Частное учреждение образования

«Колледж бизнеса и права»

ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ИГРЫ «COOL DRIVER»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к проекту по учебному предмету «Разработка и сопровождение программного обеспечения»

КП Т.293018.401

Руководитель проекта ( В.Ю. Михалевич )

Обучающийся ( Д.С. Пранович )

2025

2025

СОДЕРЖАНИЕ

Изм.

Изм.

Изм.

Изм.

Лист

Лист

Лист

Лист

№ докум.

№ докум.

№ докум.

№ докум.

Подпись

Подпись

Подпись

Подпись

Дата

Дата

Дата

Дата

Лист

Лист

Лист

Лист

3

3

3

3

КП Т.293018.401 ПЗ

КП Т.716007.401 ПЗ

КП Т.716009.401 ПЗ

КП Т.716007.401 ПЗ

Разраб.

Разраб.

Разраб.

Разраб.

*Пранович Д.С.*

Провер.

Провер.

Провер.

Провер.

Михалевич В.Ю.

Банцевич. С.В

Банцевич. С.В

Банцевич. С.В

Т. контр.

Т. контр.

Т. контр.

Т. контр.

Н. контр.

Н. контр.

Н. контр.

Н. контр.

Утверд.

Утверд.

Утверд.

Утверд.

Программная реализация игры «Cool Driver»

Лит.

У

Лит.

У

Лит.

У

Лит.

У

Листов

Листов

Листов

Листов

43

50

55

50

КБП

КБП

КБП

КБП

[Введение 4](#_Toc201047793)

[1. Объектно-ориентированный анализ и проектирование системы 5](#_Toc201047794)

[1.1. Сущность задачи 5](#_Toc201047795)

[2. Проектирование игры 6](#_Toc201047796)

[2.1. Проектирование модели 6](#_Toc201047797)

[2.2. Требования к игре и ее наполнению 11](#_Toc201047798)

[2.3. Структура игры 12](#_Toc201047799)

[2.4. Концептуальный прототип 13](#_Toc201047800)

[2.5. Организация данных 16](#_Toc201047801)

[3. Реализация игры 18](#_Toc201047802)

[3.1. Инструменты разработки 18](#_Toc201047803)

[3.2. Реализация сценария 18](#_Toc201047804)

[3.3. Реализация поведения объектов 19](#_Toc201047805)

[3.4. Функциональное тестирование 20](#_Toc201047806)

[4. Применение 24](#_Toc201047807)

[4.1. Общие сведения 24](#_Toc201047808)

[4.2. Функциональное назначение 24](#_Toc201047809)

[4.3. Условия применения 25](#_Toc201047810)

[Заключение 26](#_Toc201047811)

[Список использованных источников 27](#_Toc201047812)

[ПРИЛОЖЕНИЕ А Текст программных модулей 28](#_Toc201047813)

[ПРИЛОЖЕНИЕ Б Результаты работы программы 43](#_Toc201047814)

### Введение

В современном мире видеоигры стали неотъемлемой частью жизни миллионов людей, особенно среди молодёжи. Игры уже давно перестали быть просто развлечением: для многих они стали способом заработка, формой самовыражения, инструментом для развития когнитивных способностей и даже средой для социального взаимодействия. Сегодня видеоигры объединяют людей разных возрастов и социальных групп, формируя новые привычки, интересы и даже профессиональные навыки.

Согласно последним исследованиям, подавляющее большинство подростков играют в видеоигры, причём значительная часть делает это ежедневно. Более того, игры часто выступают как площадка для общения: многие молодые люди находят друзей именно в виртуальных мирах, а совместные игровые сессии помогают развивать навыки командной работы и стратегического мышления. Кроме того, игры оказывают положительное влияние на развитие когнитивных функций - они помогают улучшать память, внимание, скорость принятия решений и умение анализировать сложные ситуации.

Однако не стоит забывать и о рисках: чрезмерное увлечение играми может привести к нарушению режима сна, снижению успеваемости и даже к социальной изоляции. Поэтому важно соблюдать баланс между игрой и другими аспектами жизни, использовать игры как инструмент для отдыха, обучения и саморазвития.

Примером современной игры, которая не только развлекает, но и тренирует умственные способности, может служить «Cool Driver». Эта игра, предлагает пользователю не только приятно провести время, но и активно развивать стратегическое мышление и логику. Игроку необходимо просчитывать вражеские ходы, минимизировать затраты ресурсов и стремиться к максимальному результату, что делает процесс увлекательным и полезным для мозга.

Таким образом, видеоигры сегодня выполняют множество функций: от простого отдыха до профессионального заработка, от развития социальных навыков до тренировки умственных способностей. Они стали неотъемлемой частью современной культуры, оказывая значительное влияние на формирование личности и образа жизни молодёжи и взрослых людей. Введение новых игровых технологий и развитие цифровых сервисов открывают перед пользователями всё новые возможности для самореализации и развития, что делает тему особенно актуальной и востребованной в научной и практической деятельности.

### 1. Объектно-ориентированный анализ и проектирование системы

### 1.1. Сущность задачи

Требуется разработать программное средство для симулятора интерактивных сражений «Люмен», представляющего собой RPG с открытым миром, тактическими элементами и продуманной системой сражений. Предметной областью является ролевая игра с видом сверху, в которой игрок управляет персонажем в открытом мире, выполняет квесты, взаимодействует с NPC и развивает персонажа через систему навыков и предметов, как отмечается в книге А. И. Петрова «C# и Unity: основы разработки 2D-игр» [[6]](https://skillbox.ru/media/gamedev/kak_sozdat_prostuyu_2d_igru_na_unity/).

Объектом решаемой задачи является тактическая игра с видом сверху, в которой игрок управляет колодой карт, сражается с противниками в современном и увлекательном формате, используя различные тактики и комбинации действий для достижения победы.

Игроку предоставляется возможность собирать и улучшать снаряжение, сражаться с различными противниками, использовать разнообразные тактические комбинации. Как указывается в книге М. Робертса «Unity 2023 Game Development in 24 Hours», успешная реализация таких механик требует внимательной проработки классов и систем управления ресурсами [10].

Враги делятся на несколько типов: обычные и босс. Каждый враг обладает уникальными характеристиками, которые разработчик может изменить для каждого врага индивидуально. Похожий подход к системам карточных эффектов подробно рассматривается в руководстве Microsoft по программированию на C#, где акцентируется внимание на адаптивной логике взаимодействий [[7](https://unity.com/ru/how-to/optimize-performance-2d-games-unity-tilemap)].

Игровой процесс включает в себя продуманные стратегические ходы и адаптацию под тактику противника, как это отмечается в руководстве от O’Reilly Media «Learning C# by Developing Games with Unity 2023», где подчеркивается значимость сценариев ИИ и систем анализа состояний боя [9].

Сюжет игры разворачивается в полуразрушенной деревушке, охваченной хаосом. Главный герой – местный житель, решивший бросить вызов сильнейшим противникам, стремясь восстановить баланс сил и отбить свою родину. Разработка подобных игровых сюжетов требует чёткого моделирования мира и логики его развития через механики, что также отражается в полном руководстве по C# и .NET на METANIT.COM [[8](https://qna.habr.com/q/62312)].

С точки зрения реализации, необходимо создать базовую систему управлением персонажа, логику взаимодействия между врагом и героем, а также систему использования предметов и прокачки навыков. Здесь важно учитывать особенности разработки 2D-игр на Unity, включая оптимизацию и управление ресурсами, что подробно описано в книге А. И. Петрова [[6](https://skillbox.ru/media/gamedev/kak_sozdat_prostuyu_2d_igru_na_unity/)].

На основании исследованной предметной области следует выполнить следующие пункты:

* управление персонажем с помощью клавиш WASD;
* заговорить с НПС;
* прохождение линейных квестов (с фиксированным вознаграждением);
* финальный босс (концовка игры);
* выбрать варианты ответа;
* использование предметов;
* получить урон;
* нанести урон;
* переключение на атаку дальнего боя (например стрельба из пистолета или лука);
* статистика пользователя (количество убитых врагов и т.д.);
* убить врага;
* умереть самому и возродиться;
* система инвентаря (использовать предмет (некоторые предметы могут влиять на характеристики персонажа или как-то влиять на его состояние), выбросить предмет, добавить в инвентарь);
* система покупки и продажи каких-либо игровых ценностей (Система магазина);
* функция загрузки предыдущих игровых сессий (множество);
* функция создания новой игровой сессии;
* функция изменения настроек звука;
* функция изменения разрешения экрана;
* функция изменения экранного режима.

Аналогами данного проекта являются игры с стратегическим уклоном, такие как «The Legend of Zelda». Суть игры состоит в следующем: игрок, когда появляется на локацию и у него не будет базового оружия, для его получения ему потребуется вкачать определенный навык, после чего он сразу же сможет использовать начальное оружие. Основная цель – победить всех врагов, включая босса, получить максимальное снаряжение и выполнить все задания. Игроку предстоит продумывать тактику и использовать различные способности, чтобы добиться максимального результата.

### 2. Проектирование игры

### 2.1. Проектирование модели

Диаграмма сущность-связь, представленная на рисунке 2.1, отражает ключевые объекты системы и их взаимодействие в рамках игровой механики. Центральной сущностью диаграммы выступает «Игрок», который связан с рядом других сущностей, определяющих его возможности и поведение в игре.

Игрок обладает характеристиками, такими как здоровье, скорость и урон, которые напрямую влияют на его эффективность в бою. Сущность «Характеристика» связана не только с игроком, но и с врагом, что позволяет реализовать гибкую систему сравнения и взаимодействия между персонажами. Взаимодействие игрока с врагом происходит через сущность «Бой», в рамках которой учитываются параметры наносимого и получаемого урона.

Игрок также обладает навыками, которые зависят от наличия определённого оружия, например, лука или меча. Навыки могут быть улучшены за счёт золота, что стимулирует развитие персонажа. Получение золота возможно как результат успешных боёв или выполнения заданий.

Важной частью игровой системы являются задания, которые игрок получает через взаимодействие с неигровыми персонажами (NPC). Выполнение заданий приводит к получению награды, которая может выражаться в выплате или других ценных ресурсах. Каждое задание имеет свою суть и условия выполнения, что разнообразит игровой процесс.

Торговля реализована через взаимодействие игрока с NPC, обладающими собственным ассортиментом товаров. Это позволяет игроку приобретать необходимые предметы и улучшения, используя заработанное золото.

Таким образом, представленная диаграмма обеспечивает комплексное представление всех ключевых объектов игры и их взаимосвязей, что способствует целостному пониманию логики работы системы и реализации разнообразных игровых сценариев.

Такая структура позволяет наглядно представить все ключевые объекты и их связи, обеспечивая целостное понимание логики работы системы, что описывалось в книге Unity 2023 Game Development in 24 Hours [10].

