Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

ЛЕСОСИБИРСКИЙ ПЕДАГОГИЧЕСКИЙ ИНСТИТУТ –
филиал Сибирского федерального университета

Высшей математики, информатики, экономики и естествознания
кафедра

УТВЕРЖДАЮ
Заведующий кафедрой
Л.Н. Храмова
подпись инициалы, фамилия
«14» июня 2024 г.

БАКАЛАВРСКАЯ РАБОТА

09.03.02 Информационные системы и технологии
код-наименование направления

ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА 3D ИГРЫ В ЖАНРЕ ГОЛОВОЛОМКА НА ПЛАТФОРМЕ UNITY

Руководитель 14.06.2024 доцент, канд. пед. наук Е.В. Киргизова
подпись, дата должность, ученая степень инициалы, фамилия

Выпускник 14.06.2024 К.В. Макаренко
подпись, дата инициалы, фамилия

Нормоконтролер 14.06.24 А.В. Фирер
подпись, дата инициалы, фамилия

Лесосибирск 2024

РЕФЕРАТ

Выпускная квалификационная работа по теме «Проектирование и разработка 3D игры в жанре головоломка на платформе Unity» содержит 62 страницы текстового документа, 44 иллюстрации, 2 формулы, 4 приложения, 40 использованных источников, 5 таблиц.

КОМПЬЮТЕРНАЯ ИГРА, UNITY, ЖАНР, ГОЛОВОЛОМКА.

Цель исследования – теоретически обосновать и разработать 3D игру в жанре головоломка на платформе Unity с учетом тенденций 21 века в игровой индустрии.

Объект исследования – процесс проектирования 3D игры в жанре головоломка на платформе Unity.

Предмет исследования – организация игрового процесса и механик в жанре головоломка на платформе Unity.

Для достижения поставленной цели необходимо решить следующие задачи исследования:

– проанализировать учебную и техническую литературу по теме исследования для определения теоретических основ создания 3D игр головоломок на платформе Unity;

– проанализировать популярные игры жанра головоломка, с целью выявления наиболее интересных механик, дизайна уровней и стилистических решений;

– разработать концепцию игры, игровые механики и 3D модели окружения, персонажей, освещение сцены, разработка анимации и пользовательского интерфейса, создать систему уровней с постепенным усложнением, реализовать систему управления персонажем/камерой.

В результате выполнения выпускной квалификационной работы будет разработан проект компьютерной игры головоломки.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Жанр головоломок компьютерных игр на протяжении многих лет сохраняет свою популярность, привлекая игроков интеллектуальными вызовами, оригинальными механиками и возможностью развить логическое мышление. Современные технологии, в частности, игровой движок *Unity,* предоставляют разработчикам мощные инструменты для создания интерактивных 3D миров, открывая новые горизонты для жанра головоломок.

В частности, игра *Portal 2*, выпущенная в 2011 году, стала настоящим прорывом в жанре, благодаря инновационной механике порталов и захватывающему сюжету. Игра продемонстрировала потенциал головоломок как основы для создания глубоких и эмоционально вовлекающих игровых опытов.

По мере развития индустрии видеоигр создавались различные высокобюджетные проекты, которые включали в себя несколько поджанров чтобы добавить погружение в атмосферу и виртуальный мир. Так головоломки стали основой для увлекательного игрового опыта в самых крупных проектах. В настоящее время ни одна высокобюджетная компьютерная игра не обходиться без элементов головоломок.

Цель исследования – теоретически обосновать и разработать 3D игру в жанре головоломка на платформе Unity с учетом тенденций 21 века в игровой индустрии.

Объект исследования – процесс проектирования 3D игры в жанре головоломка на платформе Unity.

Предмет исследования – организация игрового процесса и механик в жанре головоломка на платформе Unity.

Для достижения поставленной цели необходимо решить следующие задачи исследования:

– проанализировать учебную и техническую литературу по теме исследования для определения теоретических основ создания 3D игр головоломок на платформе Unity;

– проанализировать популярные игры жанра головоломка, с целью выявления наиболее интересных механик, дизайна уровней и стилистических решений;

– разработать концепцию игры, игровые механики и 3D модели окружения, персонажей, освещение сцены, разработка анимации и пользовательского интерфейса, создать систему уровней с постепенным усложнением, реализовать систему управления персонажем/камерой.

В результате выполнения выпускной квалификационной работы будет разработан проект компьютерной игры головоломки.

Проведены исследования на следующих научных мероприятиях:

1. VII Всероссийская научно-практическая конференция «Актуальные проблемы преподавания дисциплин естественнонаучного цикла» (г. Лесосибирск, ЛПИ – филиал СФУ, 01-02 ноября 2023 г., диплом II степени);

2. VIII Всероссийский (IX Региональный) молодежный форум «Российское могущество пристать будет Сибирью…» (г. Лесосибирск, ЛПИ – филиал СФУ, 14 декабря 2023 г., диплом II степени);

3. III Всероссийский молодежный научный форум «Современное педагогическое образование: теоретический и прикладной аспекты» (г. Лесосибирск, ЛПИ – филиал СФУ, 9 апреля 2024 г.);

4. III Всероссийский конкурс научных работ «Молодежный научный потенциал» (г. Лесосибирск, ЛПИ – филиал СФУ, 10 апреля 2024 г., диплом II степени);

5. XX Международная научная конференция студентов, аспирантов и молодых ученых «Проспект Свободный – 2024» (г. Красноярск, СФУ, 15-20 апреля 2024 г., диплом II степени).

Методы исследования:

– теоретические: анализ учебной и научно-технической литературы по теме исследование, обобщение и сравнительный анализ;

– эмпирические: проектирование, разработка и тестирование 3D игры.

По результатам исследования приняты к публикации статьи:

1. Макаренко, К. В. Применение методов телепортации объектов в процессе разработки 3D игры в жанре головоломка / К. В. Макаренко – Сборник III Всероссийского молодежного научного форума «Современное педагогическое образование: теоретический и прикладной аспекты». – 2024;

2. Макаренко, К. В. Методы телепортации объектов в компьютерной игре в жанре головоломка / К. В. Макаренко – XX Международная научная конференция студентов, аспирантов и молодых ученых «Проспект Свободный – 2024». – 2024.

Структура работы. Работа состоит из введения, двух глав, заключения, приложений и списка использованных источников, включающего 40 наименований. Результаты работы представлены в 44 иллюстрациях, 5 таблицах. Общий объем работы – 62 страницы.

1 Теоретические основы проектирования и разработки 3D игры в жанре головоломка на платформе Unity

1.1 Жанр головоломок в игровой индустрии

Как утверждают А.Г. Леонов и А.В. Ермолович [16] «Компьютерная игра это:

1. Взаимодействие по определённому алгоритму (с целью развлечения, обучения или тренировки) человека (группы людей) с компьютером или группы людей друг с другом посредством компьютера.

2. Компьютерная программа, которая организует игровой процесс и управляет им, а также осуществляет взаимодействие между соперниками и партнёрами по игре или сама выступает в качестве игрока.

В процессе компьютерной игры на экране дисплея наглядно отображается игровая ситуация; анализируя эту ситуацию, играющий реагирует на неё с помощью устройств ввода (клавиатуры, мыши, джойстика и др.). Ответные ходы или соответствующее изменение игровой ситуации компьютер воспроизводит на экране, часто со звуковым сопровождением».

В.Е. Колесникова [12] считает, что головоломка головоломка (англ. *puzzle*) – «жанр видеоигр, построенный на решении пользователем логических задач. Часто эти задачи сопровождаются временными ограничениями или другими элементами, усложняющими прохождение».

Жанр головоломок занимает особое место в истории компьютерных игр, предлагая игрокам интеллектуальные вызовы и возможность развить логическое мышление. Его эволюция тесно связана с развитием технологий и появлением новых игровых платформ.

Ранние годы (1970-е – 1980-е):

– текстовые приключения: первые головоломки часто представляли собой текстовые приключения, где игрок взаимодействовал с миром, вводя команды и решая загадки с помощью текста. Примеры таких игр: *Colossal Cave Adventure* (1976) и *Zork* (1979);

– логические игры: с появлением графических возможностей, начали появляться игры, основанные на классических головоломках, таких как *Sokoban* (1982) с задачами перемещения ящиков, и *Tetris* (1984) с укладкой падающих фигур;

– приключения *point-and-click*: в конце 1980-х годов стали популярными приключения *point-and-click*, такие как *Maniac Mansion* (1987) и *The Secret of Monkey Island* (1990), где решение головоломок требовало исследования окружения, сбора предметов и их применения в различных ситуациях.

1990-е годы:

– расцвет жанра: 1990-е годы стали золотым веком для игр-головоломок. Появились такие хиты, как *Myst* (1993) с его загадочным островом и сложными головоломками, *The 7th Guest* (1993) с использованием FMV-видео и 3D-графики, и *Lemmings* (1991), где игрок должен был провести леммингов к выходу, используя их уникальные способности;

– рост популярности на персональные компьютеры: персональные компьютеры стали основной платформой для игр головоломок, благодаря гибкости управления с помощью клавиатуры и мыши, а также возможности создавать сложные и графически насыщенные игры.

2000-е годы – настоящее время:

– инновационные механики: с развитием технологий, игры-головоломки стали использовать новые механики, такие как манипуляции с физикой (P*ortal*, 2007), временные петли (*Braid*, 2008), манипуляции с перспективой (*The Witness*, 2016) и другие;

– расширение платформ: головоломки стали популярны на различных платформах, включая консоли, мобильные устройства и VR/AR системы. Это привело к появлению игр с сенсорным управлением, использованием гироскопов и другими инновационными способами взаимодействия;

– инди-разработка: инди-разработчики стали играть важную роль в развитии жанра, предлагая свежие идеи и экспериментируя с механиками.

Примеры успешных инди-головоломок: Fez (2012), The Swapper (2013) и *BabaIs You* (2019).

Таким образом, представленный анализ теоретических основ и истории развития игр головоломок демонстрирует значимость этого жанра в мире интерактивных развлечений. Головоломки, основанные на решении логических задач, не только развлекают игроков, но и способствуют развитию их когнитивных способностей, логического мышления, пространственного воображения.

1.2 Анализ игр в жанре головоломка

Для успешной разработки видеоигры важно изучить опыт существующих проектов, проанализировать их сильные и слабые стороны, а также выявить ключевые элементы, которые обеспечивают увлекательный и захватывающий игровой опыт. В параграфе рассмотрим популярные проекты игр головоломок: *Portal, The Witness, The Talos Principle, Monument Valley, Braid*.

Выделенные проекты игр головоломок представляют собой яркие примеры инновационного подхода к дизайну головоломок и разработке различных механик.

Portal 2 от *Valve*, ставший классикой жанра, демонстрирует элегантность простоты и глубину игрового процесса, построенного на манипулировании пространством с помощью порталов. Пример геймплея представлен на рисунке 1.

The Witness от *Jonathan Blow* предлагает игроку исследовать таинственный остров, наполненный сотнями головоломок, основанных на проведении линий. Пример геймплея представлен на рисунке 2.

The Talos Principle 2 от *Croteam* сочетает сложные пространственные головоломки с философским сюжетом. Пример геймплея представлен на рисунке 3.

Рисунок 1 – Скриншот геймплея *Portal 2*

Рисунок 2 – Скриншот геймплея *The Witness*

Рисунок 3 – Скриншот геймплея *The Talos Principle 2*

Для более глубокого понимания специфики 3D игр жанра головоломок, проведем сравнительный анализ. Рассмотренные примеры 3D игр головоломок от первого лица позволяют выделить основные характеристики, представленные в таблице 1.

Таблица 1 – Основные характеристики 3D игр головоломок от первого лица

Характеристика	**Portal 2**	**The Witness**	**The Talos Principle 2**
Механики	Порталы	Проведение линий	Лазеры, время, клонирование
Дизайн уровней	Линейный, с возрастающей сложностью	Открытый мир, тематические зоны	Нелинейны, множественные решения
Визуальный стиль	Минималистичный	Яркий, красочный	Реалистичный, футуристический

На основе выделенных характеристик 3D игр головоломок от первого лица можно сделать вывод о том, что ключевым элементом их геймплея является манипуляция с объектами в пространстве. Игрокам предлагается решать задачи, связанные с перемещением, вращением и комбинированием объектов для достижения цели.

Monument Valley – яркий пример игры с изометрическим видом. Игрок манипулирует элементами самого уровня, поворачивая и перемещая платформы и другие архитектурные конструкции, чтобы провести принцессу через серию иллюзорных уровней. Необычный вид камеры и стильная графика создают уникальную атмосферу. Пример геймплея представлен на рисунке 4.

Braid головоломка, которая дает возможность игроку манипулировать временем. Классический вид сбоку позволяет эффективно использовать механику управления временем и визуализировать её воздействия для решения головоломок. *Braid* не только бросает вызов интеллекту игрока, но и рассказывает трогательную историю, переплетенную с темой времени и сожаления. Пример геймплея представлен на рисунке 5.

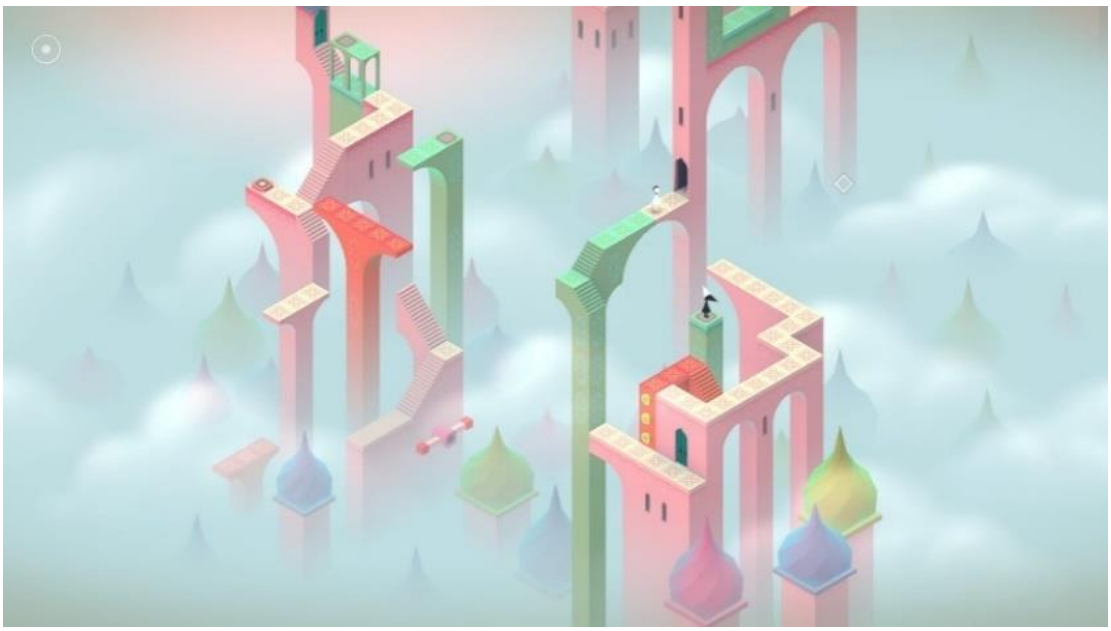


Рисунок 4 – Скриншот геймплея *Monument Valley*

Рисунок 5 – Скриншот геймплея *Braid*

Для более глубокого понимания специфики жанра головоломок-платформеров, проведем сравнительный анализ. Рассмотренные примеры игр головоломок-платформеров позволяют выделить основные характеристики, представленные в таблице 2.

