

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»
Профиль подготовки: «Программное обеспечение средств вычислительной техники и автоматизированных систем»

БАКАЛАВРСКАЯ РАБОТА

на тему

ПРИЛОЖЕНИЕ В ЖАНРЕ ИЗОМЕТРИЧЕСКАЯ
ГОЛОВОЛОМКА. ФУНКЦИОНАЛЬНАЯ СОСТАВЛЯЮЩАЯ

Студент



Пантелеев Иван Алексеевич

(подпись, дата) 20.05.2024 ФИО полностью)

Руководитель



Юрова О.В.

(подпись, дата) 16.05.2024 (фамилия, инициалы)

Нормоконтролёр



Бычков А. С.

(подпись, дата) 20.05.2024 (фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 27.05.2024 г. № 11)

Заведующий кафедрой



Митрохин М. А.

(подпись) (фамилия, инициалы)

Работа защищена с отметкой хорошо (протокол заседания ГЭК от

24.05.2024 г. № 35)

Секретарь ГЭК



Е.И. Гурин

(подпись) (фамилия, инициалы)

Пенза 2024

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
М.А. Митрохин
« 19 » 04 2024 г.

**ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ**

1. Студент Пантелеев Иван Алексеевич гр. 20ВВП1 факультета вычислительная техника
2. Тема работы «Приложение в жанре изометрическая головоломка. Функциональная составляющая»
Тема утверждена приказом ПГУ №405/о от "19" апреля 2024г.
3. Руководитель бакалаврской работы к.т.н. доцент кафедры ВТ ПГУ Юрова О.В.
4. Задание на работу (назначение разработки, исходные данные и т.п.)
Назначение разработки – Приложение в жанре изометрическая головоломка.
Разработать сюжетное игровое приложение с головоломками разного типа. Подготовить и протестировать программную реализацию взаимодействия пользователя с объектами и игровыми локациями для решения головоломок. Язык программирования C#, среда разработки Visual Studio Code, игровой движок Unity, программное обеспечение для создания 3D моделей Blender, приложение для создания графических элементов Figma.
Программа должна обладать многообразием корректно реализованных механик, интуитивно понятным интерфейсом и качественной оптимизацией.
5. Перечень подлежащих разработке вопросов
Требуется произвести анализ предметной области, определить архитектуру приложения, выбрать инструменты для разработки, реализовать приложение.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Анализ предметной области	10 %	19.04.24 - 25.04.24	<i>фис</i>
2.	Выбор архитектуры системы и технологий, применяемых при проектировании и разработке	15 %	26.04.24 - 02.05.24	<i>фис</i>
3.	Разработка сценария, логики головоломок	20 %	03.05.24 - 07.05.24	<i>фис</i>
4.	Разработка уровней, моделей, текстур и интерфейса	25 %	08.05.24 - 19.05.24	<i>фис</i>
5.	Тестирование и отладка	15 %	20.05.24 - 22.05.24	<i>фис</i>
6.	Выполнение задания по БЖД	5 %	23.05.24 - 25.05.24	<i>фис</i>
7.	Оформление выпускной работы	10 %	25.05.23 - 31.05.23	<i>фис</i>

Дата выдачи задания "19" апреля 2024г.

Руководитель бакалаврской работы _____

(подпись, дата)

Задание к исполнению принял студент _____

(подпись, дата)

Работу к защите допустить
Декан факультета _____

(подпись, дата)

фис Юрова В.В.

Пантеев. И.А.

фис 21.06.2024 Рыкова Л.Р.

Реферат

Бакалаврская работа содержит 97 листов, 38 рисунков, 6 источников, 3 приложения

ГОЛОВОЛОМКА, UNITY, РАЗРАБОТКА НА C#, ГЕЙМДИЗАЙН, ИЗОМЕТРИЯ, ИНТЕРФЕЙС, ИГРОВОЕ ПРИЛОЖЕНИЕ

Цель работы – разработать приложение в жанре изометрическая головоломка. Основными элементами разработки приложения являются:

- создание концепции игрового мира, включающей в себя уровни, локации, объекты и головоломки;
- реализация дополнений игрового движка и различных игровых механик;
- внедрение и реализация различных типов головоломок и заданий, требующих логического мышления и решения;
- создание визуальных эффектов и анимаций;
- создание звукового оформления, включающее в себя фоновую музыку и звуковые эффекты;
- создание удобного интерфейса пользователя, обеспечивающего эффективное управление игровым процессом;
- интегрирование средств управления, таких как клавиатура и мышь;
- тестирование и оптимизация приложения.

В результате выполненных работ было разработано приложение, которое полностью соответствует поставленной задаче.

ПГУ1.090301.19.001 ПЗ				
Изм.	Лист	№ докум.	Подпись	Дата
Разраб.		Пантелеев И. А.		20.06.24
Провер.		Юрова О. В.		20.06.24
Н. Контр		Бычков А. С.		20.06.24
Утверд.		Митрохин М. А.		20.06.24
Приложение в жанре изометрическая головоломка. Функциональная составляющая. Пояснительная записка				
		Лист	Лист	Листов
		1	2	97
ФВТ, каф. ВТ гр. 20ВВП1				

Содержание

Введение.....	5
1 Приложение в жанре изометрическая головоломка.....	6
1.1 Анализ предметной области	6
1.2 Анализ аналогов	7
1.2.1 «Monument Valley»	7
1.2.2 «Tiny Room Stories: Town Mystery»	9
1.2.3 «The Almost Gone»	11
1.3 Постановка задачи	13
1.4 Выбор инструментов для разработки приложения	14
1.4.1 Игровой движок	14
1.4.2 Вспомогательные инструменты	15
1.5 Описание приложения	17
1.5.1 Игрок	17
1.5.2 Игровой мир:	17
1.5.3 Объекты взаимодействия:	21
1.5.4 Головоломки.....	21
1.5.5 Система сообщений	22
1.5.6 Инвентарь	23
1.5.7 Интерфейс пользователя (UI)	24
1.6 Разработка приложения.....	24
1.6.1 Основные компоненты	24
1.6.2 Игровые механики	25
1.6.2.1 Механика вращения комнаты	25
1.6.2.2 Механика смены комнат	28
1.6.2.3 Механика приближение к объектам	29
1.6.2.4 Механика вывода сообщений	31

1.6.2.5 Механика инвентаря	33
1.6.3 Паттерны проектирования	35
1.6.3.1 Одиночка.....	35
1.6.3.2 Команда.....	36
1.6.4 Пользовательский интерфейс	37
2 Охрана труда.....	40
2.1 Организация рабочего места при работе на персональном компьютере.....	40
2.1.1 Освещение	40
2.1.2 Общие эргономические требования к рабочему месту оператора ...	41
2.1.3 Климатические условия	43
2.2 Психофизиологические аспекты труда программиста	44
2.3 Электробезопасность	44
2.3.1 Использование исправного оборудования	44
2.3.2 Заземление и защитные устройства	44
2.4 Пожарная безопасность	45
Заключение	46
Список использованных источников	47
Приложение А	48
Приложение Б	52
Приложение В.....	89

Введение

Современные технологии позволяют создавать интерактивные и увлекательные игровые приложения, которые становятся неотъемлемой частью досуга миллионов людей по всему миру. Игровая индустрия активно развивается, предлагая все больше возможностей для творчества и инноваций. В этом контексте игровые приложения охватывают широкий спектр жанров и стилей, каждый из которых привносит уникальные элементы и подходы к игровому процессу.