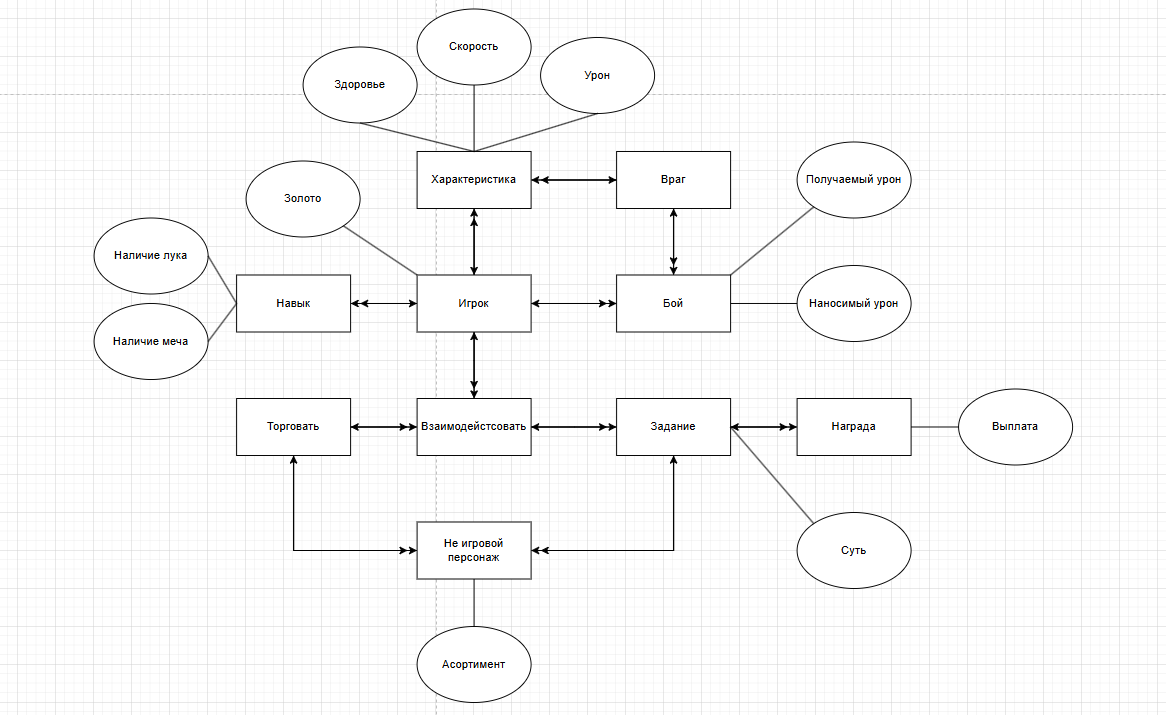


Рисунок 2.1

Диаграмма вариантов использования находится на листе 1 в графической части и отражает функциональность игровой системы с точки зрения взаимодействия игрока с различными элементами игрового процесса. Центральным действующим лицом на диаграмме является сущность «Игрок», которая обладает широким набором возможностей.

Игрок может свободно перемещаться по игровому миру с помощью клавиш WASD. В боевой системе он способен наносить урон врагам, получая в ответ урон, а также может убивать обычных врагов и боссов, что расширяет основное боевое взаимодействие. В случае смерти игрока активируется процесс возрождения, что делает цикл жизни персонажа завершённым.

Кроме боевых действий, игрок может открывать и просматривать свою статистику, а также прокачивать навыки, улучшая способности персонажа. Игрок может включать режим атаки, что, вероятно, активирует боевую стойку или особые действия.

Взаимодействие с неигровыми персонажами (НИП) также занимает важное место. Игрок может говорить с НИПами, брать у них задания, выполнять их и получать соответствующие награды. Также предусмотрена торговля с НИПами, в рамках которой игрок может покупать и продавать предметы.

Инвентарь - ещё один ключевой элемент. Игрок может взаимодействовать с ним: использовать, поднимать, выбрасывать и выбирать предметы. Это даёт возможность управлять экипировкой и ресурсами во время игры.

Игровая система предусматривает работу с сохранениями. Игрок может создавать новые игровые сессии и загружать предыдущие. Это обеспечивает удобное продолжение игрового процесса.

Дополнительно игрок может взаимодействовать с настройками игры. Он способен изменять экранный режим, разрешение экрана и уровень звука, что позволяет адаптировать игру под индивидуальные предпочтения.

Таким образом, диаграмма вариантов использования охватывает все основные и расширенные действия, доступные игроку в рамках игры, от перемещений и сражений до управления инвентарём, взаимодействия с персонажами, настройками и сохранениями.

Диаграмма классов представлена в графической части на листе 2 и отражает ключевые элементы игровой системы и их взаимосвязи. В центре диаграммы находится класс **PlayerMovement**, который отвечает за передвижение игрока и взаимодействует с другими элементами системы. Он содержит такие атрибуты, как компоненты физики (Rigidbody2D), анимации (Animator), а также переменные для управления боем и направлением движения. В классе реализованы методы для перемещения, прыжка, смены направления и обработки отталкивания от врагов.

Класс **EnemyCombat** управляет боем врага. Он взаимодействует с PlayerMovement, вызывая метод отталкивания игрока, когда происходит атака. Этот класс содержит данные о наносимом уроне, радиусе атаки и временном интервале между ударами.

Класс **SkillManager** управляет навыками игрока и реализует методы активации и обработки навыков. Он работает совместно с SkillList, который представляет собой список всех доступных навыков. SkillList содержит информацию о каждом навыке, включая UI-элементы для отображения и кнопки для разблокировки. Кроме того, SkillList взаимодействует с классом **SkillSO**, содержащим описание конкретного навыка, его название, максимальный уровень и иконку.

Класс **SkillTreeManager** управляет деревом навыков и взаимодействует с SkillManager, подписываясь на события, связанные с улучшением навыков.

Класс **StatusManager** является центральным хранилищем характеристик игрока и реализован в виде синглтона. Он содержит такие параметры, как здоровье, урон, время отталкивания и перезарядки, количество очков умений, скорость, а также методы для их обновления. Класс **PlayerMovement** активно использует StatusManager для получения данных о характеристиках и их изменения.

Интерфейс игрока реализуется через классы **StatusUI** и **ToggleSkillTree**. StatusUI отображает значения характеристик игрока, а ToggleSkillTree управляет отображением дерева навыков.

Классы **ElevationEntry**, **ElevationExit**, **UpperElevationEntry** и **UpperElevationExit** обрабатывают столкновения с определёнными участками локации и используются для управления переходами между уровнями или участками карты.

Также в системе присутствуют классы **NPCDialogue** и **NPCMovement**, которые отвечают за движение NPC и взаимодействие с игроком через диалоги. Диалоги управляются через класс **QuestManager**, который хранит активные задания и проверяет их выполнение. Сами задания описываются в классе **Quest**, содержащем параметры состояния, описания и флаг выполнения.

Таким образом, диаграмма иллюстрирует архитектуру взаимодействий между игровыми объектами, системой навыков, интерфейсом пользователя и логикой заданий, обеспечивая комплексное управление поведением игрока, противников, NPC и прогрессией в игре.

Диаграмма последовательности представлена на листе и иллюстрирует этапное взаимодействие между ключевыми объектами игровой системы во время сессии. В качестве участников взаимодействия выступают: **Игрок**, **Враг**, **Босс** и **НИП**, каждый из которых представляет отдельную роль в игровом процессе.

Сценарий начинается с боевого взаимодействия между игроком и противниками. Игрок первым атакует **врага**, после чего враг отвечает ударом, нанося урон игроку. Далее игрок добивает врага, что приводит к уничтожению объекта врага в системе.

Аналогичный сценарий реализован и для **босса**, однако, в отличие от обычного врага, босс также наносит урон перед своей гибелью. После ответной атаки со стороны босса, игрок побеждает его, и объект босса также уничтожается.

Далее следует взаимодействие игрока с **НИП** (неигровым персонажем). Игрок инициирует диалог, после чего НИП отвечает и предоставляет возможность принять задание. После принятия задания, игрок его выполняет и возвращается за наградой, которую НИП вручает в качестве завершения квеста.

В дополнение к квестовой системе, игрок может осуществлять **торговлю** с НИП: покупку или продажу предметов. После совершения сделки НИП отправляет игроку подтверждение.

Помимо взаимодействий с внешними объектами, игрок также совершает **внутренние действия**, отражённые в виде обращений к самому себе. К ним относятся: использование, поднятие или выбрасывание предметов, прокачка навыков, просмотр статистики, создание и загрузка игровых сессий, а также изменение настроек.

В завершение игровой сессии возможна ситуация, при которой игрок получает смертельный урон, что приводит к уничтожению объекта игрока. Однако предусмотрен и сценарий **возрождения**, позволяющий игроку вернуться в игровой процесс.

Таким образом, диаграмма демонстрирует все основные типы взаимодействий в системе: боевые действия, квестовые события, торговлю, управление предметами и навыками, а также события, связанные с жизненным циклом игрока. Это позволяет получить целостное представление о логике взаимодействия между объектами и последовательности вызова операций в течение одной игровой сессии.

### 2.2. Требования к игре и ее наполнению

Стилистическое оформление игры должно соответствовать жанру RPG с элементами. Визуальный стиль предполагает верхнюю перспективу с четкими и яркими визуальными элементами, создающими атмосферу фэнтези. Цветовая гамма должна быть сбалансированной, сочетая яркие и насыщенные тона для интерфейса и эффектов, в зависимости от ситуации на карте. Интерфейс должен быть минималистичным, интуитивно понятным и не перегруженным, с удобным расположением элементов управления и информации о персонаже. Элементы интерфейса, такие как иконки и счетчики, должны быть легко читаемы и подчеркивать атмосферу игры.

Игровой интерфейс будет выполнен в средневековом стиле с использованием текстур, напоминающих дерево и камень. Панели меню будут иметь вид старинных деревянных табличек или досок с вырезанными узорами или вставными камнями. Кнопки будут выполнены в виде стилизованных деревянных или металлических элементов, с элементами старинной гравировки. Карта будет выполнена в пиксельном и ярком стиле, подчеркивающими атмосферу фэнтезийного мира, добавляющие визуальную глубину и эстетику старинного мира.

Требования к стилистическому оформлению включают строгое соответствие жанру фэнтезийной RPG. Все визуальные элементы должны создавать ощущение погружения в мир средневековых сражений, магии и героизма. Особое внимание уделяется деталям: визуальные эффекты, анимации и переходы между сценами должны быть плавными, атмосферными и дополнять художественную концепцию игры. Иллюстрации должны быть выполнены в едином художественном стиле с высокой детализацией.

Требования к графическому дизайну предполагают визуальную целостность всех элементов интерфейса и игровых объектов. Используемые текстуры (дерево, камень, металл, ткань) должны быть высококачественными и гармонично сочетаться друг с другом, не создавая визуального шума. Анимации интерфейса и игровых эффектов должны быть ненавязчивыми, но выразительными. Значки, иконки и визуальные индикаторы должны быть интуитивно понятными и легко различимыми при любом разрешении экрана.

Требования к шрифтовому оформлению предполагают один несколько шрифтов, которые сочетают читаемость с тематической выразительностью. Основной шрифт должен быть стилизован под средневековую антикву или каллиграфию, но без потери читабельности в динамике игры. Заголовки и названия могут использовать более декоративный стиль, в то время как игровой текст и интерфейсные надписи должны оставаться чёткими даже при уменьшении масштаба.

Требования к формату экрана и его адаптивности включают поддержку широкого диапазона разрешений и соотношений сторон, включая 16:9, 16:10 и 21:9. Интерфейс должен масштабироваться и перестраиваться без искажений, сохраняя читаемость текста, правильное позиционирование элементов и эстетическую целостность. Важно предусмотреть гибкую систему адаптивной верстки, чтобы обеспечить комфортный пользовательский опыт на различных устройствах и экранах.

### 2.3. Структура игры

Игровая сцена является стартовой сценой, где игрок получает доступ к основным функциям. Интерфейсы реализуются с помощью системы Canvas в Unity. Элементы меню включают логотип игры, кнопки и фоновое изображение, создающее атмосферу. Кнопки стилизованы в фэнтезийной тематике: они напоминают старинные таблички, а шрифт имитирует рукописный стиль. Фоновая палитра выполнена в тёплых и приглушённых тонах, создавая ощущение уюта и таинственности. Вся визуализация реализована с помощью UI-компонентов Unity (Image, Text, Button, Panel и т.д.) и встроенной системы навигации между сценами (SceneManager).

