МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(ФГБОУ ВО «КубГУ»)

Факультет компьютерных технологий и прикладной математики
Кафедра прикладной математики

Допустить к защите
И.о. заведующего кафедрой
канд. физ.-мат. наук, доц.
_____ А.В. Письменский
_____ 2023 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
(БАКАЛАВРСКАЯ РАБОТА)

РАЗРАБОТКА ИЗОМЕТРИЧЕСКОЙ ИГРЫ НА UNITY

Работу выполнил _____ Я. Шушкина

Направление подготовки 01.03.02 Прикладная математика и информатика

Направленность (профиль) Математическое и информационное обеспечение экономической деятельности

Научный руководитель
кандидат физ-мат наук, доц. _____ Н. М. Сеидова

Нормоконтролер
преподаватель _____ Е.С. Троценко

Краснодар
2023

РЕФЕРАТ

Выпускная квалификационная работа 60 с., 3 ч., 37 рис., 15 источн.

РАЗРАБОТКА ИЗОМЕТРИЧЕСКОЙ ИГРЫ НА UNITY

Объект исследования – проектирование изометрических игр среде движка Unity.

Предмет исследования – игры жанра RPG и их особенности, средства проектирования в виде движка Unity.

Актуальность данной курсовой работы обусловлена тем, что на сегодняшний день существует большая потребность в изучении средств проектирования для создания компьютерных игр. Так как эта область является одной из самых востребованных в данный момент.

Целью работы является использование движка Unity для реализации компьютерной игры жанра RPG.

В ходе работы был сделан прототип игр на игровом движке Unity.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Разработка игр является актуальной и постоянно развивающейся областью в современном мире. В настоящее время игры стали неотъемлемой частью культуры и развлечений многих людей по всему миру. Они предоставляют возможность погрузиться в увлекательные виртуальные миры, испытать новые эмоции и взаимодействовать с уникальными персонажами и сюжетами. Социальные платформы, мобильные устройства и компьютерные системы предоставляют все больше возможностей для разработки и распространения игр. Рынок игровой индустрии растет с каждым годом, привлекая миллионы игроков и генерируя значительные доходы. Это создает благоприятные условия для разработчиков, стимулируя их внести свой вклад в эту динамичную область.

В таком контексте разработка изометрической игры представляет большой интерес. Изометрическая графика и вид от третьего лица обеспечивают особый визуальный стиль, который позволяет создавать уникальные игровые миры и геймплейные механики. Это позволяет разработчикам воплотить свои творческие идеи, предлагая игрокам новые впечатления и возможности.

Цель данной работы заключается в разработке изометрической игры, что представляет собой увлекательный проект. Используя современные технологии и творческий подход, студент сможет создать захватывающий игровой опыт, привлекающий и удерживающий внимание игроков. Разработка такой игры дает возможность расширить границы возможностей в игровой индустрии и принести свой вклад в развитие этой популярной области.

Исходя из названной цели, в работе поставлен ряд задач:

– провести анализ предметной области, изучить технологии для решения поставленной задачи, изучить актуальность проекта;

– изучить основные этапы создания игры: подготовка, препродакшн, продакшн, релиз.;

– создание или нахождение ассетов;

– создание сюжета и истории игры;

– разработка геймплея и метагеймплея

– тестирование игры

– собрать статистические данные от игроков, дабы оценить игру

– сборка рабочей версии игры

Предметом дипломного исследования является процесс создания изометрической игры на игровом движке Unity. Объектом дипломного исследования является проектирование изометрических игр на движке Unity.

Общую теоретико-методологическую базу дипломного проекта составили труды исследователей, а также разработчиков, посвященные созданию игр на движке Unity и не только на нем: с чего сначала стоит начать разработку, что потребуется для создания игры, как правильно продумать механики и архитектуру кода, геймдизайн, левелдизайн и т. д.

Дипломная работа состоит из трех разделов. Первый раздел посвящен анализу предметной области. В этом разделе проводится исследование игровой индустрии и рынка, а также изучаются тенденции и требования игроков. В результате анализа формулируются основные принципы и концепции, которые будут использованы при разработке игры. Второй раздел представляет собой разработку и реализацию игрового проекта. Здесь подробно описывается архитектура игры, функциональные возможности и особенности игровых механик. В этом разделе также описываются использованные технологии и инструменты разработки, а также проводится тестирование и оптимизация проекта. Третий раздел посвящен оценке и сбору данных от игроков. В данном разделе проводится сбор обратной связи от пользователей, анкетирование или интервьюирование игроков для оценки их впечатлений от игры. Анализируются данные о понравившихся и не понравившихся аспектах игры, выявляются сильные и слабые стороны проекта. На основе полученных данных делаются выводы и предлагаются

рекомендации для улучшения игрового опыта и удовлетворения потребностей пользователей.

1 Анализ предметной области

1.1 Используемые технологии

Игровой движок играет важную роль в процессе разработки игры, поскольку он помогает разработчикам в создании сцены, персонажей, графических изображений, звука, искусственного интеллекта, анимации, работы в сети и т. д. Игровой движок похож на интегрированную среду разработки с готовым набором визуальных средств разработки и многократно используемых программных компонентов. Он упрощает сложную задачу разработки игр, предоставляя уровень абстракции, который делает многие большие задачи очень простыми, он также выполняет всю тяжелую работу в фоновом режиме. Другими словами, это фреймворк (программная платформа, определяющая структуру программной системы), разработанный специально для создания и разработки видеоигр. Разработчики используют эти игровые движки для создания игр для консолей, мобильных устройств и персональных компьютеров. Однако не стоит думать, что разработка игр является чем–то простым благодаря игровым движкам.

Компоненты игрового движка.

Игровой движок создан для разработки игр, как и любая другая IDE для программирования на любом конкретном языке. Все компоненты игрового движка построены и интегрированы для поддержки разработки игры.

Вводимые данные (Input): Игра – ничто, если в нее нельзя играть, игровой движок обеспечивает поддержку множества устройств ввода, таких как мышь, геймпад, сенсорный экран и т. д., а также обеспечивает поддержку таких устройств, как геймпад, джойстики и т. д. Существует множество различных способов обработки ввода, два наиболее часто используемых – это сквозные: события и опрос. События ввода фиксируются компьютером (щелчок правой кнопкой мыши, нажатие клавиши со стрелкой вверх), и ваш собственный код запускается в зависимости от того, какой ввод был получен.

Опрос используется для получения значений положения, например, по каким координатам (x, y) находится указатель мыши, или угол наклона игровой ручки или смартфона, с помощью которого вы играете в игру.

Графика: Графика в игре решает многое. Трехмерная графика создается с использованием трехмерных ресурсов, которые разрабатываются и создаются во внешних программах трехмерного рендеринга, таких как Maya, Blender и т. д., а затем импортируются в игровой движок. В двумерных играх используется плоская графика, называемая спрайтами, которая не имеет трехмерной геометрии. Спрайты отображаются на экране как плоские изображения, а камера не имеет перспективы (ортогональная проекция). Хороший игровой движок должен поддерживать несколько форматов импорта.

Игровой движок предоставляет множество функций, таких как световые эффекты, тени, карты рельефа, анимацию смешивания и т. д., чтобы импортированный ассет(ресурс) выглядел реалистичным. Физика: В игровом движке есть подкомпонент, известный как Physics Engine. Физические движки – это программное обеспечение, которое позволяет выполнять довольно точное моделирование большинства реальных физических систем, таких как движение твердого тела, изменение массы и скорости мягкого тела, а также гидродинамику, подвижность. и т.д. Это сложные движки, интегрированные в новейшие игровые движки, в основном используются в видеоиграх (обычно в качестве промежуточного программного обеспечения), где необходимо отображать симуляцию в реальном времени и в реальной жизни. Гравитация, обнаружение столкновений, вращение и вращение, скорость объектов и другие подобные приложения обрабатываются физическим движком в игре.

Искусственный интеллект: в наши дни искусственный интеллект играет значительную роль в разработке игр. Знание того, какое оружие будет использовать игрок в зависимости от ситуации или поведения игрока, записывается и действия выполняются соответствующим образом, это может быть выполнено с использованием специализированного программного

обеспечения, встроенного в игры. Внедрение ИИ в игры обычно осуществляется с помощью готовых скриптов, разработанных и написанных программистами, специализирующимися на ИИ.

Звук: Механизмы аудио и рендеринга являются частью движка игры, которые используются для управления звуковыми эффектами и создания трехмерной анимированной графики на вашем экране. Они обеспечивают программную абстракцию графического процессора с использованием API интерфейсов мульти рендеринга, таких как Direct3D или OpenGL, для рендеринга видео и API, таких как Open AL, SDL audio, X Audio 2, Web Audio для аудио.

Сеть: уже долгие годы игры поддерживают многопользовательские онлайн–игры и социальные игры, которые объединяют игровые приключения игроков с разных концов земли. Большинство игровых движков предоставляют полную поддержку и сценарии для таких требований, поэтому вам не нужно беспокоиться о трафике TCP / UDP, интеграции социальных API.

Как сделать игру? Это самый неуловимый вопрос игровой индустрии. Фактически, вся индустрия программного обеспечения в целом открыто говорила о том, насколько незрелы процессы разработки программного обеспечения в целом. Программисты прилагали много усилий для улучшения процесса создания программного обеспечения.

В течение 60-х и 70-х годов были достигнуты большие успехи в увеличении силы языков программирования от Fortran и COBOL до C. В 80-е годы микрокомпьютеры значительно улучшили рабочее место программирования. У каждого разработчика могла быть собственная рабочая станция, на которой он редактировал, запускал, отлаживал код. В конце 70-х как раз–таки и появился первый игровой движок ZIL.

В конце 80-х – начале 90-х годов программисты говорили об эффективности объектно–ориентированного программирования и сильных сторонах C ++ для крупных проектов.

Оптимизация компиляторов сделала программирование на ассемблере устаревшим, инструменты визуального макета интерфейса сделали программирование гораздо легче. Со всеми этими фантастическими улучшениями неудивительно, что количество создаваемых игровых движков возросло в разы. В конце 90-х было создано 17 игровых движком.

Сейчас игровых движком насчитывается не мало – около 70. Первые из этого списка уже не поддерживаются разработчиками и не идут в ногу с современными технологиями, однако некоторые разработчики–энтузиасты используют их для своих целей и даже улучшают. Это такие движки как: Dark Engine, Freescape, Infinity Engine. Вторые были специально разработаны для игр определённой коммерческой студии и их нельзя найти в открытом доступе. Примеры: Frostbite (EA), AnvilNext 2.0 (Ubisoft), RE Engine (Capcom), Luminous Studio (Square Enix), REDEngine 3(CD Project).

Большое значение для всей игровой индустрии имеют движки, которые распространяются бесплатно или же по платной подписке для всех пользователей. Такие движки пользуются большим спросом среди начинающих и опытных разработчиков, ведь каждый с лёгкостью может их опробовать и подобрать движок лично для себя. Наибольшее влияние или перспективу в этой среде имеют такие игровые движки как: Unity Engine 4, Unreal Engine, GodotEngine, Game Maker Studio.

Почему игры так важны в наше время? Игры давно перешли из категории детских развлечений в серьезный и значимый вид индустрии, ставшей неотъемлемой частью современной культуры. С развитием технологий игры стали более доступными, что привело к росту популярности и прибыльности игровой индустрии. В настоящее время игровая индустрия является одной из наиболее прибыльных и быстрорастущих отраслей в мире, с ожидаемой стоимостью более 200 миллиардов долларов в 2023 году, обгоняя кино и музыку. Более того, они стали не только формой развлечения, но и платформой для образования и развития навыков. Игры могут помочь людям научиться новым вещам, улучшить реакцию и координацию движений,

развить логическое и креативное мышление, а также улучшить коммуникативные навыки.

Таким образом, разработка игр – это не только выгодный бизнес, но и серьезная и важная часть нашей культуры и образования.

Однако наш конечный продукт – это видеоигра. В процессе разработки мы ставим перед собой цель создать уникальную игру, которая позволит игрокам окунуться в фантастический мир, полный приключений и загадок, поэтому передем непосредственно к реализации задуманного.

1.2 Актуальность

Развлечение:

Игровая индустрия – это многомиллиардная индустрия, и нетрудно понять, почему. Видеоигры предлагают уникальную форму развлечения, которая может переносить игроков в разные миры и впечатления. В 2020 году мировой рынок видеоигр оценивался в 159,3 млрд долларов США, и ожидается, что совокупный годовой темп роста (CAGR) в период с 2021 по 2028 год составит 9,3%. Это показывает, что спрос на видеоигры не только высок, но и быстро растущий.

Образование:

Игры также могут быть эффективным средством обучения. Согласно опросу, проведенному Центром Джоан Ганц Куни, 74% учителей K 8 сообщили об использовании цифровых игр для обучения в 2014 году, а 55% использовали игры еженедельно. Видеоигры могут сделать обучение более интерактивным и увлекательным, помогая учащимся лучше запоминать информацию. Кроме того, они могут обучать таким навыкам, как критическое мышление, решение проблем и принятие решений, которые ценны как для академического, так и для реального успеха.

Социализация:

Видеоигры также могут служить социальной отдушиной, особенно во время социального дистанцирования и изоляции. Согласно опросу, проведенному Ассоциацией развлекательного программного обеспечения, 70% геймеров играют с другими в сети, а 53% с друзьями. Многопользовательские игры предлагают чувство общности и связи, позволяя игрокам взаимодействовать с людьми со всего мира.

Психическое благополучие:

Видеоигры также могут оказывать положительное влияние на психическое здоровье. Исследование, опубликованное в Журнале киберпсихологии, поведения и социальных сетей, показало, что видеоигры помогают снять стресс и тревогу. Более того, игры могут быть источником релаксации, отвлечения внимания и даже терапии для некоторых людей.

В заключение следует отметить, что игровая индустрия важнее, чем когда либо, предоставляя развлечения, образование, социализацию и психологическое благополучие миллионам людей во всем мире. Учитывая, что в ближайшие годы ожидается рост индустрии, нет сомнений в том, что видеоигры будут продолжать играть важную роль в нашей жизни.

2 Реализация

2.1 Концепция игры

Концепция игры – это первоначальное описание того, как будет выглядеть и работать игра. Она должна содержать информацию о теме и жанре игры, геймплее, механиках, структуре игры и т. д. В этом разделе мы должны определить, что мы хотим сделать, почему и как мы это сделаем. Определение темы и жанра игры один из ключевых шагов при разработке игры. Тема игры описывает ее основную идею, тогда как жанр определяет, как будут представлены главные элементы игры, такие как геймплей, механики, искусство и звук. Важно выбрать тему и жанр, которые будут интересны целевой аудитории и соответствовать навыкам и ресурсам разработчика.

Используя данную информацию, можно составить описание нашей игры:

Название игры: Devoid of Everglow;

Жанр: Action, RPG, Adventures;

Платформы: PC, Mobile;

Целевая аудитория: Фанаты фантастических жанров, Женщины и Мужчины от 12 до 30 лет.

Краткое описание: «Devoid of Everglow» – это игра в жанре научной фантастики, которая позволяет игроку окунуться в альтернативную реальность в роли тестера корпорации EM. Игроку предстоит исследовать виртуальное пространство и решать различные проблемы, связанные с происходящими в сети событиями.

Цель игры – помочь герою победить опасных существ сети, решить различные головоломки, и найти способ выбраться из виртуального мира. При этом игроку предстоит исследовать уникальный мир игры, взаимодействовать с другими персонажами и совершать различные действия, чтобы преодолеть препятствия на пути к восстановлению своей памяти.

Так как мы стараемся создать уникальный продукт, то важно обозначить особенности игры:

Одной из главных особенностей игры является атмосфера, создаваемая виртуальным миром. Игрок погружается в альтернативную реальность. Уникальный дизайн локаций, музыкальное сопровождение и звуковые эффекты помогают создать неповторимую атмосферу, которая погружает игрока в игровой мир.

У игры изометрический вид. Вместо стандартного видеоизображения, игроки будут играть из высоко установленной камеры, позволяющей получить широкий обзор на игровой мир и создать ощущение, что игроки смотрят на игровой мир сверху вниз. Такой вид игры дает уникальный геймплей и помогает игрокам получить новые впечатления.

Передача уютного одиночества в мире игры, где игрок может взаимодействовать с окружающей средой. Это поможет создать атмосферу уединения и погружения.

Концепция перемещения между разными пространствами, где игрок переносится из сети в реальный мир после каждого прохождения уровня, с изменением вида камеры от первого лица.

Простая графика и необычный сеттинг, который поможет создать уникальную атмосферу и подчеркнуть уникальность игры.

Интересное управление персонажем, где игрок может комбинировать атаки в соответствии с задачами и испытывать свою реакцию и способности.

Чувство сопричастности через сюжетную линию, которую игрок должен разобраться, что поможет создать более глубокую связь между игроком и игровым миром.

После определения темы, жанра игры, целевой аудитории, ее описания и т. д., необходимо создать концептАрт, который поможет визуализировать игровой мир и персонажей. КонцептАрт является важным инструментом в разработке игр, так как он помогает определить стиль и атмосферу игры, а также визуализировать ее главные элементы. Для создания концептАрта

можно использовать различные техники и программы, такие как рисование вручную, 3D моделирование, компьютерное рисование и т. д.