Среди множества жанров особое внимание привлекают головоломки, которые требуют от игроков не только реакции и координации, но и активного использования логического мышления и планирования. Головоломки стимулируют умственную деятельность, предлагая игрокам решать сложные задачи и преодолевать разнообразные интеллектуальные вызовы. В последние годы особую популярность приобретают изометрические головоломки, объединяющие в себе увлекательный игровой процесс и визуально привлекательную изометрическую перспективу.

Изометрическая головоломка представляет собой жанр, в котором игроки взаимодействуют с псевдотрёхмерной средой, представленной под углом, создающим иллюзию объема. Это сочетание стратегического мышления и эстетически приятного визуального исполнения делает изометрические головоломки особенно интересными и популярными среди широкого круга игроков.

Целью данной работы является разработка игрового приложения в жанре изометрическая головоломка с использованием языка программирования C# и игрового движка Unity. Эта работа нацелена на создание уникального игрового опыта, объединяющего захватывающие визуальные эффекты и глубокую, продуманную механику игры.

1 Приложение в жанре изометрическая головоломка

1.1 Анализ предметной области

Игры в жанре изометрических головоломок представляют собой уникальный тип игр, объединяющий в себе элементы головоломок, стратегии и визуально привлекательного дизайна. Этот жанр игр ставит перед игроками разнообразные задачи и головоломки, решение которых требует логического мышления, творческого подхода и стратегического планирования. Важными характеристиками таких игр являются:

- уникальный геймплей. Игры в жанре изометрических головоломок должны предложить игрокам оригинальные и увлекательные головоломки, которые будут вызывать интерес и мотивировать на преодоление новых уровней.

- привлекательный визуальный стиль. Визуальное оформление игры играет важную роль в создании атмосферы и привлечении внимания игроков. Игры в этом жанре обычно имеют яркую и красочную графику, создающую уникальные и запоминающиеся игровые миры.

- логическая сложность. Хорошие игры в жанре изометрических головоломок должны предложить игрокам разнообразные и сложные задачи, требующие тщательного анализа, экспериментов и инновационного мышления для их решения.

- интуитивное управление и интерфейс. Управление игрой должно быть простым и интуитивно понятным, чтобы игроки могли сосредоточиться на решении головоломок, а не на изучении управления.

- разнообразие уровней и контента. Хорошие игры в этом жанре предлагают разнообразие уровней и контента, чтобы поддерживать интерес игроков на протяжении всей игры. Разнообразие головоломок, обстановок и сюжетных элементов делает игру увлекательной и непредсказуемой.

Для создания хорошей игры в жанре изометрических головоломок необходимо учитывать эти ключевые аспекты при проектировании и разработке игрового приложения. Использование современных технологий разработки, качественной графики, интересных головоломок и увлекательного сюжета поможет создать игру, которая будет привлекательной для широкой аудитории игроков.

1.2 Анализ аналогов

На рынке существует множество игр в жанре изометрических головоломок, предлагающих разнообразные механики игры и уникальные решения. Среди наиболее известных представителей этого жанра можно выделить игры такие как «Monument Valley», «Tiny Rooms Story» и «The Almost Gone». Изображения игрового процесса представлены на рисунках 1.1, 1.2, 1.3.

1.2.1 «Monument Valley»

Игра «Monument Valley» представляет собой выдающийся пример изометрической головоломки, она предлагает уникальный игровой опыт, объединяя в себе выдающийся визуальный дизайн и интригующий геймплей.

Данная игра представляет собой последовательность уровней-головоломок каждый из которых вдохновлен популярными постройками или оптическими иллюзиями. Фрагмент игрового процесса представлен на рисунке 1.1.



Рисунок 1.1 – Фрагмент игрового процесса «Monument Valley»

Сильными сторонами этой игры можно назвать:

– визуальное оформление. Игра обладает великолепным дизайном уровней и персонажей, который вдохновлен монохромными геометрическими формами и оптическими иллюзиями. Это создает уникальный и неповторимый атмосферный мир;

– инновационный геймплей. "Monument Valley" предлагает игрокам уникальный геймплей, основанный на взаимодействии с архитектурными элементами и перспективными иллюзиями. Игровые механики строятся на вращении и сдвиге структур для создания путей к целям;

– атмосферный саундтрек. Один из ярких аспектов игры – это красивая музыка и звуковое сопровождение, которые подчеркивают атмосферу загадочного мира и добавляют эмоциональную глубину к игровому опыту;

– креативные головоломки. Уровни в игре представляют собой уникальные головоломки, которые требуют от игрока нестандартного мышления и внимательности к деталям. Это поддерживает интерес к игре и стимулирует игрока на новые открытия.

Слабыми сторонами игры можно назвать:

– короткая длительность игры. Одним из основных недостатков игры является ее относительно короткое время прохождения;

– отсутствие глубокого сюжета. В «Monument Valley» сюжет игры остается довольно абстрактным, что может не удовлетворить тех игроков, которые ищут более глубокие и захватывающие повествования;

– отсутствие нарастающей сложности уровней. После определенного этапа, все головоломки в игре перестают быть для игрока сложными, их прохождение становится однообразным.

«Monument Valley» является вдохновляющим примером игры в жанре изометрической головоломки, который привлекает внимание своим уникальным дизайном, инновационным геймплеем и красивой атмосферой. Несмотря на свои недостатки, игра успешно демонстрирует, как с помощью креативных идей и качественного исполнения можно создать запоминающийся игровой опыт.

1.2.2 «Tiny Room Stories: Town Mystery»

«Tiny Room Stories: Town Mystery» – это увлекательная головоломка, разработанная Kiary Games. Игра предлагает пользователю продвигаясь по сюжетной линии, исследовать город и решать различные головоломки, чтобы раскрыть загадки его прошлого. Фрагмент игрового процесса представлен на рисунке 1.2.

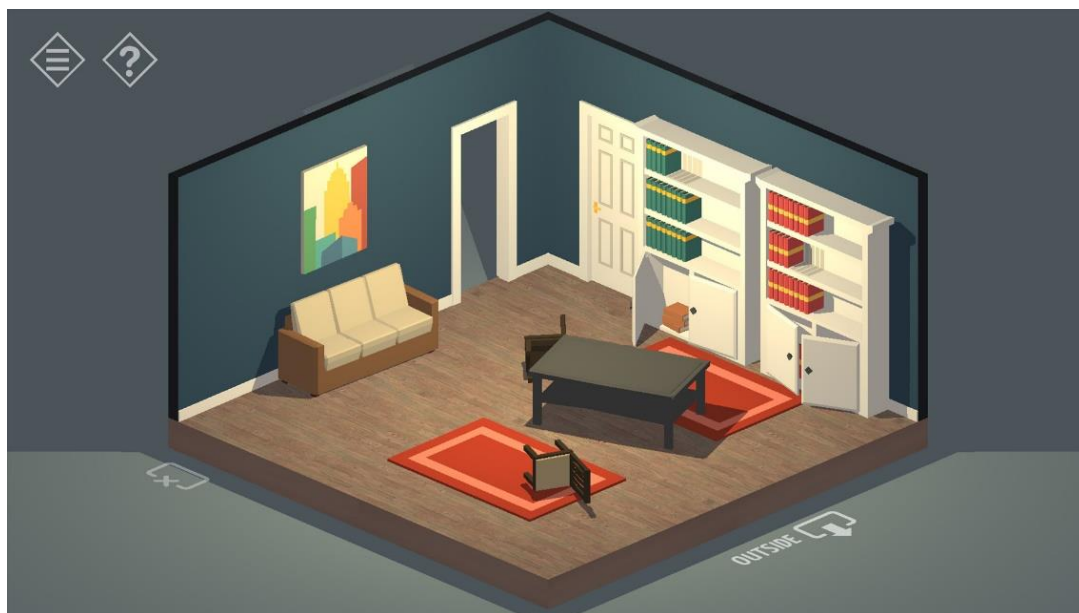


Рисунок 1.2 – Фрагмент игрового процесса «Tiny Room Stories: Town Mystery»