Таблица 2 – Основные характеристики игр головоломок-платформеров

Характеристика	**Monument Valley**	**Braid**
Механики	Манипуляции перспективой и иллюзии, изменение геометрии уровней	Манипуляции временем: перемотка, замедление, ускорение
Дизайн уровней	Иллюзорные лабиринты с геометрическими головоломками и визуальными обманами	Линейный, с постепенным усложнением механик
Визуальный стиль	Минималистичный, вдохновленный работами Эшера	Нарисованный от уки, сказочный

Анализ игр головоломок от первого лица и игр-платформеров позволил выявить ключевые элементы коммерчески успешных головоломок:

– оригинальные механики такие как порталы, манипуляции со временем, геометрические иллюзии или проведение линий, создают основу для увлекательных и разнообразных головоломок;

– дизайн уровней обеспечивает плавное обучение игрока новым механикам и постепенно увеличивать сложность задач;

– вовлеченность в решение головоломок повышает предоставление игроку свободы выбора и возможности экспериментировать с различными подходами к решению головоломок;

– визуальное оформление и звуковое сопровождение создают иммерсивную атмосферу и соответствуют тематике игры.

Согласно данным из открытых источников в сети интернет, продажи игр *The Witness* и *The Talos Principle 2* в первые недели после релиза составили около 100 тысяч копий для каждого проекта. В отличие от них, продажи *Portal 2* за аналогичный период превысили 600 тысяч копий. Этот существенный разрыв можно объяснить несколькими факторами, включая поддержку со стороны крупной студии разработчика *Valve*, активное освещение игры в средствах массовой информации и вовлеченность игрового сообщества по всему миру.

Вышеизложенное позволяет сделать вывод о том, что коммерческий успех игры не определяется исключительно выбором конкретных игровых

механик. Более важную роль играет общий гейм-дизайн проекта и его способность заинтересовать целевую аудиторию. Ключевым фактором является создание концепции, способной вызвать эмоциональный отклик у игроков и предложить им уникальный и захватывающий игровой опыт.

1.3 Компонентно-ориентированная модель архитектуры игры. Принципы проектирования и разработки игры в жанре головоломка

Современная разработка игр, в том числе игр в жанре головоломка, сталкивается с рядом проблем, связанных с архитектурой игровых проектов и необходимостью переписывать значительную часть кода при создании новых игр, даже если они используют схожие механики и компоненты. Это связано с тем, что код зачастую оказывается специфичным для конкретного проекта и плохо поддается повторному использованию.

Чтобы уменьшить риск повторного переписывания компонента под определенную игру, существует компонентно-ориентированная модель разработки. Эта модель позволяет разбивать весь проект на конкретные компоненты, каждый из которых предназначен для выполнения определенной роли. В этом случае важной проблемой при разработке игр является большое количество взаимосвязей между компонентами.

В книге «Проектирование и архитектура игр» Э. Роллингза и Д. Морриса [21] представлена архитектура типовой игры, в соответствии с рисунком 6, состоящая из различных систем. Каждая система может быть декомпозирована на большое количество компонентов, которые могут быть связаны как между собой, так и между другими системами и их компонентами. Такая модель архитектуры позволит создавать универсальные компоненты, которые можно будет применять во всех последующих проектах.

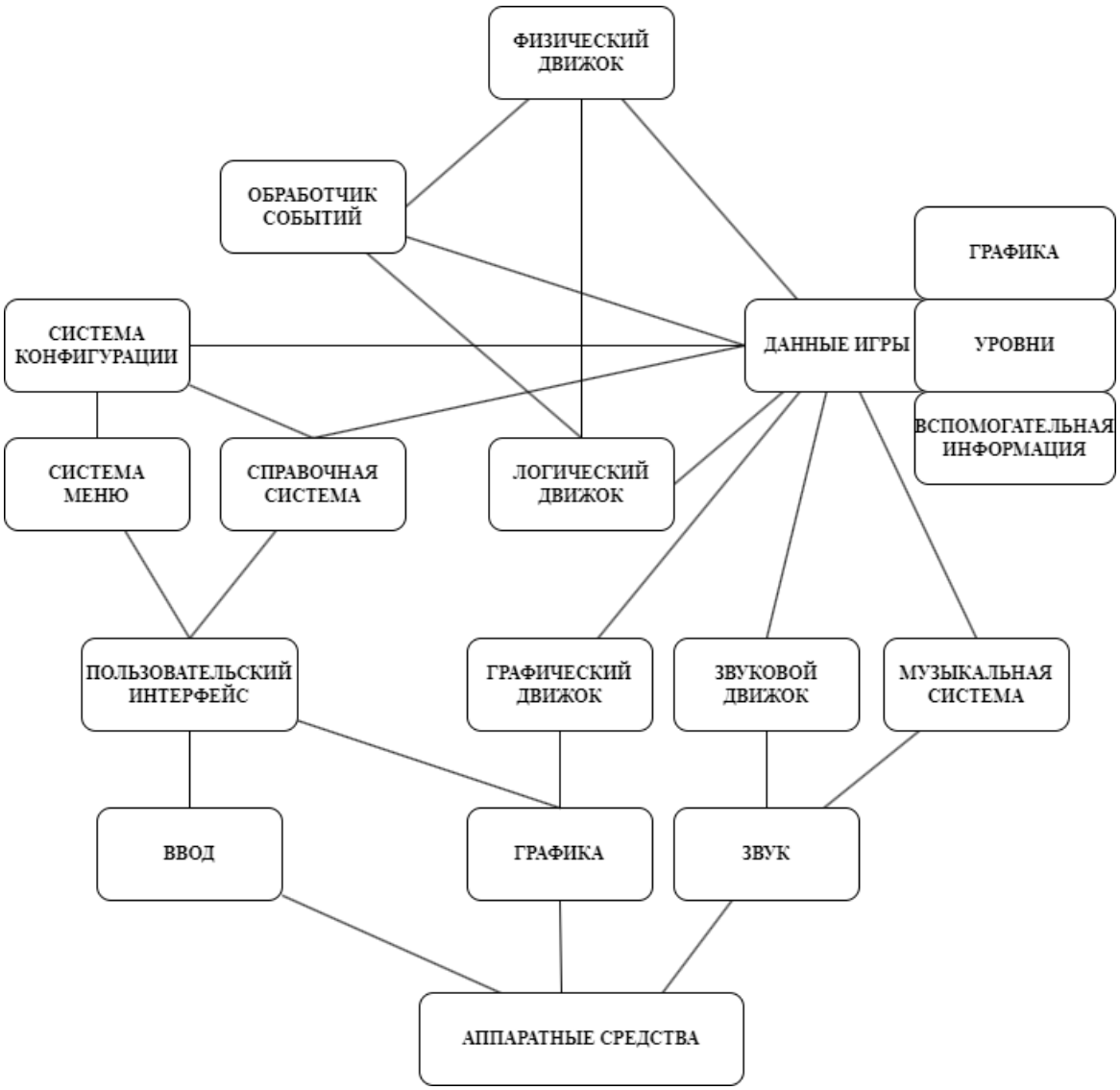


Рисунок 6 – Архитектура типовой игры

С учетом компонентно-ориентированной модели выделим универсальные критерии, которые, по мнению Н.Ю. Тилининой и Н.Е. Губенко [28], «помогут в создании хорошей архитектуры компьютерной игры:

– эффективность системы. В первую очередь программа должна решать поставленные задачи и выполнять свои функции, причем в различных условиях;

– гибкость системы. Любое приложение приходится менять со временем, когда изменяются требования, добавляются новые. Чем быстрее и удобнее можно внести изменения в существующий функционал, чем меньше проблем и ошибок это вызовет – тем более гибкая и конкурентоспособная система;

– расширяемость системы. Возможность добавлять в систему новые сущности и функции, не нарушая ее основной структуры. На начальном этапе в систему имеет смысл закладывать лишь основной и самый необходимый функционал. Но при этом архитектура должна позволять легко наращивать дополнительный функционал по мере необходимости. Причем так, чтобы внесение наиболее вероятных изменений требовало наименьших усилий;

– возможность повторного использования. Систему желательно проектировать так, чтобы ее фрагменты можно было повторно использовать в других системах».

Так же современная разработка игр опирается на принципы игрового дизайна.

Р. Рауз [20, с. 660] в своей книге писал: «Игровой дизайн (также геймдизайн, англ. game design) – процесс создания формы и содержания игрового процесса (геймплея) разрабатываемой игры».

И.С. Кудряшов [14] утверждает, что «Геймплей (англ. gameplay), одна из ключевых составляющих видеоигры, описывающая основные моменты взаимодействия игры и игрока. Геймплеем обычно называют процесс взаимодействия с игровым миром, который распадается на серию действий игрока (вводы) и реакций игры на них (выводы). В других случаях говорят о моделях и структурах, на основе которых подобный процесс разворачивается. Данный термин широко распространён в игровой среде и понимается в основном контекстуально (в том числе геймплеем могут называть и геймплейный ролик с предварительным показом игры)».

Для поддержания интереса игрока необходимо качественно подойти к дизайну уровней и игровым механикам.

Б. Бейтс [2, с 107] в своей книге вводит термин «дизайн уровней» (англ. level desing) – «дисциплина в разработке компьютерных игр, которая включает в себя создание уровней для игр – локации, миссии, задания и прочее окружение».

М. Сикрат [25] в своей статье писал: «Игровая механика (англ. game mechanics) – набор правил и способов, реализующий определённым образом некоторую часть интерактивного взаимодействия игрока и игры. Все множество игровых механик игры формируют конкретную реализацию её игрового процесса».

Помимо дизайна уровней и игровых механик, важно учитывать игровой нарратив, который помогает игроку получить уникальный игровой опыт.

Н. Андрианова и С. Яковлева [1, с. 15] в своей книге утверждают: «Игровой нарратив – в контексте компьютерных игр это не повествование, а полнота игрового опыта игрока, полученного путем уникального прохождения игры. Это история, которую игрок выстраивает сам себе по мере прохождения».

Обеспечение игрового баланса, создает гармоничное соотношение сложности уровней и награды за решение головоломки.

А. Беккер [3] в своей статье пришел к выводу что «игровой баланс – это направление игрового дизайна, направленное на улучшение игрового процесса и пользовательского опыта за счет баланса сложности и справедливости».

Перед началом разработки следует выбрать игровой движок, на котором будет создана вся игра. Для этого произведен анализ, в соответствии с приложением А, самых популярных игровых движков используемых в разработке игр по статистике платформ-магазинов игр.

Современная разработка игр-головоломок опирается на использование игровых движков, таких как Unity, предоставляющих разработчикам инструменты для создания 3D-графики, программирования игровой логики и управления проектом. Одним из популярных языков программирования, используемых в Unity, является C#, обладающий объектно-ориентированной структурой и широкими возможностями.

Р. Валенсия-Гарсия [4, с. 146] утверждает в своей книге, что «игровой движок – это программная платформа, в первую очередь предназначенная для разработки видеоигр и обычно включающая соответствующие библиотеки и вспомогательные программы, такие как редактор уровней».

В руководстве по C# от Microsoft [22] определено, что «C# – кроссплатформенный язык общего назначения...».

Создание визуального стиля игры требует использования программного обеспечения для 3D-моделирования, таких как Blender, Maya или 3ds Max. С помощью этих инструментов создаются модели персонажей, объектов и окружения, а также текстуры, материалы и освещение.

Для создания 3D-контента использовали графический редактор Blender. На официальном сайте Blender [8] введено определение, что «это бесплатный пакет для создания 3D-графики с открытым исходным кодом. Он поддерживает весь 3D-конвейер – моделирование, риггинг, анимацию, симуляцию, рендеринг, композитинг и отслеживание движения, даже редактирование видео и создание игр».

Анимация добавляет жизнь в игровой мир, делая персонажей и объекты более реалистичными и выразительными. Инструменты для анимации позволяют создавать движения персонажей, эффекты частиц и анимацию интерфейса. Для этого использовались инструменты внутри Unity.

Дополнением к атмосфере игры и получением уникального игрового опыта является музыкальное сопровождение. Оно задает тон игры и усиливает эмоциональное воздействие на игрока. Выбор музыкального стиля и композиции должен соответствовать тематике и атмосфере головоломки.

Для создания музыкального сопровождения использовалось программное обеспечение FL Studio – это среда создания музыки, включающая в себя более 100 инструментов и огромную библиотеку звуков.

Из книги Д. Шрейера [37] можно выделить, что понимание мотивации игроков является важным аспектом разработки головоломок. Игроки могут быть мотивированы различными факторами, такими как желание решить сложную задачу, испытать чувство достижения, исследовать игровой мир или просто получить удовольствие от процесса игры.

Вышеизложенное позволяет сделать вывод, что разработка компьютерных игр головоломок – это сложный и многогранный процесс,

требующий понимания различных теоретических основ и практических навыков. Успешная игра-головоломка должна сочетать в себе оригинальные механики, увлекательный геймплей, эффектный дизайн уровней, атмосферный звуковой дизайн и продуманный нарратив. Учитывая мотивацию игроков и потенциальное влияние игры на человека, разработчики могут создавать игры, которые не только развлекают, но и способствуют развитию логического мышления и позитивному опыту.

1.4 Этапы проектирования и разработки игры в жанре головоломка на платформе Unity

В динамичном и конкурентном мире разработки игр, определение четких этапов является ключевым фактором успеха. Структурированный подход позволяет оптимизировать использование ресурсов, контролировать процесс разработки, своевременно выявлять и устранять потенциальные проблемы. Это особенно важно для игр головоломок, где баланс сложности, увлекательный геймплей и интуитивно понятный дизайн играют решающую роль в обеспечении положительного пользовательского опыта.

Проанализировав различные источники в сети интернет, установлено, что разработка игры головоломки на платформе Unity включает в себя несколько ключевых этапов, представленных на рисунке 7, которые организуют процесс создания игры от первоначальной идеи до финального релиза.

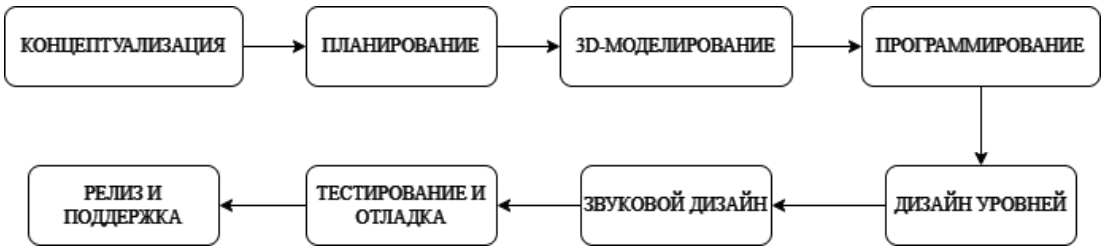


Рисунок 7 – Этапы разработки игр

Рассмотрим каждый из этапов подробнее:

1. Концептуализация: на этом этапе формируется основная идея игры, определяется целевая аудитория, жанр, особенности геймплея и визуальный стиль. Создается концепт-документ, который описывает все основные аспекты игры и служит руководством для дальнейшей разработки.

2. Планирование: на этом этапе распределяются сроки, а также устанавливаются ключевые вехи проекта.

3. 3D-моделирование и анимация: разработка 3D-моделей персонажей, объектов и окружения, а также создание анимаций для придания жизни игровому миру. Качество моделей и анимаций напрямую влияет на визуальное восприятие игры и погружение игрока в атмосферу головоломки.

4. При прочтении книги Р. Найстрома [19] выявлено, что программирование – это реализация игровой логики, механик и других систем игры с помощью C#. Программирование является основой функциональности игры и обеспечивает взаимодействие всех элементов игрового мира.

5. Дизайн уровней: создание уровней с возрастающей сложностью, сбалансированным геймплеем и интересными вызовами для игрока является центральным элементом разработки головоломок. Дизайн уровней должен учитывать особенности механик игры, поддерживать интерес игрока и обеспечивать постепенное обучение новым концепциям.

6. Звуковой дизайн и музыка: создание звуковых эффектов, музыки и озвучки для создания погружения в атмосферу игры. Звуковое оформление играет важную роль в формировании эмоционального отклика игрока и усилении впечатления от игры.

7. Тестирование и отладка: выявление и исправление ошибок, тестирование баланса игры, производительности и удобства пользовательского интерфейса. Тестирование является важным этапом, обеспечивающим качество игры и устранение потенциальных проблем перед релизом.

8. Релиз и поддержка: выпуск игры на выбранных платформах и обеспечение дальнейшей поддержки, включая исправление ошибок и выпуск

обновлений. Поддержка игры после релиза позволяет сохранить интерес игроков и укрепить репутацию разработчика.

