x - from.posi-
tion.x, transform.position.y - from.position.y);
    inDamage = true;
    float horizontalSpeed = intialDamageSpeed * Mathf.Cos(angle *
Mathf.Deg2Rad);
    float verticalSpeed = intialDamageSpeed * Mathf.Sin(angle *
Mathf.Deg2Rad);
    var horizontalDirection = directionVector.x > 0f ? Vector2.right : Vec-
tor2.left;
    rb.AddForce(horizontalDirection * horizontalSpeed, ForceMode2D.Im-
pulse);
    rb.AddForce(Vector2.up * verticalSpeed, ForceMode2D.Impulse);
    damageOnGroundCooldown = damageOnGroundInitial;
    Invoke(nameof(CancelDamaged), damageDuration);
}
```

Метод `MakeDamageMove` рассчитывает направление и силу отбрасывания на основе позиции атакующего (from) и позиции персонажа. Это добавляет элемент стратегии, требуя от игрока избегать атак и планировать свои действия.

3.5 Реализация камеры

Правильное управление камерой играет ключевую роль в создании комфортного игрового процесса. Камера должна следить за персонажем, обеспечивая игроку хороший обзор игрового пространства и помогая ориентироваться в уровне. В данном разделе описана реализация камеры в игре с использованием объекта `CameraController`.

Камера прикреплена к комнате, в которой находится игрок, и плавно переходит за ним, когда он перемещается в другую комнату. Такое поведение камеры позволяет игроку лучше понимать структуру уровня и выбирать оптимальные маршруты его прохождения. Реализация камеры представлена в листинге 7.

Листинг 7 – Код реализации камеры

```
public class CameraController : MonoBehaviour
{
    [SerializeField] private float speed;
    [SerializeField] private PlayerMovement playerMovement;
    private float currentPositionX;
    private float currentPositionY;
    private Vector3 velocity = Vector3.zero;

    private void Update()
    {
        transform.position = Vector3.SmoothDamp(transform.position, new 
Vector3(currentPositionX, currentPositionY, transform.position.z), ref ve-
locity, speed);
    }
    public void MoveToNewRoom(Transform _newRoom)
    {
        currentPositionX = _newRoom.position.x;
        currentPositionY = _newRoom.position.y;
    }
}
```

В этом коде класс `CameraController` использует метод `SmoothDamp` для плавного перемещения камеры. Параметры `currentPositionX` и

currentPositionY определяют текущие координаты камеры, которые обновляются при переходе в новую комнату методом MoveToNewRoom.

Объяснение работы камеры

1. Плавное следование камеры: метод Update вызывает SmoothDamp для плавного перемещения камеры к новым координатам. Это обеспечивает мягкое и непрерывное движение камеры, что улучшает визуальное восприятие игры.

2. Переход между комнатами: метод MoveToNewRoom обновляет координаты камеры в соответствии с позицией новой комнаты. Когда игрок переходит в новую комнату, камера плавно следует за ним, обеспечивая непрерывность игрового процесса.

Действие камеры, когда игрок переходит из одной комнаты в другую. Вызывается метод MoveToNewRoom, и камера начинает плавное перемещение к новой позиции. Реализация перехода камеры в другую комнату показана в листинге 8.

Листинг 8 – Код реализации перехода камеры между комнатами

```
void OnTriggerEnter2D(Collider2D other)
{
    if (other.CompareTag("Player"))
    {
        CameraController.Instance.MoveToNewRoom(transform);
    }
}
```

Плавное перемещение камеры и ее точное следование за персонажем позволяют игроку лучше ориентироваться в игровом пространстве и наслаждаться игровым процессом.

3.6 Анимации

Анимация в Unity – это процесс изменения свойств объектов с течением времени. Unity использует систему анимации Mecanim, которая предоставляет мощный инструмент для создания, редактирования и управления анимацией. Mecanim поддерживает как 2D, так и 3D анимации, предлагая гибкость для разработчиков игр.

В системе анимации Unity существует несколько ключевых компонентов. Ниже перечислены основные компоненты системы анимации Unity.