Дизайн всех сцен подчёркивает фэнтезийную тематику игры, оставаясь визуально лаконичным, чтобы не перегружать внимание игрока и сохранять комфортное восприятие интерфейса.

Структура проекта в Unity

Проект структурирован следующим образом:

* **scripts** - папка содержит скрипты на C#, управляющие логикой игры, сценами, интерфейсом и взаимодействием между компонентами;
* **scenes** - папка со сценами Unity, содержащая в себе сцены игры, меню и выбора колоды;
* **prefabs** - папка с префабами игровых объектов, таких как карты, панели UI, кнопки и т.д;
* **resources** - папка со статическими ресурсами: изображениями карт, иконками, звуками и фонами;
* tests - папка с игровыми тестами.

Граф логики переходов между сценами представлен на рисунке 2.2.

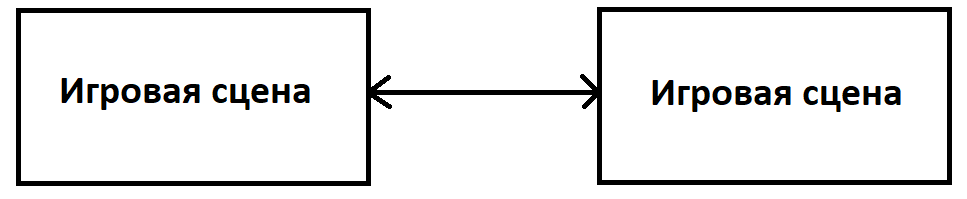


Рисунок 2.2 - Логическая структура переходов между сценами

### 2.4. Концептуальный прототип

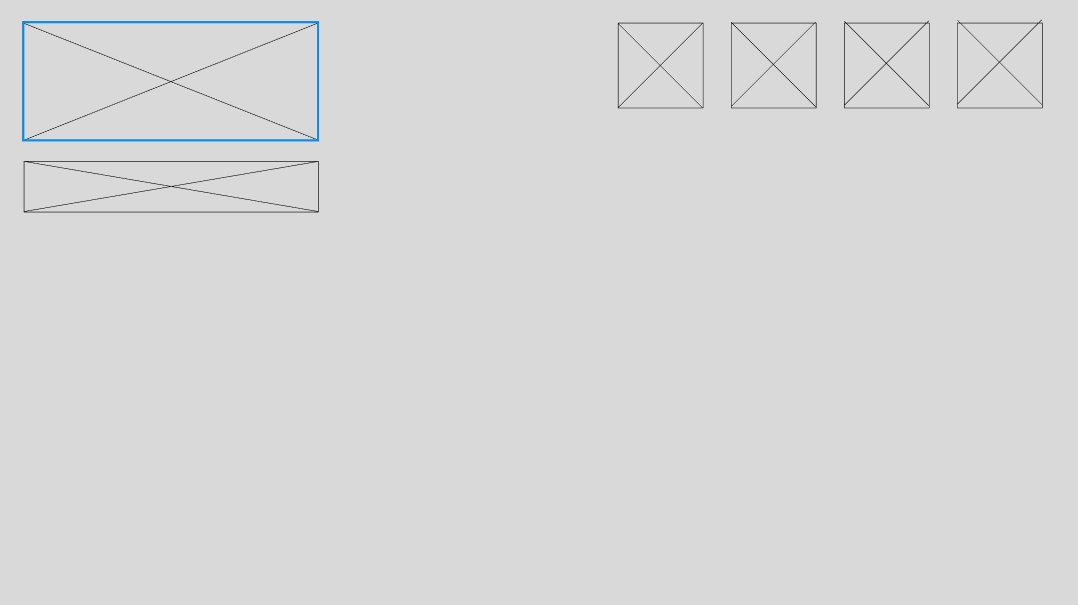
Прототип, изображенный на рисунке 2.3 представляет собой интерфейс основного окна игры. Главной задачей было создать удобный и визуально привлекательный экран, который сохраняет атмосферу средневековой стилистики и при этом прост в использовании. В левой верхней части экрана находятся полоски здоровья и опыта, отображающие шкалу здоровья и уровень игрока. В правой верхней части экрана находятся 4 слота инвентаря персонажа, где будут располагаться поднятые персонажем. Каждый предмет содержит свою иконку, которая отображается в инвентаре.

Рисунок 2.3 – прототип окна выбора колоды

Прототип магазина, изображённый на рисунке 2.4, представляет собой экран, выдержанный в средневековом стиле и сохраняющий общую атмосферу игры. В левой части экрана расположен портрет торговца - персонажа, с которым взаимодействует игрок. Рядом, в левом верхнем углу, как и на основном игровом экране, отображаются полосы здоровья и опыта персонажа, что позволяет игроку всегда видеть текущее состояние своего героя.

Центральная и правая часть экрана занимает большая панель с ассортиментом товаров, которые предлагает торговец. На этой панели расположены иконки предметов, каждая из которых сопровождается своим названием, расположенным непосредственно над иконкой. Это обеспечивает удобство выбора и позволяет быстро ориентироваться в доступных товарах.

В верхней части панели находятся три кнопки, позволяющие переключаться между разными категориями магазина - например, оружие, броня, зелья и другие типы предметов. Это упрощает навигацию и делает процесс покупки более организованным.

В правом верхнем углу экрана, как и в основном окне игры, присутствуют четыре слота инвентаря персонажа с иконками предметов, которые игрок уже поднял и может использовать. Это даёт возможность быстро оценить текущее снаряжение и принять решение о покупке или обмене.

В целом, интерфейс магазина сочетает в себе удобство использования и стилистическую выдержанность, используя элементы средневекового дизайна - деревянные рамки, текстуры, стилизованные шрифты и декоративные детали, что поддерживает целостность визуального образа игры и способствует погружению игрока в её мир.

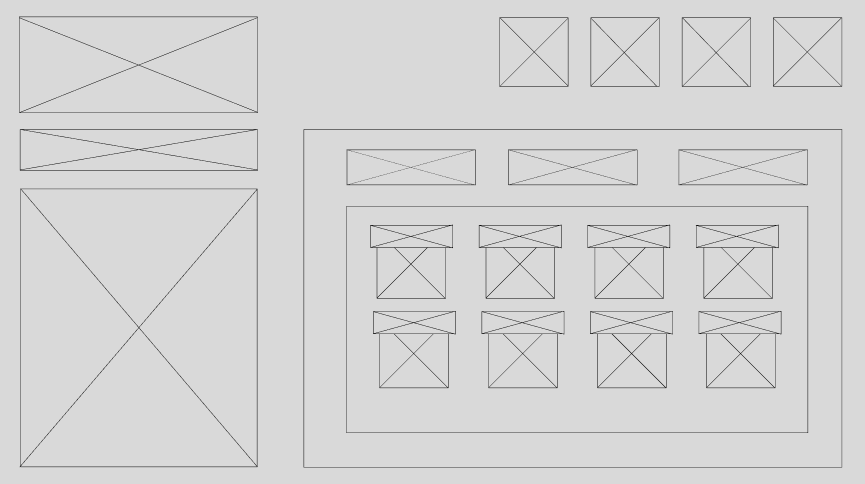


Рисунок 2.4 - прототип окна магазина

Прототип меню характеристики представлен на рисунке 2.5 и отражает удобный и информативный интерфейс для отображения текущих параметров персонажа. В правой части экрана расположена панель, которая частично перекрывает область инвентаря. На этой панели размещены текстовые поля с актуальной статистикой и характеристиками пользователя — такими как здоровье, уровень, сила, ловкость, опыт и другие важные параметры, влияющие на игровой процесс.

Дизайн панели выдержан в общей стилистике игры, поддерживая атмосферу средневековья и визуальную целостность интерфейса. Такое расположение панели позволяет игроку одновременно видеть инвентарь и основные характеристики персонажа, обеспечивая быстрый доступ к важной информации без необходимости переключаться между окнами. Это повышает удобство использования и способствует более эффективному управлению развитием героя.

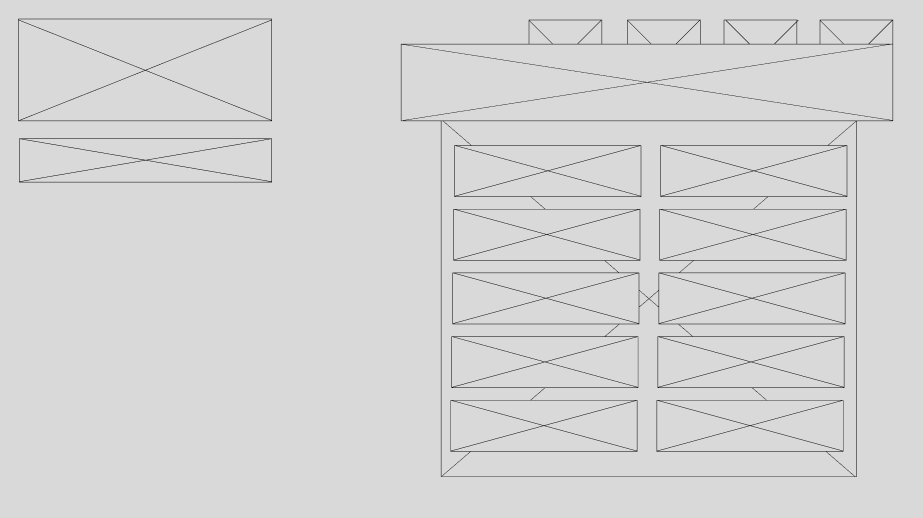


Рисунок 2.5 – прототип окна меню характеристик

Прототип меню скиллов представлен на рисунке 2.6 и отражает удобный и информативный интерфейс для отображения доступных и закрытых от глаза пользователя набора способностей. В правой части экрана расположена панель, которая частично перекрывает область инвентаря. На этой панели размещены кнопки, позволяющие улучшать способности персонажа.

Дизайн панели выдержан в общей стилистике игры, поддерживая атмосферу средневековья и визуальную целостность интерфейса. Такое расположение панели позволяет игроку одновременно видеть инвентарь и основные характеристики персонажа, обеспечивая быстрый доступ к важной информации без необходимости переключаться между окнами. Это повышает удобство использования и способствует более эффективному управлению развитием героя.