Платформа Unity предоставляет разработчикам широкий спектр инструментов и функций, необходимых для реализации каждого этапа разработки. Её гибкость, доступность и обширное сообщество делают Unity идеальным выбором для создания игр головоломок любого уровня сложности.

Таким образом, соблюдение этапов разработки от концептуализации до релиза и поддержки, является залогом успешного создания игры головоломки на платформе Unity, предоставляющая все необходимые инструменты для реализации каждого этапа.

2 Проектирование и разработка 3D игры в жанре головоломка на платформе Unity

Проектирование игровой механики является ключевым этапом разработки, определяющим основные принципы взаимодействия игрока с игровым миром и создающим основу для увлекательного и захватывающего игрового опыта. Дизайн уровней обеспечивает постепенное усложнение задач, введение новых механик и поддержание интереса игрока на протяжении всей игры.

Данная глава посвящена проектированию игровой механики и уровней для компьютерной 3D игры головоломки, разрабатываемой с использованием Unity. В этой главе будет представлена основная концепция игры, подробно описаны механики порталов, лазеров и интерактивных объектов, а также создание дизайна уровней.

2.1 Проектирование и создание материалов для игры

2.1.1 Планирование концепции игры. Создание концепт-документа

Общая информация:

– название: TwT;

– жанр: головоломка;

– целевая платформа: персональный компьютер;

– целевая аудитория: 12+; все люди, которые хотят окунуться в мир фантазии и пространственных загадок; количество игрового опыта совершенно не важно, игра будет интуитивно понятной.

При разработке 3D игры в жанре головоломка используется игровой движок *Unity,* который содержит большое количество бесплатных инструментов для удобного создания видеоигр. В *Unity* используется объектно-

ориентированный язык *C#*, а для программирования визуальной составляющей используется внутренний язык *ShaderLab* и низкоуровневый язык *HLSL*.

При прочтении книги К. Уэйланда [31] определил основную идею игры: персонаж просыпается в башне, не помня, как он туда попал и кто он такой. По мере прохождения игры он восстанавливает свою память и узнает причину своего заточения. Персонаж является участником загадочного эксперимента, а башня служит лабиринтом для испытания его интеллектуальных способностей. Цель – добраться до вершины башни и раскрыть секреты эксперимента.

Жанр стилистики технофентези. В игре сочетается фантастическая магия и мир научно-технических открытий. Так, например, совместно с магическими способностями телепортации и клонирования, герою предстоит встретиться с электрическими лазерами и барьерами.

Структура локаций, в которой будет находиться игрок, представляет собой последовательность из комнат. Чем больше игрок посетил комнат за свое прохождение, тем интереснее и сложнее будут головоломки. То есть игроку нужно будет пройти от начала локации, до ее конца.

В соответствии с рекомендациями В. Маргулец [17] определил графический и звуковой стиль в игре минималистичный с уклоном в сторону мультяшного и low-poly (малое количество полигонов на полигональной сетке моделей). Стены и полы комнат состоят из ровных кубов создавая впечатление что игрок находиться в созданном человеком пространстве.

План разработки:

1. Создание 3D моделей, используемых при проектировании сцен: стены, полы, двери, порталы, лазер и др..

2. Создание системы управления персонажем.

3. Создание основных игровых механик.

4. Создание визуальной составляющей, такой как: материалы, эффекты, анимации.

5. Создание сцены, на которой будут расположены все комнаты.

6. Создание системы сохранений.

7. Создание звукового сопровождения.

8. Создание дополнительных игровых механик связующих элементы игры.

9. Создание UI для управления игрой.

Таким образом, следуя созданному концепт-документу, планируется разработать увлекательную 3D игру в жанре головоломка. Игра будет сочетать в себе интригующий сюжет с минималистичным средневековым визуальным стилем и разнообразными игровыми механиками. Разработка на платформе *Unity* позволит выполнить весь план работ для создания игры.

2.1.2 Создание 3D моделей

Процесс создания моделей для компьютерных игр начинается с концептуального дизайна. На этом этапе создаются эскизы и схематичные рисунки. Для этого выбран единый визуальный средневековый стиль, в соответствии с рисунком 8.

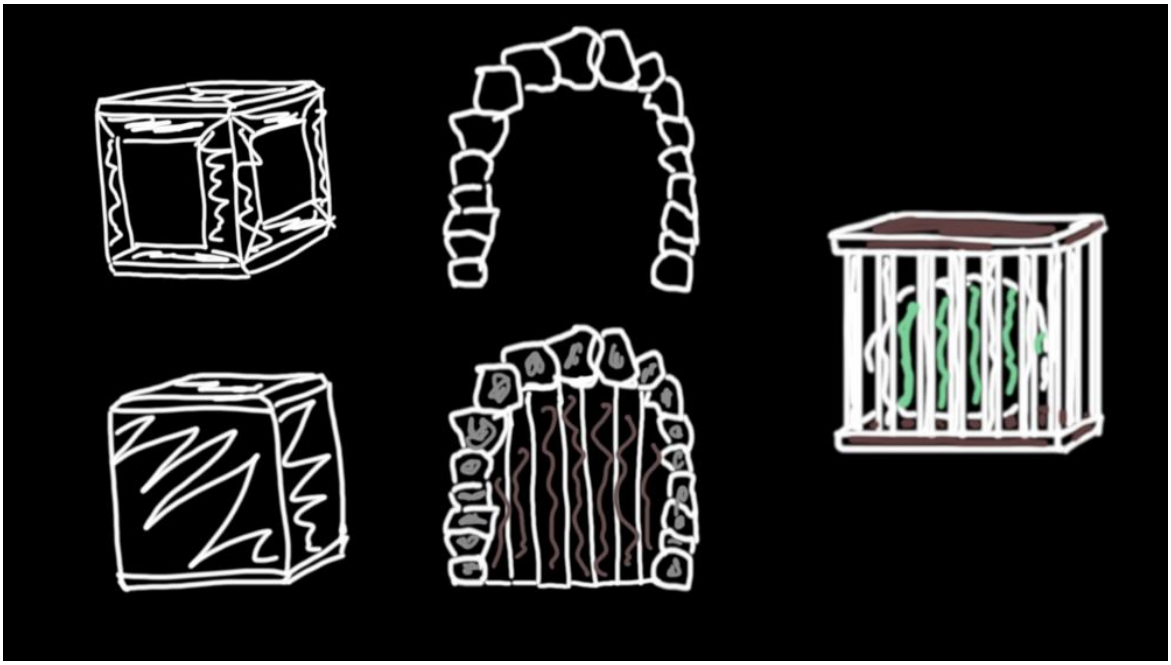


Рисунок 8 – Концепт-арт 3D объектов в игре

Следующим этапом является разработка 3D-моделей в средневековом стиле с помощью инструмента *Blender*, представленных на рисунке 9.

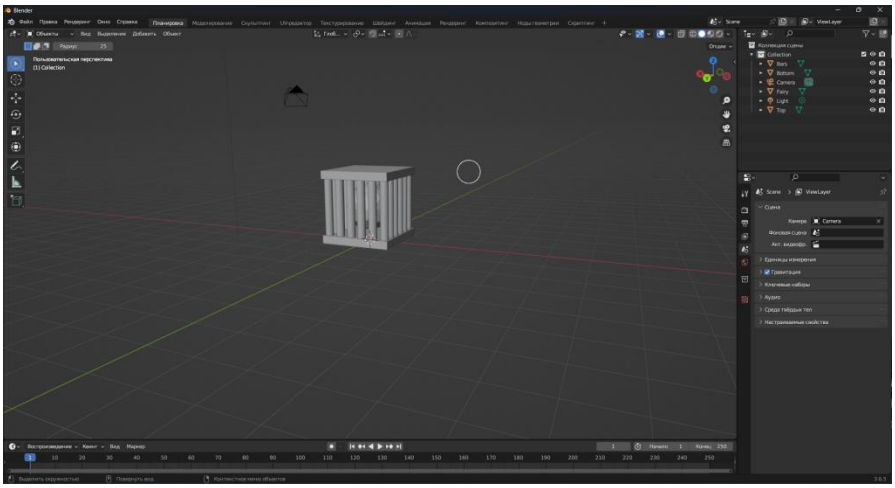


Рисунок 9 – Скриншот создания одного из объектов в Blender

Игра состоит из low-poly моделей, что позволяет оптимизировать игру за счет уменьшения количества полигонов и упрощения геометрии.

Завершающий этап – импорт 3D-моделей в Unity, представленный на рисунке 10, где будет происходить настройка материалов, физических параметров и других компонентов.

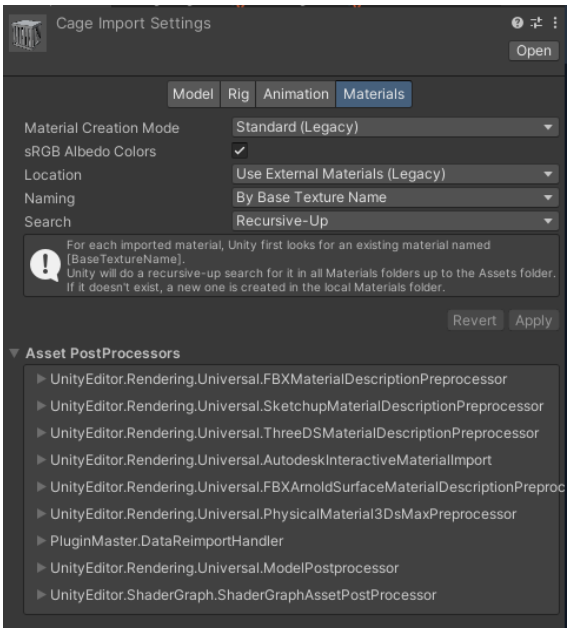


Рисунок 10 – Скриншот импорта 3D-модели в Unity

В результате работы в *Blender* были созданы все необходимые 3D-модели для игры, включая интерактивные объекты, элементы окружения, порталы и лазеры. Особое внимание было уделено оптимизации моделей для обеспечения плавной работы игры, а также созданию визуально привлекательного и соответствующего тематике игры стиля. Полученные 3D-модели готовы к импорту в *Unity* и дальнейшему использованию в игровом мире.

2.1.3 Создание визуальной составляющей

Визуальное восприятие человека напрямую виляет на решение тех или иных головоломок. Можно создать пространство, скрывающее присутствие головоломки, позволяя ей неожиданно раскрыться перед игроком. Или же, напротив, с первых мгновений привлечь внимание к интеллектуальному вызову, побуждая игрока немедленно приступить к решению.

Для раскрытия логического потенциала игрока и обеспечения комфортного процесса решения головоломок, важно создать спокойную и медитативную атмосферу. Визуальные элементы, такие как цветовая палитра, освещение и стиль окружения, могут способствовать созданию расслабляющей и сосредоточенной обстановки.

Unity предоставляет разработчикам мощные инструменты для создания визуальных эффектов и стилизации изображения. Шейдеры, представляющие собой программы для графических вычислений, используются для создания материалов, эффектов освещения и постобработки. В *Unity* существует два основных способа создания шейдеров: с помощью встроенного языка *ShaderLab* и низкоуровневого языка шейдеров *HLSL*, или с использованием инструмента визуального программирования *ShaderGraph*.

Для создания удобства и погружения в игровой мир добавлены следующие визуальные эффекты:

– эффект просвечивания стен, представленный на рисунке 11, позволяет видеть игрока за стеной. Для этого создан отдельный скрипт с помощью

метода *Raycast*, который постоянно проверяет наличие столкновений от игрока до камеры. Скрипт проверяет наличие необходимого материала, только у тех стен, которые включены в настроенный заранее слой на сцене;

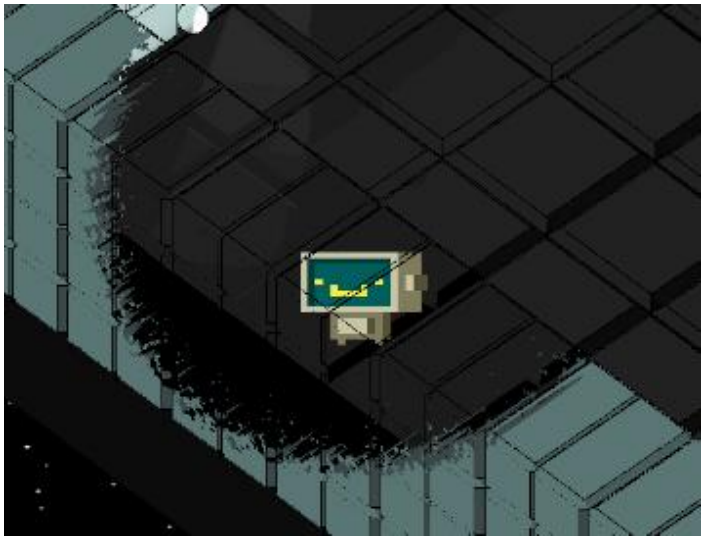


Рисунок 11 – Скриншот эффекта для стен

– эффект затенения потолка комнат, представленный на рисунке 12, добавляет сосредоточенности на определенной комнате в игре. Здесь использовался подобный подход к созданию шума и реакции потолка на персонажа игрока, чтобы персонаж всегда оставался в центре внимания пользователя;

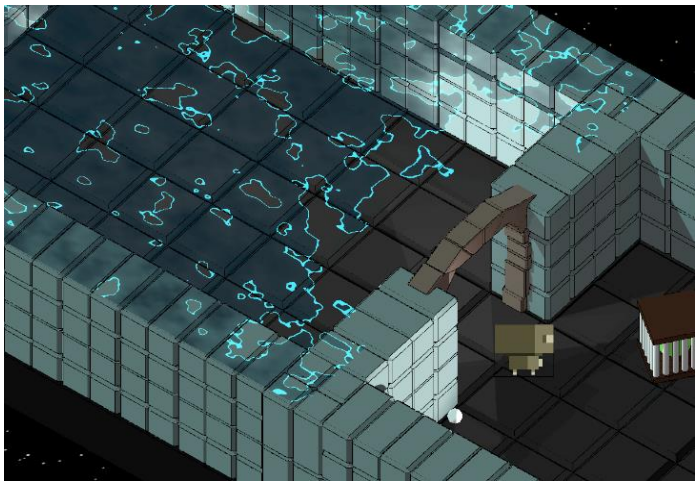


Рисунок 12 – Скриншот эффекта для потолка

– эффекты для барьеров и активирующей зоны, созданные при помощи написания шейдера вручную на языке HLSL, в соответствии с рисунком 13.

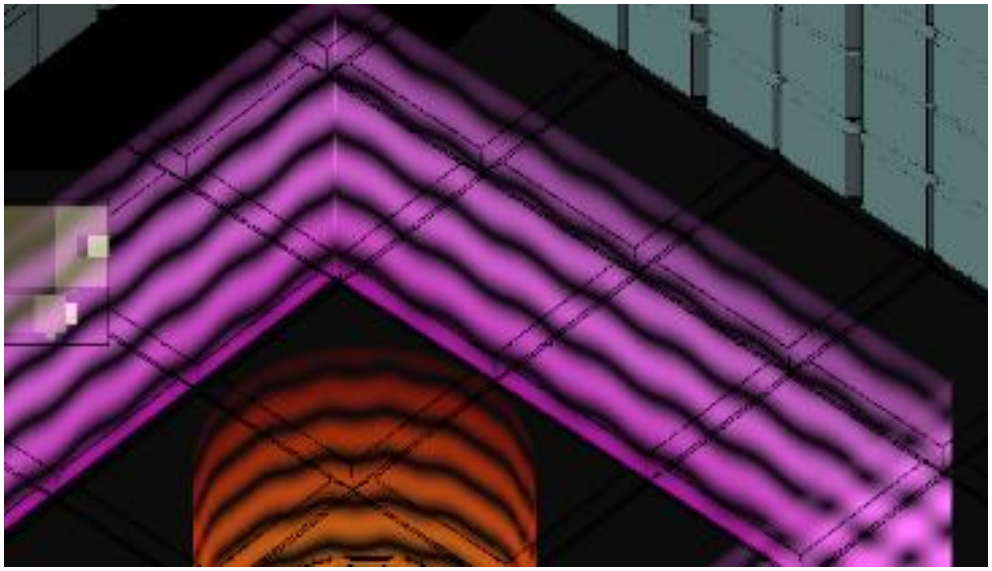


Рисунок 13 – Скриншот эффекта для барьера и активирующей зоны

Для добавления единого стиля всем объектам на сцене, создан шейдер, представленный на рисунке 14, влияющий на изменения цвета, теней и реакции на свет объектов, к которым присвоен этот материал.