Сильные стороны «Tiny Room Stories: Town Mystery»:

- увлекательный сюжет. Игра предлагает интересный сюжет, наполненный тайнами и загадками, который заставляет игрока погрузиться в атмосферу таинственного города. Отдельно стоит отметить, внедрение в сюжет элементов детектива, что хорошо сказывается на атмосфере;

- интересные головоломки. «Tiny Room Stories» предлагает разнообразные и оригинальные головоломки, которые требуют логического мышления и внимания к деталям. Это добавляет глубину и разнообразие в игровой процесс;

- простота управления. Управление в игре интуитивно понятно и легко осваивается, что делает процесс игры доступным для широкого круга пользователей.

Слабые стороны «Tiny Room Stories: Town Mystery»:

- ограниченный геймплей. Несмотря на интересный контекст головоломок, игра может страдать от ограниченности в игровом процессе и малого разнообразия действий;

– анимации. Для некоторых действий в игре реализована очень плохая анимация, которая заставляет игрока отвлекаться и обращать на нее лишнее внимание;

– нереалистичность. Для игры, которая преподносит свой сюжет максимально приближенным к реальности, в ней присутствует множество действий и головоломок, которые не соответствуют этому свойству.

«Tiny Room Stories: Town Mystery» представляет собой увлекательную игру с интересным сюжетом и оригинальными головоломками, которая подарит игрокам много часов увлекательного времяпрепровождения. Несмотря на некоторые ограничения, игра привлекает внимание своей атмосферой и интригующими загадками.

1.2.3 «The Almost Gone»

«The Almost Gone» – это захватывающая головоломка-приключение. Игра предлагает игроку исследовать загадочные миры, раскрывая историю главного героя через решение головоломок и разгадывание тайн. Фрагмент игрового процесса представлен на рисунке 1.3.

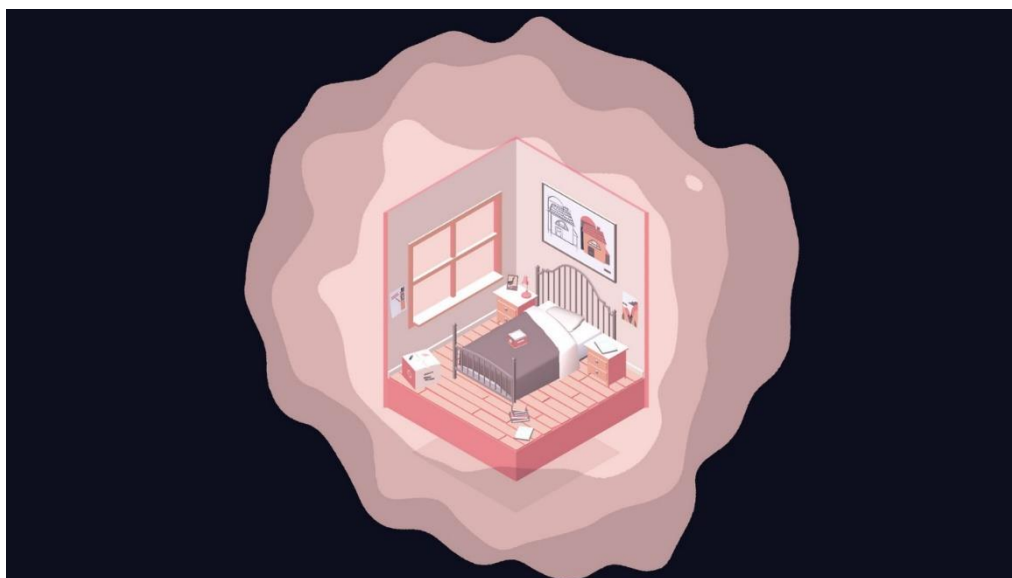


Рисунок 1.3 – Фрагмент игрового процесса «The Almost Gone»

Сильными сторонами игры можно считать:

– анимации и обработка действий. В игре очень качественно реализованы переходы и отклик на действия персонажа, все анимации соответствуют одному спокойному стилю;

– игровые механики. В реализована необычная механика осмотра предметов и групп объектов, они расположены в отдельном окне за пределами комнаты, что позволяет одновременно видеть предметы в нескольких перспективах и не пропускать важные детали;

– управление. В игре очень простое управление, зависящее только от мыши, что позволяет легко адаптироваться даже когда играешь впервые. К тому же есть возможность показывать некоторые элементы только когда они будут нужны, что хорошо сказывается на общем виде игровой сцены.

Из слабых сторон игры можно выделить:

– перемещение игрока. Зона видимости игрока ограничена не комнатой или уличным пространством, а их небольшой частью, которые в свою очередь можно поворачивать вокруг своей оси, это несомненно удобно для управления вниманием игрока, и позволяет ему сфокусироваться только на нужных предметах в каждой из таких частей, но нередко это заставляет игрока потерять своё местоположение в общем пространстве, и не понимать в каком именно месте комнаты или улицы он сейчас находится;

– перегруженность. Из-за своих небольших размеров и большого количества объектов, сцена, выглядит слишком заполненной.

«The Almost Gone» – это захватывающая и эмоциональная игра, которая предлагает игроку уникальный игровой опыт с глубоким сюжетом и оригинальными головоломками. Несмотря на некоторые недостатки, игра привлекает внимание своей атмосферой и захватывающим сюжетом.

1.3 Постановка задачи

Необходимо разработать функциональную составляющую игрового приложения в жанре изометрическая головоломка. Разработка должна включать в себя следующие аспекты:

– проектирование игрового мира:

1) разработать концепцию игрового мира, включающую в себя уровни, локации, объекты и головоломки;

2) создать игровой дизайн уровней с учетом сложности и прогрессии игрового процесса;

– реализация дополнений игрового движка и различных игровых механик:

1) разработать механику изучения локации, перемещения между локациями, и взаимодействия с объектами в изометрической среде;

2) внедрить и реализовать различные типы головоломок и заданий, требующих логического мышления и решения;

– создание визуальных и звуковых эффектов:

1) разработать визуальные элементы, включая арт-аспекты уровней, персонажей и объектов;

2) создать звуковое оформление, включающее фоновую музыку, звуковые эффекты и звуковые индикации действий игрока;

– интеграция средств управления:

1) создать удобный интерфейс пользователя, обеспечивающий эффективное управление игровым процессом;

2) интегрировать средства управления, такие как клавиатура и мышь

– тестирование и оптимизация. Провести тестирование игры на различных устройствах для выявления ошибок и недочетов.