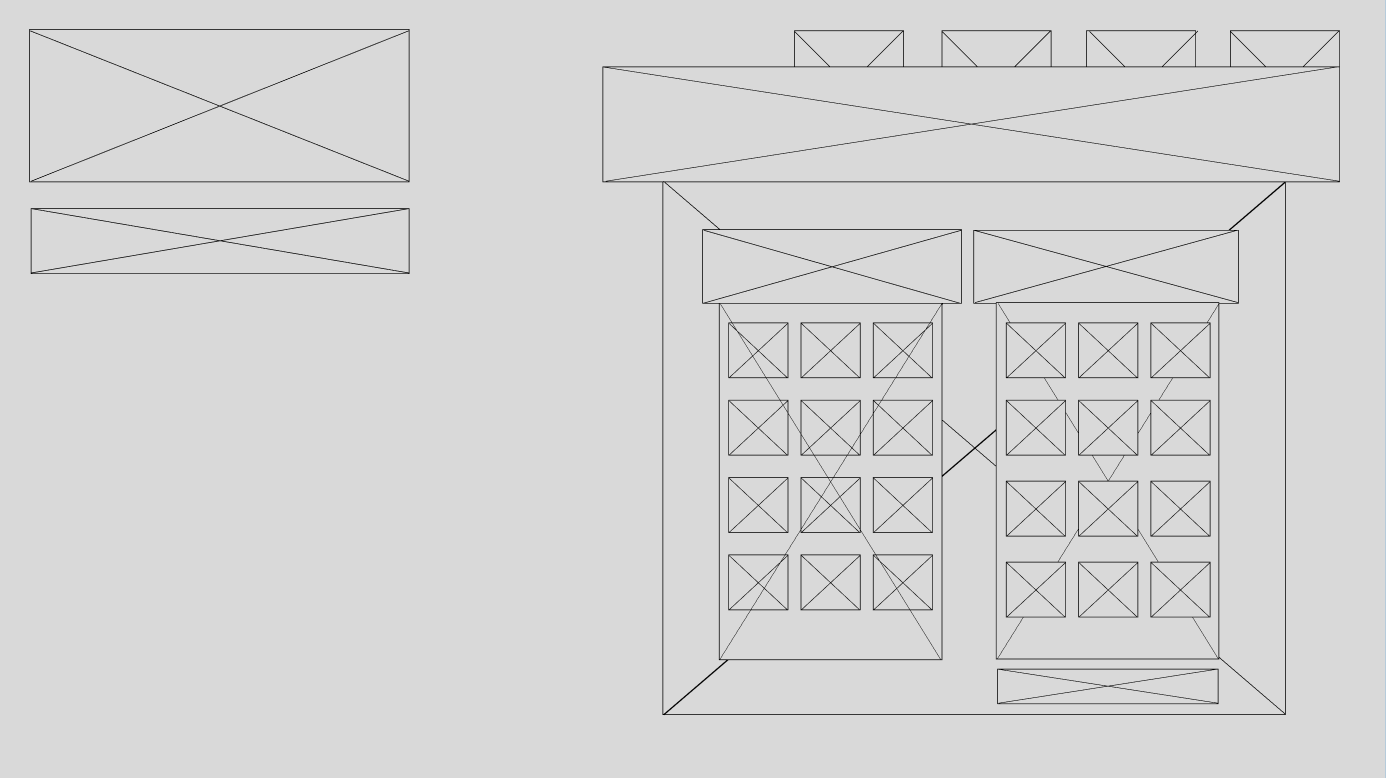


Рисунок 2.6 -прототип окна скиллов

### 2.5. Организация данных

Физическая структура дерева каталогов игры на Unity организована таким образом, чтобы обеспечить удобное управление ресурсами, кодом и игровыми данными. В корневой папке проекта (Assets) располагаются ключевые директории:

* **scripts** - содержит скрипты на C#, управляющие логикой игры, обработкой сцен, интерфейсом пользователя и взаимодействием между различными игровыми компонентами. Здесь реализованы основные игровые модули: карты, колоды, фракции и способности;
* **scenes** - папка со сценами Unity;
* **prefabs** - хранит префабы игровых объектов: карты, панели интерфейса, кнопки и другие элементы, которые можно переиспользовать в разных частях игры;
* **resources** - содержит статические ресурсы: изображения карт, иконки, звуки и фоны, необходимые для визуализации и звукового оформления игры;
* **tests** – содержит функции для тестирования в режиме редактирования.

Каждая из этих директорий отвечает за определённую область функциональности и взаимодействует с другими компонентами, что обеспечивает стабильную и эффективную работу игры. Такая организация позволяет легко масштабировать проект, добавлять новые возможности и поддерживать его в будущем.

Модуль Inventory&Shop реализует логику управления инвентарём и торговыми операциями. Он отвечает за хранение, использование, перемещение и удаление предметов игрока, а также за взаимодействие с торговыми системами, представленными NPC. Модуль обеспечивает корректную обработку покупок и продаж, проверку условий сделок, а также обновление визуального и логического состояния инвентаря после каждой операции. Его взаимодействие с НИП отражено в сценарии торговли, где происходит подтверждение сделки и изменение содержимого инвентаря.

МодульNPCScripts управляет поведением неигровых персонажей (НИП), включая взаимодействие с игроком, выдачу заданий, обработку диалогов и предоставление наград. Он реализует механику общения, контроля активных квестов, условий их выполнения и завершения. На диаграмме последовательности отражены все ключевые этапы взаимодействия: инициирование диалога, передача задания, принятие выполненного задания и выдача награды. Модуль также может обрабатывать торговые предложения и сопровождать игрока в сюжетных ветках.

Модуль PlayerScripts отвечает за обработку всех пользовательских действий: атаки, получение урона, взаимодействие с объектами, управление статистикой и смерть персонажа. В рамках диаграммы этот модуль реализует действия по атаке врагов, получению урона, использованию предметов, управлению сессиями и настройками, а также обработку смерти и возрождения игрока. Он играет центральную роль в игровых взаимодействиях, обеспечивая корректную реакцию системы на действия пользователя.

Модуль SkillTree реализует систему развития персонажа, включая прокачку навыков и распределение очков умений. Он управляет данными об активных и доступных навыках, проверяет условия улучшений и обновляет способности игрока после прокачки. В диаграмме последовательности модуль проявляется в момент прокачки, где игрок самостоятельно активирует развитие навыков. Этот модуль тесно связан с механиками прогресса и персонализации, позволяя формировать уникальный стиль игры.

### 3. Реализация игры

### 3.1. Инструменты разработки

Вид и характеристика операционной системы будут предусматривать основную поддержку Windows 10/11 (64-bit), так как данная платформа является наиболее популярной среди пользователей и оптимальна для разработки в среде Unity. В дальнейшем будет рассматриваться возможность адаптации игры под macOS и Linux, однако на начальном этапе основной упор будет сделан на стабильную работу на Windows. Игра будет оптимизирована для средних и высоких конфигураций ПК с графической поддержкой на базе DirectX 10. Хранение данных будет реализовано в локальной папке пользователя, где будут сохраняться игровые настройки и прогресс. Игрокам будет предоставлен выбор разрешений экрана и режимов отображения - полноэкранного или оконного. При расширении аудитории возможен порт на другие платформы, включая macOS и SteamOS.

Для разработки игры будут использоваться следующие инструменты и языки программирования: основной игровой движок - Unity 6000.0.37f1, обеспечивающий удобство создания 2D-игр, а также Visual Studio 2022 как основная среда разработки. Программная логика будет писаться на языке C#, что обеспечит гибкость, удобную работу с объектами и интеграцию с Unity. Для контроля версий будет применяться Git, а управление ассетами будет осуществляться с помощью встроенных инструментов Unity Asset Management.

В качестве операционной системы для разработки будет использоваться Windows, так как она предоставляет необходимую совместимость и поддержку всех ключевых инструментов. Язык программирования C# будет выбран за его тесную интеграцию с Unity и возможности объектно-ориентированного программирования. Unity будет использоваться как основной движок из-за развитых инструментов анимации, физики и поддержки кроссплатформенной компиляции. Visual Studio 2022 будет обеспечивать удобную работу с кодом, отладкой и интеграцией с Unity. Git будет применяться для отслеживания версий проекта и командной работы при необходимости.

Сетевая поддержка в игре не будет предусмотрена, поскольку проект ориентирован на одиночное прохождение с акцентом на бесконечный геймплей.

### 3.2. Реализация сценария

Реализация сценария в игре, представляет собой сложную систему, которая связывает ключевые игровые события с динамическими реакциями персонажей и мира, создавая живое и погружающее повествование. В основе такой системы лежит набор триггеров - условий, при наступлении которых запускаются соответствующие сценарные элементы, включая изменения в игровом процессе. При начале партии сценарий задаёт атмосферу и тон, активируя вступительные элементы, которые помогают игроку погрузиться в сюжет и понять контекст происходящего. В течение игры каждое действие игрока и противника, будь то разыгрывание карты, использование способности лидера или завершение раунда, сопровождается вызовом событий, которые запускают соответствующие реакции.

Система сценария интегрируется с игровым кодом через механизмы событий и слушателей, которые постоянно отслеживают состояние игры и реагируют на изменения, обеспечивая плавный и логичный переход между этапами. Это позволяет не только поддерживать последовательность сюжета, но и учитывать выборы и действия игрока, влияя на развитие истории и создавая эффект нелинейности. Например, успешное применение определённой карты может вызвать уникальную реакцию персонажа. Благодаря такой гибкости сценарий становится неотъемлемой частью геймплея, придавая глубину и эмоциональную насыщенность даже в рамках карточной механики.

Каждое событие связано с конкретным игровым состоянием или выбором, позволяя создавать богатые и разнообразные сценарные ветвления. В итоге, такая система обеспечивает живой и адаптивный игровой геймплей, который органично вплетается в игровой процесс, делая каждую партию уникальной и наполненной смыслом.

### 3.3. Реализация поведения объектов

Реализация поведения объектов в игре «Люмен» основывается на описании свойств игровых элементов - карт, рядов, персонажей - и закреплении за ними функций, которые управляют их состояниями и взаимодействиями в процессе игры. Каждый объект обладает набором характеристик, например, у карт - сила, способность, принадлежность к ряду и фракции, а у рядов - тип (ближний, дальний, осадный) и текущее состояние (подсвечен, занят картой и т.д.). Эти свойства реализованы в виде полей и компонентов в соответствующих классах, таких как CardStats, CardDisplay, RowManager и других.

Функции, управляющие поведением объектов, связаны с элементами управления и игровыми событиями. Например, метод OnPointerClick в классе CardBtn реагирует на клики мышью по карте, инициируя выбор карты или её разыгрывание, а в RowClick реализована логика выбора ряда при клике. В DeckController и SceneController сосредоточена основная игровая логика - проверка состояния хода, размещение карт на поле, обновление состояния игры. Взаимодействие между функциями происходит через вызовы методов и передачу параметров, обеспечивая последовательное выполнение действий: выбор карты, подсветка возможных ходов, подтверждение хода, обновление интерфейса и игрового состояния.

Например, при клике на карту вызывается метод, который устанавливает её как выбранную, подсвечивает допустимые ряды для разыгрывания, а при подтверждении хода вызывается функция размещения карты на выбранном ряду с проверками состояния игры. Аналогично,

Таким образом, поведение объектов реализовано через совокупность классов и методов, тесно связанных с элементами управления, которые инициируют выполнение функций. В приложении к проекту приводится полный текст всех модулей с комментариями, где указаны связи между элементами управления и функциями, что позволяет понять и отследить логику взаимодействия объектов в игровом процессе.

### 3.4. Функциональное тестирование

Функциональное тестирование представляет собой проверку работоспособности функций приложения на соответствие заданным требованиям. Оно охватывает ключевые сценарии использования системы и анализирует реакцию мобильного приложения на различные типы входных данных. Цель тестирования - обнаружить сбои и недостатки на этапе подготовки программы к запуску. Этот процесс позволяет выявить возможные ошибки, которые будут устранены до передачи конечного продукта пользователю.

В таблицах с 1-6 представлены тест-кейсы для проведения функционального тестирования.

В таблице 1 представлен тест-кейс для тестирования функции получения урона игроком.

Таблица 1 – Тест-кейс для получения урона игроком

|  |  |  |  |
| --- | --- | --- | --- |
| № | Модуль / Функция | Шаги воспроизведения | Результат |
| 1 | Получение урона игроком | * + 1. Игрок подходит к врагу     2. Враг атакует игрока | Здоровье игрока уменьшается на величину урона |
| Фактический:  Здоровье уменьшилось  Пример результата получения урона игроком представлен на рисунке Б.1 |

В таблице 2 представлен тест-кейс для тестирования функции расчёта победы по сумме силы карт. Предварительно пользователь должен авторизоваться и выиграть 2 раунда.