1. Animator Controller: основной инструмент управления анимациями. Он определяет, какие анимации и когда должны воспроизводиться.

2. Animation Clips: файлы, содержащие данные об анимации. Они описывают, как меняются свойства объекта в течение времени.

3. Animator State Machine: механизм, состоящий из состояний и переходов между ними. Каждое состояние представляет собой анимацию.

4. Transitions: переходы между состояниями, которые определяют, когда и как происходит переключение между анимациями.

На рисунке 13 представлен Animator Controller для персонажа. Он управляет различными анимационными состояниями.

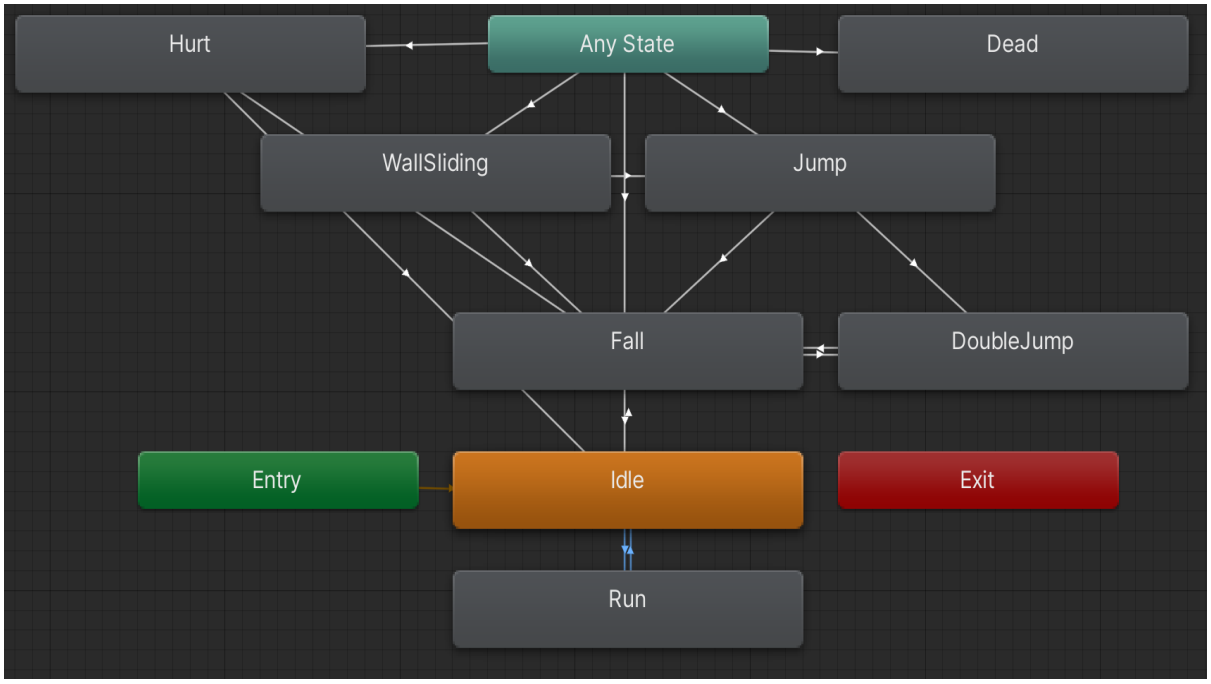

Рисунок 13 – Аниматор персонажа

Основные состояния и их взаимосвязи аниматора персонажа представлены ниже.

1. Any State: состояние, из которого можно перейти в любое другое состояние. Это позволяет быстро переключаться на критические анимации, такие как «Hurt» или «Dead».

2. Entry: начальное состояние, с которого начинается выполнение анимаций при запуске игры. Оно ведет к состоянию «Idle».

3. Idle: состояние покоя, когда персонаж не выполняет никаких действий. Это состояние может переходить в: Run, Jump, Fall, Hurt, Dead.