Рисунок 14 – Скриншот шейдера объектов на сцене

В Unity есть удобные инструменты постобработки, которые помогают добавить уникальности визуальной составляющей игры. Для добавления ощущения нахождения игрока в башне, где сливается магия и техника в единое целое были использованы такие эффекты постобработки, в соответствии с рисунком 15, как:

– виньетка (с англ. *Vignette*);

– хроматическая аберрация (с англ. *Chromatic Aberration*);

– блум (с англ. *Bloom*).

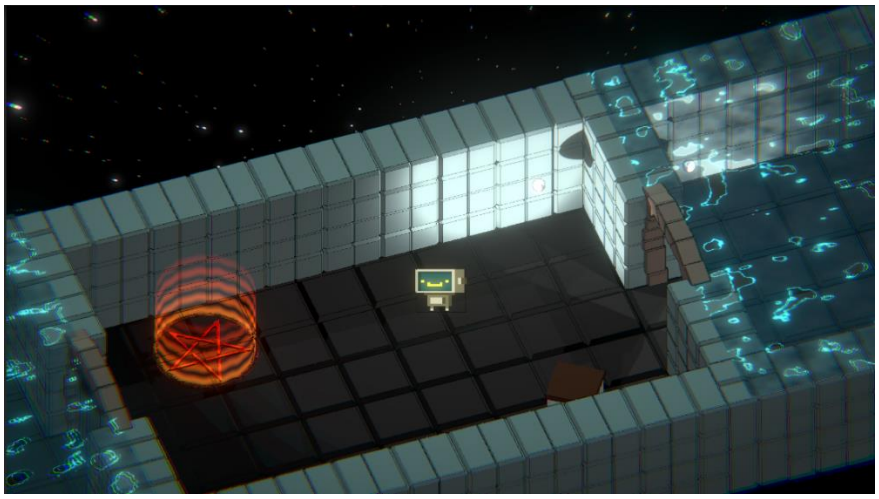


Рисунок 15 – Скриншот игры с эффектами постобработки

Следуя описанным методам, использование шейдеров и эффектов постобработки позволило создать уникальную визуальную атмосферу, соответствующую тематике игры головоломки и способствующую погружению игрока в игровой мир.

2.1.4 Создание звукового сопровождения

Добавление звукового сопровождения в игру, усиливает погружение игрока в мир и подчеркивает важные моменты геймплея, а также создает эмоциональную связь с происходящим на экране. Звуковое оформление играет важную роль в создании атмосферы и усилении эмоционального воздействия

игры. Грамотно подобранные звуковые эффекты и музыка способны не только подчеркнуть важные моменты геймплея, но и создать глубокое погружение в игровой мир.

Для подчеркивания особенностей игрового жанра головоломки, важно создать медитативное музыкальное сопровождение, которое обеспечит возможность сосредоточиться игроку на головоломках и не отвлекаться ему по мере прохождения.

С помощью программы *FL Studio* создана единая звуковая аудиодорожка, состоящая из нескольких фаз, что позволило разделить ее на несколько частей и переключать их в определенные моменты прохождения игры. Работа в программном средстве представленное на рисунке 16. Например, когда игрок находиться в игровом меню паузы или при нахождении решения головоломки, чтобы пользователю проще было сориентироваться, меняем звуковую дорожку на более интенсивную.

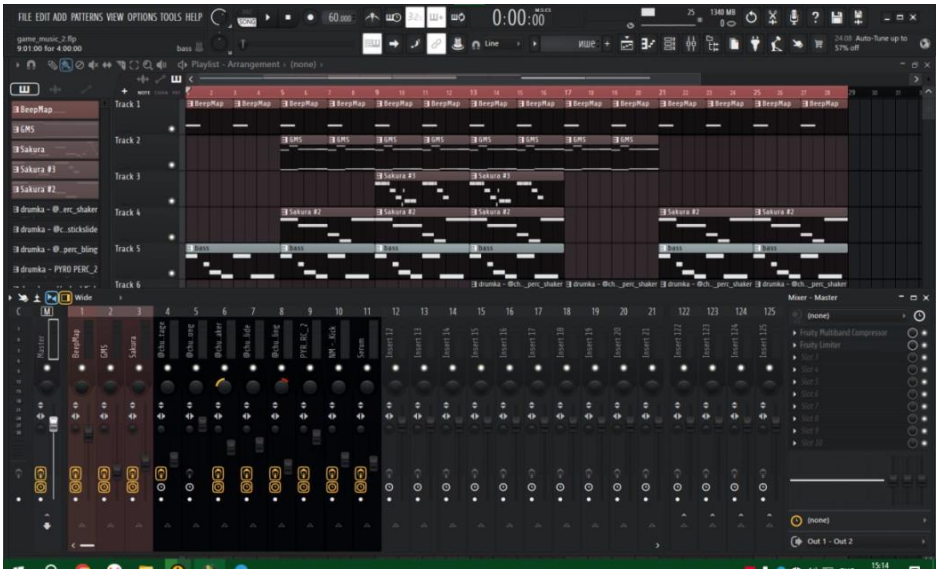


Рисунок 16 – Скриншот проекта в FL Studio

Таким образом, создание продуманного звукового сопровождения для игры-головоломки является неотъемлемой частью процесса разработки, существенно влияющей на восприятие игроком игрового мира.

2.1.5 Создание уровней

При разработке уровней учтены основные принципы дизайна головоломок, описанные в параграфе 1.3. В начальных комнатах игрок знакомится с базовыми механиками игры: использование порталов для телепортации, взаимодействие с объектами и манипуляции с лазерами. По мере продвижения по игре сложность головоломок возрастает, вводятся новые механики и комбинации существующих, требующие от игрока все более сложного и нестандартного мышления. Разнообразие уровней достигается за счет использования различных комбинаций механик, изменения размеров и конфигурации комнат, а также введения новых типов интерактивных объектов и препятствий. В игре существуют 8 комнат-головоломок, вид сверху изображен на рисунке 17.

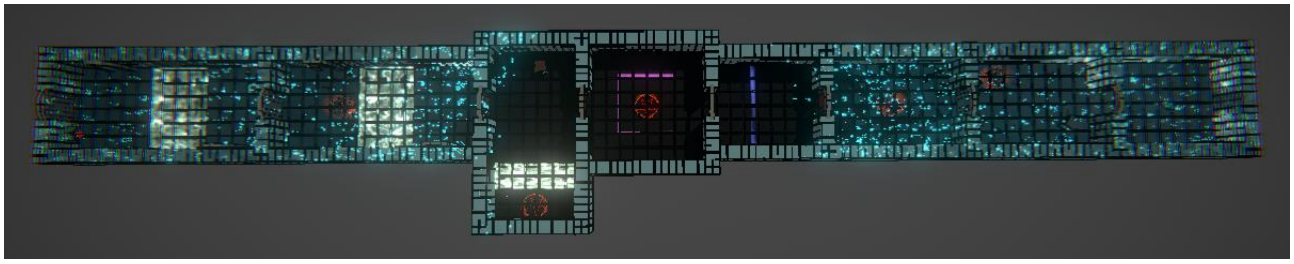


Рисунок 17 – Скриншот всех комнат уровня сверху

Переход в следующую комнату возможен только после успешного решения головоломки текущей комнаты, что создает ощущение прогрессии и достижения. Дизайн уровней играет ключевую роль в поддержании интереса игрока, обеспечивая постепенное усложнение задач и разнообразие игрового опыта.

Таким образом, дизайн уровней в виде последовательности комнат, каждая из которых представляет собой отдельную головоломку, позволяет создать увлекательный и разнообразный игровой опыт. Постепенное

усложнение задач, введение новых механик и использование разнообразных визуальных элементов поддерживают интерес игрока и мотивируют его к дальнейшему прохождению игры.

2.2 Проектирование и разработка игровых компонентов

2.2.1 Создание системы управления персонажем

Для каждой 3D игры создается собственная система управления, включающая управление игровым персонажем и пользовательским интерфейсом.

В головоломках с видом камеры top-down критически важным является интуитивное и отзывчивое управление, позволяющее игроку сосредоточиться на решении задач, не отвлекаясь на сложности управления персонажем. В разрабатываемом проекте система управления, в соответствии с рисунком 18, реализована с помощью инструмента Input System и обеспечивает гибкость в настройке ввода с различных устройств.

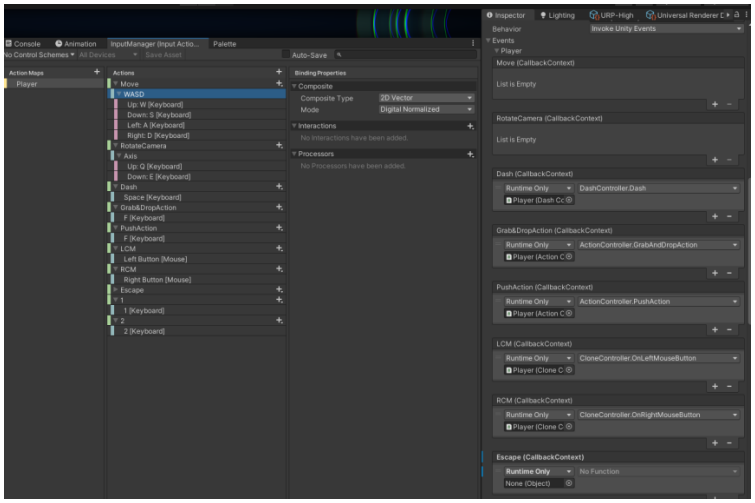


Рисунок 18 – Скриншот системы управления Unity

В top-down перспективе особое внимание уделяется плавности и отзывчивости движения персонажа. Игрок должен иметь возможность легко перемещаться по игровому миру, используя WASD-управление.

Компонент *PlayerController*, в соответствии с рисунком 19, обрабатывает ввод с клавиатуры (WASD) и преобразует его в вектор движения *movement*. Этот вектор учитывает направление камеры, чтобы движение персонажа было корректным в top-down перспективе.

Затем компонент *PlayerMovement* использует вектор movement для перемещения персонажа с помощью компонента *Rigidbody*. Плавность и отзывчивость движения обеспечиваются использованием метода *MovePosition* и умножением вектора движения на скорость персонажа и время кадра.

```
public void MovePlayer()
{
    Vector3 movement = _playerController.movement * (_playerController.speed * Time.fixedDeltaTime);
    _rb.MovePosition(_rb.position + movement);
}
```

Рисунок 19 – Листинг компонента PlayerController

Компонент *ActionController*, в соответствии с рисунком 20, отвечает за взаимодействие игрока с предметами. При нажатии клавиши «F» компонент проверяет наличие интерактивного объекта в радиусе действия. Если объект найден, он «поднимается» и прикрепляется к персонажу. Перемещение происходит вместе с персонажем.

Удержание клавиши «F» активирует «зарядку» броска. При отмене удержания объект отсоединяется от персонажа и получает импульс силы в направлении взгляда. Сила импульса зависит от времени «зарядки».

```
public void GrabAndDropAction(InputAction.CallbackContext context)
{
    if (context.performed && !OpenMenuUI.GameIsPaused)
    {
        if (heldObject == null)
        {
            if (Physics.Raycast(origin: transform.position, direction: 2 * (holdParent.position - transform.position), out hit, pickUpRange, (int)interactiveLayerMask))
            {
                if(hit.collider.tag == "Interactive")
                    PickupObject(hit.transform.gameObject);
            }
        }
        else
        {
            DropObject();
        }
    }
}

public void PushAction(InputAction.CallbackContext context)
{
    if (context.performed && (heldObject != null && !_isCharging) && !OpenMenuUI.GameIsPaused)
    {
        StartCharging();
    }
    else if (context.canceled && _isCharging && !OpenMenuUI.GameIsPaused)
    {
        StopCharging();
    }
```

Рисунок 20 – Листинг компонента ActionController

Компонент *DashController*, представленный на рисунке 21, отвечает за реализацию рывка персонажа при нажатии клавиши «SPACE».

Если рывок доступен, то есть не находится в состоянии перезарядки, то запускается сопрограмма *PerformDash*. В течение короткого промежутка времени *dashTime* к персонажу применяется импульс силы в направлении последнего движения.

Одновременно с этим, переменная *isDashing* в компоненте *PlayerController* принимает значение *true*, что позволяет корректно отобразить анимацию рывка.

После завершения рывка переменная *isDashing* принимает значение *false* и начинается отсчет времени перезарядки *dashCooldown*, в течение которого рывок недоступен.

```
private bool CanDash()
{
    return Time.time >= _lastDashTime + dashCooldown;
}

public void Dash(InputAction.CallbackContext context)
{
    if ((context.performed && !_playerController.isDashing && CanDash()) && _playerController.lastMovementDirection != Vector3.zero)
    {
        if (_playerController.movement.magnitude > 0.01f)
        {
            StartCoroutine(routine:PerformDash());
        }
    }
}
```

Рисунок 21 – Листинг компонента DashController

Фрагмент кода компонента *CameraController* демонстрирует отслеживание нажатия клавиш «Q» и «E». При нажатии одной из этих клавиш вызывается метод *RotateCamera*, который обеспечивает вращение камеры вокруг целевого объекта (персонажа) в соответствующую сторону. Такая ситуация полезна в тех случаях, когда игроку необходимо увидеть объекты и элементы окружения, скрытые за стенами или другими препятствиями. Скорость вращения определяется параметром *keyboardRotationAmount*.

Одновременно с этим, камера плавно перемещается к позиции, определяемой положением целевого объекта (персонажа) и смещением *offset*. Скорость перемещения задается параметром *smoothSpeed*. Листинг компонента CameraController() представлен на рисунке 22.

```
void FixedUpdate()
{
    float keyboardRotateInput = Input.GetAxis("RotateCamera");
    RotateCamera(keyboardRotateInput);
    HandleMovementInput();

    Vector3 desiredPosition = target.transform.position + offset;
    transform.position = Vector3.Lerp( a: transform.position,  b: desiredPosition, smoothSpeed);
}

private void RotateCamera(float rotationInput)
{
    transform.RotateAround( point: target.transform.position,
                            axis: Vector3.up,
                            angle: rotationInput * keyboardRotationAmount * Time.fixedDeltaTime);
}

private void HandleMovementInput()
{
    transform.position = Vector3.Lerp( a: transform.position,  b: offset,  t: Time.fixedDeltaTime * movementTime);
}
```

Рисунок 22 – Листинг компонента CameraController

В результате реализации описанных компонентов, в игре создана комплексная система управления персонажем, адаптированная к top-down перспективе. Игрок может плавно перемещаться по игровому миру, взаимодействовать с объектами, вращать камеру для лучшего обзора, а также совершать кратковременные рывки, что обогащает игровой процесс и делает его более динамичным.

2.2.2 Проектирование и разработка основных игровых механик

Основой для игр в жанре головоломок являются ее игровые механики, определяющие правила взаимодействия игрока с миром и создающие основу для интеллектуальных вызовов.

В игре были реализованы следующие механики:

1. Клон – это способность игрока, которую он может использовать когда захочет. Способность заключается в том, что игрок может создавать клона своего персонажа в указанном им месте с помощью курсорного указателя.

Создание клона реализовано через префаб («образец» объекта с заранее настроенными компонентами) персонажа без компонентов управления и камеры. Создание и удаление производится посредством нажатия на левую и правую кнопку мыши в определенной точке на экране, то есть игрок указывает курсором, где именно он хочет создать клона.