Таблица 2 – Тест-кейс для проверки отсутствия лечения при атаке врага

|  |  |  |  |
| --- | --- | --- | --- |
| № | Модуль / Функция | Шаги воспроизведения | Результат |
| 1 | Проверка отсутствия лечения при атаке врага | * + 1. Игрок подходит к врагу     2. Враг атакует игрока | Ожидаемый:  Здоровье игрока не увеличивается после атаки врага |
| Фактический:  Здоровье понизилось, т.е. не увеличилось  Пример проверки отсутствия лечения при атаке врага представлен на рисунке Б.2 |

В таблице 3 представлен тест-кейс для тестирования анимации атаки пользователя.

Таблица 3 – Тест-кейс для проверки анимации атаки

|  |  |  |  |
| --- | --- | --- | --- |
| № | Модуль / Функция | Шаги воспроизведения | Результат |
| 1 | Проверка анимации атаки | * + 1. Игрок нажимает кнопку атаки | Ожидаемый:  Анимация атаки запускается, таймер перезагружается |
| Фактический:  Анимация и таймер обновлены  Пример результата проверки анимации атаки представлен на рисунке Б.3 |

В таблице 4 представлен тест-кейс для проверки сброса анимации атаки.

Таблица 4 – Тест-кейс для проверки сброса анимации атаки

|  |  |  |  |
| --- | --- | --- | --- |
| № | Модуль / Функция | Шаги воспроизведения | Результат |
| 1 | Завершение анимации атаки | * + 1. Игрок нажимает кнопку атаки     2. Анимация проигрывается | Ожидаемый:  Анимация атаки останавливается |
| Фактический:  Анимация атаки останавливается  Пример результата остановки атаки представлен на рисунке Б.4 |

В таблице 5 представлен тест-кейс для обновления текста здоровья игрока.

Таблица 5 – Тест-кейс для обновления здоровья игрока

|  |  |  |  |
| --- | --- | --- | --- |
| № | Модуль / Функция | Шаги воспроизведения | Результат |
| 1 | Обновления текста здоровья | 1. Игрок получает урон | Ожидаемый:  Текст здоровья обновляется и корректно отображает текущее здоровье |
| Фактический:  Текст обновлен корректно  Пример результата счетчика победы представлен на рисунке Б.5 |

В таблице 6 представлен тест-кейс для тестирования смерти игрока при нулевом здоровье.

Таблица 6 – Тест-кейс для тестирования смерти игрока при нулевом здоровье

|  |  |  |  |
| --- | --- | --- | --- |
| № | Модуль / Функция | Шаги воспроизведения | Результат |
| 1 | Превышение лимита карт в колоде | * + 1. Игрок получает урон, снижающий здоровье до 0 или ниже | Ожидаемый:  Активируется событие смерти игрока |
| Фактический:  Событие смерти успешно вызвано  Пример результата превышение лимита карт представлен на рисунке Б.6 |

### 4. Применение

### 4.1. Общие сведения

Цель разработки игры - создать увлекательное одиночное приключение с интересным сюжетом, интерактивными диалогами и системой линейных квестов. Игроку предстоит исследовать мир, взаимодействовать с NPC, сражаться с врагами и развивать персонажа.

Для функционирования игры требуется программное обеспечение: операционная система Windows 8/10/11, установщик. Аппаратные требования включают процессор не ниже AMD Phenom, 512 МБ оперативной памяти и видеокарту с поддержкой DirectX 10.

Процесс установки игры заключается в скачивании и распаковке дистрибутива, после чего пользователь может запустить исполняемый файл. Запуск игры осуществляется через ярлык на рабочем столе или вручную через файловый менеджер.

В базовых настройках игрок сможет изменить громкость звуков, разрешение экрана, режим отображения (полноэкранный или оконный).

Интернет-подключение не требуется, так как игра предназначена для офлайн-прохождения без сетевых функций.

### 4.2. Функциональное назначение

Игра «Cool Driver» предназначена для одиночного прохождения и предлагает пользователю увлекательное погружение в атмосферный фэнтезийный мир, где ключевую роль играют тактические сражения и выполнение заданий различной сложности. Игроку предстоит взять на себя роль спасителя деревни, вступающего в серию испытаний, турниров и сюжетных поединков.

В процессе игры пользователь сможет участвовать в поединках, где исход зависит не только от силы снаряжения, но и от его прыткости.

Жанр игры можно охарактеризовать как RPG. Геймплей реализован в двухмерном пространстве с интуитивно понятным интерфейсом, ориентированным как на новичков, так и на опытных поклонников RPG. Особое внимание уделено атмосфере и глубине механик: игроку предлагается не только сражаться, но и совершенствовать свою колоду и открывать новые сюжетные ветки.

Разработчики сосредоточили усилия на создании насыщенного тактического опыта, в котором важны как грамотный сбор снаряжения, так и принятие решений в ходе каждой битвы. Благодаря этому игра «Cool Driver» способна заинтересовать широкий круг пользователей, предлагая уникальное сочетание RPG, сюжетной глубины и фэнтезийной атмосферы.

Потенциальную аудиторию проекта составляют любители одиночных игр, особенно те, кто предпочитает играть в RPG. Основная возрастная категория - подростки от 12 лет и взрослые игроки, которым близка атмосфера подобных игр, минимализм в управлении и увлекательное повествование без необходимости подключаться к интернету или взаимодействовать с другими пользователями в реальном времени. Игра также будет интересна поклонникам ретро-стилистики, инди-проектов и 2D-графики.

На данный момент проект разрабатывается под операционную систему Windows (версии 7, 10, 11 - 64-bit), что обусловлено удобством работы с игровым движком Unity и распространённостью данной платформы среди пользователей. Кроссплатформенность на данном этапе не реализуется, однако в дальнейшем, при наличии интереса аудитории и технической возможности, планируется адаптация игры под другие операционные системы, включая macOS и Linux. Также рассматривается перспектива расширения на мобильные устройства и консоли, если игра получит положительный отклик. Таким образом, функциональное назначение игры заключается в предоставлении пользователю качественного одиночного игрового опыта, построенного на интересном сюжете, проработанных механиках и доступной реализации.

### 4.3. Условия применения

Для корректной работы игры «Люмен» потребуется устройство с достаточной производительностью и стабильным подключением к интернету. Минимальные системные требования включают операционную систему Windows 7/8/10 (64-bit), процессор не ниже Intel Celeron G1820 или AMD A4-7300, 4 ГБ оперативной памяти и видеокарту уровня NVIDIA GeForce GT 730 либо её аналог.

В качестве основной задачи при разработке игры ставилось создание полноценного цифрового карточного симулятора с уникальной механикой, сбалансированными фракциями и возможностью онлайн-сражений между игроками. Эта задача была успешно реализована: игра разработана на движке Unity, обеспечивающем кроссплатформенную совместимость и высокую производительность. Были реализованы все ключевые функции, включая выбор фракций, сбор колод, пошаговую механику боёв, рейтинговые и тренировочные матчи, а также визуальные и звуковые эффекты. Преимуществами игры являются увлекательный геймплей, разнообразие стратегий, визуальная проработка карт и стабильная работа в сетевом режиме. Среди недостатков можно отметить высокую зависимость от интернет-соединения и необходимость постоянных обновлений для поддержания баланса и актуальности игрового процесса.

### Заключение

В ходе разработки игры **«**Cool Driver**»** была успешно реализована одиночная 2D-RPG игра с элементами стратегии, вдохновлённая механиками популярной игры The Legend of Zelda. Проект выполнен на игровом движке Unity с использованием языка C# в среде Visual Studio 2022.

В процессе разработки была создана и интегрирована ключевая игровая механика для RPG-игры, включающая управление персонажем, систему боя с использованием различных умений и тактическое разделение на ближний и дальний бой. Особое внимание уделено уникальным навыкам персонажей, способным существенно менять ход боя и влиять на исход сражений.

Реализована система прокачки персонажа с возможностью выбора специализации и развития навыков, что расширяет тактические возможности игрока и позволяет создавать разнообразные стили игры. Для удобства пользователя создан простой и интуитивно понятный интерфейс с меню и быстрым доступом к основным функциям.

Особенностью проекта является продуманная боевая система, разнообразие стратегий за счёт различных классов и умений, а также комфортное управление и качественная визуализация, что делает игру доступной для широкой аудитории. Игра удачно сочетает глубину тактических решений с лёгкостью освоения.

Тем не менее, в текущей версии отсутствует многопользовательский режим, что ограничивает взаимодействие между игроками. Также планируется дальнейшая оптимизация искусственного интеллекта врагов и расширение контента для повышения интереса и реиграбельности.

В целом, разработка RPG-игры продемонстрировала успешную реализацию комплексной боевой и прокачной системы в 2D-формате, обеспечив стабильную и увлекательную игровую платформу. Полученный опыт создаёт прочную основу для дальнейшего развития проекта с добавлением новых функций, режимов и улучшений.

Результаты курсовой работы полностью соответствуют поставленным задачам, система успешно реализована и протестирована, а полученные навыки и знания могут быть использованы в дальнейшем обучении и профессиональной деятельности.

### Список использованных источников

1. Багласова, Т.Г., Якимович, К.О. Методические указания по оформлению курсовых проектов, дипломных проектов и отчетов для учащихся специальности 2-40 01 01 «Программное обеспечение информационных технологий» / Т.Г. Багласова, К.О. Якимович. – Минск : КБП, 2022. – 41 с.
2. Михалевич, В.Ю. Методические указания к курсовому проектированию по Конструированию программ и языки программирования для обучающихся специальности 2-40 01 01 «Программное обеспечение информационных технологий» / В.Ю. Михалевич. – Минск : КБП, 2023. – 25 с.
3. Общие требования к тестовым документам : ГОСТ 2.105-95. – Введ. 01.01.1996. – Минск : Межгос. совет по стандартизации, метрологии и сертификации, 1995. – 84 с.
4. Программа и методика испытаний. Требования к содержанию, оформлению и контролю качества : ГОСТ 19.301-2000. – Введ. 01.09.2001. – Минск : Межгос. совет по стандартизации, метрологии и сертификации, 2000. – 14 с.
5. Текст программы. Требования к содержанию, оформлению и контролю качества : ГОСТ 19.401-2000. – Введ. 01.09.2001. – Минск : Межгос. совет по стандартизации, метрологии и сертификации, 2000. – 16 с.
6. Петров, А. И. C# и Unity: основы разработки 2D-игр / А. И. Петров. – Москва : ДМК Пресс, 2024. – 256 с.
7. Microsoft. Руководство по программированию на C# [Электронный ресурс]. – Microsoft.com, 2025. – Режим доступа: <https://learn.microsoft.com/ru-ru/dotnet/csharp/programming-guide/>. – Дата доступа: 09.04.2025.
8. METANIT.COM. .NET и C# | Полное руководство [Электронный ресурс]. – METANIT.COM, 2025. – Режим доступа: <https://metanit.com/sharp/>. – Дата доступа: 11.04.2025.
9. O'Reilly Media. Learning C# by Developing Games with Unity 2025: A Beginner's Guide / O'Reilly Media. – Sebastopol : O'Reilly Media, 2025. – 352 p.
10. Roberts, M. Unity 2023 Game Development in 24 Hours: Learn Game Development Using Unity, C#, and the Unity Game Engine / M. Roberts. – New York : Sams Publishing, 2025. – 512 p.