В нашем случае будет использоваться программа Blender и основные два концептАрта показаны на рисунке 1 и рисунке 2.

Рисунок 1 – Первый КонцептАрт

Рисунок 2 – Второй КонцептАрт

Таким образом, вы завершили один из основных этапов в создании игры, который помог определить нам концепцию и пут развития проекта дальше.

2.2 Моделирование игровых объектов: разработка игровых объектов

Одним из основных инструментов цифрового моделирования является 3D моделирование, которое позволяет создавать трехмерные модели игровых объектов. Для этого могут использоваться различные программы, такие как Blender, Autodesk Maya, 3ds Max, ZBrush и другие. Как упоминалось в предыдущем разделе, для наших целей мы будем использовать Blender. Создание 3D моделей позволяет разработчикам более эффективно работать над визуализацией игровых объектов, их анимацией, текстурами и освещением. В результате игровые объекты выглядят более реалистично и детализировано, что улучшает общее впечатление от игры и делает ее более привлекательной для игроков.

Процесс создания любой 3d модели включает в себя несколько общих этапов: Концептуализация: определение общей идеи и формы модели; Моделирование: создание базовой геометрии модели в 3D программном обеспечении; Текстурирование: добавление цветовой и текстурной информации на поверхности модели; Риггинг: создание внутренней структуры и контрольных точек, которые позволяют анимировать модель.

Перейдем к описанию нескольких моделей для наглядности:

Модель основного игрового персонажа, за которого будет играть игрок на рисунке 3, можно увидеть первоначальный концепт, итоговый результат, а также сетку, которая подтверждает качество модели.

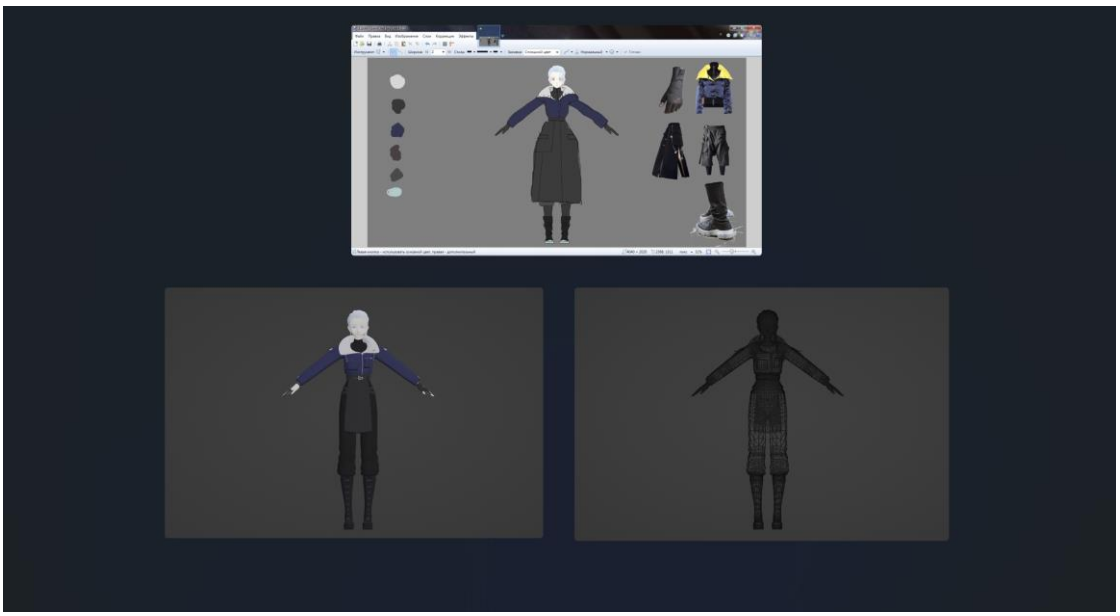


Рисунок 3 – Основной персонаж: от концепта до реализации

Модель врага, который будет мешать персонажу, можно увидеть на рисунке 4.

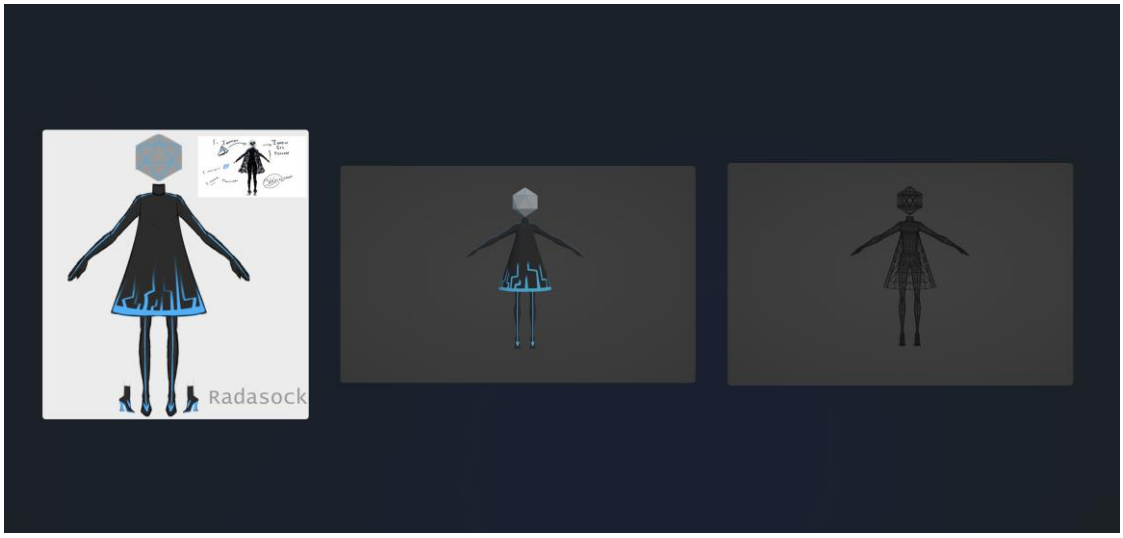


Рисунок 4 – Модель врага: от концепта до реализации

Модели окружения, из которых выстроены локации игрового мира, показаны на рисунке 5.

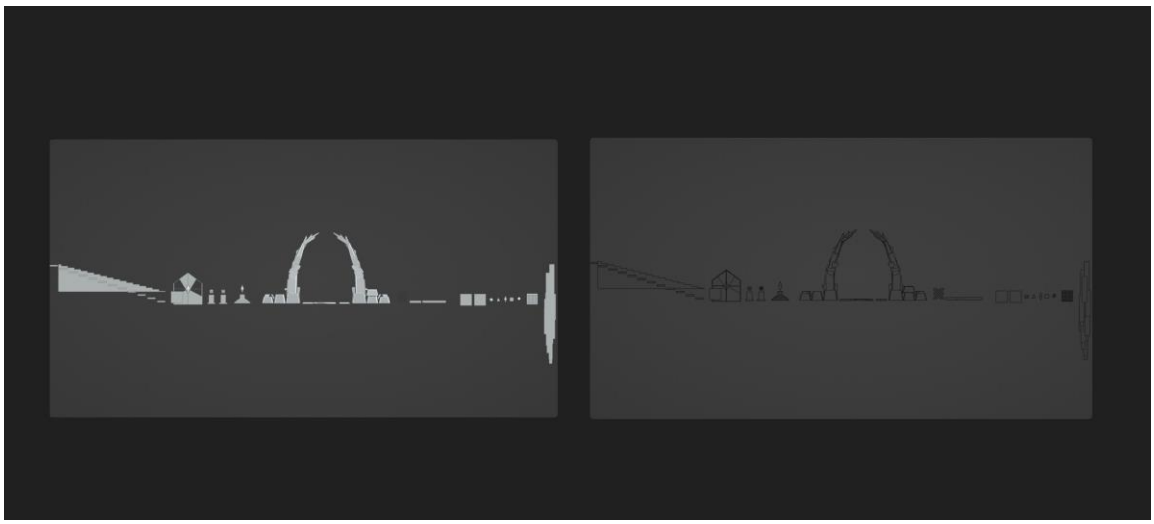


Рисунок 5 – Модели

Таким образом, моделирование игровых объектов является важным этапом разработки видеоигр и позволяет создавать высококачественные и привлекательные игры, которые будут иметь успех у пользователей. Для успешной разработки игры необходимо уметь создавать 3D модели объектов, используя специальные программы для цифрового моделирования, а также обладать знаниями и навыками в области проектирования игровых объектов. Цифровое моделирование и проектирование позволяют ускорить процесс создания игры, улучшить качество и детализацию моделей, а также экономить ресурсы и сократить время разработки.

2.3 Интерфейс пользователя: разработка пользовательского интерфейса

Разработка пользовательского интерфейса (UI) играет важную роль в создании игры. В этом разделе мы рассмотрим основные шаги, которые следует сделать для разработки UI в игре.

Первым шагом в разработке UI является определение цели, которую должен достигать пользовательский интерфейс. Определение цели поможет

разработчикам понять, какие элементы необходимы для достижения этой цели.

Перед нами стоит цель создать удобный и приятный пользовательский интерфейс, который будет помогать игроку взаимодействовать с игрой. Например, основное меню, в котором можно начать игру, настроить игру под себя или выйти или же HUD (Heads Up Display), который должен помогать игроку отслеживать разные характеристики по типу здоровья, маны, очков и т. д.

После того, как цель определена, следующим шагом является создание концепта UI. Это включает в себя определение общего дизайна и выбор цветовой схемы, шрифтов и стилей, которые будут использоваться в игре. И дальше предстоит разработать макеты. Макеты UI представляют собой прототипы пользовательского интерфейса, которые могут быть использованы для проверки функциональности и улучшения дизайна.

Разработанный макет для игры можно увидеть на рисунке 6.

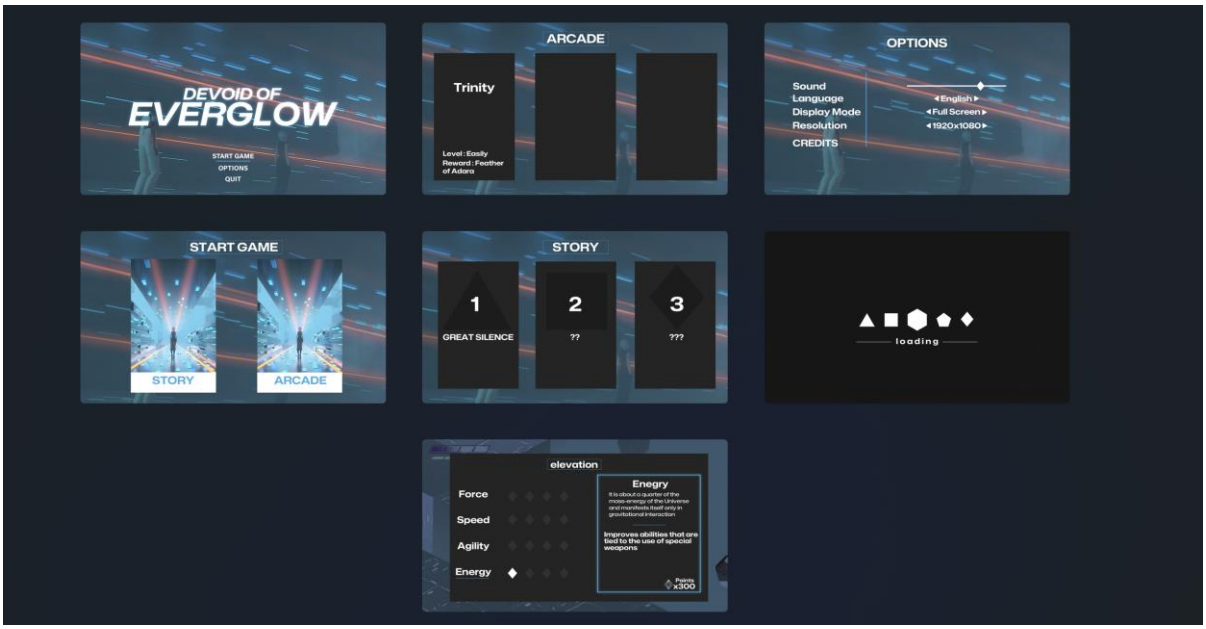


Рисунок 6 – Макет UI

Главным является то, что весь UI выдержан в одном стиле, а также достаточно удобен и понятен.

После того, как макеты UI созданы и одобрены, следующим шагом является реализация UI. Реализация происходит непосредственно в среде игрового движка, однако по мере внедрения также происходит тестирование. И также происходят постоянные улучшения конечного варианта UI.

Окончательная версия пользовательского интерфейса после всех этапов можно увидеть на рисунке 7.

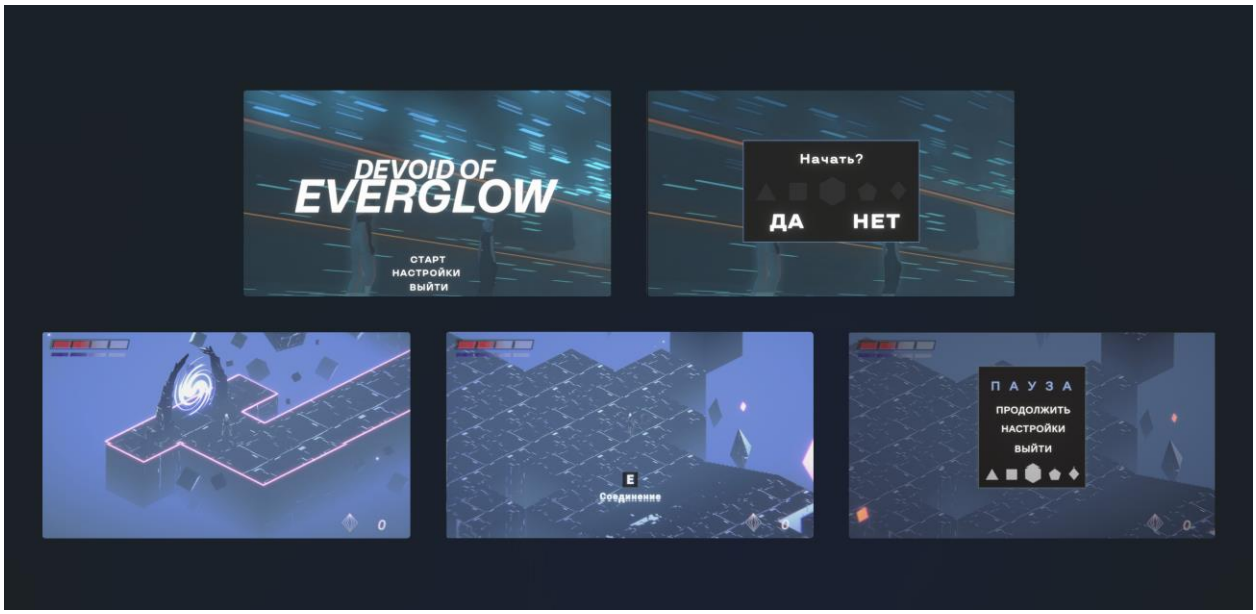


Рисунок 7 – Законченный интерфейс в игровой среде

Итак, разработка пользовательского интерфейса играет важную роль в создании игр. Она помогает обеспечить удобство использования игры пользователями.

2.4 Выбор игрового движка и языка программирования

При разработке любой игры одним из самых важных решений является выбор игрового движка и языка программирования, которые будут использоваться. В данном проекте было решено выбрать игровой движок Unity и язык программирования C#.

Первоначально выпущенная в 2005 году, Unity представляет собой среду разработки игр и движок для более чем 20 различных платформ. Он работает в MAC OS X, Linux и Windows. Unity быстро стал популярным среди людей, начинающих создавать игры, благодаря простому и понятному интерфейсу. Частые обновления, хорошая документация и еще более развитое сообщество сделали его одним из лучших движков для работы.

Преимущества Unity.

Unity предлагает моделирование физических сред, карты нормалей, прегражденне окружающего света в экранном пространстве (Screen Space Ambient Occlusion, SSAO), динамические тени и т. д. Unity обладает двумя преимуществами перед другими передовыми инструментами разработки игр. Это крайне производительный визуальный рабочий процесс и сильная межплатформенная поддержка. Визуальный рабочий процесс – достаточно уникальная вещь, выделяющая Unity.

Рабочий процесс привязан к визуальному редактору на рисунке 8. Именно там происходит компоновка сцены будущей игры, связывая игровые ресурсы и код в интерактивные объекты. Редактор особенно удобен для процессов с последовательным улучшением, например, циклов создания прототипов или тестирования. Даже после запуска игры остается возможность модифицировать в нем объекты и двигать элементы сцены.