1.4 Выбор инструментов для разработки приложения

1.4.1 Игровой движок

В качестве движка для разработки приложения выбран был выбран Unity, так как это один из самых популярных и мощных игровых движков, используемых разработчиками по всему миру. Он предоставляет обширные возможности для создания как 2D, так и 3D игр, а также поддерживает различные платформы, включая персональные компьютеры и мобильные устройства на операционных системах Android и IOS.

Преимущества использования Unity:

- изометрическое моделирование. Unity предоставляет инструменты для создания сложных сцен с изометрической проекцией, что идеально подходит для разработки изометрических головоломок. Возможности настройки камеры и управления перспективой позволяют легко создавать псевдотрёхмерные миры;

- поддержка C#. Unity использует язык программирования C#, что предоставляет разработчикам мощные возможности для написания кода. Язык программирования C# – это объектно-ориентированный язык, который обеспечивает высокую производительность и удобство при разработке сложных игровых механик;

- богатый набор инструментов и ресурсов. Движок предоставляет широкий спектр инструментов для работы с графикой, физикой, анимацией и звуком;

- кроссплатформенность. Unity поддерживает экспорт игр на различные платформы, что позволяет создать игру, доступную для широкой аудитории[1]. Это особенно важно для независимых разработчиков, стремящихся охватить как можно больше пользователей;

- обширная документация и сообщество. Unity имеет подробную документацию и активное сообщество разработчиков. Это облегчает процесс обучения и решения возникающих вопросов в ходе разработки. Разработчики

могут найти множество примеров, руководств и уроков, что упрощает освоение инструментария Unity.

1.4.2 Вспомогательные инструменты

Для успешной реализации проекта, помимо Unity, будут использованы следующие вспомогательные инструменты:

– среда разработки Visual Studio Code. Данное приложение представляет собой текстовый редактор, который широко используется для написания и редактирования кода. Он идеально подходит для разработки на языке C# в среде Unity благодаря своему богатому набору расширений и плагинов. Visual Studio Code предоставляет следующие преимущества:

1) поддержка C#. специализированные расширения, обеспечивают подсветку синтаксиса, автодополнение кода и интеллектуальные советы (IntelliSense);

2) лёгкость и скорость. Visual Studio Code является легковесным редактором, который запускается быстро и потребляет меньше ресурсов по сравнению с полноценными IDE;

3) расширяемость. Множество доступных плагинов позволяет адаптировать Visual Studio Code под конкретные потребности проекта, например, интеграция с Git или отладка C#;

4) интеграция с Unity. Благодаря расширениям, таким как Unity Tools, Visual Studio Code легко интегрируется с Unity, поддерживая запуск редактора прямо из игрового движка и предоставляя инструменты для отладки;

– графический редактор Figma. Данный редактор является мощным инструментом для создания и редактирования графики, который особенно эффективен для разработки интерфейсов и прототипов. В отличие от традиционных графических редакторов, Figma может работать в браузере и поддерживает совместную работу в реальном времени. Основными преимуществами использования Figma в проекте, являются:

1) прототипирование UI. Figma идеально подходит для проектирования и создания интерфейсов пользователя. Она позволяет легко создавать макеты экранов и интерфейсных элементов;

2) совместная работа. Благодаря облачной платформе, Figma позволяет нескольким разработчикам и дизайнерам работать над проектом одновременно, что упрощает совместное редактирование и обсуждение изменений;

3) Экспорт и интеграция. Figma поддерживает экспорт графических ресурсов в форматы, совместимые с Unity, такие как PNG и SVG, что облегчает интеграцию визуальных элементов в игру;

– программа 3D-моделирования Blender. Blender — это бесплатное и мощное программное обеспечение для 3D-моделирования, которое будет использоваться для создания трёхмерных объектов и моделей, необходимых для изометрических сцен. Blender поддерживает экспорт моделей в форматы, совместимые с Unity, что позволяет легко интегрировать 3D-контент в игровой проект. Преимуществами использования Blender, являются:

1) широкие возможности моделирования. Blender предлагает множество инструментов для создания сложных 3D-моделей, текстурирования и анимации;

2) экспорт и интеграция с Unity. Поддержка экспорта в форматы FBX и OBJ делает процесс интеграции созданных моделей в Unity простым и быстрым;

3) активное сообщество. Большое сообщество пользователей Blender предоставляет множество обучающих материалов и ресурсов, что облегчает процесс обучения и использования программы.

1.5 Описание приложения

Основной концепцией приложения является процесс управления игровым миром, с целью найти и собрать различные предметы, чтобы узнать больше о событиях игрового мира и обеспечить продвижение истории.

1.5.1 Игрок

В данном приложении нет физического персонажа, которого мы видим на экране. Вместо этого пользователь осуществляет действия и взаимодействовать с миром через интерфейс и курсор, что создает иллюзию присутствия.

Игрок может перемещаться по комнатам и взаимодействовать с элементами окружения путем наведения курсора и нажатия на объекты. Таким образом, все действия выполняются через интерфейс и курсор, позволяя игроку чувствовать себя активным участником игрового процесса, без необходимости отображения физического персонажа на экране.

Такой подход делает игру более абстрактной и сфокусированной на головоломках и взаимодействии с окружающим миром.

1.5.2 Игровой мир:

Игровое приложение представляет собой набор комнат, выполненных в изометрической проекции. Комнаты содержат различные объекты, такие как мебель, механизмы, и элементы интерьера, с которыми игрок может взаимодействовать. На рисунках 1.4 – 1.6 продемонстрированы комнаты игрового мира.

У игрока есть возможность управлять вращением комнаты, для более детального изучения её содержимого. Вращение комнаты продемонстрировано на рисунках 1.6, 1.7.

Также игрок может выполнять приближение к конкретному объекту или группе объектов, для получения доступа к взаимодействию к более мелким или отдаленным объектам. Приближение к объектам комнаты показано на рисунках 1.8, 1.9.

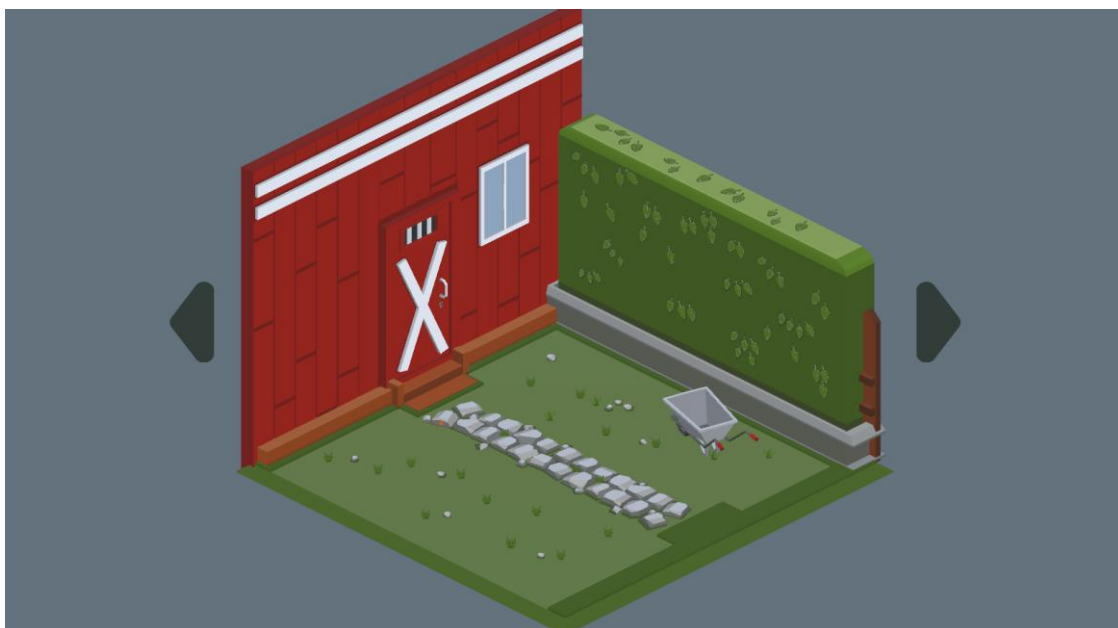


Рисунок 1.4 – Игровая комната «Улица»