### ПРИЛОЖЕНИЕ А Текст программных модулей

Файл EnemyMovement.cs

public class EnemyMovement : MonoBehaviour

{

// Скорость перемещения врага

public float speed;

// Дистанция, на которой враг может атаковать игрока

public float AttackRange = 2f;

// Время перезарядки между атаками

public float attackCooldown = 2f;

// Радиус обнаружения игрока

public float playerDetectRange = 5f;

// Точка, откуда происходит проверка обнаружения игрока

public Transform detectionPoint;

// Слой, на котором находится игрок

public LayerMask playerLayer;

// Таймер перезарядки атаки

private float attackCooldownTimer;

// Направление, в которое смотрит враг (1 — вправо, -1 — влево)

private int facingDirection = 1;

// Текущее состояние врага (Idle, Chasing, Attacking, Knockback)

private EnemyState enemyState;

// Компонент Rigidbody2D для управления физикой врага

private Rigidbody2D rb;

// Ссылка на трансформ игрока, когда он обнаружен

private Transform player;

// Компонент Animator для управления анимациями врага

private Animator anim;

private void Start()

{

// Получаем компоненты Rigidbody2D и Animator на старте

rb = GetComponent<Rigidbody2D>();

anim = GetComponent<Animator>();

// Устанавливаем начальное состояние врага — ожидание (Idle)

ChangeState(EnemyState.Idle);

}

private void Update()

{

// Если враг не находится в состоянии отбрасывания (Knockback), выполняем логику

if (enemyState != EnemyState.Knockback)

{

// Проверяем наличие игрока в зоне обнаружения

CheckForPlayer();

// Обновляем таймер перезарядки атаки

if (attackCooldownTimer > 0)

{

attackCooldownTimer -= Time.deltaTime;

}

// В зависимости от состояния враг либо преследует игрока, либо стоит на месте при атаке

if (enemyState == EnemyState.Chasing)

{

Chase();

}

else if (enemyState == EnemyState.Attacking)

{

// Останавливаем движение во время атаки

#pragma warning disable CS0618 // Используется устаревший член, но для совместимости оставлен

rb.velocity = Vector3.zero;

#pragma warning restore CS0618

}

}

}

// Метод преследования игрока

void Chase()

{

// Если игрок находится слева, а враг смотрит вправо, или наоборот — меняем направление взгляда

if (player.position.x < transform.position.x && facingDirection == 1 || player.position.x > transform.position.x && facingDirection == -1)

{

Flip();

}

// Вычисляем направление к игроку и нормализуем вектор

Vector2 direction = (player.position - transform.position).normalized;

// Задаём скорость врага в направлении игрока

#pragma warning disable CS0618

rb.velocity = direction \* speed;

#pragma warning restore CS0618

}

// Метод для смены направления взгляда врага

private void Flip()

{

// Меняем знак направления

facingDirection \*= -1;

// Отзеркаливаем спрайт по оси X

transform.localScale = new Vector3(transform.localScale.x \* -1, transform.localScale.y, transform.localScale.z);

}

// Метод проверки наличия игрока в зоне обнаружения

private void CheckForPlayer()

{

// Получаем все коллайдеры игрока в радиусе обнаружения

Collider2D[] hits = Physics2D.OverlapCircleAll(detectionPoint.position, playerDetectRange, playerLayer);

if (hits.Length > 0)

{

// Если игрок найден, запоминаем его трансформ

player = hits[0].transform;

// Если игрок в зоне атаки и атака готова — меняем состояние на атаку

if (Vector2.Distance(transform.position, player.position) <= AttackRange && attackCooldownTimer <= 0)

{

attackCooldownTimer = attackCooldown;

ChangeState(EnemyState.Attacking);

}

// Если игрок вне зоны атаки и враг не атакует — преследуем игрока

else if (Vector2.Distance(transform.position, player.position) > AttackRange && enemyState != EnemyState.Attacking)

{

ChangeState(EnemyState.Chasing);

}

}

else

{

// Если игрок не обнаружен — останавливаем движение и переходим в состояние ожидания

#pragma warning disable CS0618

rb.velocity = Vector2.zero;

#pragma warning restore CS0618

ChangeState(EnemyState.Idle);

}

}

// Метод смены состояния врага и управления анимациями

public void ChangeState(EnemyState newState)

{

// Сбрасываем текущие анимационные флаги в зависимости от предыдущего состояния

if (enemyState == EnemyState.Idle)

{

anim.SetBool("isIdle", false);

}

else if (enemyState == EnemyState.Chasing)

{

anim.SetBool("isChasing", false);

}

else if (enemyState == EnemyState.Attacking)

{

anim.SetBool("isAttacking", false);

}

// Обновляем текущее состояние

enemyState = newState;

// Включаем анимацию, соответствующую новому состоянию

if (enemyState == EnemyState.Idle)

{

anim.SetBool("isIdle", true);

}

else if (enemyState == EnemyState.Chasing)

{

anim.SetBool("isChasing", true);

}

else if (enemyState == EnemyState.Attacking)

{

anim.SetBool("isAttacking", true);

}

}

// Визуализация радиуса обнаружения игрока в редакторе

private void OnDrawGizmosSelected()

{

Gizmos.color = Color.red;

Gizmos.DrawWireSphere(detectionPoint.position, playerDetectRange);

}

}

// Перечисление состояний врага для удобства работы с логикой и анимациями

public enum EnemyState

{

Idle, // Ожидание

Chasing, // Преследование игрока

Attacking, // Атака игрока

Knockback, // Отбрасывание (например, после удара игрока)

}

Файл QuestNPC.cs

public class QuestNPC : MonoBehaviour

{

// Перечисление состояний текущего квеста

public enum QuestState

{

Mushrooms, // Квест на сбор грибов

Boss, // Квест на убийство босса

Completed // Все квесты выполнены

}

// Текущее состояние квеста у NPC

public QuestState currentQuest = QuestState.Mushrooms;

[Header("Mushroom Quest")]

// Диалоговые строки для квеста с грибами

public string mushroomsDialogueBefore = "Bring me 3 mushrooms."; // До выполнения квеста

public string mushroomsDialogueNotEnough = "You don't have enough mushrooms."; // Если недостаточно грибов

public string mushroomsDialogueComplete = "Thank you for the mushrooms!"; // После сдачи квеста

public ItemSO mushroomsRequiredItem; // Требуемый предмет (грибы)

public int mushroomsRequiredAmount = 3; // Количество необходимых грибов

[Header("Boss Quest")]

// Диалоговые строки для квеста с боссом

public string bossDialogueBefore = "Kill the boss at the end of this location."; // До убийства босса

public string bossDialogueNotKilled = "You haven't killed the boss yet!"; // Если босс не убит

public string bossDialogueComplete = "Great job! You killed the boss!"; // После убийства босса

public string bossTag = "Enemy"; // Тег объекта босса

private bool bossKilled = false; // Флаг, отмечающий убийство босса

// Флаг, указывающий, находится ли игрок в зоне взаимодействия с NPC

private bool isPlayerInRange = false;

// Ссылка на менеджер отображения диалогов

public DialogueUIManager dialogueUIManager;

private void Start()

{

// Находим объект босса по тегу и подписываемся на событие его смерти

GameObject boss = GameObject.FindGameObjectWithTag(bossTag);

if (boss != null && boss.GetComponent<EnemyHealth>() != null)

{

boss.GetComponent<EnemyHealth>().OnDeath += OnBossDeath;

}

// Инициализируем менеджер диалогов с этим NPC

dialogueUIManager.Initialize(this);

}

private void Update()

{

// Если игрок находится рядом и нажимает клавишу F — показываем диалог

if (isPlayerInRange && Input.GetKeyDown(KeyCode.F))

{

ShowDialogueOptions();

}

}

// Отобразить текущий диалог в UI

private void ShowDialogueOptions()

{

string dialogueText = GetCurrentDialogueText();

dialogueUIManager.ShowDialogue(dialogueText);

}

// Получить текст диалога в зависимости от текущего состояния квеста

private string GetCurrentDialogueText()

{

switch (currentQuest)

{

case QuestState.Mushrooms: return mushroomsDialogueBefore;

case QuestState.Boss: return bossDialogueBefore;

case QuestState.Completed: return "All quests completed!";

default: return "Error: Unknown quest state";

}

}

// Обработка входа игрока в зону взаимодействия

private void OnTriggerEnter2D(Collider2D collision)

{

if (collision.CompareTag("Player"))

{

isPlayerInRange = true;

}

}

// Обработка выхода игрока из зоны взаимодействия

private void OnTriggerExit2D(Collider2D collision)

{

if (collision.CompareTag("Player"))

{

isPlayerInRange = false;

dialogueUIManager.HideDialogue();

}

}

// Метод взаимодействия игрока с NPC (вызывается извне)

public void Interact()

{

switch (currentQuest)

{

case QuestState.Mushrooms: HandleMushroomQuest(); break;

case QuestState.Boss: HandleBossQuest(); break;

case QuestState.Completed: FinalDialogue(); break;

}

}

// Обработка квеста на сбор грибов

private void HandleMushroomQuest()

{

// Получаем количество грибов в инвентаре игрока

int count = InventoryManager.Instance.GetItemCount(mushroomsRequiredItem);

if (count >= mushroomsRequiredAmount)

{

// Если достаточно грибов — удаляем их из инвентаря, выдаём награду и переключаем квест на убийство босса

InventoryManager.Instance.RemoveItem(mushroomsRequiredItem, mushroomsRequiredAmount);

InventoryManager.Instance.gold += 10;

InventoryManager.Instance.goldText.text = InventoryManager.Instance.gold.ToString();

currentQuest = QuestState.Boss;

dialogueUIManager.ShowDialogue(mushroomsDialogueComplete + "\n" + bossDialogueBefore);

}

else

{

// Если грибов недостаточно — показываем соответствующий диалог

dialogueUIManager.ShowDialogue(count > 0 ? mushroomsDialogueNotEnough : mushroomsDialogueBefore);

}

}

// Обработка квеста на убийство босса

private void HandleBossQuest()

{

if (bossKilled)

{

// Если босс убит — выдаём награду и отмечаем квест как завершённый

currentQuest = QuestState.Completed;

InventoryManager.Instance.gold += 50;

InventoryManager.Instance.goldText.text = InventoryManager.Instance.gold.ToString();

dialogueUIManager.ShowDialogue(bossDialogueComplete);

}

else

{

// Если босс ещё жив — информируем игрока

dialogueUIManager.ShowDialogue(bossDialogueNotKilled);

}

}

// Диалог после завершения всех квестов

private void FinalDialogue()

{

dialogueUIManager.ShowDialogue("Come back later!");

}

// Обработчик события смерти босса — устанавливает флаг убитого босса

private void OnBossDeath()

{

bossKilled = true;

}

}

Файл SettingsManager.cs

public class SettingsManager : MonoBehaviour

{

// Ссылка на аудиомикшер для управления громкостью

public AudioMixer audioMixer;

// Dropdown для выбора разрешения экрана

public TMPro.TMP\_Dropdown resolutionDropdown;

// Объект с фоновым музыкальным сопровождением

public GameObject BGMusic;

// Массив объектов с тегом "Sound" для определения наличия звука в сцене

public GameObject[] objs;

// CanvasGroup для управления видимостью и взаимодействием с меню настроек

public CanvasGroup settingsCanvas;

// Слайдер для регулировки громкости (устанавливается через инспектор)

[SerializeField] private Slider volumeSlider;

// Флаг, указывающий, открыто ли меню настроек

private bool settingsOpen = false;

// Компонент AudioSource, связанный с этим объектом

private AudioSource audiosrc;

// Массив доступных разрешений экрана

private Resolution[] resolutions;

private void Awake()

{

// Находим все объекты с тегом "Sound" в сцене

objs = GameObject.FindGameObjectsWithTag("Sound");

if (objs.Length > 0 )

{

// Если объекты звука есть, создаём экземпляр фоновой музыки и делаем её неуничтожаемой при загрузке сцен

BGMusic = Instantiate(BGMusic);

BGMusic.name = "BGMusic";

DontDestroyOnLoad(BGMusic.gameObject);

}

else

{

// Если объекты звука отсутствуют, пытаемся найти уже существующий объект BGMusic

BGMusic = GameObject.Find("BGMusic");

}

}

private void Start()