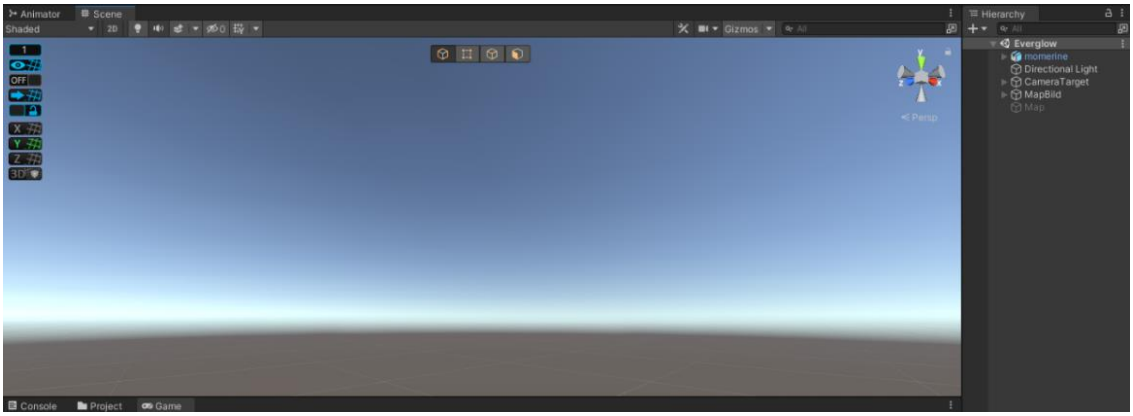


Рисунок 8 – Визуальный редактор Unity

Так в Unity есть преимущество в виде модульной системы компонентов, которая используется для конструирования игровых объектов. «Компоненты» в такой системе представляют собой комбинируемые пакеты функциональных элементов, поэтому объекты создаются как наборы компонентов, а не как жесткая иерархия классов. В результате получается альтернативный (и обычно более гибкий) подход к объектно ориентированному программированию, в котором игровые объекты создаются путем объединения, а не наследования на рисунке 9.

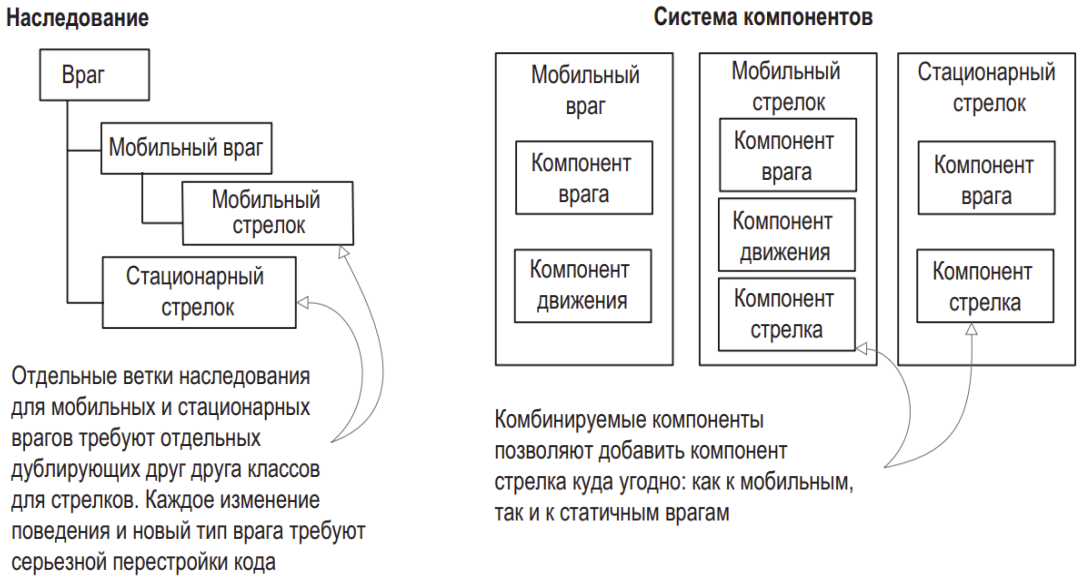


Рисунок 9 – Сравнение наследования с компонентной

В компонентной системе объект существует в горизонтальной иерархии, поэтому объекты состоят из наборов компонентов, а не из иерархической структуры с наследованием, в которой разные объекты оказываются на разных ветках дерева. Такая компоновка облегчает создание прототипов, потому что взять нужный набор компонентов куда быстрее и проще, чем перестраивать цепочку наследования при изменении каждого объекта. Эта система дает возможность не только управлять компонентами программным образом, но и соединять и разрывать связи между ними в редакторе.

Разработчики игр пишут сценарии (Script) в Unity на C#, мощном объектно ориентированном языке программирования, разработанном Microsoft. С тех пор он стал одним из самых популярных языков программирования. Команда Unity выбрала C # в качестве основного языка программирования, потому что он хорошо документирован, прост в изучении и достаточно гибок.

Недостатки Unity. Многочисленные преимущества приложения Unity превращают его в замечательное средство разработки игр, но было бы упущением не упомянуть о его недостатках. В частности, затруднения может вызвать нетипичное сочетание визуального редактора со сложным кодом, несмотря на всю его эффективность в рамках компонентной системы Unity. В сложных сценах из виду могут потеряться присоединенные компоненты. Некоторых программистов может обескуражить и то, что Unity не поддерживает ссылки на внешние библиотеки кода. Все библиотеки, которые вы планируете задействовать, следует вручную копировать в проект, вместо того чтобы дать ссылку на одну папку общего доступа. Отсутствие единой папки с библиотеками затрудняет коллективное использование функционала разными проектами. Это неудобство можно обойти, рационально воспользовавшись системами контроля версий, но готовое решение данной проблемы в Unity отсутствует.

Третий недостаток связан с шаблонами создания экземпляров (prefabs). Шаблоны экземпляров предлагают гибкий подход к визуальному созданию интерактивных объектов. Эта крайне мощная концепция существует исключительно в Unity (и, естественно, она связана с компонентной системой Unity), но редактирование таких шаблонов порой оказывается на удивление труднореализуемым.

Язык программирования C# был выбран, как основной язык для программирования игровой логики и скриптов в Unity. C# является современным объектно ориентированным языком программирования,

который поддерживает различные парадигмы программирования и позволяет разработчикам создавать высококачественный код с легкостью.

Вместе Unity и C# предоставляют мощную и эффективную платформу для разработки игр, которая позволяет создавать игры любой сложности и жанра. Они также обеспечивают богатый набор инструментов и ресурсов, которые помогают разработчикам создавать игры быстро и эффективно.

В целом, выбор Unity и C# для разработки игры был обоснованным и эффективным решением, которое позволило создать высококачественную игру с использованием современных технологий и инструментов.

2.5 Определение игровых механик и разработка игровой логики

Одним из важных этапов создания игры было определение игровых механик и разработка игровой логики. Важно сфокусироваться на создании увлекательного геймплея, который будет держать в напряжении игроков на протяжении всей игры. Разберем какие игровые механики были разработаны и как они были реализованы в игре.

Начать стоит с описания основных игровых механик, которые создают Core Loop. Core Loop – это цикл основных игровых действий, который определяет, как игрок взаимодействует с игрой и какие действия ему нужно выполнить для достижения цели. Основные моменты данного цикла можно увидеть на рисунке 10.

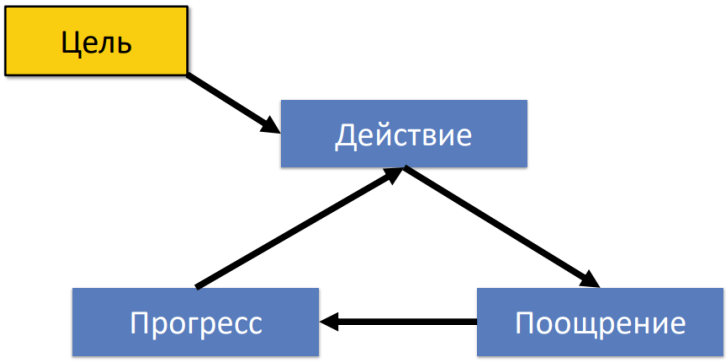


Рисунок 10 – Core Loop

В игре игроку поставлена цель – найти выход. Чтобы достичь этой цели, ему необходимо использовать различные игровые механики. Он будет сталкиваться с врагами, которых нужно побеждать, чтобы продвинуться дальше, а также с головоломками, которые нужно решать. Для успешного выполнения этих задач игровой персонаж должен обладать соответствующими механиками и умениями. Давайте рассмотрим, какие механики были применены в игре для взаимодействия с игровыми объектами.

Самое основное с чего можно начать – создание компонента, который отвечает за передвижение персонажа. Его можно посмотреть в Приложении А.

Основная особенность реализации движения в данном случае заключается в том, что в игре изометрическая проекция. При использовании изометрической проекции реализация передвижения персонажа будет отличаться от реализации в обычной перспективной камере.

В изометрической проекции персонаж не будет двигаться по прямой линии, а будет двигаться по диагонали, чтобы соответствовать перспективе. Для этого необходимо использовать вектор, который будет повернут на угол камеры или проекции, и перемещать персонажа по этому вектору. Это можно реализовать, используя матрицы поворота и умножение вектора на эту матрицу.

Скрипт будет являться компонентом движения для объекта в Unity– игре и отвечает за передвижение и поворот персонажа. Внутри скрипта используются математические методы векторной алгебры, такие как работа с векторами и кватернионами. Общая работа скрипта скрипты выглядет следующим образом: когда игрок нажимает на кнопку движения, метод GatherInput принимает данные и преобразует их в вектор направления движения. Метод Look поворачивает персонажа в направлении, указанном в векторе движения, используя кватернионы и методы RotateTowards и LookRotation. Когда игрок движется, метод Move () перемещает персонажа вперед по вектору направления и устанавливает соответствующую скорость анимации, используя метод SetFloat из Animator компонента. Также в скрипте

используются методы OnEnable и OnDisable, чтобы включать и выключать InputAction, когда скрипт активен и неактивен соответственно. Метод ToIso из класса Helpers принимает 3D вектор и преобразует его в изометрическую плоскость с помощью матрицы преобразования.

Теперь рассмотрим подробнее какие методы математической алгебры используются для работы с движением и вращением в Unity.

Vector3 – это структура в Unity, которая представляет собой трехмерный вектор в пространстве. В нашем случае Vector3 используется для представления направления движения объекта, которое определяется в методе GatherInput на основе входных данных игрока.

Quaternion – это структура в Unity, которая используется для представления вращения объекта в пространстве. Quaternion используется для вращения персонажа в методе Look.

Нужно понимать разницу между углами Эйлера (значениями X, Y и Z, которые мы видим в инспекторе для поворота GameObject) и базовым значением Quaternion, которое Unity использует для хранения фактического поворота GameObjects. При работе с поворотами в скриптах мы должны использовать класс Quaternion и его функции для создания и изменить значения вращения. В некоторых ситуациях допустимо использование углов Эйлера, но мы должны помнить:

Почему надо использовать кватернионы вместо углов Эйлера? Самое большое преимущество кватернионов – интерполяция. Кватернионы могут интерполироваться с помощью сферической линейной интерполяции (SLERP). Данный вид интерполяции позволяет найти кратчайший поворот на поверхности сферы. Точно так же можно повернуть объект из текущего вращения в новое по кратчайшему пути. Такого вы никогда не добьетесь с помощью интерполяции углов Эйлера. Непосредственное создание кватернионов и управление имиКласс Quaternion в Unity имеет ряд функций, которые позволяют создавать повороты и управлять ими без использования

углов Эйлера, и именно эти функции следует использовать в большинстве типичных случаев.

Метод Quaternion.LookRotation используется для поворота объекта в направлении указанного вектора. В методе Look он используется для определения направления взгляда персонажа в направлении движения.

Метод Quaternion.RotateTowards используется для плавного поворота объекта от его текущего положения до заданной ориентации. В методе Look он используется для плавного вращения персонажа в нужном направлении.

Rigidbody – это компонент в Unity, который позволяет объектам двигаться в пространстве под воздействием сил и гравитации. В данном случае Rigidbody управляет движением персонажа и перемещает его вперед по вектору направления в методе Move.

Matrix4x4 – это матрица 4x4, которая используется для преобразования точек в пространстве. Используется Matrix4x4 для преобразования вектора из обычной 3D– координаты в изометрическую плоскость в методе ToIso.

Таким образом, видно, что используются основные математические методы векторной алгебры и кватернионов, которые позволяют перемещать и вращать объекты в Unity в зависимости от входных данных игрока.

Теперь рассмотрим создание компонент, который помогает персонажу совершать рывок вперед. Dash – это компонент для игрового движка Unity, который позволяет реализовать способность быстрого перемещения персонажа по направлению к курсору мыши с помощью рывка (dash). Эта функциональность может быть полезна в играх с быстрым темпом, где игрокам нужно быстро перемещаться по уровням, избегая препятствий и врагов. Посмотреть компонент рывка можно в Приложение Б.

Данный скрипт использует следующие компоненты:

Rigidbody – компонент физического тела, используемый для перемещения персонажа.

ParticleSystem – компонент, используемый для создания эффектов частиц.

PlayerInput – класс, позволяющий получать ввод от пользователя.

Скрипт содержит два метода:

Метод DashM вызывается при нажатии игроком на кнопку "Dash" и устанавливает переменную "_isdashing" в значение "true", что означает, что персонаж находится в режиме dash. Кроме того, метод устанавливает время следующего dash, чтобы предотвратить повторный вызов этой функции слишком часто.

Метод Dashing отвечает за само перемещение персонажа в режиме dash. Он проверяет, есть ли препятствия на пути персонажа при помощи функции Physics.Raycast и, если их нет, перемещает персонажа вперед на фиксированную дистанцию, используя Vector3 и переменную "_isdashing". Кроме того, метод также воспроизводит звуковой эффект и эффект частиц при использовании dash. Рывок показан на рисунке 11.

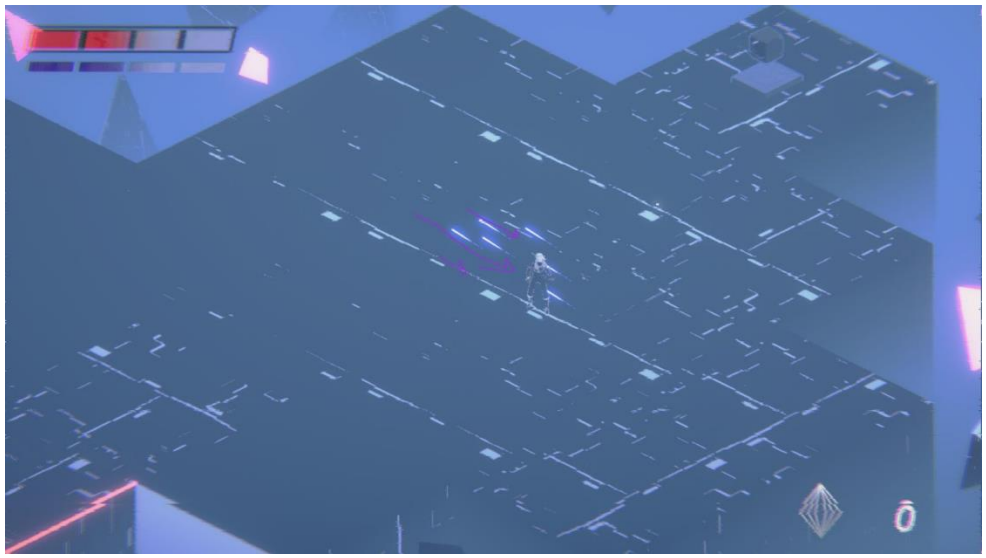


Рисунок 11 – Рывок

Далее мы немного углубимся в создание анимаций с помощью такой утилиты, как DoTween. Dotween – это библиотека, которая предоставляет простой и удобный способ создания анимаций, не требующий написания сложного кода.

С помощью Dotween разработчики могут создавать различные типы анимаций, такие как перемещение, вращение, масштабирование и изменение цвета объектов. Она также поддерживает создание петельных анимаций, использование эффектов затухания и плавности движения. Хотя для использования Dotween в Unity не требуется особого знания математики, но понимание принципов линейной алгебры и геометрии может значительно облегчить создание сложных и высококачественных анимаций. Векторы, матрицы и кривые Безье – все это является важными компонентами при работе с DoTween. Кроме того, знание математики может помочь разработчикам лучше понимать, как работает сама библиотека, и настраивать ее параметры для достижения нужных результатов.

Теперь на примере рассмотрим, как данная утилита может быть применена в разработке игры.

Для нашей первой задачи требуется, чтобы объект вращался и это было зациклено. Библиотека Dotween позволяет создавать анимации с помощью вызова методов на объектах типа Transform (используется для хранения значений позиции, вращения и т.д) и других классах Unity. В данном случае можно использовать метод DOLocalRotate (), который позволяет вращать объект относительно его локальных координат. Метод принимает вектор с тремя углами, задающими вращение по осям x, y, z. В данном случае мы можем настроить, чтобы объект вращался на 360 градусов по всем осям в течение 1.5 секунды. А метод SetEase используется для установки типа интерполяции, используемой для плавного изменения скорости анимации. В данном случае можно использовать Ease.Flash, который обеспечивает моментальную остановку анимации и мерцание на конечной точке каждой итерации. Результат виден на рисунке 12.

Для применения данного метода нужно иметь представление о математике вращения объектов в 3D– пространстве. Вектор с тремя углами, передаваемый методу, задает вращение объекта вокруг осей x, y и z. Понимание углов Эйлера и кватернионов, а также матричных преобразований

и композиции вращений может помочь в настройке сложных анимаций с использованием библиотеки Dotween. Кроме того, знание математики интерполяции может помочь выбрать наиболее подходящий тип интерполяции для достижения желаемого эффекта анимации.