Рисунок 1.5 – Игровая комната «Гостиная»

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.19.001 ПЗ

Лист

18



Рисунок 1.6 – Игровая комната «Детская», первое состояние поворота



Рисунок 1.7 – Игровая комната «Детская», второе состояние поворота



Рисунок 1.8 – Игровая комната «Гостинная», приближение к группе объектов, при первом состоянии поворота



Рисунок 1.9 – Игровая комната «Гостинная», приближение к группе объектов, при втором состоянии поворота

1.5.3 Объекты взаимодействия:

Игровое приложение включает множество интерактивных объектов. Таких как рычаги, ключи, детали и механизмы. Каждый объект может играть роль в решении головоломок, открывая новые возможности и продвигая игрока к цели. На рисунке 1.10 продемонстрированы интерактивные игровые объекты.



Рисунок 1.10 – Игровые объекты «Шахматные фигуры»

1.5.4 Головоломки

Основной игровой процесс построен вокруг решения головоломок. Эти головоломки варьируются от простых задач, таких как поиск ключа, до сложных механических головоломок, требующих логического мышления и последовательных действий. Головоломки могут включать в себя комбинации предметов, последовательности действий и управление различными механизмами. На рисунке 1.11 показана головоломка с кубиками, целью которой является последовательный поиск кубиков по возрастанию числа их точек скрепления.

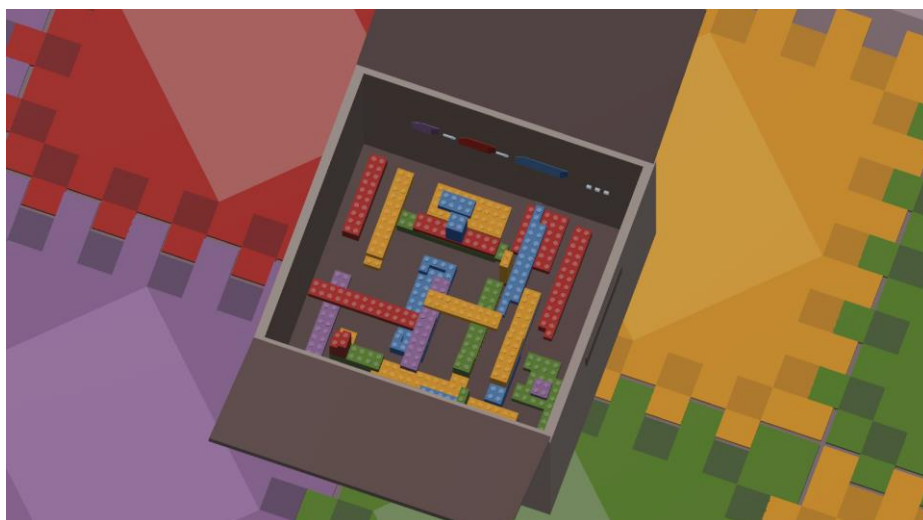


Рисунок 1.11 – Головоломка «Кубики»

1.5.5 Система сообщений

Для поддержки игроков, которые могут столкнуться с трудностями, и для более глобального раскрытия игрового сюжета, игра содержит систему сообщений. Эти сообщения привязаны к конкретным объектам или событиям в игре и могут быть активированы по нажатию. Они будут предоставлять намек, описания предметов или мысли персонажа. Сообщения исчезают по истечению 5 секунд после появления. На рисунках 1.12, 1.13 показан простой пример вывода сообщения в ответ на нажатие персонажа по объекту.

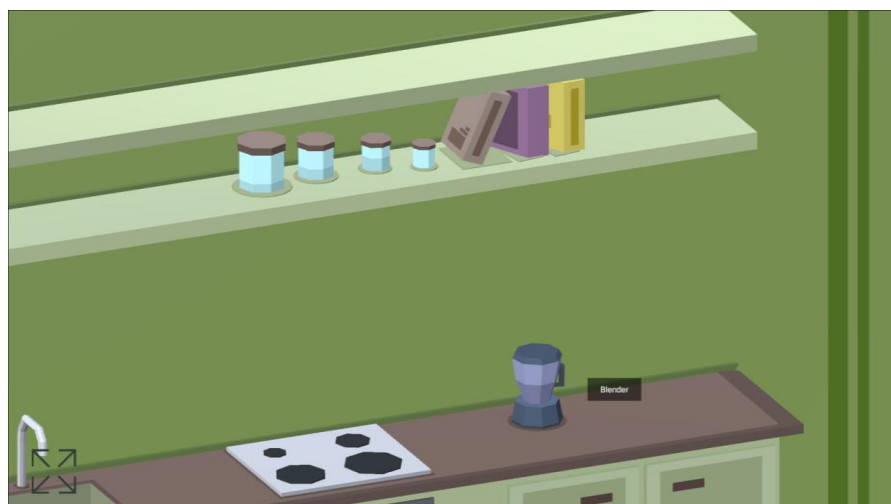


Рисунок 1.12 – Вывод сообщения, по нажатию на объект, первое состояние поворота



Рисунок 1.13 – Вывод сообщения, по нажатию на объект, второе состояние поворота

1.5.6 Инвентарь

Персонаж будет иметь инвентарь, в который он может собирать найденные предметы. Игрок сможет просматривать содержимое инвентаря и использовать предметы для решения головоломок и взаимодействия с объектами. На рисунке 1.14 изображен инвентарь, содержащий предмет – «Ножницы».



Рисунок 1.14 – Демонстрация инвентаря пользователя

Изм.	Лист	№ докум.	Подпись	Дата

1.5.7 Интерфейс пользователя (UI)

Интерфейс включает в себя:

- элементы инвентаря, содержащий предметы для решения головоломок и выполнения различных действий на локации;
- кнопки управления комнатой, обеспечивающие простой способ управлять состоянием комнаты;
- выводимые на экран сообщения, предоставляющие игроку информацию о найденных предметах и возможности применения конкретных предметов;
- меню приложения, позволяющее настроить громкость звуковых эффектов и музыки, а также выйти из игры.

Пользовательский интерфейс приложения довольно простой и в то же время понятный, что позволяет пользователю не отвлекаться от игрового процесса.