Регистрация нажатий осуществлена через используемый инструмент InputSystem и mousePosition – это свойство системы ввода, которое передает текущее положение мыши в координатах. Так же добавлена фильтрация по слоям на сцене, то есть игрок сможет поставить клона только там, где разрешил разработчик. Работает это так: на сцене есть слои стен и полов, соответственно нужно разрешить ставить клона только на полу, чтобы не возникало различных ошибок в прохождении игры.

Компонент CloneController, в соответствии с рисунком 23, отвечает за создание и удаление клона персонажа в игре.

Рисунок 23 – Схема работы компонента CloneController

При нажатии левой кнопки мыши компонент *CloneController*, в соответствии с рисунком 24, получает координаты курсора мыши и проецирует их на игровой мир с помощью *Raycast*. Если *Raycast* попадает в объект на слое, разрешенном для создания клона, создается экземпляр префаба клона в точке попадания. При нажатии правой кнопки мыши, если клон уже создан, он уничтожается.

```
public void OnLeftMouseButton(InputAction.CallbackContext context)
{
    if (context.performed && _clone == null && !OpenMenuUI.GameIsPaused)
    {
        _mousePosition = Input.mousePosition;

        Vector3 viewportPosition = Camera.main.ScreenToViewportPoint(_mousePosition);

        Ray ray = Camera.main.ViewportPointToRay(viewportPosition);
        RaycastHit hit;

        if (Physics.Raycast(ray, out hit, maxDistance: Mathf.Infinity, (int) layerMask))
        {
            Debug.Log(hit.collider);
            _spawnPosition = hit.point;
        }

        _clone = Instantiate(playerClone, _spawnPosition, rbPlayer.transform.rotation);
        _clone.tag = "Clone";

        isClone = true;
    }
}

public void OnRightMouseButton(InputAction.CallbackContext context)
{
    if (context.performed && _clone != null && !OpenMenuUI.GameIsPaused)
    {
        Destroy(_clone);
        isClone = false;
    }
}
```

Рисунок 24 – Листинг компонента CloneController

2. Лазеры – представляют собой в игре не просто яркие лучи электрического света, а динамические элементы геймплея. При попытке пересечь луч лазера, игрок будет откинут назад. Лазеры могут быть прерваны физическими объектами, такими как кубы, стены или специальные преграды. Это позволяет игроку манипулировать траекторией лазеров, направляя их в нужные точки или блокируя их воздействие. Клоны также могут прерывать лазеры, позволяя игроку создавать «живые» барьеры.

Работа лазеров построена на компоненте *LineRenderer* в *Unity*. Схематичное изображение работы компонента *LaserController* представлено на рисунке 25. Этот компонент позволяет создать линию исходя из заданных точек, например, чтобы создать прямую линию нужны две точки начало и конец координат этой линии. Начало координат всегда будет известно изначально, это то откуда будет выходить лазер, то есть из объекта на сцене. Конец координат необходимо искать исходя из случая:

– столкнулся со стеной или другим физическим объектом;

– столкнулся с персонажем;

– не сталкивается ни с каким объектом на сцене.

При столкновении с лазером игрок отталкивается. Если игрок задел лазер, перемещаясь с помощью клавиш «WASD», его отталкивает в направлении касательной от лазера. Если же игрок использовал рывок с использованием клавиши «SPACE», анимация и сам рывок прерываются, а игрок отталкивается с удвоенной силой.

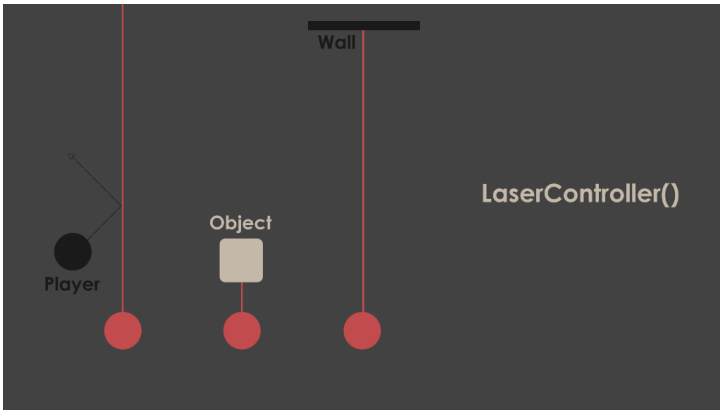


Рисунок 25 – Схема работы компонента LaserController

Для взаимодействия лазера с порталом и передатчиком необходим коллайдер, размер которого динамически изменяется в зависимости от длины *LineRenderer*. Для этой цели, в соответствии с рисунком 26, разработан метод *CurrentColliderSize()*, который определяет новый размер коллайдера на основе размера *LineRenderer* на сцене.

```
private void CurrentColliderSize(float newSize)
{
    if (_laserCollider is BoxCollider boxCollider)
    {
        boxCollider.size = new Vector3(boxCollider.size.x, boxCollider.size.y, newSize);
        boxCollider.center = new Vector3(boxCollider.center.x, boxCollider.center.y, boxCollider.size.z / 2f);
    }
}
```

Рисунок 26 – Листинг компонента LaserController

3. Порталы – являются одной из ключевых механик игры, предлагающей игроку уникальную возможность мгновенного перемещения в пространстве и манипуляции с объектами и лазерными лучами. В комнатах могут находятся два статических портала, заранее размещенные в определенных точках. При входе в один из порталов, игрок мгновенно перемещается в другой портал, расположенный в той же комнате. Игрок может переносить через порталы объекты, которые он берет в руки. Лазерные лучи также могут проходить через порталы, продолжая свою траекторию с другой стороны.

Если в игре есть перемещение, то добавление портала добавит огромное количество вариаций уровней или решений этих уровней.

В рамках исследования выявлен ряд принципов телепортации (с англ. *teleportation* – перемещать на расстоянии) объектов:

– перемещение объектов из одной точки в другую (в случае перемещения объекта через портал);

– реализация эффекта непрерывного перемещения через порталы (например, персонаж проходит через дверь или окно);

– реализация визуальной части портала (чтобы игрок видел, что находится перед выходом из портала).

3.1. Для реализации телепортации объектов и игроков в компьютерной игре используется несколько методов:

Метод *PlayerTeleport* – предназначен для корректной телепортации игрока из одной точки в другую, на основании которого:

а) вычислили вектор *a* от портала (*Portal*) до игрока (*Player*) по формуле (1), для того чтобы определить, где находится объект по отношению к порталу в соответствии с рисунком 27:

$$a = Player - Portal \tag{1}$$

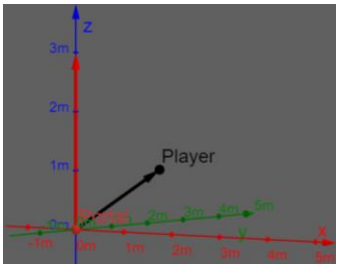


Рисунок 27 – График вектора портала и игрока

б) для того, чтобы определить местоположение игрока на рисунке относительно портала (игрок перед порталом и внутри портала), нашли скалярное произведение *Dot* вектора портала (B1) и вектора (нормали) от портала до игрока (A1, A2), в соответствии с таблицей 3.

Таблица 3 – Определение позиции игрока относительно портала

Случай, когда игрок находится перед порталом	**Случай, когда игрок находится внутри портала**
$A_1 = Player - Portal = \begin{pmatrix}1\\1\\1\end{pmatrix} - \begin{pmatrix}0\\0\\0\end{pmatrix} = \begin{pmatrix}1\\1\\1\end{pmatrix}$	$A_2 = Player - Portal = \begin{pmatrix}1\\1\\-1\end{pmatrix} - \begin{pmatrix}0\\0\\0\end{pmatrix} = \begin{pmatrix}1\\1\\-1\end{pmatrix}$
$B_1 = \begin{pmatrix}0\\0\\3\end{pmatrix}$	$B_1 = \begin{pmatrix}0\\0\\3\end{pmatrix}$
$Dot = A_1 * B_1 = \begin{pmatrix}1\\1\\1\end{pmatrix} * \begin{pmatrix}0\\0\\3\end{pmatrix} = 3$	$Dot = A_2 * B_1 = \begin{pmatrix}1\\1\\-1\end{pmatrix} * \begin{pmatrix}0\\0\\3\end{pmatrix} = -3$

Если *Dot* > 0, то игрок зашел в портал, иначе игрок находится перед порталом.

в) для корректной телепортации игрока относительно выхода вычислили угол поворота игрока к входу по формуле (2), и нашли разницу между ориентациями первого и второго порталов:

$$rotationDiff = -Qiuaternion.Angle(Q_1, Q_2) + 180^{\circ}, \quad (2)$$

где *rotationDiff* – угол поворота игрока к входу;

Q1 – ориентация первого портала;

Q2 – ориентация второго портала.

Используемая функция *Quaternion.Angle* позволяет вычислить угол между двумя кватернионами (ориентациями). При вычислении угла поворота игрока к входу добавили 180º, для того чтобы учесть различия ориентаций порталов.

В соответствии с рекомендациям в Т. Хилтона [36], находим смещение позиции игрока к вектору портала (*portalToPlayer*) после поворота с использованием функции *Quaternion.Euler*.

3.2. Для реализации эффекта непрерывного физического перемещения используется разработанный метод *CloneObject*, представленный на рисунке 28.

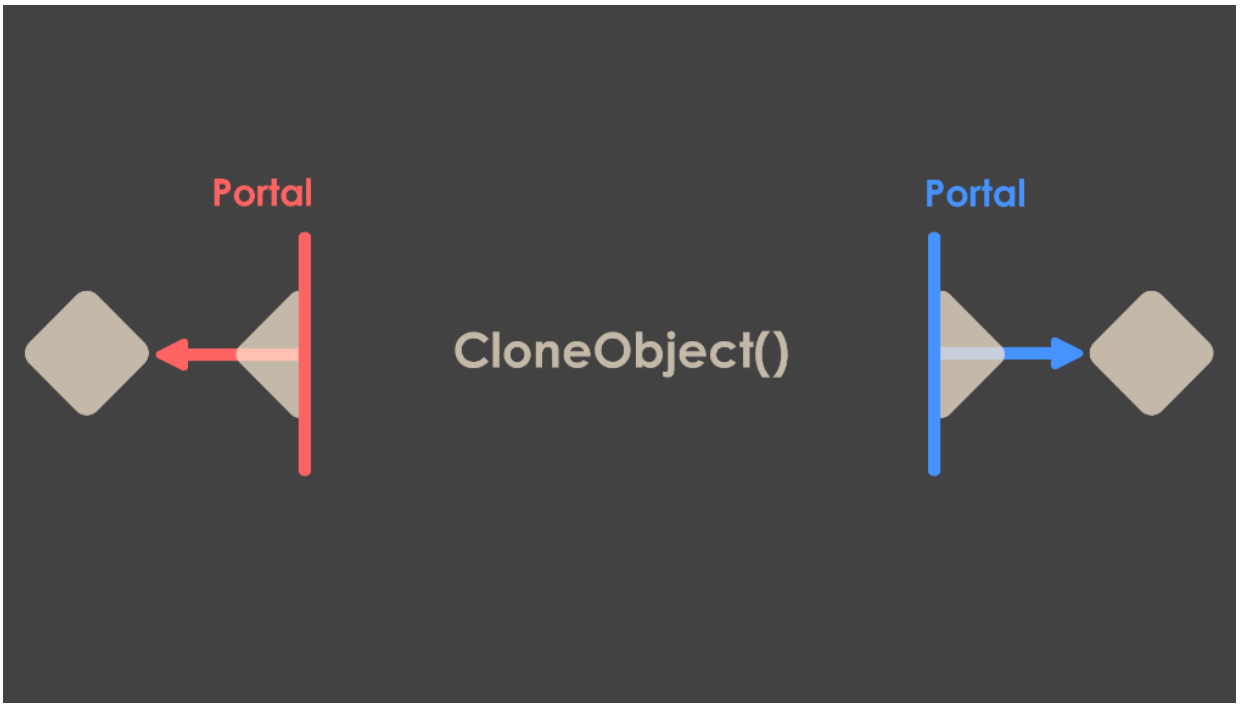


Рисунок 28 – Схема работы метода CloneObject

Для создания клона выполнили следующее:

а) при создании клона объекта (всех кроме лазера) присвоили оригинальный угол поворота и назначили позицию портала, из которого будет выходить объект его клону;

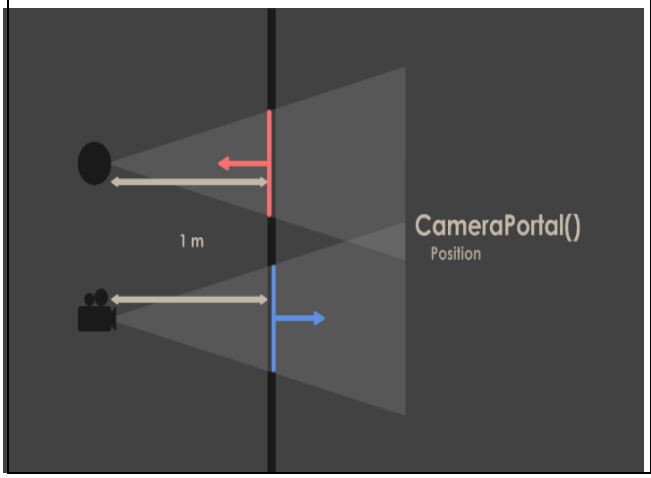
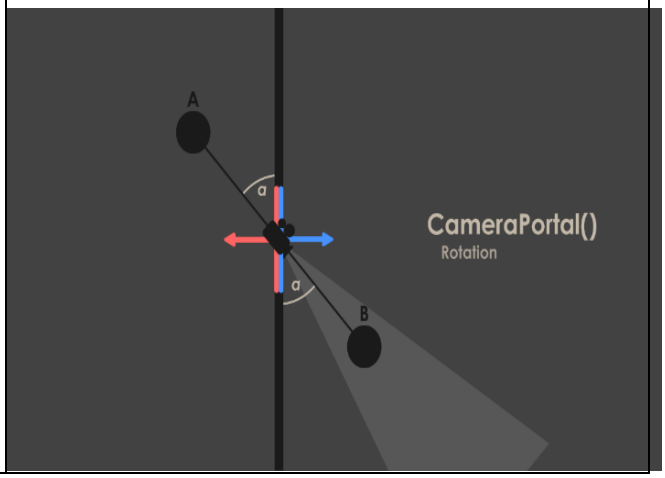
б) при создании клона объекта лазера присвоили оригинальный угол поворота и позицию начальной точки, которую переводим в локальные координаты портала. Затем создаем новую позицию начальной точки и переводим полученный поворот и начальную точку в мировые координаты дочернего портала.

3.3 При прочтении документации *Unity* [9] определил методы, используемые для визуальной части:

3.3.1 Метод *Start* – создает текстуру рендера для камеры дочернего портала и назначает ее как текстуру материала родительского портала, чтобы отображать изображение, видимое через дочерний портал;

3.3.2 Метод *CameraPortal* – выполняет следующие вычисления корректного перемещения камер порталов, которые передают изображение, в соответствии с таблицей 4.

Таблица 4 – Применение метода CameraPortal

Нахождение позиции игрока в локальных координатах первого портала и присвоили эти координаты камере второго портала	**Нахождение разницы в повороте между текущим порталом и дочерним порталом и корректировка поворота камеры портала**
	

4. Активаторы дверей/механизмов – это базовые кнопки, которые могут открывать проход куда-либо, только визуально выглядят по-другому. Один из таких механизмов активируется, когда внутрь помещается какой-то физический объект (куб или клон). Другой активируется, когда лазер коснется его.

Первый тип активаторов, который работает, когда в него помещают физический объект. Здесь используется *Trigger* (свойство компонента коллайдера, который делает объект областью способной взаимодействовать с входящими объектами в эту область) для создания области взаимодействия. С помощью кода создается работа этой зоны, представленная на рисунке 29, когда объект помещается в зону этот объект активирует механизм (например, открытие двери) и притягивается к центру этой зоны.