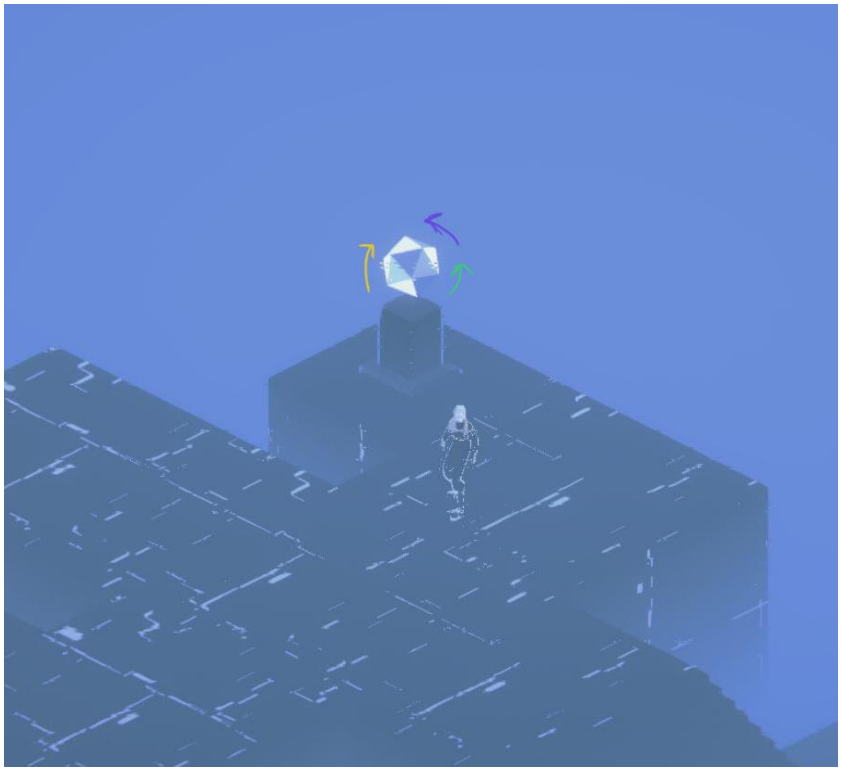


Рисунок 12 – Вращение объекта

Для задачи, где объект сначала поднялся по оси, затем произошла задержка и в самом конце началось зацикленное движение влево– вправо. В библиотеке DoTween есть специальный метод, с помощью которого можно объявлять различные анимационные последовательности и различные параметры времени и интервалов – Sequence. В нашем случае будет следующая последовательность действий: объект с помощью метода DOMove () плавно перемещается в заданную позицию, затем происходит пауза с помощью метода AppendInterval (), а затем объект снова перемещается в другую позицию с заданным временем и параметрами интерполяции. В этом случае объект перемещается туда и обратно с использованием методов

SetLoops () и SetEase () и продолжает обновляться с помощью метода SetUpdate (). Результат виден на рисунке 13.

Для использования DOTween в данных случаях не требуется прямое использование математических методов, однако важно понимать, что нужно знать основы векторной алгебры и не только. Например, во втором случае первая строка использует векторную алгебру для определения новой позиции объекта. Вторая строка использует метод AppendInterval для задержки на 2 секунды, а третья строка использует векторную алгебру для определения новой позиции объекта. Знание векторной алгебры поможет понять, как определяется новая позиция объекта, а знание принципов анимации и DOTween позволит понять, как создается и управляется анимация перемещения объекта. В сочетании с этим разработчики смогут добиваться результатов, которые им нужны всего – то несколькими строчками кода.

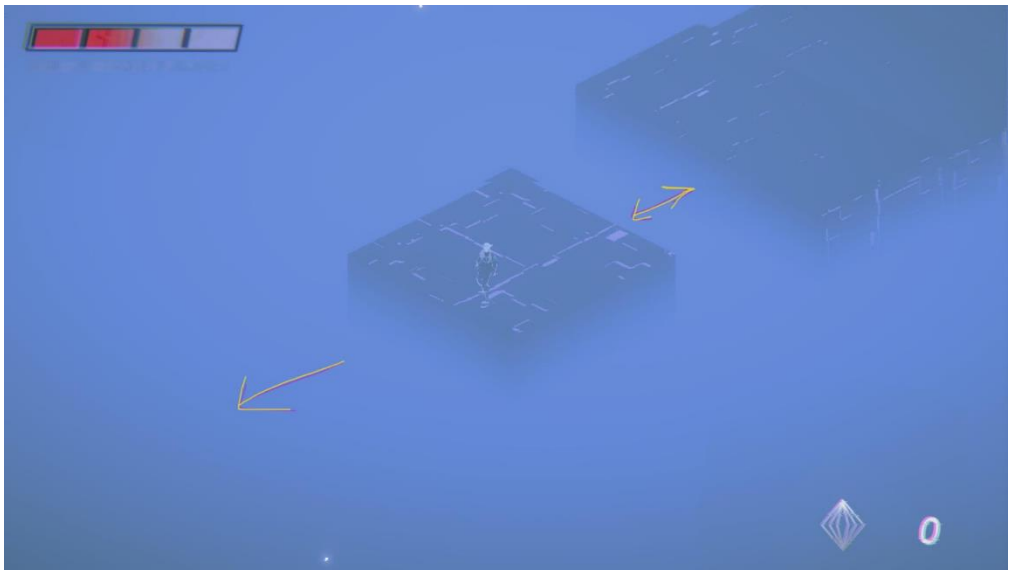


Рисунок 13 – Передвижение объекта

Процесс создание компонентов понятен. Теперь опишем все компоненты на игроке и как они взаимодействуют друг с другом. На рисунке 14 можно увидеть основные механики персонажа и какие компоненты потребовались для ее реализации.

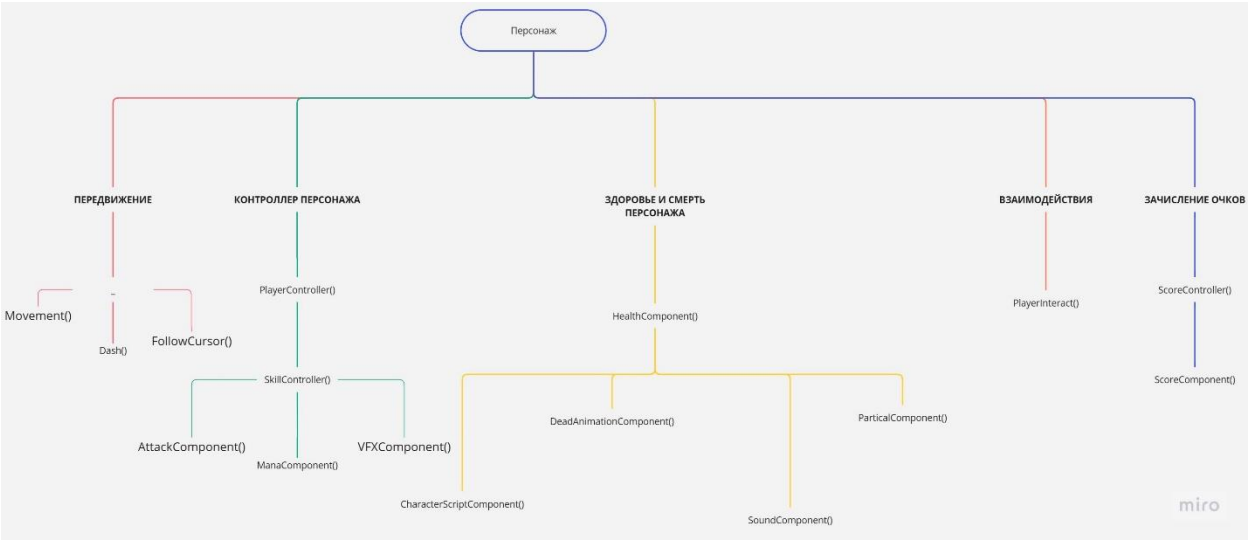


Рисунок 14 – Компоненты Персонажа

Передвижение. Компонент Movement уже был описан выше, поэтому на нем останавливаться не будем помимо него есть еще два компонента, которые отвечают за передвижение. FollowCursorотвечает за то, чтобы персонаж поворачивался по направлению курсора. Dash () позволяет реализовать способность быстрого перемещения персонажа по направлению к курсору мыши с помощью рывка (dash).

Контроллер Персонажа. Он отвечает за умения персонажа и следит за нажатыми игроком клавишами. Для этого контроллер состоит из двух компонентов: PlayerController и SkillController. Первый следит за нажатой клавишей, а второй определяет, какое именно умение будет использоваться по данной клавише. SkillController также контролирует запуск анимации, расход маны и запуск эффектов. Он дает указания AttackComponent, ManaComponent и VFXComponent, которые отвечают за выполнение соответствующих действий. Таким образом, контроллер персонажа является ключевым элементом игровой механики, позволяющим игроку управлять персонажем и выполнять необходимые действия.

Здоровье и смерть персонажа. Он отвечает за проигрыш персонажа, то есть смерть или получения урона. HealthComponent представляет собой

компонент здоровья, который может быть прикреплен к игровому объекту в Unity. Он содержит переменные для текущего здоровья и максимального здоровья, а также методы для получения урона и увеличения здоровья. В скрипте есть метод TakeDamage, который принимает в качестве параметра количество урона, наносимого объекту, и вычитает это значение из текущего здоровья. Если здоровье становится равным 0, вызывается событие onDead. При каждом получении урона вызывается событие onDamage. Вызываемые события находятся в скриптах, которые находятся ниже по ветке.

Взаимодействие. Помимо взаимодействий с помощью умений персонажа, нужно, чтобы было взаимодействие игрока с объектами в окружающей среде. Например, перемещение между уровнями, начало диалогов и т.д все это можно сделать с помощью компонента PlayerInteract. В методе Update он проверяет, была ли кнопка "E" нажата в этом кадре, и если была, то получает ближайший объект, с которым может взаимодействовать игрок (реализующий интерфейс IInteract). Если такой объект найден, вызывается метод Interact () на этом объекте, передавая в него трансформ игрока. Метод GetInteract отвечает за поиск ближайшего объекта взаимодействия в радиусе заданной величины (interactRange). Он использует метод Physics.OverlapSphere для поиска всех коллайдеров в заданном радиусе от позиции игрока. Затем он проверяет, реализует ли каждый коллайдер интерфейс IInteract, и добавляет его в список. Затем метод находит ближайший объект из списка по расстоянию и возвращает его.

Зачисление очков. ScoreController отвечает за управление очками игрока. В скрипте определены две переменные – scoreCube и scoreRadasock, отвечающие за количество очков, получаемых за уничтожение куба и радасока (врагов) соответственно. При начале игры в методе OnEnable скрипта происходит подписка на события, которые возникают при уничтожении куба и радасока (эти события определены в компоненте ScoreComponent.

Метод IncreaseCube увеличивает текущий счет на значение scoreCube при уничтожении куба. Аналогично, метод IncreaseRadasock увеличивает

текущий счет на значение scoreRadasock при уничтожении радасока. При выключении объекта, к которому привязан данный скрипт, происходит отписка от событий уничтожения куба и радасока (врагов) в методе OnDisable. Текущее количество очков можно получить через свойство score.

Теперь перейдем к описанию врагов в игре, которые играют важную роль. В игре реализован искусственный интеллект для противников. В нашем случае было использовано дерево состояний, чтобы реализовать эту логику. Что такое вообще дерево состояний и почему мы его использовали для решения нашей задачи?

Конечный автомат или дерево состояний – это математическая модель, используемая для описания последовательности действий и состояний объекта в ответ на внешние воздействия.

Дерево состояний представляет собой графическую структуру, состоящую из узлов, которые представляют состояния объекта, и ребер, которые связывают состояния и определяют, как объект должен переходить из одного состояния в другое. Каждое состояние объекта имеет определенные свойства и поведение, которые описываются при помощи кода. При использовании дерева состояний для реализации искусственного интеллекта в игре каждый враг будет иметь свое собственное дерево состояний. Это дерево будет определять поведение врага, основываясь на текущей ситуации в игре. Когда враг находится в определенном состоянии, он может выполнять определенные действия и переходить в другое состояние, если существуют соответствующие условия.

Использование дерева состояний позволяет создавать гибкую логику поведения врагов в игре и управлять ими с помощью простых правил, что делает процесс разработки игры более простым и эффективным.

Перейдем к описанию. На рисунке 15 можно увидеть основные механики врага, а также с помощью каких компонентов они были реализованы.

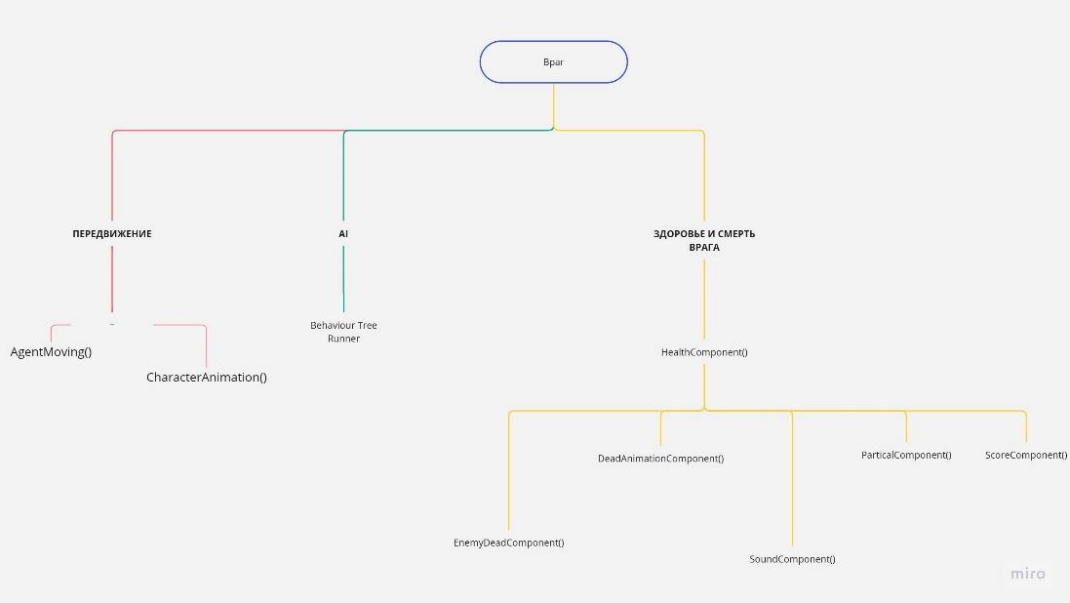


Рисунок 15 – Компоненты Врага

Компонент HealthComponent мы рассматривали выше у персонажа игрока здесь логика такая т. к. компонент сам по себе универсальный. Акцентируем большее внимание на AI (artificial intelligence) здесь находится переход на дерево состояний, которое можно увидеть на рисунке 16.

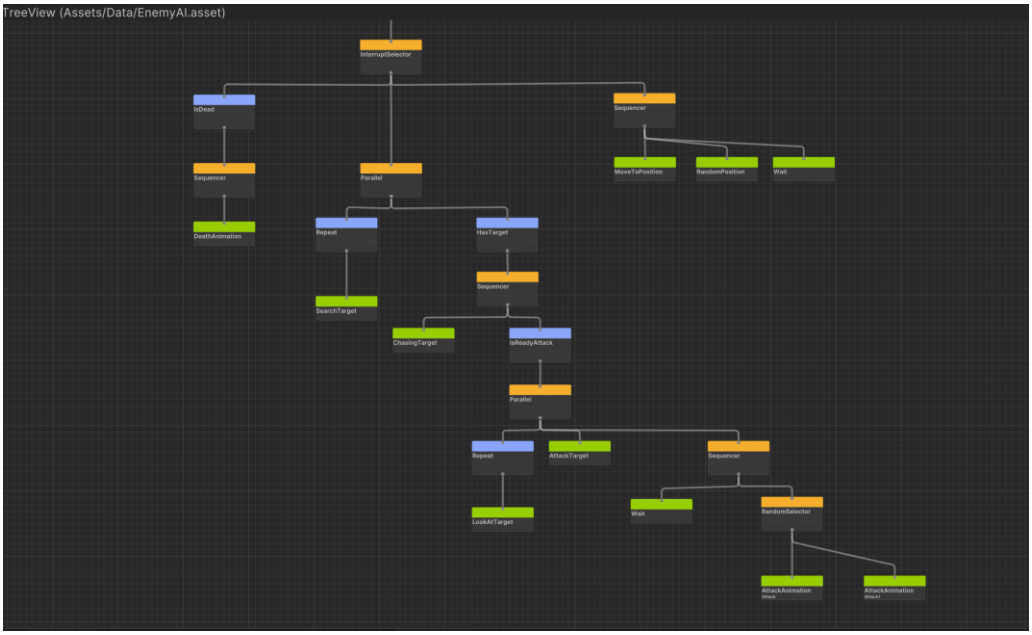


Рисунок 16 – Дерево состояний врага в среде движка

Каждая ячейка дерева состояний представляет собой отдельный компонент, ответственный за выполнение определенной логики. Некоторые компоненты отвечают за определенные условия, при выполнении которых происходят определенные действия, а другие – за частоту повторения проверок условий и переход на другие ветки дерева.