{

// Получаем компонент AudioSource

audiosrc = GetComponent<AudioSource>();

// Загружаем сохранённое значение громкости, если есть, иначе ставим 0.5

float savedVolume = PlayerPrefs.GetFloat("Volume", 0.5f);

volumeSlider.value = savedVolume;

SetVolume(savedVolume);

// Подписываемся на событие изменения значения слайдера громкости

volumeSlider.onValueChanged.AddListener(SetVolume);

// Получаем все доступные разрешения экрана

resolutions = Screen.resolutions;

// Очищаем текущие опции в dropdown

resolutionDropdown.ClearOptions();

var options = new List<string>();

int currentResolutionIndex = 0;

// Формируем список строк с разрешениями и находим индекс текущего разрешения

for (int i = 0; i < resolutions.Length; i++)

{

string option = resolutions[i].width + " x " + resolutions[i].height;

options.Add(option);

if (resolutions[i].width == Screen.currentResolution.width && resolutions[i].height == Screen.currentResolution.height)

{

currentResolutionIndex = i;

}

}

// Добавляем опции в dropdown и устанавливаем текущее разрешение по умолчанию

resolutionDropdown.AddOptions(options);

resolutionDropdown.value = currentResolutionIndex;

resolutionDropdown.RefreshShownValue();

}

private void Update()

{

// Обработка нажатия кнопки для открытия/закрытия меню настроек

if (Input.GetButtonDown("ToggleSettings"))

{

if (settingsOpen)

{

// При закрытии меню обновляем список сохранений и возобновляем время

SaveSelector.Instance.RefreshSaveList();

Time.timeScale = 1.0f;

settingsCanvas.alpha = 0f; // Скрываем меню

settingsCanvas.blocksRaycasts = false; // Отключаем взаимодействие с меню

settingsOpen = false;

}

else

{

// При открытии меню останавливаем игру и показываем меню

Time.timeScale = 0f;

settingsCanvas.alpha = 1f; // Показываем меню

settingsCanvas.blocksRaycasts = true; // Включаем взаимодействие с меню

settingsOpen = true;

}

}

}

// Метод установки разрешения экрана по индексу из списка доступных разрешений

public void SetResolution(int resolutionIndex)

{

Resolution resolution = resolutions[resolutionIndex];

Screen.SetResolution(resolution.width, resolution.height, Screen.fullScreen);

}

// Метод установки громкости звука на основе значения слайдера (0..1)

public void SetVolume(float sliderValue)

{

// Преобразуем значение слайдера в децибелы (-80..0)

float decibels = Mathf.Lerp(-80f, 0f, sliderValue);

// Устанавливаем громкость в аудиомикшере

audioMixer.SetFloat("volume", decibels);

// Сохраняем значение громкости в PlayerPrefs для последующего использования

PlayerPrefs.SetFloat("Volume", sliderValue);

}

// Метод переключения полноэкранного режима

public void SetFullScreen(bool isFullScreen)

{

Screen.fullScreen = isFullScreen;

}

}

Файл SaveSelector.cs

public class SaveSelector : MonoBehaviour

{

// Singleton-экземпляр класса для удобного доступа из других скриптов

public static SaveSelector Instance;

private void Awake()

{

// Реализация паттерна Singleton: если экземпляр ещё не создан, присваиваем текущий объект

if (Instance == null)

{

Instance = this;

}

else

{

// Если экземпляр уже существует, уничтожаем дубликат

Destroy(gameObject);

}

}

// Префаб кнопки для отображения сохранений

public GameObject saveButtonPrefab;

// Родительский объект для кнопок сохранений (контейнер в UI)

public Transform saveButtonParent;

// Ссылка на менеджер сохранений для загрузки выбранного сохранения

public SaveManager saveManager;

// Расширение файлов сохранений (например, ".xml")

public string saveFileExtension = ".xml";

// Панель UI для выбора сохранений (открывается и закрывается)

public GameObject saveLoadPanel;

// Список имён найденных файлов сохранений без расширения

private List<string> saveFiles = new List<string>();

private void Start()

{

// При старте обновляем список сохранений

RefreshSaveList();

}

// Открыть панель выбора сохранений и обновить список

public void OpenSaveLoadPanel()

{

saveLoadPanel.SetActive(true);

RefreshSaveList();

}

// Закрыть панель выбора сохранений

public void CloseSaveLoadPanel()

{

saveLoadPanel.SetActive(false);

}

// Обновить список сохранений: очистить старые кнопки и создать новые по найденным файлам

public void RefreshSaveList()

{

Debug.Log("RefreshSaveList called!");

// Удаляем все дочерние объекты (старые кнопки) из контейнера

foreach (Transform child in saveButtonParent)

{

Destroy(child.gameObject);

}

saveFiles.Clear();

// Путь к папке с сохранениями (папка persistentDataPath)

string saveDirectory = Application.persistentDataPath;

DirectoryInfo directoryInfo = new DirectoryInfo(saveDirectory);

// Получаем все файлы с нужным расширением

FileInfo[] fileInfos = directoryInfo.GetFiles("\*" + saveFileExtension);

Debug.Log("Found " + fileInfos.Length + " save files.");

// Для каждого файла создаём кнопку в UI

foreach (FileInfo fileInfo in fileInfos)

{

// Получаем имя файла без расширения для отображения

string saveName = Path.GetFileNameWithoutExtension(fileInfo.Name);

saveFiles.Add(saveName);

// Создаём кнопку из префаба и помещаем в контейнер

GameObject saveButtonGO = Instantiate(saveButtonPrefab, saveButtonParent);

if (saveButtonGO == null)

{

Debug.LogError("saveButtonGO is null! saveButtonPrefab: " + saveButtonPrefab);

continue;

}

// Находим компонент текста внутри кнопки и устанавливаем имя сохранения

TextMeshProUGUI buttonText = saveButtonGO.GetComponentInChildren<TextMeshProUGUI>();

if (buttonText != null)

{

buttonText.text = saveName;

}

// Добавляем обработчик нажатия, который загрузит выбранное сохранение

Button button = saveButtonGO.GetComponent<Button>();

if (button != null)

{

button.onClick.AddListener(() => LoadSelectedSave(saveName));

}

}

}

// Загрузить выбранное сохранение по имени

public void LoadSelectedSave(string saveName)

{

// Устанавливаем имя файла сохранения в менеджере сохранений с расширением

saveManager.saveFileName = saveName + saveFileExtension;

// Вызываем загрузку игры

saveManager.LoadGame();

Debug.Log("Loading save: " + saveName);

// Закрываем панель выбора сохранений

CloseSaveLoadPanel();

}

}

Файл PlayerBow.cs

public class PlayerBow : MonoBehaviour

{

// Точка, из которой выпускается стрела

public Transform launchPoint;

// Префаб стрелы для создания экземпляров при выстреле

public GameObject arrowPrefab;

// Ссылка на скрипт управления движением игрока

public PlayerMovement playerMovement;

// Направление прицеливания, по умолчанию вправо

private Vector2 aimDirection = Vector2.right;

// Время перезарядки между выстрелами

public float shootCooldown = .5f;

// Таймер для отслеживания времени до следующего выстрела

private float shootTimer;

// Компонент Animator для управления анимациями игрока

public Animator anim;

void Update()

{

// Уменьшаем таймер перезарядки с течением времени

shootTimer -= Time.deltaTime;

// Обработка направления прицеливания

HandleAiming();

// Если нажата кнопка "Shoot" и перезарядка завершена

if (Input.GetButtonDown("Shoot") && shootTimer <= 0)

{

// Устанавливаем флаг стрельбы в скрипте движения игрока

playerMovement.isShooting = true;

// Запускаем анимацию стрельбы

anim.SetBool("isShooting", true);

}

}

// При активации объекта переключаем веса слоёв анимации для стрельбы

private void OnEnable()

{

anim.SetLayerWeight(0, 0); // Отключаем базовый слой анимации

anim.SetLayerWeight(1, 1); // Включаем слой анимации стрельбы

}

// При деактивации объекта возвращаем веса слоёв анимации обратно

private void OnDisable()

{

anim.SetLayerWeight(0, 1); // Включаем базовый слой анимации

anim.SetLayerWeight(1, 0); // Отключаем слой анимации стрельбы

}

// Обработка направления прицеливания на основе ввода с клавиатуры или геймпада

private void HandleAiming()

{

// Получаем значения по горизонтали и вертикали (WASD, стрелки или аналог)

float horizontal = Input.GetAxisRaw("Horizontal");

float vertical = Input.GetAxisRaw("Vertical");

// Если есть ввод по любой оси — обновляем направление прицеливания

if (horizontal != 0 || vertical != 0)

{

aimDirection = new Vector2(horizontal, vertical).normalized;

// Передаём значения направления в аниматор для корректной анимации прицеливания

anim.SetFloat("aimX", aimDirection.x);

anim.SetFloat("aimY", aimDirection.y);

}

}

// Метод, вызываемый для выпуска стрелы

public void Shoot()

{

// Проверяем, что перезарядка завершена

if (shootTimer <= 0)

{

// Создаём стрелу в позиции launchPoint с нулевым вращением

Arrow arrow = Instantiate(arrowPrefab, launchPoint.position, Quaternion.identity).GetComponent<Arrow>();

// Устанавливаем направление полёта стрелы

arrow.direction = aimDirection;

// Сбрасываем таймер перезарядки

shootTimer = shootCooldown;

}

// Отключаем анимацию стрельбы и флаг стрельбы в скрипте движения

anim.SetBool("isShooting", false);

playerMovement.isShooting = false;

}

}

Файл PlayerCombat.cs

public class PlayerCombat : MonoBehaviour

{

// Компонент Animator для управления анимациями игрока

public Animator anim;

// Точка атаки — позиция, откуда наносится урон

public Transform attackPoint;

// Слой, на котором находятся враги для определения попадания

public LayerMask enemyLayer;

// Таймер для отслеживания времени между атаками (перезарядка)

private float timer;

// Метод вызывается каждый кадр

public void Update()

{

// Уменьшаем таймер, если он больше или равен нулю

if (timer >= 0)

{

timer -= Time.deltaTime;

}

}

// Метод начала атаки игрока

public void Attack()

{

// Проверяем, что перезарядка завершена

if (timer <= 0)

{

// Запускаем анимацию атаки

anim.SetBool("isAttacking", true);

// Сбрасываем таймер перезарядки на значение из StatsManager

timer = StatsManager.Instance.cooldown;

}

}

// Метод нанесения урона врагам в зоне атаки

public void DealDamage()

{

// Получаем всех врагов в радиусе атаки вокруг attackPoint

Collider2D[] enemies = Physics2D.OverlapCircleAll(attackPoint.position, StatsManager.Instance.weaponRange, enemyLayer);

if (enemies.Length > 0)

{

// Наносим урон первому врагу в списке

enemies[0].GetComponent<EnemyHealth>().ChangeHealth(-StatsManager.Instance.damage);

// Применяем эффект отбрасывания и оглушения к врагу

enemies[0].GetComponent<EnemyKnockback>().Knockback(transform, StatsManager.Instance.knockbackForce, StatsManager.Instance.knockbackTime, StatsManager.Instance.stunTime);

}

}

// Метод, вызываемый по окончании анимации атаки для сброса состояния

public void FinishAttaking()

{

anim.SetBool("isAttacking", false);

}

// Визуализация радиуса атаки в редакторе

private void OnDrawGizmosSelected()

{

Gizmos.color = Color.red;

Gizmos.DrawWireSphere(attackPoint.position, StatsManager.Instance.weaponRange);

}

}

Файл PlayerMovement.cs

public class PlayerMovement : MonoBehaviour

{

// Singleton-экземпляр для удобного доступа к PlayerMovement из других классов

public static PlayerMovement Instance;