1.6 Разработка приложения

1.6.1 Основные компоненты

Архитектура данного приложения построена с учетом стандартов и практик разработки игр на движке Unity, и включает в себя следующие основные компоненты:

- сцена (Scene). Сцена представляет собой визуальное пространство, где размещается игровой мир. В Unity сцена может содержать объекты, свет, камеры и другие элементы, создающие игровую среду[2];
- объекты (Game Objects). Объекты – это основные строительные блоки игрового мира. Они могут быть представлены в виде 3D-моделей, 2D-спрайтов или простых геометрических форм. Каждый объект имеет свои компоненты, определяющие его поведение и внешний вид;
- скрипты (Scripts). Скрипты в Unity пишутся на языке программирования C#. Они представляют собой программный код, который определяет логику и

поведение игровых объектов. Скрипты могут управлять движением объектов, обработкой взаимодействий, анимациями и многим другим;

- префабы (Prefabs). Префабы – это шаблоны объектов, которые могут быть многократно использованы в игре. Они позволяют быстро создавать и модифицировать объекты без необходимости повторного создания каждый раз;

- элементы UI (User Interface). UI элементы используются для создания пользовательского интерфейса игры. Они могут включать в себя кнопки, текстовые поля, панели и другие элементы, позволяющие взаимодействовать с игрой.

1.6.2 Игровые механики

Игровые механики – это фундаментальные элементы и правила, которые определяют, как игра функционирует и как игрок взаимодействует с игровым миром. Они включают в себя все аспекты игрового процесса, от базовых действий, которые может выполнять игрок, до сложных систем, которые определяют, как игра реагирует на действия игрока. Ниже представлены механики, реализованные в данном игровом приложении.

1.6.2.1 Механика вращения комнаты

Для реализации полноценной механики поворота комнаты, были созданы два основных скрипта, их описание представлено ниже.

RoomManager – менеджер для управления комнатами в игре, этот скрипт содержит поверхностную реализацию методов для управления комнатами.

Также в нем содержится поле текущей комнаты, с которой происходит взаимодействие. Обращаясь к его методам, игрок может осуществлять повороты комнаты и переход в другую комнату. Ниже представлено описание основных методов RoomManager. Полный листинг программы описан в приложении А.

Метод Start вызывается при запуске объекта, инициализирует начальные настройки и получает из редактора стартовую комнату. Метод скрывает все комнаты, кроме стартовой.

Метод Update вызывается на каждом кадре и отслеживает пользовательский ввод для начала и завершения процесса поворота комнаты.

Метод TryRotate вызывается, чтобы попытаться повернуть текущую комнату в указанном направлении. Если это возможно.

RoomController – этот скрипт описывает внутреннюю функциональность комнат, обеспечивая их плавный поворот и синхронизацию отображения объектов и стен в соответствии с текущим состоянием комнаты. Ниже представлено описание метода RoomController обеспечивающего вращение комнаты.

Метод SmoothRotate запускает процесс плавного поворота комнаты в зависимости от параметра clockwise комната поворачивается по часовой стрелке или против. В процессе поворота комната масштабируется, чтобы создать впечатление плавности. Данный метод использует корутину SmoothRotateCoroutine для поэтапного изменения ротации и масштаба комнаты[3]. Также при повороте обновляется состояние поворота комнаты, описанное в перечислении RoomRotation, и обновляется отображение стен относительно этого состояния. Описание классов, при помощи которых реализована данная механика, отображено на рисунке 1.17. Демонстрация работы механики показана на рисунках 1.15, 1.16.