Давайте подробнее разберем принцип действия дерева состояний и его влияние на поведение врага в игре. В первой ветке дерева происходит проверка на смерть врага с помощью компонента HealthComponent. Если здоровье врага равно нулю, то запускается анимация смерти, а две последующие ветки дерева не выполняются.

Во второй ветке дерева происходит поиск игрока. Если персонаж находится в радиусе действия врага, то последний начинает приближаться к нему и, когда расстояние между ними становится достаточно маленьким, начинает атаковать. Если расстояние между ними снова увеличивается, то враг сначала должен снова приблизиться к игроку.

В третьей ветке дерева происходит так называемое "патрулирование". Если враг не обнаружил игрока, то случайным образом выбирается точка, куда враг должен отправиться, затем он ждет несколько секунд и вновь начинает двигаться к случайно выбранной точке. Для выбора точки при патрулировании врага используется локальная область вокруг него, определяемая допустимым радиусом. Только в пределах этой зоны может быть выбрана случайная точка на основе положения врага.

Теперь, когда описаны основные механики персонажа и врага мы можешь понять, как должен выглядеть основной цикл. Его можно увидеть на рисунке 17.

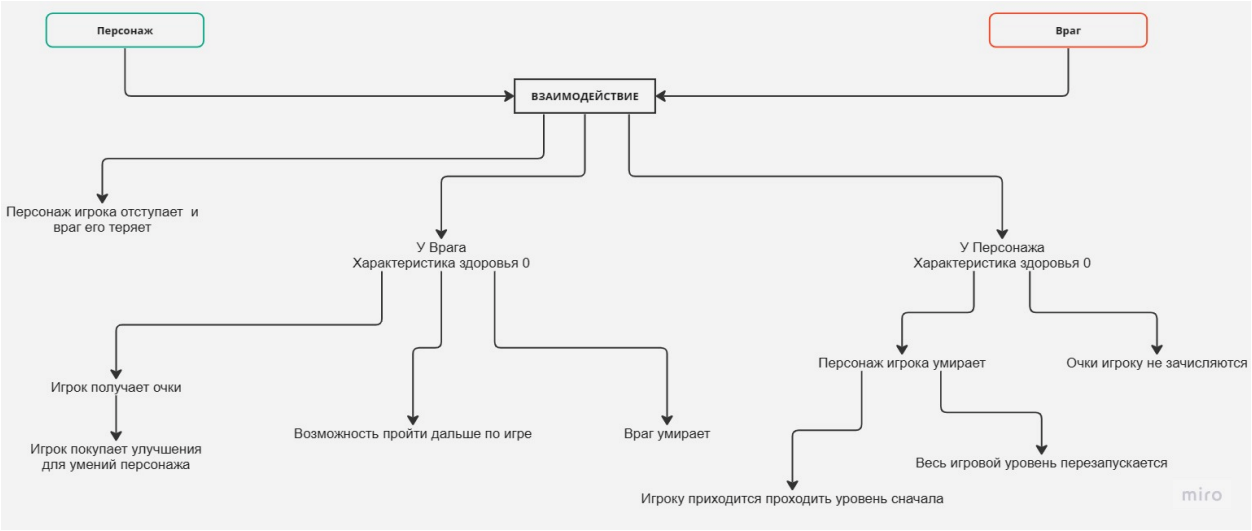


Рисунок 17 – Схема взаимодействия персонажа игрока и врага

Таким образом, была описана разработка компонентов для персонажа и врага, а также их взаимодействия. На основе этого можно сделать вывод, что разработанная игровая логика является достаточно универсальной для обеих сторон и обеспечивает эффективное взаимодействие между персонажем и врагом. Кроме того, реализация дерева состояний для искусственного интеллекта врага является инновационной и может быть полезна для будущих разработок в данной области.

2.6 Изометрическая камера

Изометрическая камера – это специальный тип камеры, который используется в играх для создания эффекта трехмерности в двухмерном игровом мире. В данном разделе я расскажу о том, как я использовал изометрическую камеру при разработке моей игры, а также какие преимущества и недостатки она имеет.

Изометрическая камера используется для создания иллюзии трехмерного пространства в двухмерном игровом мире. Она создает изображение, которое позволяет игроку видеть игровой мир с углом обзора сверху, при этом сохраняя вертикальные линии и не искажая геометрические

пропорции. Изометрическая камера часто используется в играх жанра стратегии, где игроку необходимо управлять большим количеством объектов на экране, таких как юниты, здания, ресурсы и т. д. Как выглядит обычный куб в изометрической проекции можно увидеть на рисунке 18.

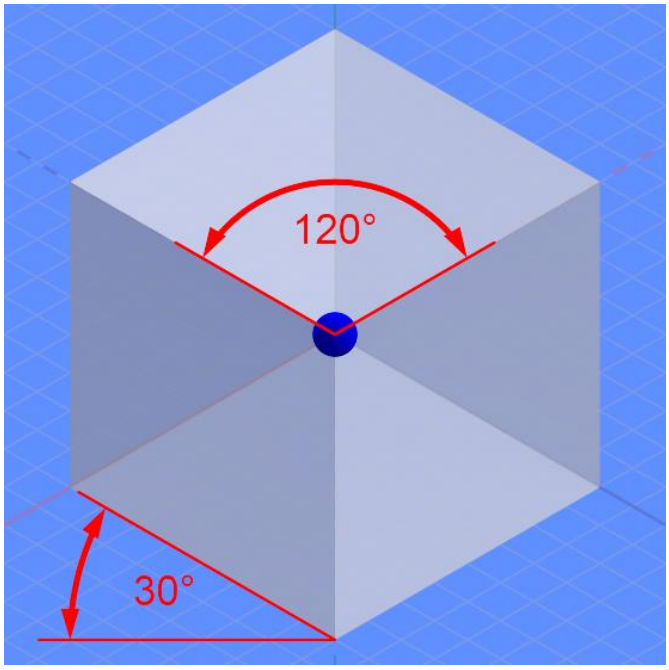


Рисунок 18 – Куб в изометрической проекции

Изометрическая камера имеет ряд преимуществ, которые делают ее популярной среди разработчиков игр. Первое, она позволяет создавать очень детализированные и красочные игровые миры, которые выглядят очень реалистично и объемно. Второе, она обеспечивает игроку лучший обзор на игровое поле, что очень важно в играх жанра стратегии, где игроку необходимо иметь полное представление о ситуации на поле боя. Третье, изометрическая камера позволяет создавать игры с низкими требованиями к железу, что делает ее привлекательной для игроков с устаревшим оборудованием.

Недостатки изометрической камеры. Несмотря на все преимущества, изометрическая камера также имеет некоторые недостатки. Первое, она может создавать некоторые проблемы с точностью позиционирования объектов на

экране, что может приводить к ошибкам в игре. Второе, из-за особенностей угла обзора изометрической камеры, игрок может иметь затруднения с определением расстояний и размеров объектов на игровом поле. Также, из-за особенностей изометрического вида, возможны проблемы с управлением персонажами, особенно в тесных пространствах, где может быть сложно определить точные позиции и направления движения персонажей.

В разрабатываемой игре используется изометрическая камера. В начале разработки данная камера была выбрана из-за того, что она подходила для многих визуальных решений, а также реализация игры с таким видом камера казалась наименее сложной, нежели с камерой от третьего лица.

Для реализации изометрической камеры в игре использовались следующие аспекты:

В предыдущих версиях камеры использовалась ручная настройка основной камеры, а через компонент CameraController настраивались другие необходимые параметры: слежение камеры за игроков, приближение и отдаление камеры. Однако данная настройка пусть и была оптимальной на тот момент, однако не предлагала никакой гибкости, поэтому, когда нашлась альтернатива было принято решение перейти на нее.

Cinemachines – это стандартный пакет от Unity, который управляет камерой. Он может следить за персонажем, работать в катсценах, накладывать эффект дрожания камеры, делать движения камеры плавными, ограничивать ее движения в пространстве, делать "орбитальную" камеру для игры от 3– го лица, от первого лица и многое, многое другое.

С помощью новых настроек, которые теперь доступны в несколько кликов, работа стала более простой и удобной. Однако, чем больше функционала, тем больше нужно разбираться в его работе и понимать, как он взаимодействует с другими аспектами игры, чтобы избежать возможных багов и визуальных артефактов.

Дабы Cinemachines работал в нашей игру необходимо сделать следующие действия. На основную камеру повесить компонент CinemachinesBrain. Он показан на рисунке 19.

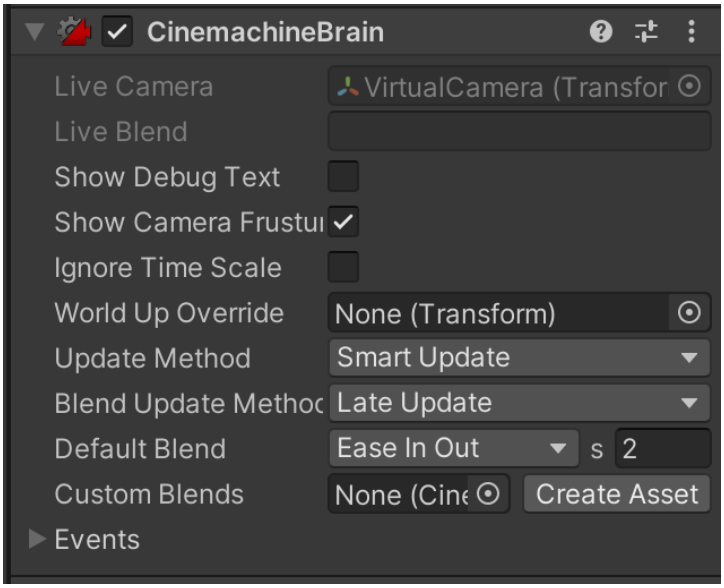


Рисунок 19 – CinemachinesBrain

Далее автоматически появляется объект VirtualCamera, на котором находится компонент CinemachinesVirtualCamera. С помощью него мы как раз таки сможем настраивать камеру, как нам нужно. На данный момент было настроено следующие: слежение за персонажем, изометрическая перспектива, тряска камеры и другие аспекты. Сам компонент можно увидеть на рисунке 20.

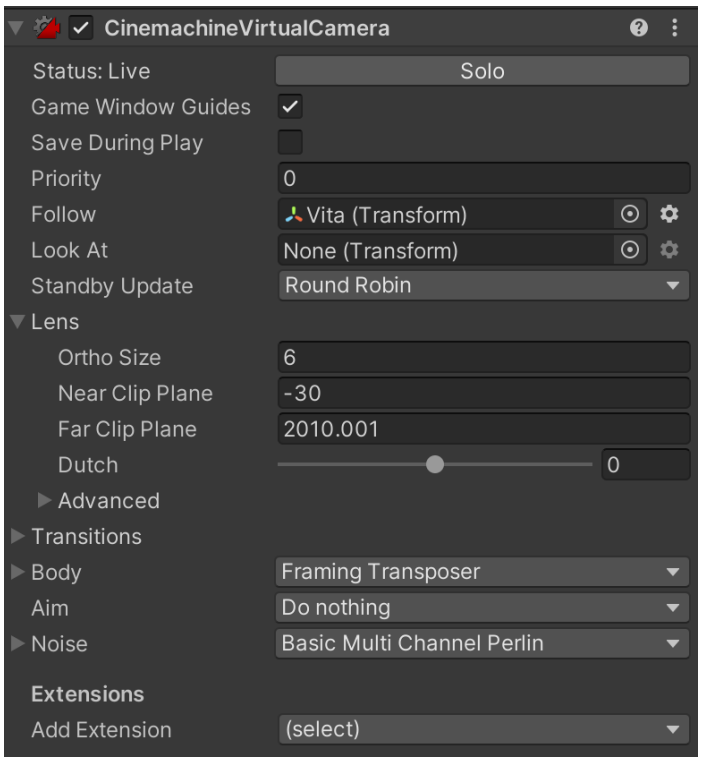


Рисунок 20 – CinemachinesVirtualCamera

Таким образом, использование изометрической камеры позволяет создавать красивые и детализированные игровые миры с эффектом объемности и глубины. А благодаря такому инструменту, как Cinemachines мы смогли сделать более качественно.

2.7 Визуальная составляющая

Конечная цель разработки игры – достижение высокого уровня визуального качества графики, которая будет в полной мере отображать и передавать задуманную атмосферу и настроение игры. Для достижения этой цели используется графический движок Unity, который предоставляет разработчикам мощный инструментарий для создания красивой и эффектной графики.

Один из ключевых компонентов визуальной составляющей игры – это графические ресурсы, которые используются для создания 3D моделей персонажей, врагов, окружения и других объектов в игре. Для этого в игре

используется инструментарий Unity для создания и импорта моделей, который позволяет создавать высококачественные 3D объекты с высоким уровнем детализации.

Важным аспектом графической составляющей является также освещение и шейдинг. Unity позволяет создавать реалистичное освещение и использовать различные техники шейдинга, такие как бамп– мэппинг, отражения и преломления, которые способствуют созданию эффектной и красивой графики.

Наконец, Unity обеспечивает возможности для создания эффектов, таких как частицы, взрывы, дым и другие, которые добавляют дополнительную атмосферность и динамизм в игру.

Разработанный визуальный стиль игры можно увидеть на рисунке 21.

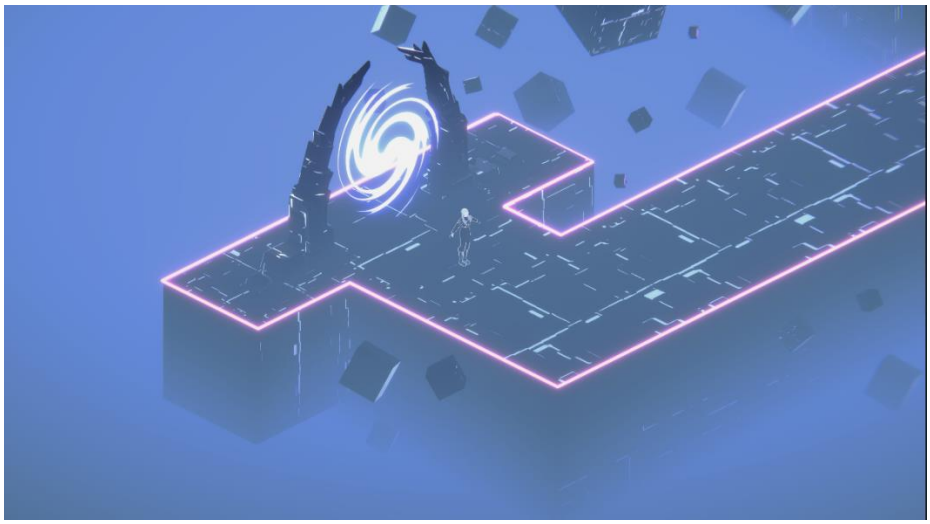


Рисунок 21 – Визуальная составляющая

Для достижения окончательного визуального стиля нашей игры в Unity, было использовано множество графических элементов, таких как текстуры, нормали, шейдеры, материалы и т. д. Каждый из этих элементов был тщательно разработан и настроен для достижения наилучшего эффекта. Разберем несколько визуальных элементов из рисунка 21.

Первостепенное значение в создании атмосферы игры имеют PostProcessing и шейдер тумана. Постобработка – это метод обработки изображения, который добавляет дополнительные эффекты к буферу изображения перед его отрисовкой на экране, чтобы улучшить качество изображения. В нашей игре было использовано множество эффектов, таких как цветовая обработка изображения, свечение, глитчи, гамма и другие. Это видно на Рисунке 22, где мы сравниваем изображения без и с постобработкой.

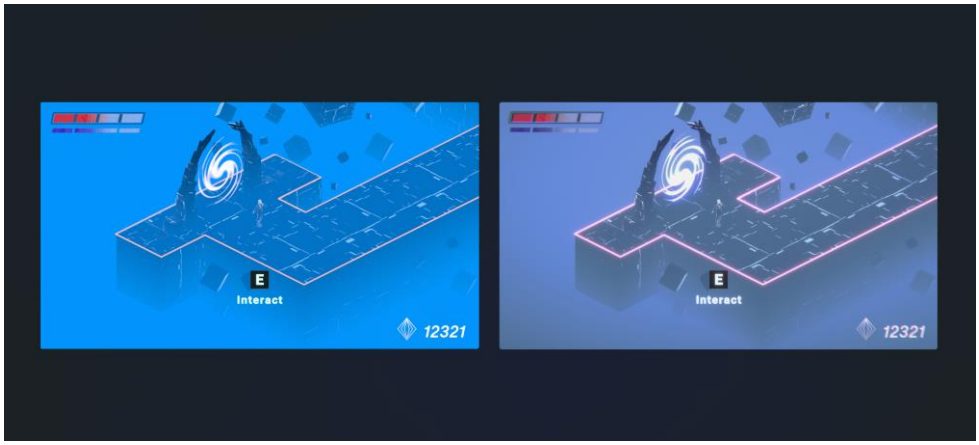


Рисунок 22 – Изображение без и с постобработкой

Шейдер тумана также играет важную роль в создании атмосферы игры. Он добавляет объемность и глубину к изображению, создавая иллюзию тумана, что делает игровой мир более реалистичным и захватывающим. Насколько туман важен видно на рисунке 23, где мы сравниваем изображения без и с туманом.