Рисунок 29 – Схема работы метода TriggerAreaController

Второй тип активаторов работает от лазера. Как и в первом варианте активатора используется *Trigger*. Работа этого объекта еще проще, когда попадает лазер активирует механизм, когда лазер выходит из триггера механизм деактивируется. Схематично компонент представлен на рисунке 30.

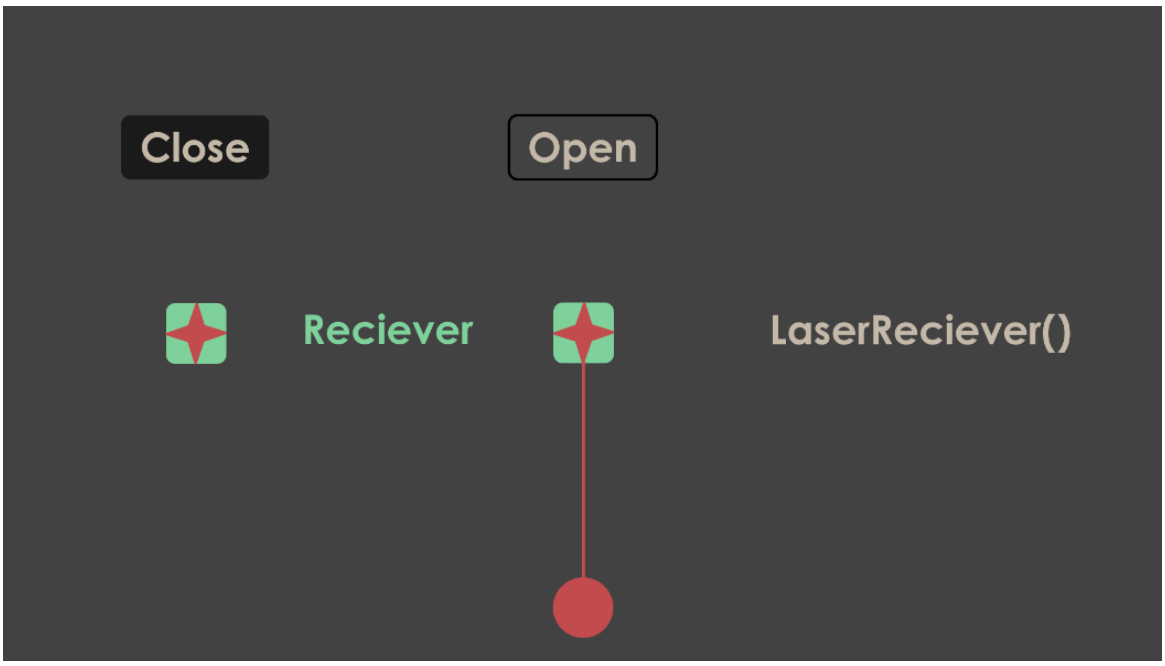


Рисунок 30 – Схема работы метода LaserReciever

Компоненты *TriggerAreaController* и *LaserReciever*, в соответствии с рисунком 31, работают по одинаковому принципу, когда в их коллайдер попадает объект – они активируются.

```
public class LaserReciever : MonoBehaviour
{
    [HideInInspector] public bool door;

    private void OnTriggerEnter(Collider cylinderCollider)
    {
        if (cylinderCollider.CompareTag("Laser"))
        {
            door = true;
            //Debug.Log("door: open");
        }
    }

    private void OnTriggerExit(Collider cylinderCollider)
    {
        if (cylinderCollider.CompareTag("Laser"))
        {
            door = false;
            //Debug.Log("door: close");
        }
    }
}
```

Рисунок 31 – Листинг компонента LaserReciever

5. Передатчик лазера внешне напоминающий обычный куб, представленный на рисунке 32, обладает способностью перенаправлять лазерный луч. Попадания лазера обрабатываются с помощью столкновения (*Collision*) двух коллайдеров. При попадании лазером в объект передатчика, компонент внутри лазера создает клон лазера в заранее определенной точке с одной из сторон.

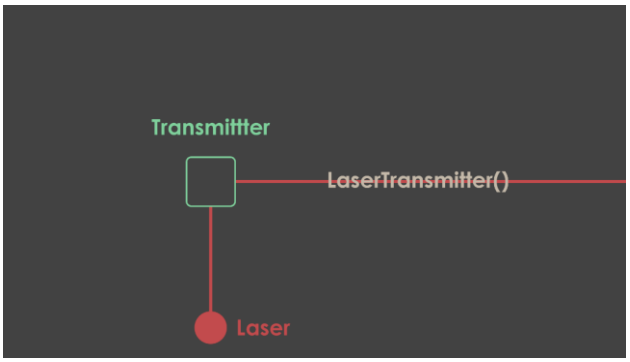


Рисунок 32 – Схема работы метода LaserTransmitter

При попадании лазера в объект передатчика, компонент *LaserTransmitter*, в соответствии с рисунком 33, создает клон лазера в заранее определенной точке (*point*) с одной из сторон передатчика. Это позволяет игроку легко менять направление лазера, взяв в руки объект передатчика.

```
private void OnCollisionEnter(Collision other)
{
    if (other.collider.CompareTag("Laser") && !_isTransmit)
    {
        if (_laserController == null) _laserController = other.collider.GetComponent<LaserController>();

        if (!_laserController.laserHitPlayer) CloneObject(other);

        _isTransmit = true;
    }
}

private void OnCollisionStay(Collision other)
{
    if (other.collider.CompareTag("Laser") && _cloneObject != null && !_laserController.laserHitPlayer && !_isTransmit)
    {
        LineRenderer cloneLineRenderer = _cloneObject.GetComponent<LineRenderer>();

        // Установка новой позиции клонированного объекта относительно портала
        cloneLineRenderer.SetPosition( index: 0,  position: new Vector3(point.position.x, point.position.y, point.position.z));

        _cloneObject.transform.position = cloneLineRenderer.GetPosition( index: 0);
        _cloneObject.transform.rotation = point.rotation;

        _isTransmit = true;
    }
}

private void CloneObject(Collision originalCollider)
{
    Quaternion originalRotation = originalCollider.transform.rotation;
    Vector3 clonePosition = new Vector3(point.position.x, point.position.y, point.position.z);

    _cloneObject = Instantiate(originalCollider.gameObject, clonePosition, originalRotation);
}
```

Рисунок 33 – Листинг компонента LaserTransmitter

6. Барьеры – это электрические поля, которые:

– не пропускают игрока и никак не влияют на движение других объектов или лазера;

– пропускают только игрока.

Барьеры в игре работают через столкновения коллайдеров и свойство компонента *Rigidbody* – *ExcludeLayers* (в переводе на русский исключенные слои).

Работа барьеров, представленная на рисунке 34, разделяется на следующие виды:

– для первого барьера, который пропускает только игрока, в созданном компоненте составлено условие «если игрок пытается пройти через него с объектом в руках, или если объект сам проходит через барьер, объект автоматически выбрасывается из рук игрока»;

– для второго барьера, который пропускает все объекты кроме игрока, в созданном компоненте составлено условие «когда у персонажа в руках находится объект, и он задевает этот барьер, то объект выбрасывается из его рук»;

– для третьего типа барьеров жидкости в пропастях создан компонент, который возвращает в начало головоломки игрока или объекты, попавшие в эту пропасть.

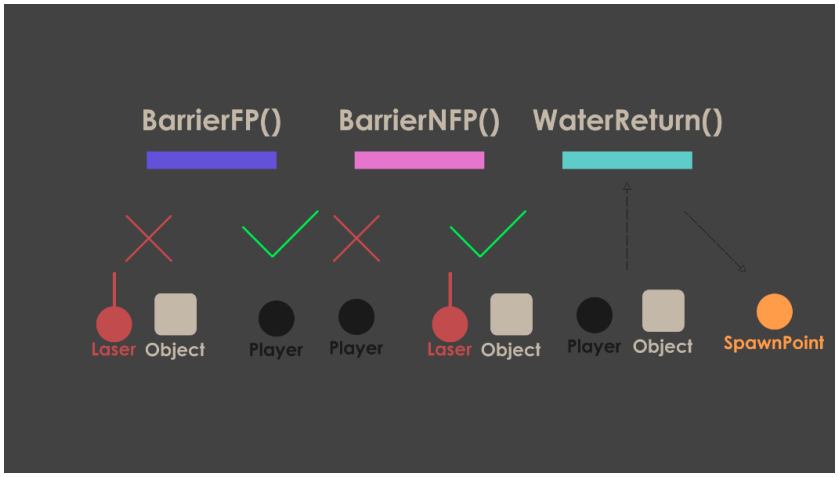


Рисунок 34 – Схема работы компонентов барьеров

Таким образом, реализация основных механик взаимодействия с окружающим миром, таких как подъем и бросок предметов, прохождение через барьеры и взаимодействие с лазерами, обогащает игровой процесс и создает основу для разнообразных головоломок. Сочетание этих механик с уникальной способностью клонирования персонажа открывает перед игроком широкие возможности для экспериментирования и поиска нестандартных решений.

2.2.3 Создание дополнительных игровых механик

Создание дополнительных механик очень важный процесс в разработке 3D компьютерных игр. Такие механики либо избавляют пользователя от рутинных действий, либо являются игровыми условностями, ограничивающими пользовательские действия.

Переход игрока из одной комнаты в другую является опасным фактором для разработчика, так как игрок может встать между комнатами или попытаться пробраться сквозь этот переход, не решив загадку, сломав основную логику построения уровня или совсем выведя из строя приложение. Для этой цели создан компонент, фрагмент кода которого представлен на рисунке 35: игрок не может находиться между комнатами, его всегда будет отталкивать в следующую комнату, а если пользователь решит пробраться сквозь дверь, существует коллайдер, размещенный и настроенный таким образом, что это сделать практически невозможно.

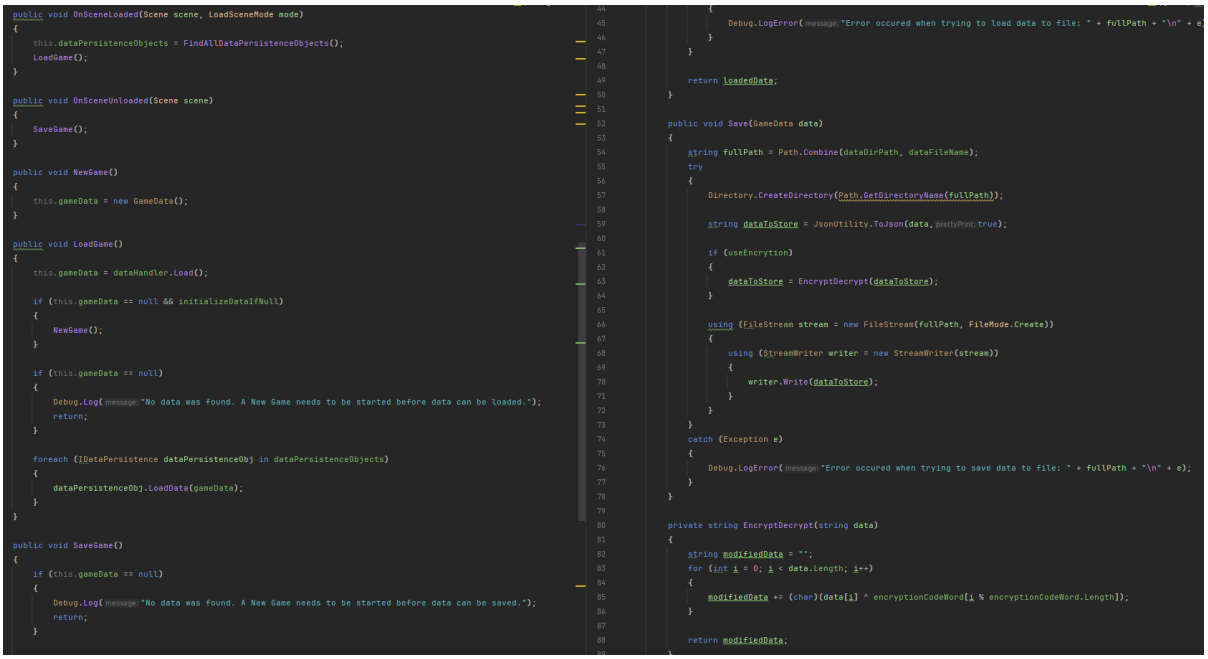
```
private void OnTriggerEnter(Collider other)
{
    if (other.CompareTag("Player"))
    {
        playerRB = other.GetComponent<Rigidbody>();
        isPlayerInside = true;
        CalculateTarget();
        _isExiting = !_isExiting;

        StartCoroutine(routine: SmoothAlphaValue(startRoof: _isExiting ? _preRoofMr : _postRoofMr,
            endRoof: _isExiting ? _postRoofMr : _preRoofMr,
            startOpacity: 0f, _targetOpacity));

        _isFirstExit = true;
    }
}
```

Рисунок 35 – Листинг компонента RoofController

Одной из главных особенностей новейших компьютерных игр, являются процессы сохранения и загрузки, позволяющие игроку сохранять полученный прогресс перед сложными моментами прохождения. Для этого разработана собственная система сохранений, в соответствии с рисунком 36, включающая в себя: компонент сохраняющихся данных, методы сохранений и загрузки через JSON файлы, а также метод шифрования данных.



```
public void OnSceneLoaded(Scene scene, LoadSceneMode mode)
{
    this.dataPersistenceObjects = FindAllDataPersistenceObjects();
    LoadGame();
}

public void OnSceneUnloaded(Scene scene)
{
    SaveGame();
}

public void NewGame()
{
    this.gameData = new GameData();
}

public void LoadGame()
{
    this.gameData = dataHandler.Load();

    if (this.gameData == null && initializeDataIfNull)
    {
        NewGame();
    }

    if (this.gameData == null)
    {
        Debug.Log(message:"No data was found. A New Game needs to be started before data can be loaded.");
        return;
    }

    foreach (IDataPersistence dataPersistenceObj in dataPersistenceObjects)
    {
        dataPersistenceObj.LoadData(gameData);
    }
}

public void SaveGame()
{
    if (this.gameData == null)
    {
        Debug.Log(message:"No data was found. A New Game needs to be started before data can be saved.");
        return;
    }
```

```
            {
                Debug.LogError(message:"Error occured when trying to load data to file: " + fullPath + "\n" + e);
            }
        }

        return loadedData;
    }

    public void Save(GameData data)
    {
        string fullPath = Path.Combine(dataDirPath, dataFileName);
        try
        {
            Directory.CreateDirectory(Path.GetDirectoryName(fullPath));

            string dataToStore = JsonUtility.ToJson(data, prettyPrint:true);

            if (useEncrytion)
            {
                dataToStore = EncryptDecrypt(dataToStore);
            }

            using (FileStream stream = new FileStream(fullPath, FileMode.Create))
            {
                using (StreamWriter writer = new StreamWriter(stream))
                {
                    writer.Write(dataToStore);
                }
            }
        }
        catch (Exception e)
        {
            Debug.LogError(message:"Error occured when trying to save data to file: " + fullPath + "\n" + e);
        }
    }

    private string EncryptDecrypt(string data)
    {
        string modifiedData = "";
        for (int i = 0; i < data.Length; i++)
        {
            modifiedData += (char)(data[i] ^ encryptionCodeWord[i % encryptionCodeWord.Length]);
        }

        return modifiedData;
    }
```

Рисунок 36 – Листинг компонентов системы сохранений

Для работы системы сохранений необходимо создать точки, в которые будет загружаться игрок. В игре созданы контрольные точки, размещенные в начале каждой комнаты, что предоставляет игроку свободу действий и возможность экспериментировать без риска потерять прогресс.