4. Run: состояние бега, активируется, когда персонаж движется. Переходы аналогичны состоянию «Idle».

5. Jump: состояние прыжка, активируется, когда персонаж прыгает. Это состояние может переходить в: DoubleJump, Fall.

6. DoubleJump: состояние двойного прыжка. Оно может переходить в состояние «Fall».

7. Fall: состояние падения, активируется, когда персонаж находится в воздухе и не прыгает. Может переходить в: WallSliding, Hurt, Dead.

8. WallSliding: состояние скольжения по стене. Может переходить в «Fall».

9. Hurt: состояние получения урона. Активируется при получении персонажем урона. Может переходить в «Fall» или «Dead».

10. Dead: состояние смерти персонажа. Активируется, когда здоровье персонажа опускается до нуля.

Переходы между состояниями определяются условиями, которые проверяются в реальном времени. Например: переход из состояния «Idle» в «Run» может быть основан на условии, что скорость персонажа больше нуля. Переход из состояния «Jump» в «Fall» происходит, когда скорость по оси Y становится отрицательной (персонаж начинает падать). Переходы могут быть мгновенными или с плавными интерполяциями, чтобы обеспечить более естественные и реалистичные анимации.

4. ТЕСТИРОВАНИЕ

Тестирование является ключевым этапом разработки, обеспечивающим соответствие реализованного приложения заявленным функциональным требованиям и спецификациям, описанным на этапе проектирования. В ходе данного тестирования проверялось работоспособность приложения, выявление ошибок и несоответствий, а также оценивалось общее качество игрового процесса.

Функциональное тестирование

Функциональное тестирование направлено на проверку того, что все элементы и функции приложения работают корректно и в соответствии с ожиданиями. Ниже представлена таблица 1 с результатами функционального тестирования реализованного игрового приложения.

Таблица 1 – Результаты функционального тестирования

№	Название теста	Описание теста	Ожидаемый результат	Фактический результат	Статус
1	Передвижение персонажа	Проверка передвижения с помощью клавиш WASD	Персонаж движется влево, вправо, вверх и вниз	Персонаж движется в соответствии с вводом	Пройден
2	Прыжок персонажа	Проверка прыжка при нажатии пробела	Персонаж прыгает	Персонаж прыгает при нажатии пробела	Пройден
3	Двойной прыжок	Проверка двойного прыжка при повторном нажатии пробела	Персонаж выполняет двойной прыжок	Персонаж выполняет двойной прыжок	Пройден
4	Скольжение по стене	Проверка скольжения по стене при соприкосновении с ней	Персонаж скользит по стене	Персонаж скользит по стене	Пройден
5	Отбрасывание при получении урона	Проверка отбрасывания персонажа при получении урона	Персонаж отбрасывается назад	Персонаж отбрасывается в правильном направлении	Пройден
6	Перемещение камеры	Проверка следования камеры за персонажем	Камера плавно следует за персонажем	Камера плавно следует за персонажем	Пройден
7	Переход камеры между комнатами	Проверка перемещения камеры при переходе персонажа в новую комнату	Камера плавно перемещается в новую комнату	Камера перемещается в новую комнату	Пройден

Анализ результатов тестирования

Проведенное функциональное тестирование показало, что все основные функции игрового приложения работают корректно и в соответствии с предъявленными требованиями. Ниже приведены ключевые моменты анализа.

1. Передвижение персонажа: персонаж корректно реагирует на ввод с клавиатуры, что обеспечивает удобство управления.

2. Прыжок и двойной прыжок: механика прыжков работает без сбоев, позволяя персонажу выполнять как обычные, так и двойные прыжки.

3. Скольжение по стене: персонаж корректно скользит по стенам, что добавляет дополнительные возможности для навигации по уровню.

4. Отбрасывание при получении урона: персонаж правильно отбрасывается при получении урона, что делает игровой процесс более реалистичным.

5. Перемещение камеры: камера плавно следует за персонажем и корректно перемещается между комнатами, обеспечивая хороший обзор игрового пространства.

Результаты тестирования подтверждают, что реализованное игровое приложение соответствует функциональным требованиям и спецификациям. Все основные механики управления и взаимодействия работают корректно, обеспечивая положительный пользовательский опыт.