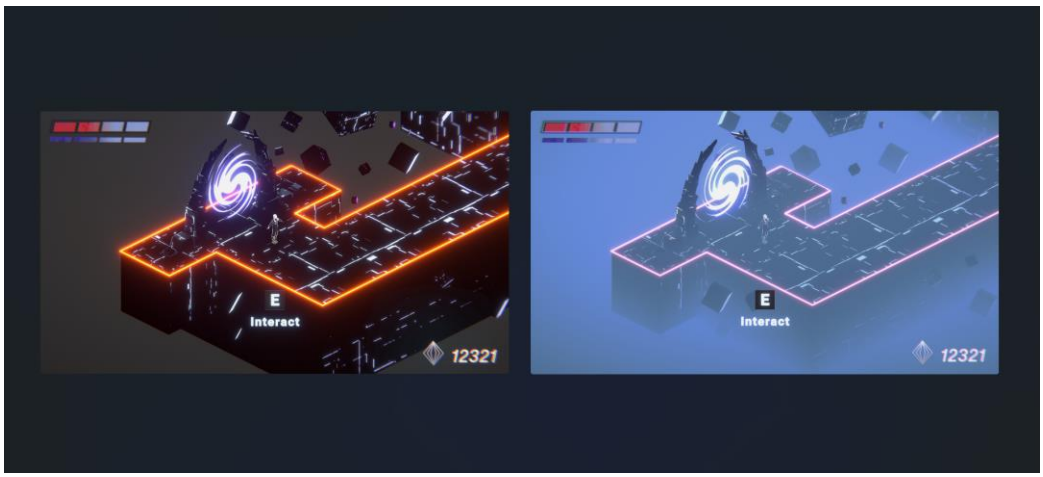


Рисунок 23 – Изображение без и с туманом

Эти два элемента – PostProcessing и шейдер тумана – существенно повысили качество визуальной составляющей нашей игры и сделали ее более привлекательной для игроков. Дабы убедится в этом наверняка на рисунке 24 можно увидеть изображение с постобработкой и туманом и изображение без них.

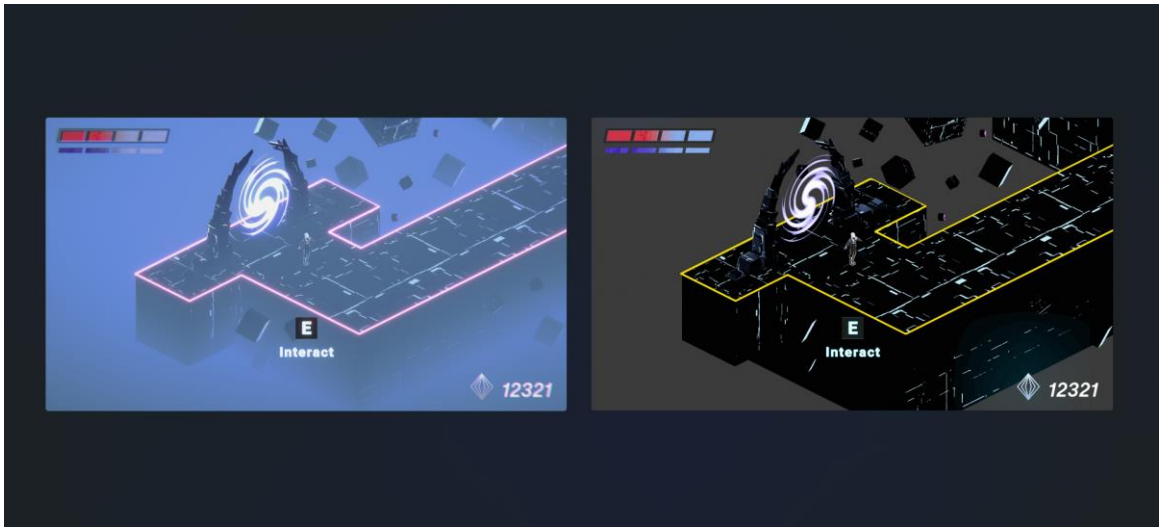


Рисунок 24 – Изображение с туманом и постобработкой и изображение без них

На рисунке 21 можно увидеть портал, который, как и туман был создан с помощью Шейдер Графа (Shader Graph). Shader Graph позволяет с легкостью разрабатывать шейдеры с обновлением результатов в реальном времени.

Графовая система передает инструмент в руки художникам и другим сотрудникам – достаточно просто соединить узлы в блок– схему. На рисунке 25 можно увидеть, как выглядит графовая система для портала:

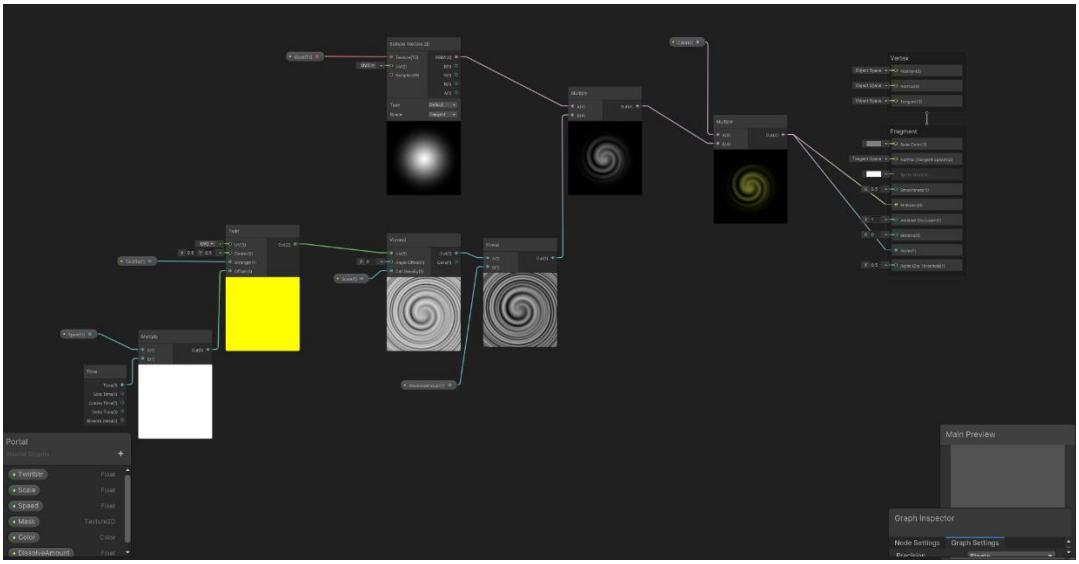


Рисунок 25 – Графовая система для портала

Для того чтобы передать на экране компьютера наглядное представление о мире, который мы создали, необходимо разработать визуальную составляющую игры. Это может включать в себя различные элементы, такие как 3D модели персонажей и объектов, текстуры, освещение и эффекты. Одним из ключевых инструментов для создания визуальной составляющей является Toon шейдер.

Toon – это специальный вид шейдеров, который используется для создания анимационного или комиксного эффекта в играх и анимациях. Он также известен как шейдер Cel, поскольку его основной целью является создание визуального эффекта, похожего на рисованный стиль, который можно часто встретить в аниме и мультфильмах. Один из его простейших реализаций можно увидеть на рисунке 26.

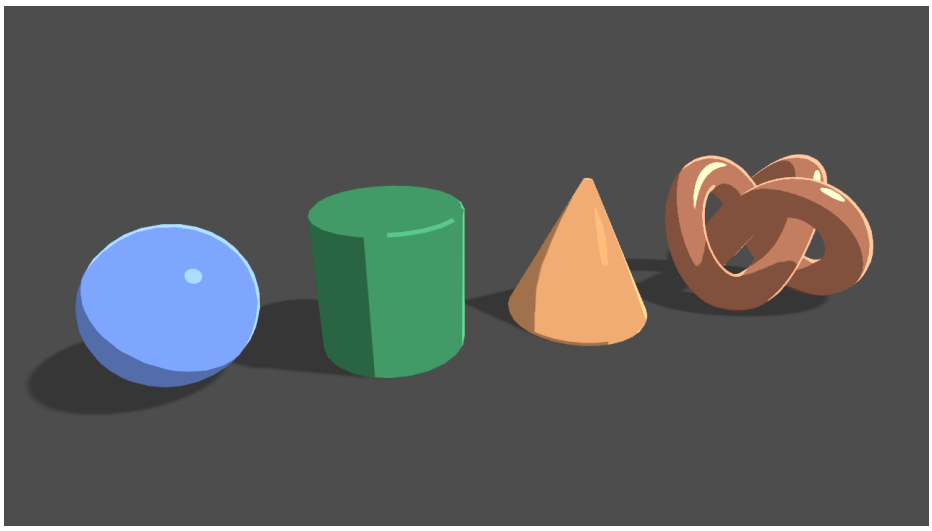


Рисунок 26 – Простой ToonShader в Unity

Шейдер Toon применяется для создания различных эффектов, таких как затенение, обводка и текстурирование, которые помогают создать живые и выразительные персонажи и объекты в игре. Затенение при использовании шейдера Toon имитирует теневые пятна на объектах, что добавляет им объема и глубины. Обводка или контурный рисунок придает персонажам и объектам на экране более четкий и выразительный вид, что помогает им выделяться на фоне остальных элементов игры. Кроме того, текстурирование с помощью шейдера Toon позволяет создавать различные текстурные эффекты, которые помогают дополнительно усилить визуальную составляющую игры.

Использование шейдера Toon также позволяет достичь определенного стиля визуального дизайна игры, что может быть особенно полезно для разработчиков, которые стремятся создать уникальный и запоминающийся образ игры. Например, использование шейдера Toon может быть эффективным способом создания игрового мира, который отражает определенную тематику или эмоциональный настрой, такой как комедия, мистика или приключения.

В целом, шейдер Toon – это мощный инструмент для создания уникальной и запоминающейся визуальной составляющей игры. Он позволяет разработчикам создавать живые и выразительные персонажи и объекты, а

также достигать определенного стиля визуального дизайна игры. При правильном использовании шейдер Toon может помочь сделать игру более привлекательной и увлекательной для игроков.

Еще одной важной частью визуальной составляющей является обводка (outlines). Она представляет собой черный контур, который обрамляет персонажей и объекты в игре, что позволяет им выделяться на экране и делает картинку более читабельной. Обводка также может использоваться для передачи определенного настроения или стиля игры. Например, в играх жанра нуар или киберпанк обводка может быть более тонкой и детализированной, чтобы создать атмосферу мрачности и таинственности.

На рисунке будет показан скриншот, как ToonShader реализован в нашей игре. Справа можно увидеть материалы, на каждом из которых находится шейдер. Стоит отметить, что каждый из этих материалов был настроен вручную т. к. определенный материал содержит свою текстуру, поэтому универсальной настройки нет.

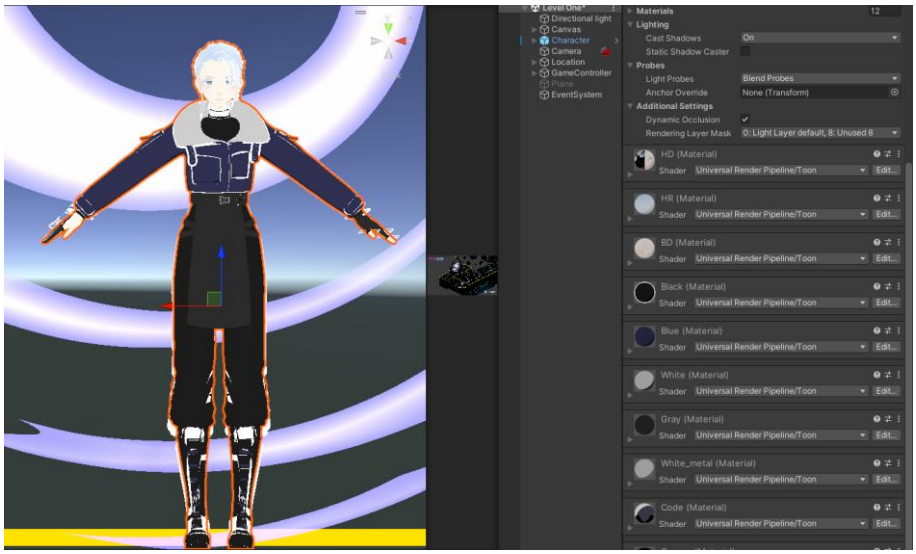


Рисунок 27 – ToonShader на персонаже

Заметно, что на персонаже обводка выходит за контуры персонажа, однако по этому поводу не стоит переживать т. к. в игре изометрическая камера. Видно, что на рисунке 21 все смотрится хорошо.

На рисунке 28 можно увидеть, как тот же шейдер выглядит на враге.

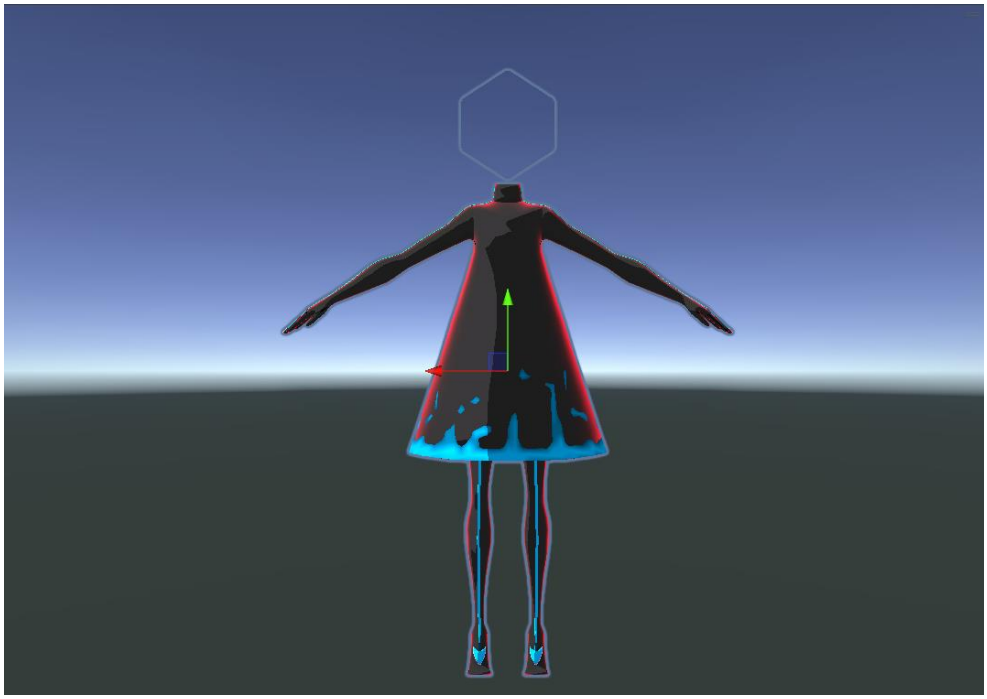


Рисунок 28 – Шейдер на Враге

Конечно, визуальная составляющая игры не ограничивается только туннельным шейдером и обводкой. Например, может быть использована анимация, чтобы сделать персонажей более живыми и реалистичными. Кроме того, можно использовать различные эффекты, такие как вспышки и взрывы, чтобы сделать игру более динамичной и захватывающей. Важно помнить, что визуальная составляющая игры – это один из ключевых факторов, который может сделать игру привлекательной и увлекательной для игроков.

Похожим образом создавалась вся визуальная составляющая игры. Была показана лишь небольшая часть всей проделанной работы.

3 Тестирование и результаты

Для проверки качества игры было проведено тестирование среди группы добровольцев. Опрос проводился в формате Google Формы. Участникам предлагалось оценить игру по различным критериям, таким как геймплей, графика, звуковое сопровождение и общее впечатление. В результате тестирования были получены следующие данные.