У каждого пользователя могут быть различные характеристики персональных компьютеров. Для этого созданы настройки игры, которые включают в себя настройки графики и звука. Эти данные тоже необходимо сохранять, иначе пользователь будет каждый запуск игры настраивать игру. В этом случае не нужно шифровать данные или выделять для них отдельный файл и можно воспользоваться сохранением настроек через класс *PlayerPrefs*

встроенный в *Unity*. Реализация сохранений через *PlayerPrefs* представлена на рисунке 37. Этот класс хранит настройки в локальном реестре платформы пользователя и удобен в использовании хранения не конфиденциальных данных.

```
public class SettingsMenuUI : MonoBehaviour
{
    public AudioSource musicSource;

    [SerializeField] private Toggle fullScreenToggle;
    [SerializeField] private Slider volumeSlider;
    [SerializeField] private TMP_Dropdown qualityDropdown;

    private void Awake()
    {
        musicSource.volume = PlayerPrefs.GetFloat(key: "volume");
        QualitySettings.SetQualityLevel(index: PlayerPrefs.GetInt(key: "quality"));
        Screen.fullScreen = PlayerPrefs.GetInt(key: "fullScreen") == 1;

        volumeSlider.value = PlayerPrefs.GetFloat(key: "volume");
        qualityDropdown.value = PlayerPrefs.GetInt(key: "quality");
        fullScreenToggle.isOn = PlayerPrefs.GetInt(key: "fullScreen") == 1;
    }

    public void SetVolume(float volume)
    {
        musicSource.volume = volume;
        PlayerPrefs.SetFloat("volume", volume);
    }

    public void SetQuality(int qualityIndex)
    {
        QualitySettings.SetQualityLevel(qualityIndex);
        PlayerPrefs.SetInt("quality", qualityIndex);
    }

    public void SetFullscreen(bool isFullscreen)
    {
        Screen.fullScreen = isFullscreen;
        PlayerPrefs.SetInt("fullScreen", isFullscreen ? 1 : 0);
    }
}
```

Рисунок 37 – Листинг компонента системы сохранений настроек

Таким образом, разработка дополнительных игровых механик, помимо основных, играет важную роль в создании комфортного и сбалансированного игрового опыта. Такое сочетание основных и дополнительных механик обеспечивает увлекательный и сбалансированный игровой процесс, способствуя положительному впечатлению игрока от игры.

2.2.4 Создание пользовательского интерфейса для управления игрой

Взаимодействие игрока с игрой осуществляется через пользовательский интерфейс, который служит «мостом» между человеком и виртуальным миром.

Одним из ключевых элементов пользовательского интерфейса является игровое меню, обеспечивающее доступ к основным функциям игры и настройкам.

В стандарте ISO/IEC/IEEE 24765:2017 [40] введен термин «Пользовательский интерфейс (с англ. user interface, далее UI) – все компоненты интерактивной системы (программные или аппаратные), которые предоставляют пользователю информацию и средства управления для выполнения конкретных задач с помощью интерактивной системы».

Игровое меню – это ключевой элемент пользовательского интерфейса, служащий мостом между игроком и функциональностью игры. Хорошо продуманное меню обеспечивает интуитивно понятную навигацию, доступ к настройкам, сохранениям и другим функциям, делая взаимодействие с игрой комфортным и приятным.

В рамках разработки 3D игры в жанре головоломка созданы два основных меню: главное меню и меню паузы. Каждое из них предназначено для решения определенных задач и содержит набор специфических функций, которые отвечают потребностям игрока на разных этапах взаимодействия с игрой.

При запуске игры, в соответствии с рисунком 38, игрок видит главное меню, которое является центральным узлом, предоставляя доступ к основным функциям игры.

Рисунок 38 – Скриншот главного меню

В главном меню пользователь может выбрать следующие элементы:

– «Новая игра»: позволяет игроку начать новую игровую сессию;

– «Продолжить»: открывает доступ к сохраненным играм, позволяя игроку продолжить прохождение с последней контрольной точки;

– «Настройки»: позволяет настроить различные параметры игры, такие как громкость звука и графические настройки, в соответствии с рисунком 39, а также рассмотреть раскладку управления игрой, в соответствии с рисунком 40;

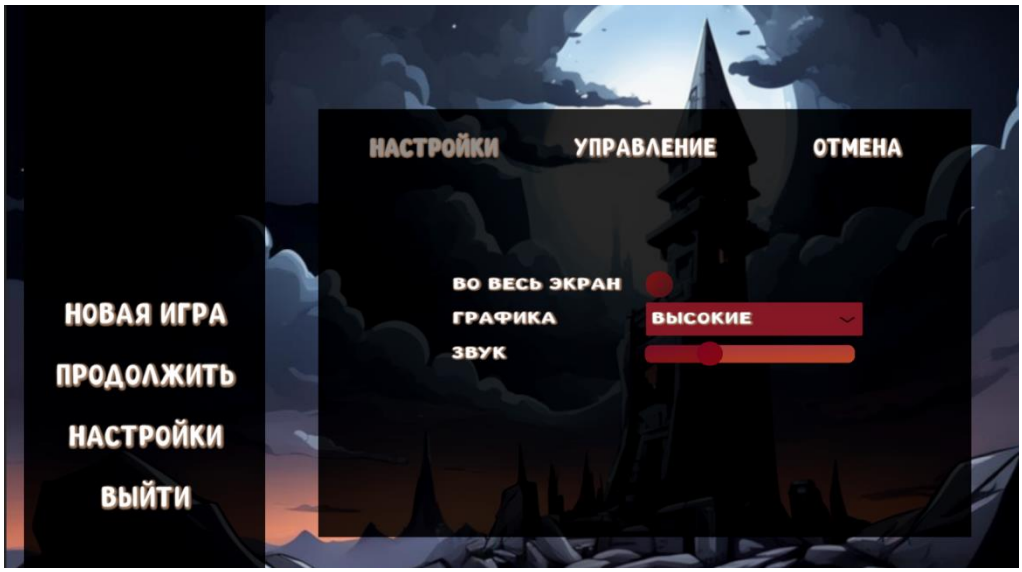


Рисунок 39 – Скриншот меню настроек

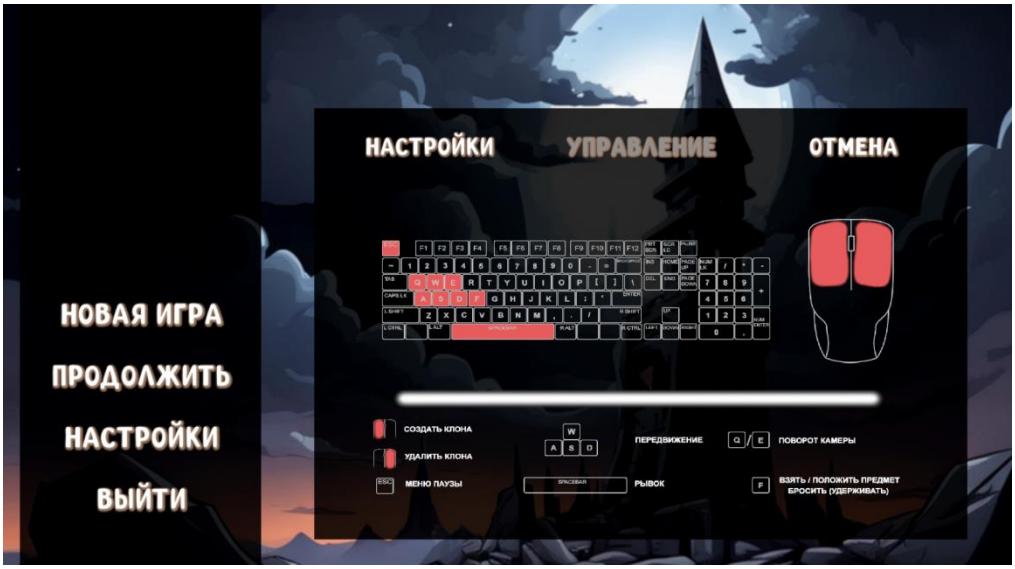


Рисунок 40 – Скриншот меню управления

– «Выйти»: завершает работу игры и возвращает пользователя на рабочий стол операционной системы.

Меню паузы, представленное на рисунке 41, становится доступно игроку непосредственно во время игрового процесса при нажатии клавиши «ESC». Оно предназначено для предоставления игроку возможности отдохнуть или загрузить игру без выхода из игровой сессии.

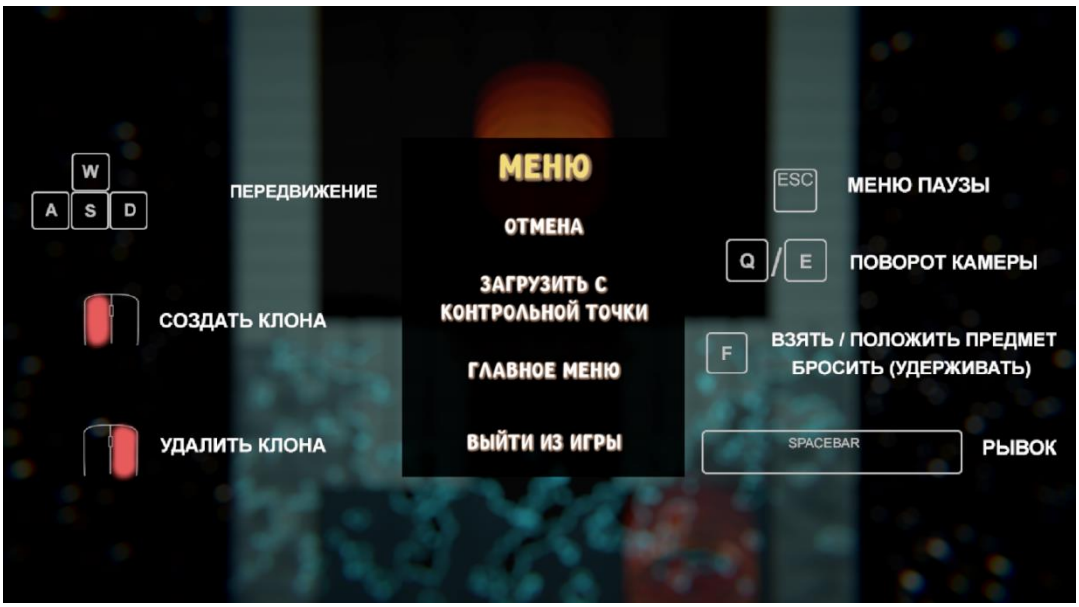


Рисунок 41 – Скриншот меню паузы

Меню паузы содержит следующие элементы:

– «Отмена»: позволяет закрыть меню паузы и вернуться к игровому процессу;

– «Загрузить с контрольной точки»: позволяет загрузить игру с последней сохраненной контрольной точки;

– «Главное меню»: возвращает игрока в главное меню, где он может начать новую игру, загрузить сохраненную игру, изменить настройки или выйти из игры;

– «Выйти из игры»: завершает работу игры;

– подсказки раскладки управления игрой.

Важно, чтобы меню паузы не перекрывало важную информацию на экране и не мешало ориентации в игровом мире, а также не отвлекало игрока от игрового процесса. Для этого создано размытие всего кроме самого меню паузы, чтобы сконцентрировать пользователя на том, что игра в данный момент неактивна.

Так же была разработана система диалогов для повествования сюжета в игре при помощи инструмента Ink – повествовательный язык программирования для игр. Работа в этом инструменте приведена в приложении Б. Для указания игроку с кем можно начать диалог, создана визуальная подсказка, в соответствии с приложением В. Которая исчезает, когда игрок закончит диалог. При разработке системы диалогов, создан UI для диалога с игровыми персонажами, приведенный в приложении Г.

Для создания меню в *Unity* использовались следующие инструменты и технологии:

– *Unity UI*: система пользовательского интерфейса в *Unity*, позволяющая создавать меню, кнопки, текстовые поля и другие элементы *UI*;

– *C# scripting*: язык программирования, используемый для написания скриптов, которые обрабатывают действия пользователя в меню, такие как нажатие кнопок и изменение настроек;

– Анимации: использованы для создания плавных переходов между меню и плавного изменения размера текста, когда пользователь наводит курсором на него.

Таким образом, создание интуитивно понятного и функционального UI для управления игрой является важной составляющей разработки любой игры, включая игры головоломки. Главное меню и меню паузы должны быть простыми в использовании и предоставлять доступ ко всем необходимым функциям игры. Платформа *Unity* предлагает разработчикам все необходимые инструменты для создания высококачественных игровых меню, которые повышают удобство и удовольствие от взаимодействия с игрой.

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной выпускной квалификационной работы была разработана 3D игра головоломка на платформе *Unity*, предлагающая игрокам увлекательный геймплей, основанный на взаимодействии механик порталов, лазеров, клонирования персонажа и интерактивных объектов. Проект реализован с использованием современных подходов к проектированию и разработке архитектуры игр, что обеспечило его гибкость, расширяемость и возможность повторного использования кода в будущих проектах.

В первой главе работы были рассмотрены теоретические основы разработки игр, включая историю развития жанра головоломок. Особое внимание было уделено изучению принципов игрового дизайна, включая дизайн уровней, игровую механику, нарратив и баланс.

Вторая глава была посвящена проектированию и разработке игровой механики, уровней и дополнительных материалов (модели объектов, музыка). Детально были проработаны механики порталов, лазеров, клонирования персонажа, а также определены принципы построения уровней с возрастающей сложностью. Были созданы 3D-модели объектов и окружения с использованием *Blender*, разработана система управления персонажем, реализованы основные и дополнительные игровые механики, такие как перенос объектов, стрельба лазерами и активация механизмов. Особое внимание было уделено динамическому звуковому сопровождению, меняющемуся в зависимости от игровой ситуации. Разработаны удобные и интуитивно понятные игровые меню с использованием *Unity UI*.

Разработка 3D игры в жанре головоломка стала ценным опытом, который позволил применить полученные знания и навыки в области разработки игр на практике. Созданный проект имеет потенциал для дальнейшего развития и может стать основой для создания коммерчески успешной игры.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Андрианова, Н. Как создавать истории. Основы игровой сценаристики и нарративного дизайна за 12 шагов / Н. Андрианова, С. Яковлева. – Москва : Альпина Паблишер, 2019. – 288 с.

2. Бейтс, Б. Дизайн игры / Б. Бейтс. – 2-е изд. – Бостон : Thomson Course Technology, 2004. – 472 с.

3. Беккер, А. Что такое баланс в игре? Исследование концепций / А. Беккер, Д. Гёрлих. – Вольфсгрубенвег : ParadigmPlus. – 2020. – С. 22–41.

4. Валенсия-Гарсия, Р. Технологии и инновации: Вторая международная конференция / Р. Валенсия-Гарсия. – Гуаякиль : Springer, 2016. – 281 с.

5. Вольф, Ю. Школа литературного и сценарного мастерства: от замысла до результата: рассказы, романы, статьи, нон-фикшн, сценарии, новые медиа / Ю. Вольф. – Лондон : The Van Lear Agency LLC, 2012. – 301 с.

6. Галузин, А. Предварительный план: как спланировать игровое окружение и дизайн уровней / А. Галузин. – 2-е изд. – Скоттс-Велли : CreateSpace Independent Publishing Platform, 2016. – 240 с.

7. Грис, С. Сделай видеоигру один и не свихнись / С. Грис. – Москва : АСТ, 2023. – 288 с.

8. Документация Blender // Blender : официальный сайт. – 2024. – URL: https://docs.blender.org (дата обращения: 04.03.2024).

9. Документация Unity // Unity : официальный сайт. – 2022. – URL: https://docs.unity3d.com (дата обращения: 04.03.2024).

10. Исбистер, К. Как игры движут нами: эмоции по замыслу (игровое мышление) / К. Исбистер. – Кембридж : Mit Pr, 2016. – 167с.

11. Клеон, О. Кради как художник. 10 уроков творческого самовыражения / О. Клеон. – Москва : МИФ, 2021. – 176 с.

12. Колесникова, В. Е. Головоломка / В. Е. Колесникова // Большая российская энциклопедия : [сайт]. – 2023 – URL: https://bigenc.ru/c/golovolomki-fa5a09 (дата обращения: 04.03.2024).

13. Кремерс, Р. Дизайн уровней: концепция, теория и практика / Р. Кремерс. – Бока-Ратон : CRC Press, 2009. – 408 с.

14. Кудряшов, И. С. Геймплей / И. С. Кудряшов // Большая российская энциклопедия [сайт]. – 2023 – URL: https://bigenc.ru/culture/text/1947088 (дата обращения: 04.03.2024).