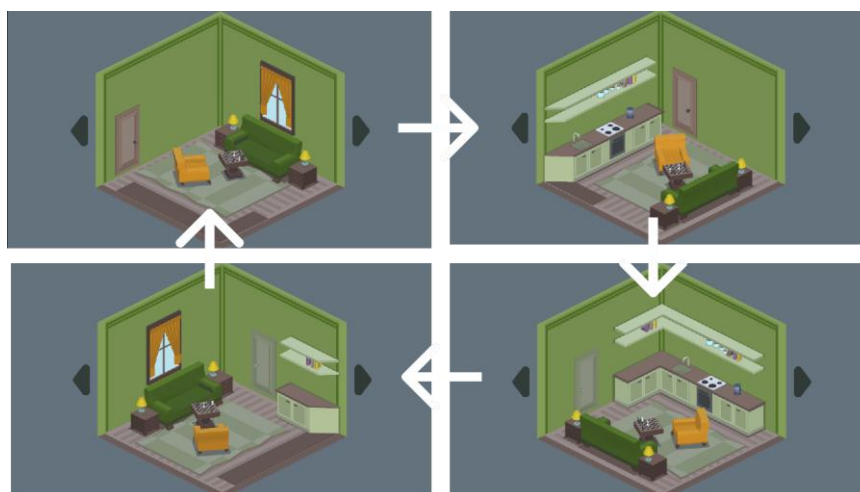


Рисунок 1.15 – Механика поворота Гостиной

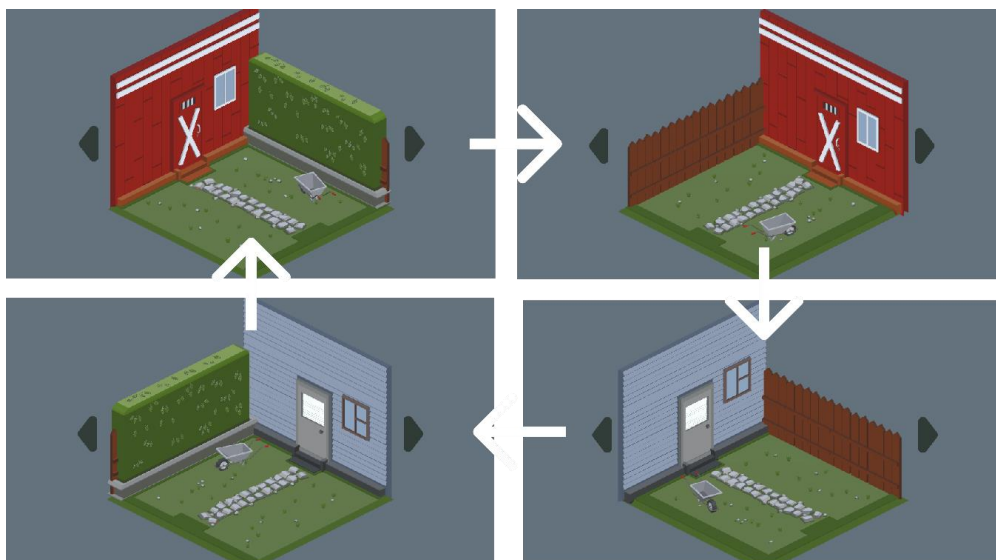


Рисунок 1.16– Механика поворота Улицы

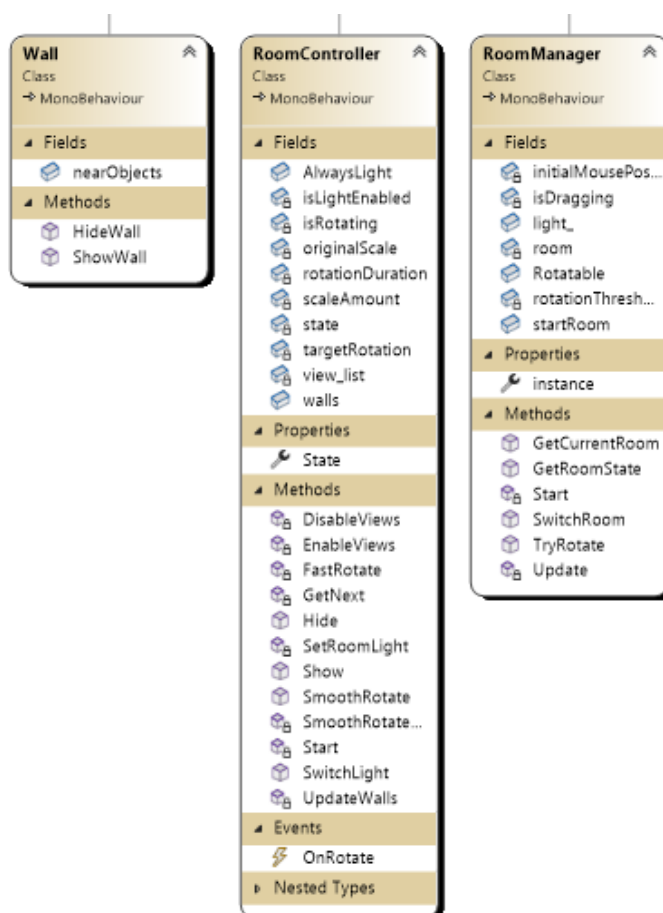


Рисунок 1.15 – Описание классов, реализующих механики управления комнатами

1.6.2.2 Механика смены комнат

Для реализации механики смены комнат были использованы вышеописанные скрипты RoomController и RoomManager.

Для них были написаны следующие дополнения:

- в RoomController добавлен метод Show, отображающий комнату, применяя начальное состояние поворота и активируя игровой объект. А также метод Hide скрывающий комнату и возвращая ее текущее состояние;

- в RoomManager был добавлен метод SwitchRoom, обеспечивающий полноценную смену комнаты, то есть вызывая последовательно методы Hide для текущей комнаты, получая ее состояние поворота, и метод Show, для новой комнаты.

Также для функционирования данной механики создан новый скрипт Door описывающий дверь между комнатами. Он привязывается к объекту двери в редакторе и содержит поле, описывающее комнату, в которую будет совершен переход. При нажатии игроком на объект двери вызывается метод SwitchRoom с параметром новой комнаты. Демонстрация работы механики смены комнат с сохранением состояния поворота показана на рисунках 1.18, 1.19. Описание классов, используемых для механики смены комнат показаны на рисунках 1.15, 1.20.

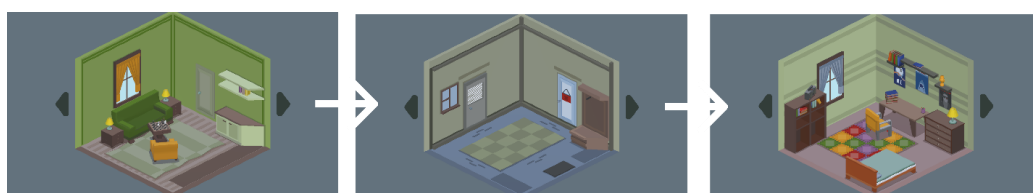


Рисунок 1.18 – Маршрут Гостиная-Холл-Детская

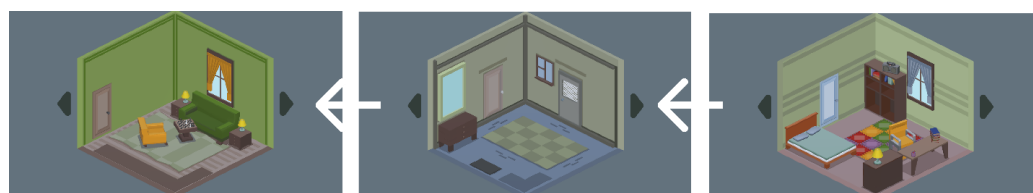


Рисунок 1.19 – Маршрут Детская-Холл-Гостиная

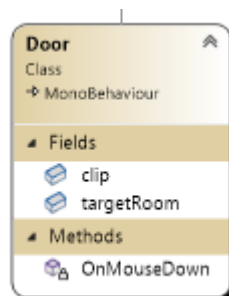


Рисунок 1.20 – Описание класса «Door»

1.6.2.3 Механика приближение к объектам

Приближение к объектам или группам объектов нужно для обеспечения игроку возможности взаимодействия с мелкими или далеко-расположенными объектами.

Эта механика реализована при помощи скрипта – менеджера ViewManager, скрипта для объекта – View, и вспомогательного скрипта – Utilities.

Менеджер ViewManager, содержит методы Zoom и UnZoom для плавного приближения камеры к объекту и возврата камеры в свою изначальную позицию. Движение камеры обеспечивается при помощи корутин, для плавной анимации движения. Значения позиции и поворота изначальной и конечной позиции камеры хранятся при помощи Vector3 полей содержащих float значения координат XYZ.

Для того чтобы обеспечить объект возможностью приближения камеры, в редакторе к этому объекту был добавлен стандартный компонент Collider, который отслеживает нажатие мыши по объекту, и позволяет вызвать метод OnMouseDown. Реализация этого метода содержится в классе View. И он обеспечивает своевременный вызов метода Zoom к конкретному объекту. Также для корректной работы этого метода нужно получить итоговые значения ротации и позиции камеры, они в свою очередь содержатся в скрипте Utils, и для каждого объекта или группы являются идентичными. Демонстрация работы механики

показана на рисунках 1.21, 1.22. Описание классов для работы данной механики, продемонстрировано на рисунке 1.23.