Игровой процесс был оценен достаточно положительно также видна оценка сложности игры на рисунке 29:

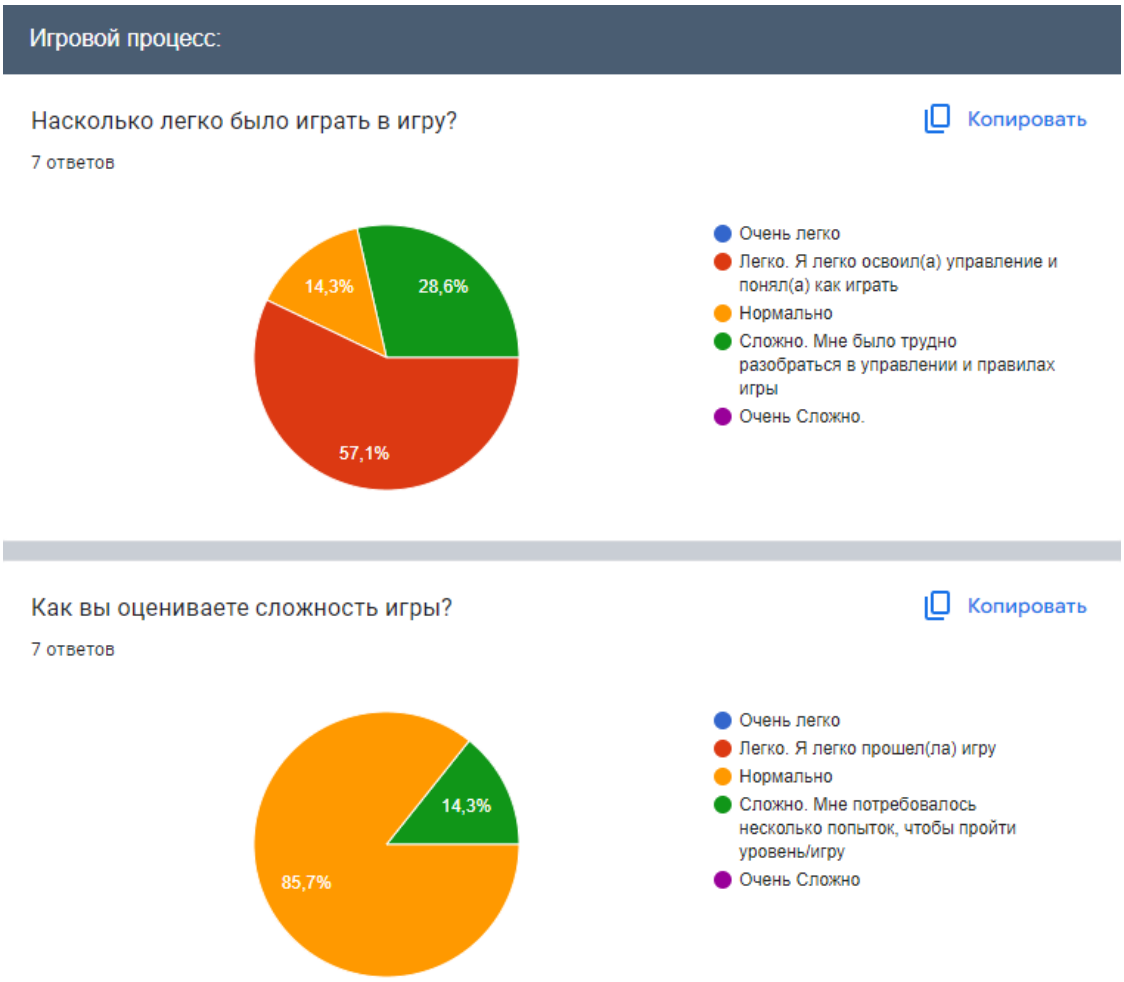


Рисунок 29 – Оценка игрового процесса и сложности

Оценка геймплея, механик и чувство импакта от оружия можно увидеть на рисунке 30.

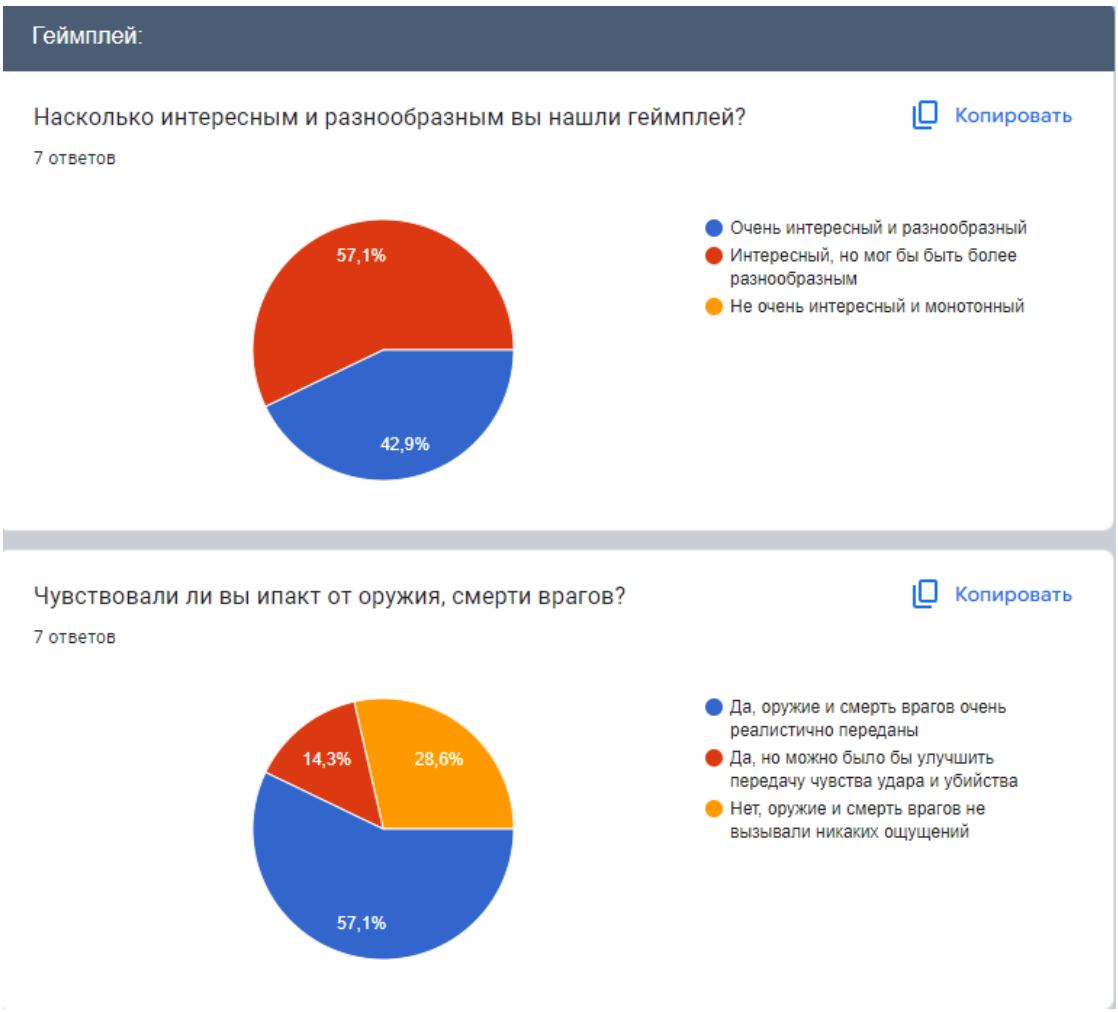


Рисунок 30 – Оценка геймплея и импакта

Оценку графики и дизайна игры можно увидеть на рисунке 31. Как видно второй пункт имеет достаточно противоречивые оценки, однако это лишь очередной повод больше уделить внимания тем или иным аспектом в создаваемой игре.

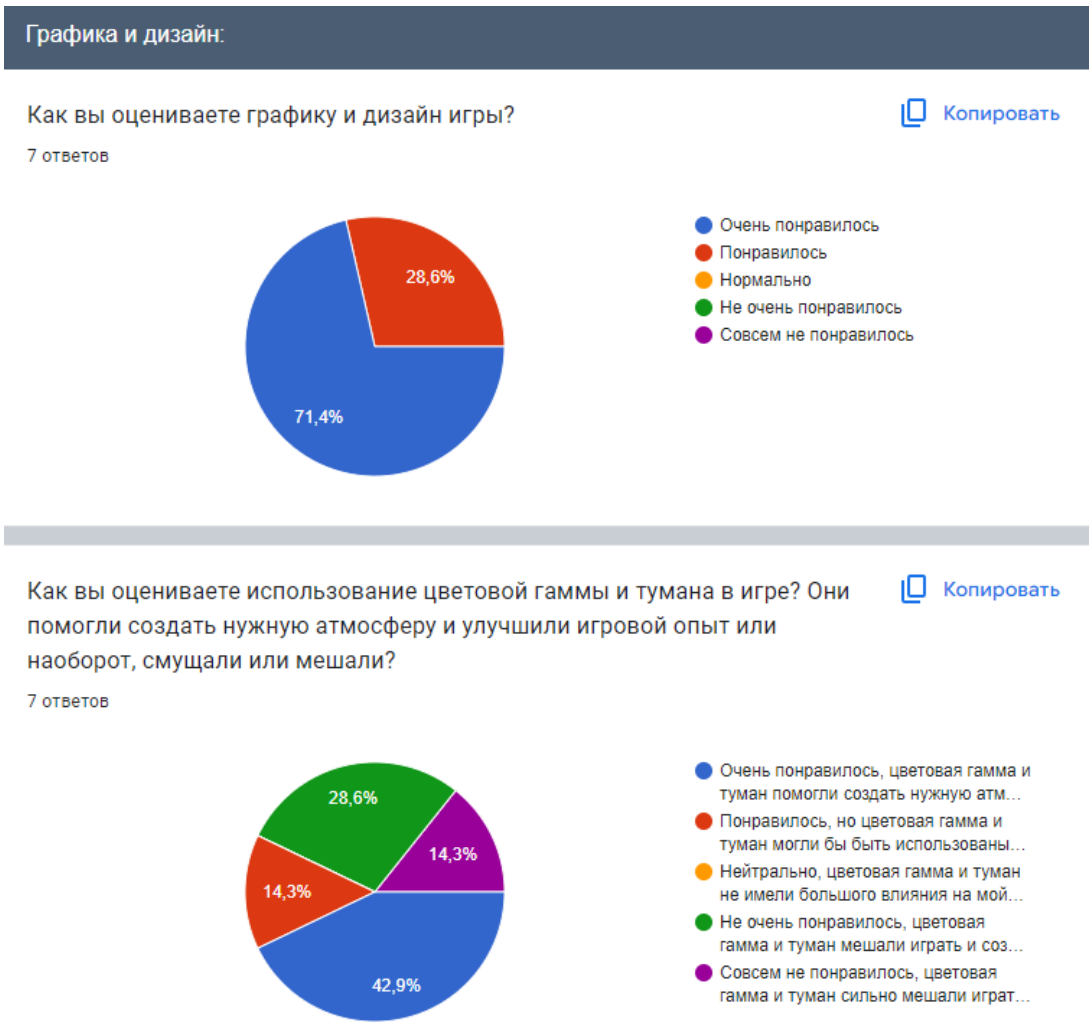


Рисунок 31 – Оценка графики и дизайна

Оценку музыки и звуковых эффектов можно увидеть на рисунке 32.

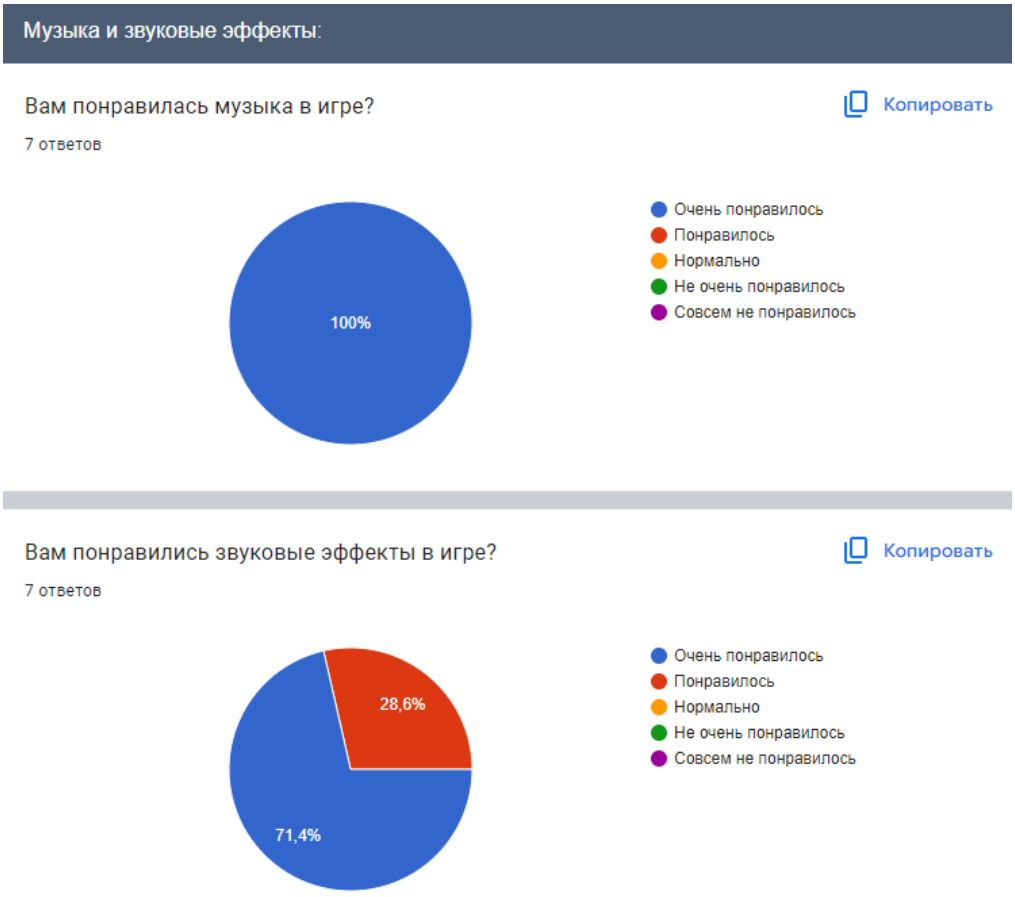


Рисунок 32 – Оценка музыки и звуковых эффектов

Оценку игрового интерфейса можно увидеть на рисунке 33. Также можно увидеть достаточно противоречивые оценки, а это значит, что в будущем игровой интерфейс будет доработан, исходя из оценок.

Рисунок 33 – Оценка игрового интерфейса

Оценку игрового сюжета и персонажей, врагов можно увидеть на рисунке 34.

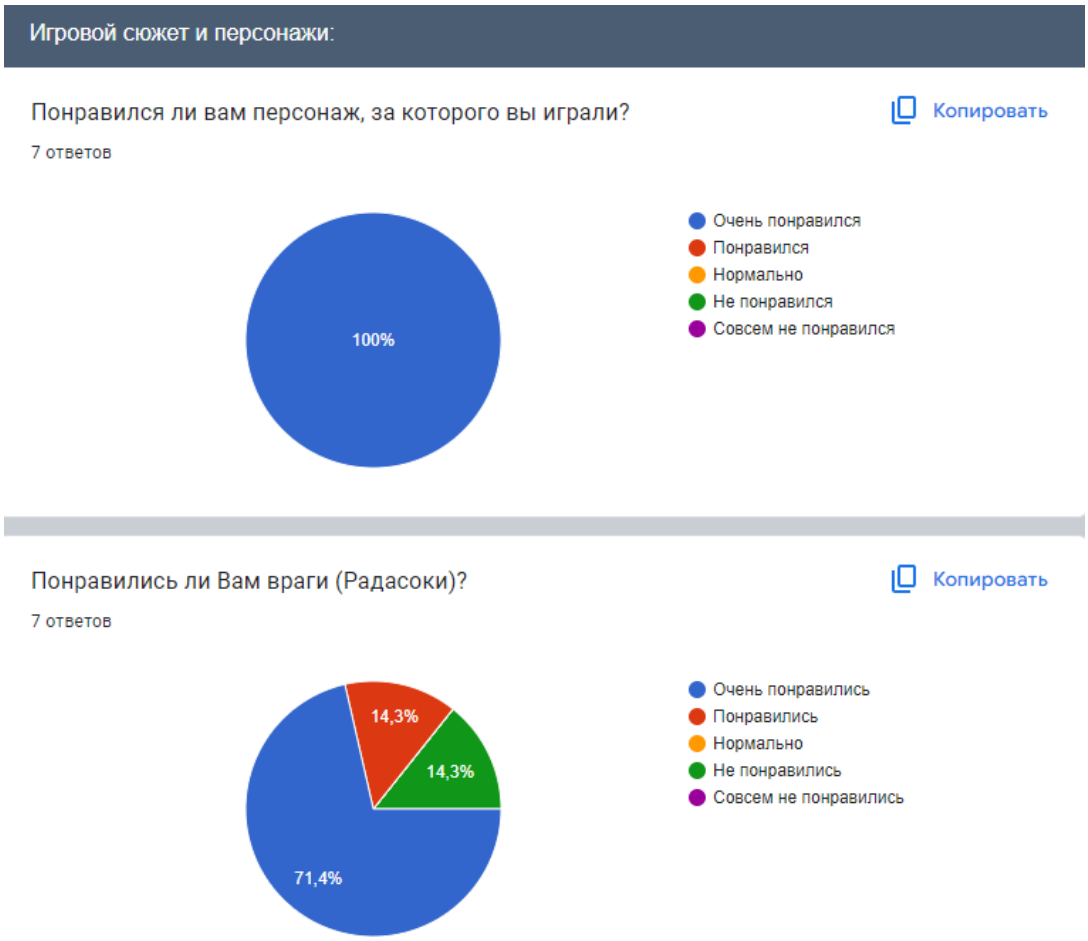
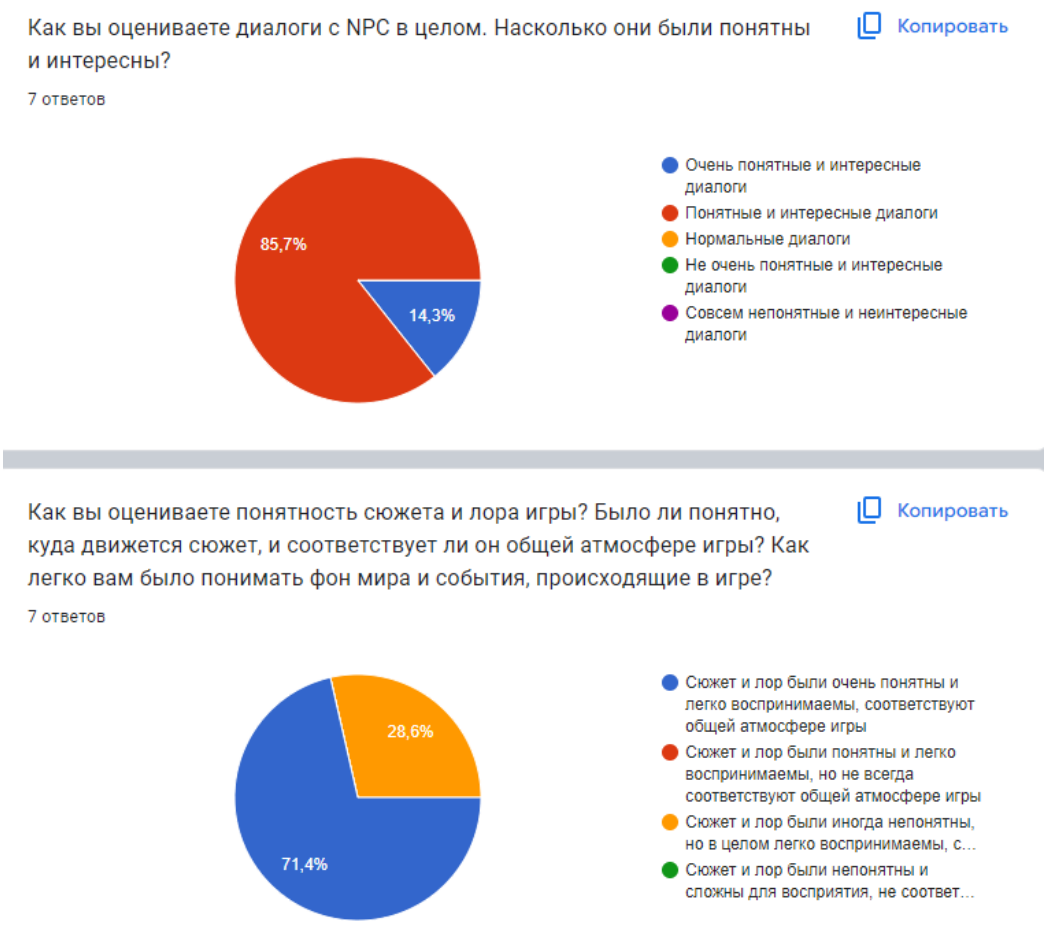