ЗАКЛЮЧЕНИЕ

В рамках данной работы была разработана компьютерная игра «Полет амфибии» на платформе Unity. Основная цель работы заключалась в создании интерактивного и увлекательного игрового приложения, которое соответствовало бы современным стандартам игрового дизайна и программирования. В процессе выполнения работы были решены следующие задачи.

1. Произведен обзор литературы: был проведен анализ современных тенденций в разработке игр, рассмотрены основные принципы и методы создания платформеров.

2. Проведен анализ предметной области и обзор аналогов: Проанализированы существующие игры в жанре платформер, определены их сильные и слабые стороны, что позволило выявить ключевые элементы, которые следовало учесть при разработке собственного приложения.

3. Спроектировано приложение: на этапе проектирования были разработаны основные компоненты игры, включая управление персонажем, взаимодействие с окружающей средой, механику прыжков и двойных прыжков, скольжение по стенам и реализацию камеры.

4. Реализовано и протестировано приложение: разработка игры включала создание кода для всех основных механик, а также их тестирование для обеспечения корректной работы.

Разработка данного игрового приложения позволила не только углубить знания в области программирования на платформе Unity, но и приобрести практический опыт в создании игр.

Проведенное тестирование подтвердило, что игра соответствует функциональным требованиям и спецификациям, что позволяет говорить о высокой степени готовности приложения к дальнейшему развитию и возможному выпуску.

ЛИТЕРАТУРА

1. Кто такой инди-разработчик. [Электронный ресурс] URL: https://hsbi.hse.ru/career/professions/razrabotchiki–indi–igr–kto–eto–i–kak–imi–stat/ (дата обращения: 04.03.2024 г.).
2. Что такое платформер. [Электронный ресурс] URL: https://dic.academic.ru/dic.nsf/ruwiki/106975 (дата обращения: 03.03.2024 г.).
3. Что такое спрайт. [Электронный ресурс] URL: https://dic.academic.ru/dic.nsf/ruwiki/42140 (дата обращения: 05.03.2024 г.).
4. Что такое скрипт. [Электронный ресурс] URL: https://ru.hostings.info/termins/chto–takoe–skript.html (дата обращения: 05.04.2024 г.).
5. Что такое геймплей. [Электронный ресурс] URL: https://kartaslov.ru/значение-слова/геймплей (дата обращения: 05.04.2024 г.).
6. Что такое скриншот. [Электронный ресурс] URL: https://clov.net/screenshot (дата обращения: 05.03.2024 г.).
7. Бонд Д. Г. Unity и C#. Геймдев от идеи до реализации. // Геймдев и прототипирование, М.: Питер, 2019.– С. 124–127.
8. Корнилов А.В. Unity. Полное руководство. // Основы Unity, СПб: Букмастер, 2020. – С. 20–22.
9. Официальная страница игры Super Mario Bros. [Электронный ресурс] URL: https://www.nintendo.ru/-/NES/Super-Mario-Bros-803853.html (дата обращения: 07.03.2024 г.).
10. Официальный сайт компании Nintendo. [Электронный ресурс] URL: https://www.nintendo.ru/ (дата обращения: 07.03.2024 г.).
11. Официальный сайт «Super Meat Boy». [Электронный ресурс] URL: http://www.supermeatboy.com/ (дата обращения: 06.03.2024 г.).
12. Metacritic – «Super Meat Boy». [Электронный ресурс] URL: https://www.metacritic.com/game/pc/super–meat–boy (дата обращения: 06.03.2024 г.)

13. Официальный сайт игры Celeste. [Электронный ресурс] URL: https://www.celestegame.com/ (дата обращения: 07.03.2024 г.).

14. Официальный сайт компании Matt Makes Games. [Электронный ресурс] URL: https://www.maddymakesgames.com/ (дата обращения: 07.03.2024 г.).

15. Официальный сайт Unity. [Электронный ресурс] URL: https://unity.com/ (дата обращения: 07.03.2024 г.).