// Компонент Rigidbody2D для управления физикой и перемещением игрока

public Rigidbody2D rb;

// Компонент Animator для управления анимациями игрока

public Animator anim;

// Ссылка на скрипт боевой системы игрока

public PlayerCombat playerCombat;

// Направление, в которое смотрит игрок (1 — вправо, -1 — влево)

public int facingDirection = 1;

// Флаг, указывающий, находится ли игрок в состоянии отбрасывания (knockback)

private bool isKnockedBack;

// Флаг, указывающий, что игрок в данный момент стреляет (ограничивает движение)

public bool isShooting;

private void Awake()

{

// Реализация паттерна Singleton: если экземпляр ещё не создан, присваиваем текущий объект

if (Instance == null)

{

Instance = this;

}

else

{

// Если экземпляр уже существует, уничтожаем дубликат

Destroy(gameObject);

}

}

private void Update()

{

// Если нажата кнопка атаки и скрипт боевой системы активен — вызываем метод атаки

if (Input.GetButton("Attack") && playerCombat.enabled == true)

{

playerCombat.Attack();

}

}

void FixedUpdate()

{

// Если игрок стреляет — останавливаем его движение

if (isShooting)

{

#pragma warning disable CS0618 // Используется устаревший член, но для совместимости оставлен

rb.velocity = Vector2.zero;

#pragma warning restore CS0618

}

// Если игрок не в состоянии отбрасывания — обрабатываем движение

else if (!isKnockedBack)

{

// Получаем ввод по горизонтали и вертикали (WASD, стрелки или аналог)

float horizontal = Input.GetAxis("Horizontal");

float vertical = Input.GetAxis("Vertical");

// Если направление движения противоположно направлению взгляда — меняем сторону (флип)

if (horizontal > 0 && transform.localScale.x < 0 || horizontal < 0 && transform.localScale.x > 0) Flip();

// Передаём абсолютные значения скорости в аниматор для плавной анимации ходьбы

anim.SetFloat("horizontal", Mathf.Abs(horizontal));

anim.SetFloat("vertical", Mathf.Abs(vertical));

// Устанавливаем скорость Rigidbody2D с учётом параметра скорости из StatsManager

#pragma warning disable CS0618

rb.velocity = new Vector3(horizontal, vertical) \* StatsManager.Instance.speed;

#pragma warning restore CS0618

}

}

// Метод для изменения направления взгляда игрока (отзеркаливание спрайта)

public void Flip()

{

// Меняем знак направления взгляда

facingDirection \*= -1;

// Отзеркаливаем спрайт по оси X

transform.localScale = new Vector3(transform.localScale.x \* -1, transform.localScale.y, transform.localScale.z);

}

// Метод применения отбрасывания игрока при получении удара

public void Knockback(Transform enemy, float force, float stunTime)

{

// Устанавливаем флаг, что игрок в состоянии отбрасывания

isKnockedBack = true;

// Вычисляем направление отбрасывания от врага к игроку и нормализуем

Vector2 direction = (transform.position - enemy.position).normalized;

#pragma warning disable CS0618

// Задаём скорость Rigidbody2D для отбрасывания с заданной силой

rb.velocity = direction \* force;

#pragma warning restore CS0618

// Запускаем корутину для отсчёта времени оглушения

StartCoroutine(KnockbackCounter(stunTime));

}

// Корутин для отсчёта времени оглушения и восстановления контроля над игроком

IEnumerator KnockbackCounter(float stunTime)

{

// Ждём заданное время оглушения

yield return new WaitForSeconds(stunTime);

#pragma warning disable CS0618

// Останавливаем движение игрока

rb.velocity = Vector2.zero;

#pragma warning restore CS0618

// Сбрасываем флаг оглушения — игрок может двигаться

isKnockedBack = false;

}

}

### 

### ПРИЛОЖЕНИЕ Б Результаты работы программы



Рисунок Б.1 – Результат получения урона игроком



Рисунок Б.2 – Результат отсутствия лечения при атаке врага

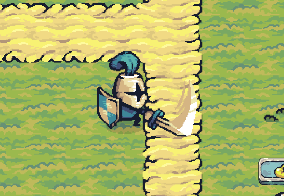


Рисунок Б.3 – Результат запуска анимации атаки пользователя

41



Рисунок Б.4 – Результат окончания анимации атаки пользователя



Рисунок Б.5 – Результат обновления текста здоровтя

42



Рисунок Б.6 – Результат смерти игрока при нулевом здоровье

43

КП Т. 293018.401 ГЧ

КБП

У

Т. Контр.

Разраб.

Пранович Д.С.

Провер. Провер.

Михалевич В.Ю.

Реценз.

Утверд.

Н. Контр.

*Диаграмма вариантов использования*

№ докум.

Подпись

Дата

Изм

Лист

Масса

Лист.

Масштаб

Лист 1

Листов 3

Инв.№подл.

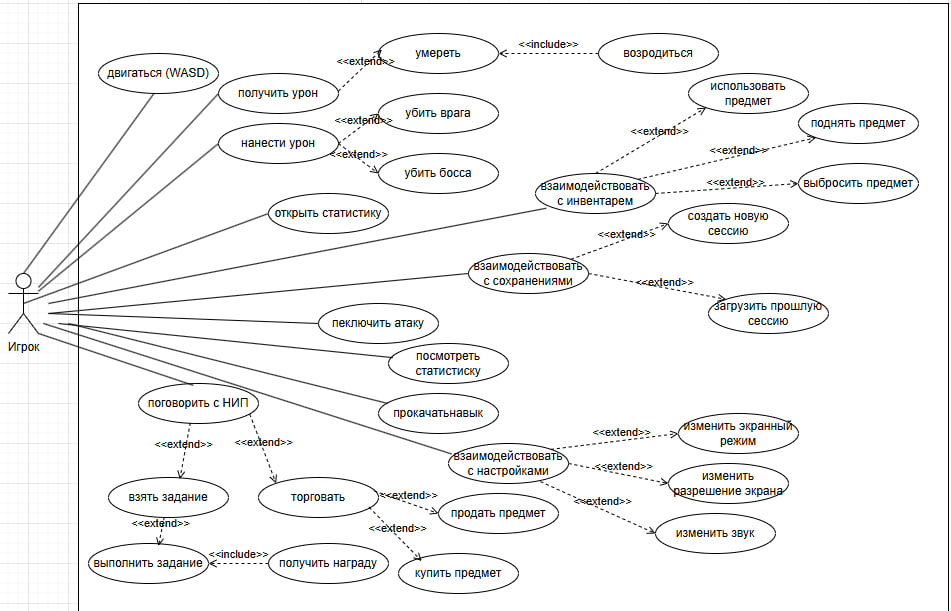
Подп. и дата

Взам.инв.№

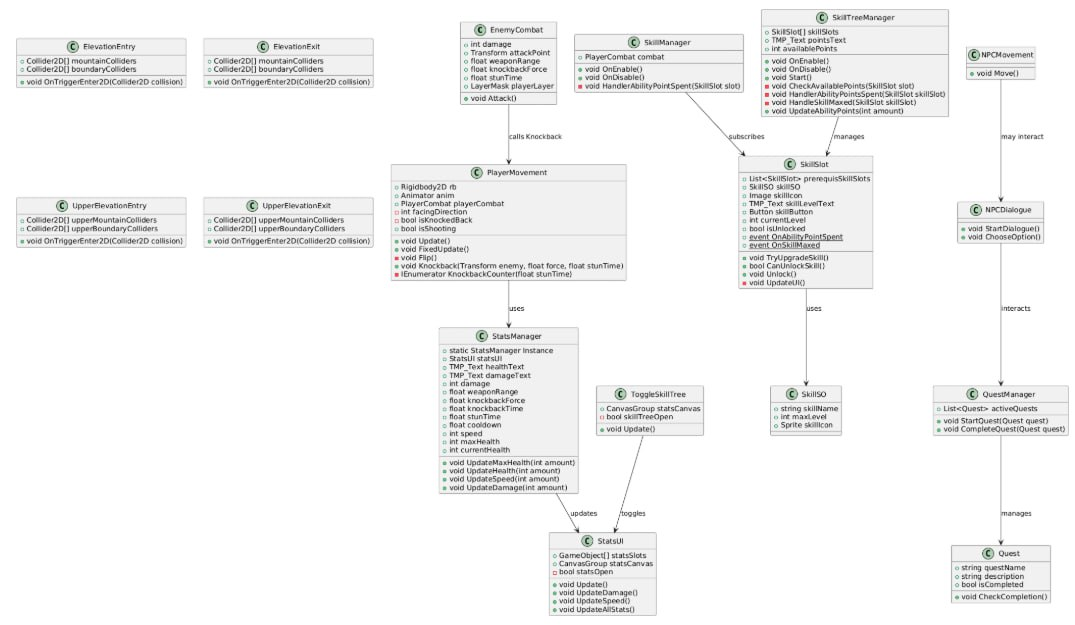
Инв.№дубл.

Подп. и дата

КП Т. 293018.401 ГЧ



Программная реализация игры «Cool Driver»



КП Т. 293018.401 ГЧ

КБП

У

Т. Контр.

Разраб. Разраб.

Пранович Д.С..

Провер. Провер.

Михалевич В.Ю.

Реценз.

Утверд.

Н. Контр.

Диаграмма классов

№ докум.

Подпис

Дата

Изм.

Лист

Масса

Лист.

Масштаб

Лист 2

Листов 3

Инв.№подл.

Подп. и дата

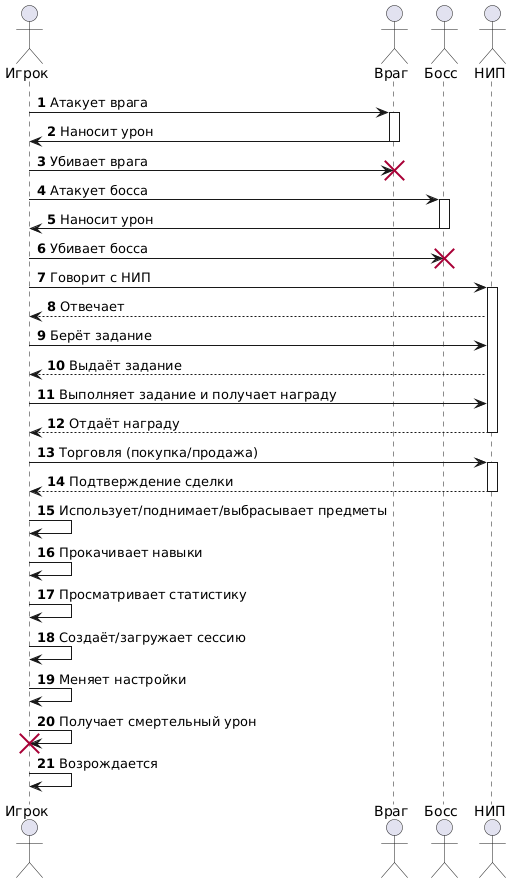
Взам.инв.№

Инв.№дубл.

Подп. и дата

КП Т. 293018.401 ГЧ

Программная реализация игры «Cool Driver»



КП Т. 293018.401 ГЧ

КБП

У

Т. Контр.

Разраб. Разраб.

Пранович Д.С.

Провер. Провер.

Михалевич В.Ю

Реценз.

Утверд.

Н. Контр.

Диаграмма последовательности

№ докум.

Подпис

Дата

Изм.

Лист

Масса

Лист.

Масштаб

Лист 3

Листов 3

Инв.№подл.

Подп. и дата

Взам.инв.№

Инв.№дубл.

Подп. и дата

КП Т. 293018.401 ГЧ

Программная реализация игры «Cool Driver»