Рисунок 34 – Оценка игрового сюжета и врагов

Оценку диалогов с NPC и понятности сюжета и истории в целом можно увидеть на рисунке 35.



Рисунки 35 – Оценка NPC и понятность сюжета

Оценку общей атмосферы в игре можно увидеть на рисунке 36.

Рисунок 36 – Оценка атмосферы в игре

И наконец оценку общего впечатления от игры можно увидеть на рисунке 37. Исходя из нее, можно сделать вывод, что игра понравилась людям, а значит ее можно развивать в будущем.

Рисунок 37 – Оценка общего впечатления пользвателей

Большинство участников положительно оценили геймплей и графику, отметив красочность и качество изображения. Звуковое сопровождение также было хорошо оценено, что дало возможность углубиться в атмосферу игры.

Несмотря на высокие оценки, были также выявлены некоторые проблемы, которые будут исправлены в будущих версиях игры. Одной из таких проблем было некоторое затруднение с игровым процессом в начальных уровнях. Некоторые участники также отметили небольшое количество багов в игре, что является нормальным явлением для игр в стадии разработки.

Общее впечатление от игры было положительным, большинство участников выразили желание играть в игру еще раз и рекомендовали ее своим друзьям.

Таким образом, результаты тестирования показали, что игра имеет хороший потенциал и может заинтересовать широкую аудиторию.

ЗАКЛЮЧЕНИЕ

В рамках данной дипломной работы была успешно разработана изометрическая игра с использованием современных технологий и инструментов разработки. Были применены следующие средства разработки: Unity, как игровой движок и среда разработки; C#, как язык программирования; Adobe Photoshop, как графический редактор; Bledner, как программное обеспечение для создания трёхмерной компьютерной графики; FMOD, как программное обеспечение для работы со звуком и музыкой.

Цель работы была достигнута путем создания увлекательного игрового опыта, который привлекает внимание игроков и предлагает интересные игровые механики. В ходе разработки были реализованы следующие основные функциональности: управление персонажа, управление врага или же его искусственный интеллект, взаимодействие между игровыми объектами.

Важным аспектом работы являлось тестирование разработанной игры для обеспечения ее качества и стабильной работы. Были проведены тесты, включающие проверку функциональности, исправление ошибок и оптимизацию производительности. Тестирование позволило выявить и устранить возможные проблемы, обеспечивая более гармоничный игровой процесс.

В результате выполненной работы была создана уникальная изометрическая игра, которая предлагает игрокам захватывающий игровой опыт, увлекательные задания и визуально привлекательную графику. Разработанная игра демонстрирует потенциал и возможности современных технологий в области разработки игр.

Таким образом, данная дипломная работа является успешным примером создания изометрической игры с применением современных средств разработки и тщательным тестированием. Полученные результаты подтверждают достигнутые цели и открывают перспективы для дальнейшего развития и улучшения игровых проектов в будущем

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Хокинг, Д. Unity в действии: Мультиплатформенная разработка C# / Д. Хокинг. – СПб.: Питер, 2019. – 352 с. – ISBN 978-5-4461-2266-0.
2. Бонд, Д. Г. Unity и C#: Геймдев от идеи до реализации / Д. Г. Бонд – СПб.: Питер, 2019.– 928с. – ISBN 978-5-4461-0715-5.
3. Unity Manual // Unity: [сайт]. – 2023. – URL: https://docs.unity3d.com/Manual/index.html (дата обращения: 06.04. 2023).
4. DoTween Documentation // DoTween: [сайт]. – 2023. – URL: http://dotween.demigiant.com/documentation.php (дата обращения: 06.05. 2023).
5. RPG Guide: 6 Types of Roles– Playing Games // Masterclass: [сайт]. – 2023. – URL: https://www.masterclass.com/articles/what– is– an– rpg#what– is– a– roleplaying– game (дата обращения: 06.05. 2023).
6. Характеристики: официальный сайт // RPGFandom: [сайт]. –2023. – URL: https://rpg.fandom.com/ru/wiki (дата обращения: 06.05. 2023).
7. История развития компьютерных игр: сайт // Gamesisart: [сайт]. – 2023. – URL: https://gamesisart.ru/istoriya_komputernyh_igr.html (дата обращения: 06.05. 2023).
8. A history of RPGs: официальный сайт // Denofgeek: [сайт]. – 2023. – URL: https://www.denofgeek.com/games/a– history– of– rpgs/ (дата обращения: 06.05. 2023).
9. Как создаются видеоигры: процесс разработки игры: официальный сайт // Itanddigital: [сайт]. – 2023. – URL: https://itanddigital.ru/videogame (дата обращения: 06.05. 2023).
10. Ролевая система: официальный сайт//RPGFandom: [сайт]. –2023. – URL: https://rpg.fandom.com/ru/wiki (дата обращения: 06.05. 2023).
11. Материалы, шейдеры и текстуры // Unity: [сайт]. – 2023. – URL: https://docs.unity3d.com/560/Documentation/Manual/Shaders.html (дата обращения: 06.05. 2023).

12. Важные классы – Quaternion // UnityHub: [сайт]. – 2023. – URL: https://unityhub.ru/manual/class–Quaternion (дата обращения: 06.05. 2023).

13. Изометрия в играх. С чего все начиналось // Habr: [сайт]. – 2023. – URL: https://habr.com/ru/articles/568194/ (дата обращения: 06.05. 2023).

14. Designing a HUD That Works for Your Game: сайт // Pluralsight: [сайт]. – 2023. – URL: https://www.pluralsight.com/blog/film– games/designing– a– hud– that– works– for– your– game (дата обращения: 06.05. 2023).

15. Кватернион: сайт // UnityCode: [сайт]. – 2023. – URL: http://unitycode.blogspot.com/2012/04/blog– post_3413.html (дата обращения: 06.05. 2023).

ПРИЛОЖЕНИЕ А

Код скрипта на C#, отвечающий за передвижение персонажа.

```
using UnityEngine;
using UnityEngine.InputSystem;
using UnityEngine.Events;

namespace DOE
{
    public class Movement: MonoBehaviour
    {
        [SerializeField] Rigidbody _rb;
        [SerializeField] float _speed = 3.0f;
        float _turnSpeed = 360;
        Animator _animator;
        Vector3 _input;


        [SerializeField] InputActionReference _moveActionRef;
        InputAction _moveAction => _moveActionRef.action;

        void OnEnable()
        {
            _moveAction.Enable();
        }

        void OnDisable()
        {
            _moveAction.Disable();
        }


        void Start()
        {
            _animator = GetComponent<Animator>();
        }

        void Update()
        {

            GatherInput();
            Look();

            _rb.angularVelocity = Vector3.zero;

        }

        void FixedUpdate()
        {
            Move();

            if(_animator.GetFloat("Speed") < 0.1f)
            _animator.SetFloat("Speed", 0.0f);
        }


        void GatherInput()
```

```
        {

            var value = _moveAction.ReadValue<Vector2>();
            _input = new Vector3(value.y, 0, value.x);

        }

        void Look()
        {
            if (_input == Vector3.zero) return;

            var rot = Quaternion.LookRotation(_input.ToIso(), Vector3.up);
            transform.rotation   =   Quaternion.RotateTowards(transform.rotation,
rot, _turnSpeed * Time.deltaTime);
        }

        void Move()
        {
            if (_moveAction.IsPressed())
            {
                _rb.MovePosition(transform.position    +    transform.forward    *
_input.normalized.magnitude * _speed * Time.deltaTime);

                _animator.SetFloat("Speed", 1 , 0.1f, Time.deltaTime);
            }
            else {_animator.SetFloat("Speed", 0, 0.1f, Time.deltaTime ); }
}
}

    public static class Helpers
    {
        private          static          Matrix4x4          _isoMatrix          =
Matrix4x4.Rotate(Quaternion.Euler(0, 45, 0));
        public     static     Vector3     ToIso(this     Vector3     input)     =>
_isoMatrix.MultiplyPoint3x4(input);}}
```

ПРИЛОЖЕНИЕ Б

Код скрипта на C#, отвечающий за рывок.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using DG.Tweening;

namespace DOE
{
    public class Dash : MonoBehaviour
    {
        [SerializeField] float _dashSpeed = 50;
        Rigidbody _rigidbody;
        bool _isdashing;

        Transform _target;

        PlayerInput _playerInput;
        float _dashRate = 1.0f;
        float _nextDash = 0.0f;

        Vector3 _targetPosition;

        void Awake()
        {
            _playerInput = new PlayerInput();

            _playerInput.Player.Dash.performed += context => DashM();
        }
        [SerializeField] ParticleSystem forwardDashParticalSystem;

        void Start()
        {
            _rigidbody = GetComponent<Rigidbody>();
        }
        void OnEnable()
        {
            _playerInput.Enable();
        }

        void OnDisable()
        {
            _playerInput.Disable();
        }

        void DashM()
        {
            _isdashing = true;
            _nextDash = Time.time + _dashRate;
        }


        private void FixedUpdate()
        {
            if (_isdashing)
            {
                Dashing();
            }
```

```
        }

        void Dashing()
        {
            //Vector3 direction = (_target.position –
transform.position).normalized;
            //Quaternion lookRotation = Quaternion.LookRotation(new
Vector3(direction.x, 0,direction.z));

            _rigidbody.AddForce(transform.forward * _dashSpeed, ForceMode.Impulse);
            //_rigidbody.DOMove(new Vector3(0,3,0), 1);
            forwardDashParticalSystem.Play();
            _isdashing = false;
        }

        void SetTarget()
        {

        }
    }

}
```

ОТЗЫВ

на выпускную квалификационную работу студентки 4 курса (направление 01.03.02 – Прикладная математика и информатика) Шушкиной Я. на тему «РАЗРАБОТКА ИЗОМЕТРИЧЕСКОЙ ИГРЫ НА UNITY»

Данная дипломная работа посвящена такой актуальной теме как разработка изометрической игры на игровом движке Unity, поскольку изометрическая графика и вид от третьего лица обеспечивают особый визуальный стиль, который позволяет создавать уникальные игровые миры и геймплейные механики, предлагая игрокам новые впечатления и возможности.

Выпускная квалификационная работа включает в себя введение, в котором убедительно обосновывается актуальность работы, ставятся цель и задачи работы. Структура основной части работы состоит из трех разделов. Первый раздел посвящен анализу предметной области, проведено исследование игровой индустрии и рынка, а также изучено основные тенденции и требования игроков, что позволила сформулировать основные принципы и концепции для разработки игры. Второй раздел представляет описание реализации игрового проекта (архитектура игры, функциональные возможности и особенности игровых механик, использованные технологии и инструменты разработки), а также проводится тестирование и оптимизация проекта. Третий раздел посвящен оценке и сбору данных обратной связи от пользователей, на основе которых делаются выводы и предлагаются рекомендации для улучшения игрового опыта и удовлетворения потребностей пользователей.

В ходе выполнения дипломной работы Шушкина Яна проявила высокий уровень самостоятельности, навыки работы с научной литературой, умение осваивать новые математические, программистские и экономические знания, профессиональный подход к проектированию и реализации программных продуктов. В целом считаю, что выпускная квалификационная работа Шушкиной Я. удовлетворяет требованиям, предъявляемым к выпускной квалификационной работе, выполнена на высоком профессиональном уровне, носит завершенный характер и заслуживает оценки «отлично», а ее автор – присвоения квалификации «Бакалавр» по направлению 01.03.02 Прикладная математика и информатика.

Научный руководитель,
канд. физ.-мат. наук,
доцент кафедры прикладной математики Н.М. Сеидова

АНТИПЛАГИАТ
ОБНАРУЖЕНИЕ ЗАИМСТВОВАНИЙ

СПРАВКА

о результатах проверки текстового документа
на наличие заимствований

Кубанский Государственный университет

ПРОВЕРКА ВЫПОЛНЕНА В СИСТЕМЕ АНТИПЛАГИАТ.ВУЗ

Автор работы: Шушкина Я. А.
Самоцитирование рассчитано для: Шушкина Я. А.
Название работы: РАЗРАБОТКА ИЗОМЕТРИЧЕСКОЙ ИГРЫ НА UNITY
Тип работы: Выпускная квалификационная работа
Подразделение: ФКТиПМ, кафедра прикладной математики

РЕЗУЛЬТАТЫ

■ **ОТЧЕТ О ПРОВЕРКЕ КОРРЕКТИРОВАЛСЯ:** НИЖЕ ПРЕДСТАВЛЕНЫ РЕЗУЛЬТАТЫ ПРОВЕРКИ ДО КОРРЕКТИРОВКИ

СОВПАДЕНИЯ	6.6%		СОВПАДЕНИЯ	6.6%
ОРИГИНАЛЬНОСТЬ	90.18%		ОРИГИНАЛЬНОСТЬ	90.18%
ЦИТИРОВАНИЯ	3.22%		ЦИТИРОВАНИЯ	3.22%
САМОЦИТИРОВАНИЯ	0%		САМОЦИТИРОВАНИЯ	0%

ДАТА ПОСЛЕДНЕЙ ПРОВЕРКИ: 19.05.2023

ДАТА И ВРЕМЯ КОРРЕКТИРОВКИ: 19.05.2023 17:58

Структура документа: Проверенные разделы: основная часть с.2-51, библиография с.52-53, приложение с.54-57

Модули поиска: ИПС Адилет; Библиография; Сводная коллекция ЭБС; Интернет Плюс; Сводная коллекция РГБ; Цитирование; Переводные заимствования (RuEn); Переводные заимствования по eLIBRARY.RU (EnRu); Переводные заимствования по Интернету (EnRu); Переводные заимствования издательства Wiley; eLIBRARY.RU; СПС ГАРАНТ: аналитика; СПС ГАРАНТ: нормативно-правовая документация; Модуль поиска "КубГУ"; Медицина; Диссертации НББ; Коллекция НБУ; Перефразирования по eLIBRARY.RU; Перефразирования по СПС ГАРАНТ: аналитика; Перефразирования по Интернету; Перефразирования по Интернету (EN); Патенты СССР, РФ, СНГ; СМИ России и СНГ; Шаблонные фразы; Кольцо вузов; Издательство Wiley; Переводные заимствования

Работу проверил: Троценко Екатерина Сергеевна
ФИО проверяющего

Дата подписи: 19.05.23 г.

Подпись проверяющего

Чтобы убедиться в подлинности справки, используйте QR-код, который содержит ссылку на отчет.

Ответ на вопрос, является ли обнаруженное заимствование корректным, система оставляет на усмотрение проверяющего. Предоставленная информация не подлежит использованию в коммерческих целях.