15. Кэмпбелл, Д. Тысячеликий герой / Д. Кэмпбелл. – Санкт-Петербург : Питер, 2021. – 518 с.

16. Леонов, А. Г. Компьютерная игра / А. Г. Леонов, А. В. Ермолович // Большая российская энциклопедия [сайт]. – 2023 – URL: https://bigenc.ru/culture/text/2088372 (дата обращения: 04.03.2024).

17. Маргулец, В. 100 советов и подсказок по игровому дизайну / В. Маргулец. – Калифорния : Unfold Games, LLC, 2020. – 89 с.

18. Мовшовиц, Д. От идеи до злодея. Учимся создавать истории вместе с Pixar / Д. Мовшовиц. – Москва : Бомбора, 2019. – 101 с.

19. Найстром, Р. Шаблоны игрового программирования / Р. Найстром. – Женева: Genever Benning, 2014. – 389 с.

20. Рауз, Р. Дизайн игры: теория и практика / Р. Рауз. – 2-е изд. – Плано : Wordware Publishing, 2004. – 698 с.

21. Роллингз, Э. Проектирование и архитектура игр / Э. Роллингз, Д. Моррис. – 2-е изд. – Москва : Вильямс, 2006. – 698 с.

22. Руководство по C# // Microsoft Learn : официальный сайт. – 2024. – URL: https://learn.microsoft.com/ru-ru/dotnet/csharp/ (дата обращения: 04.03.2024).

23. Савченко, А. Игра как бизнес. От мечты до релиза / А. Савченко. – Москва : Бомбора, 2020. – 341 с.

24. Свинк, С. Game Feel: Руководство гейм-дизайнера по виртуальным ощущениям / С. Свинк. – Бока-Ратон : CRC Press, 2008. – 376 с.

25. Сикарт, М. Определение игровых механик / М. Сикарт. – Швеция : Game Studies Foundation. – 2008. – С. 1–14.

26. Тайнан, С. Геймдизайн. Рецепты успеха лучших компьютерных игр от Super Mario и Doom до Assassin’s Creed и дальше / С. Тайнан. – Санкт-Петербург : Питер, 2022. – 448 с.

27. Тайнан, С. Проектирование игр: Путеводитель по инженерному опыту / С. Тайнан. – Себастопол : O'Reilly Media, 2013. – 416 с.

28. Тилинина, Н. Ю. Программная инженерия: методы и технологии разработки информационно-вычислительных систем / Н. Ю. Тилинина, Н. Е. Губенко // Сборник научных трудов II научно-практической конференции (студенческая секция) / ответственный редактор Н. Ю. Тилинина. – Донецк, 2018. – С. 198–202.

29. Тоттен, К. Архитектурный подход к дизайну уровней / К. Тоттен. – Бока-Ратон : CRC Press, 2014. – 472 с.

30. Уточкин, В. Н. Хочу в геймдев! Основы игровой разработки для начинающих / В. Н. Уточкин, К. С. Сахнов. – Москва : Бомбора, 2022. – 250 с.

31. Уэйланд, К. Архитектура сюжета. Как создать запоминающуюся историю / К. Уэйланд. – Москва : Альпина нон-фикшн, 2020. – 230 с.

32. Филдс, Т. Дизайн социальных игр: методы и механика монетизации / Т. Филдс. – Бока-Ратон : CRC Press, 2011. – 220 с.

33. Фриман, Д. Создание эмоций в играх: ремесло и искусство создания эмоций / Д. Фриман. – Индианаполис : New Riders Pub, 2003. – 539 с.

34. Фуллертон, Т. Семинар игрового дизайна: игровой подход к созданию инновационных игр / Т. Фуллертон. – Бока-Ратон : CRC Press, 2008. – 556 с.

35. Хилл-Уиттол, Р. Справочник разработчика инди-игр / Р. Хилл-Уиттолл. – Абингдон : Routledge, 2015. – 278 с.

36. Хилтон, Т. Рендеринг порталов с помощью внеэкранных целей рендеринга / Т. Хилтон. // Bloger : [сайт]. – 2015. – URL: http://tomhulton.blogspot.com/2015/08/portal-rendering-with-offscreen-render.html (дата обращения: 04.03.2024).

37. Шрейер, Д. Кровь, пот и пиксели. Обратная сторона индустрии видеоигр / Д. Шрейер. – 2-е изд. – Москва : Эксмо, 2018. – 331 с.

38. Эгри, Л. Искусство драматургии / Л. Эгри. – Москва : Альпина нон-фикшн, 2020. – 360 с.

39. Юнгер, Ф. Игры. Ключ к их значению / Ф. Юнгер. – Москва : Владимир Даль, 2012. – 335 с.

40. ISO/IEC/IEEE 24765:2017. Systems and software engineering – Vocabulary // ISO : официальный сайт. – 2017. – URL: https://www.iso.org/obp/ui/en/#iso:std:iso-iec-ieee:24765:ed-2:v1:en (дата обращения: 24.05.2024).

ПРИЛОЖЕНИЕ А

Сравнительный анализ популярных игровых движков

Таблица А.1 – Сравнительный анализ популярных игровых движков

Критерий	**Unity**	**Unreal Engine**	**Game Maker**	**RPG Maker**	**Ren'Py**	**Godot**
3D Графика	+++	+++	+	-	-	+++
2D Графика	+++	+	+++	++	+++	+++
Программирование	+++	+++	++	+	+	+++
Скорость разработки	+++	++	++	++	+++	+++
Сообщество разработчиков	+++	+++	++	++	++	++
Поддержка платформ	+++	+++	+	+	+	+++
Стоимость лицензии для компаний с большим потенциалом	--	---	-	-	+++	+++
Документация	+++	+++	+	+	+	+++
Производительность проектов	+++	+++	++	++	++	+++
Магазин ассетов	+++	+++	+	+	-	+++
Выбор крупных разработчиков	++	+++	+	-	-	++
Выбор независимых разработчиков	+++	+++	++	++	++	+++

ПРИЛОЖЕНИЕ Б

Скриншот системы диалогов с помощью инструмента Ink

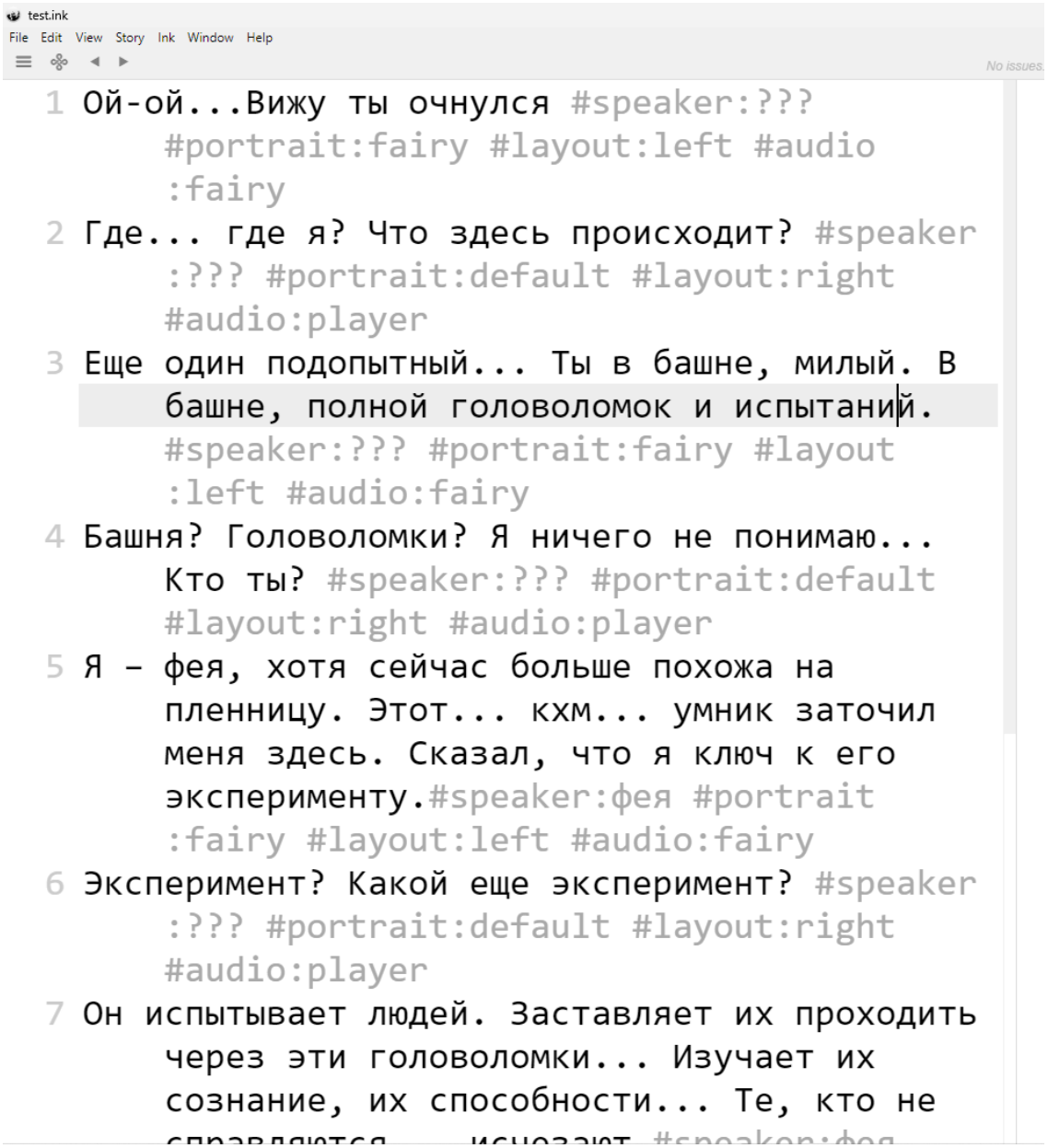


Рисунок Б.1 – Скриншот системы диалогов с помощью инструмента Ink

ПРИЛОЖЕНИЕ В

Визуальное представление UI диалога

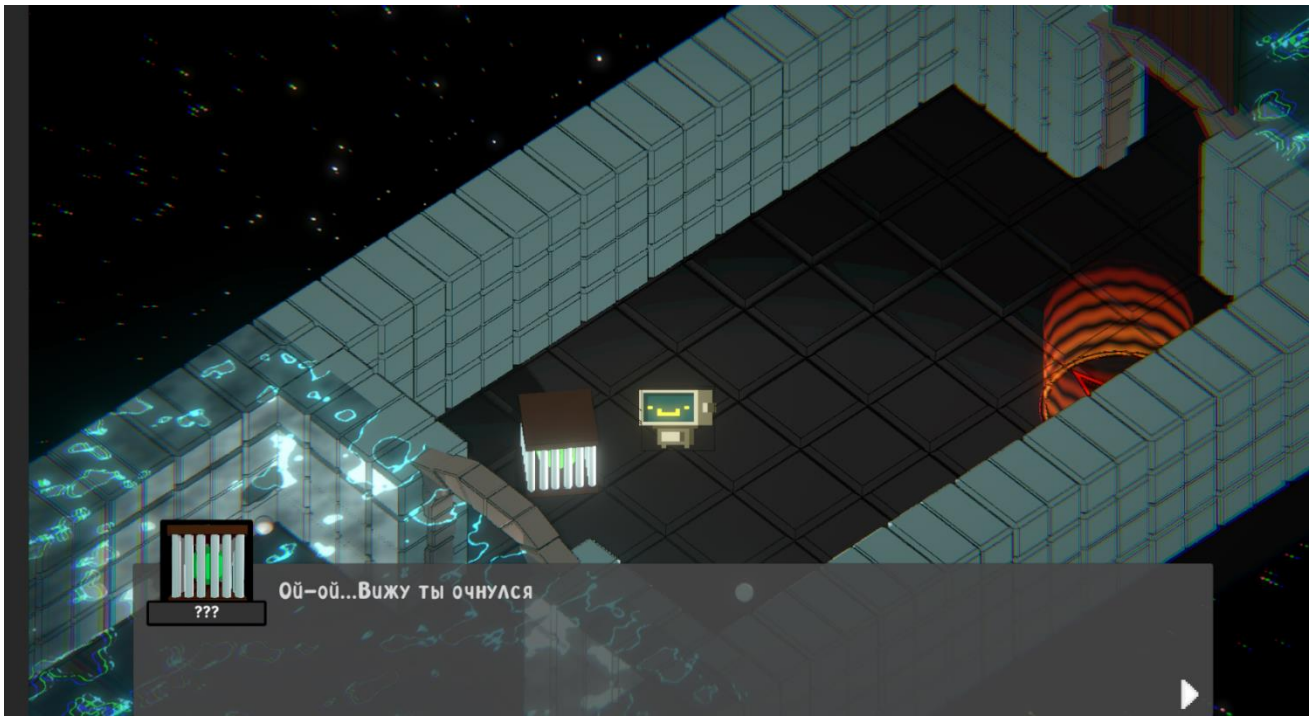


Рисунок В.1 – Визуальное представление UI диалога

ПРИЛОЖЕНИЕ Г

Визуальное представление всплывающей подсказки диалога

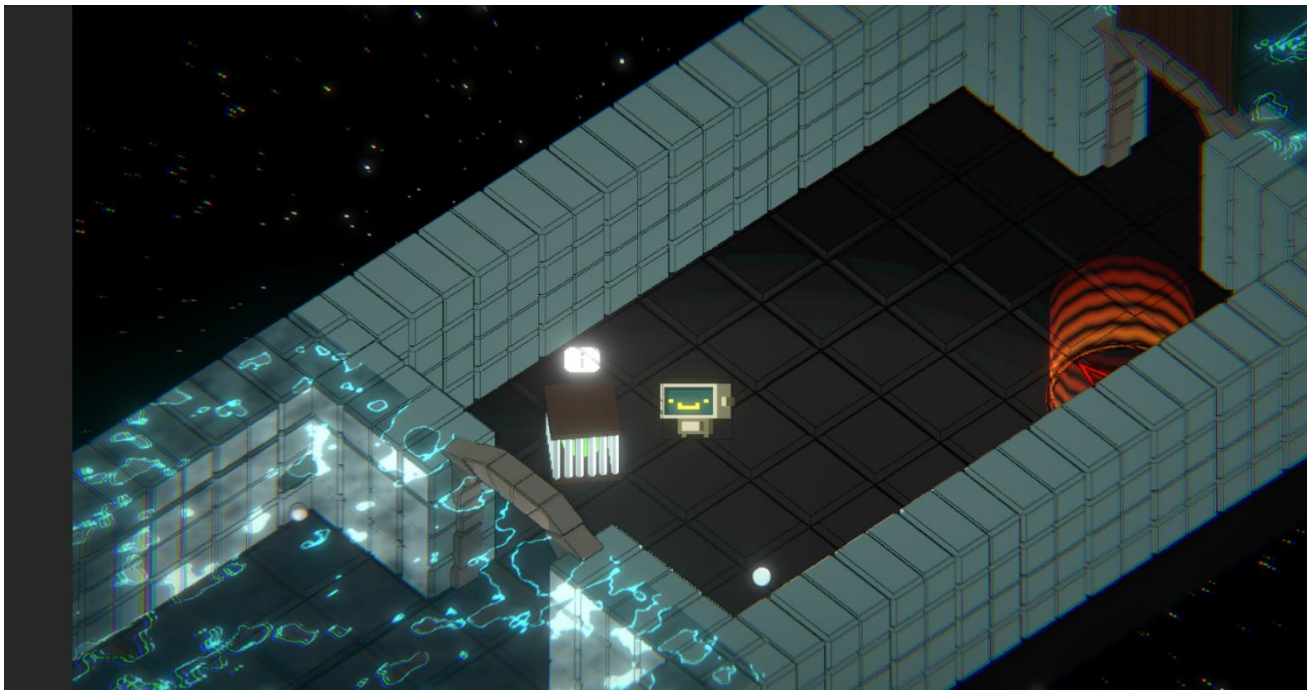


Рисунок Г.1 – Визуальное представление всплывающей подсказки диалога