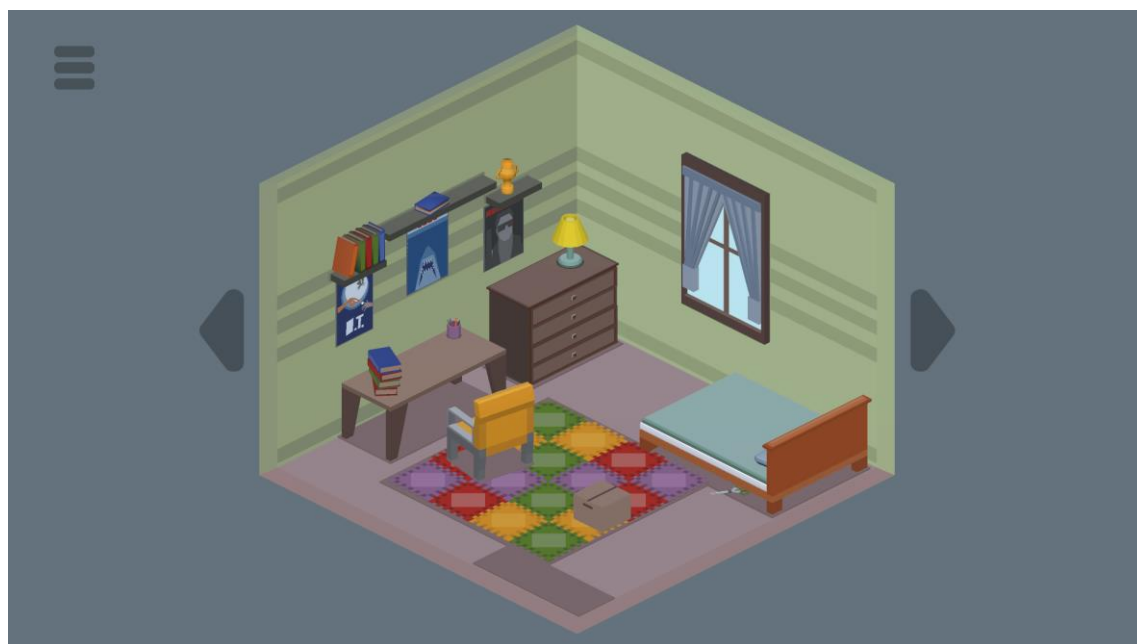


Рисунок 1.21 – Состояние игры до приближения к объекту



Рисунок 1.22 – Состояние игры после приближение к объекту

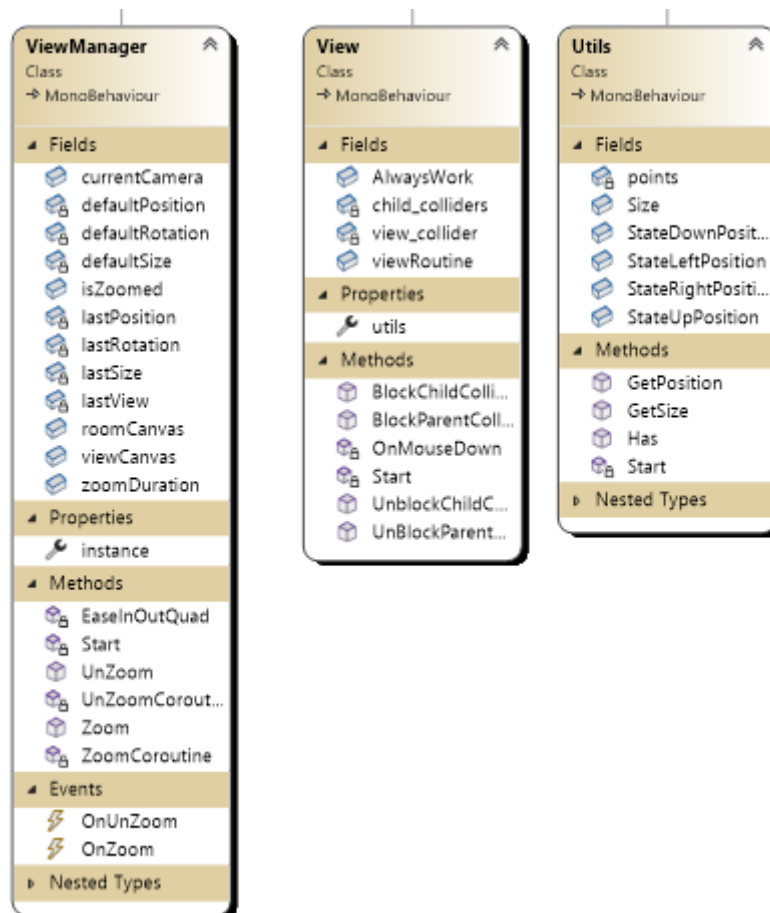


Рисунок 1.23 – Классы, реализующие механику приближения к объектам

1.6.2.4 Механика вывода сообщений

Вывод сообщений на экран создан, чтобы обеспечить пользователя дополнительной, полезной для него информацией, это могут быть подсказки к головоломке, или комментарий на определенное действие пользователя. В данном приложении эта механика реализована при помощи скрипта MessageController, который хранит в себе объект, содержащий шаблон сообщения без конкретного текста[4]. Для корректной работы механики также понадобится объект с обработчиком нажатия мыши OnMouseDown, такой же как описан выше, но с изменением реализации нажатия. В данном объекте будет содержаться текст сообщения и позиция этого объекта, для того чтобы при выводе сообщения, оно появлялось рядом с объектом, для которого оно

вызывается. При нажатии по объекту будет происходить вызов метода ShowToolTip из MessageController, который обеспечит вывод соответствующего сообщение, и его скрытие через 5 секунд. Время показывания сообщения можно изменить. Демонстрация работы механики показана на рисунках 1.24, 1.25. Класс, используемый для реализации данной механики, отображен на рисунке 1.26.



Рисунок 1.24 – Вывод сообщения при стандартном состоянии камеры

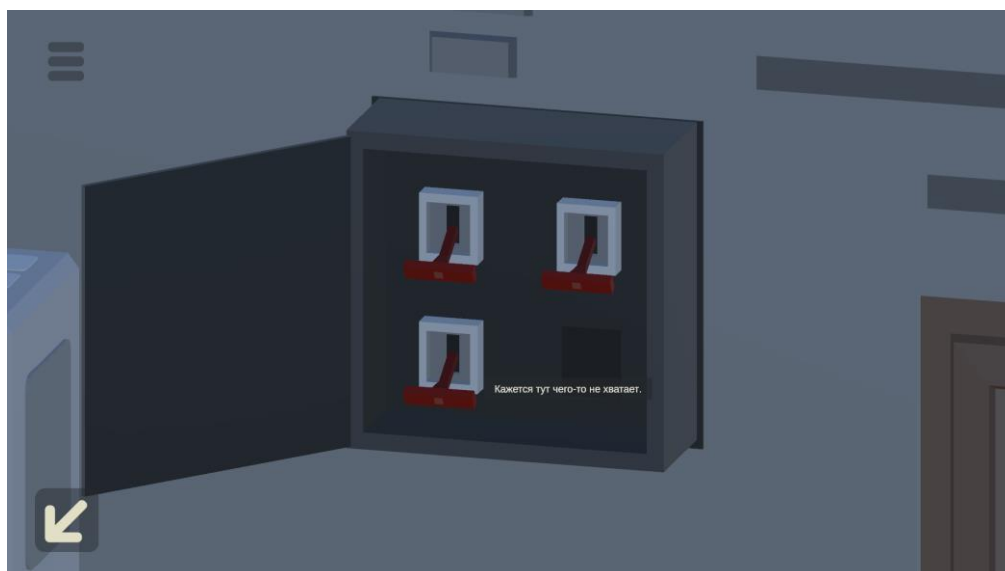


Рисунок 1.25 – Вывод сообщения при приближении к объекту

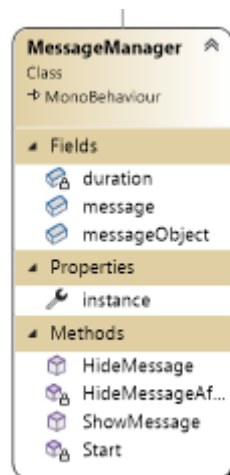


Рисунок 1.26 – Класс, для механики вывода сообщений

1.6.2.5 Механика инвентаря

Механика инвентаря предоставляет удобный способ управления предметами в игре. Пользователь может легко собирать предметы с игровой сцены, которые автоматически добавляются в инвентарь. Инвентарь организован так, чтобы отображать все собранные предметы в компактной и интуитивно понятной форме. Основная сложность этой системы заключается в координации взаимодействия между игровыми объектами и UI, что достигается за счет использования скриптов и UI компонентов в Unity. В конкретном приложении механика инвентаря реализована при помощи скрипта InventoryManager, префаба InventorySlot, который хранит одну ячейку инвентаря, и контейнера InventoryPanel для ячеек инвентаря, где каждая ячейка отображает конкретный предмет. Так же объект, который может содержаться в инвентаре должен содержать созданный компонент InventoryItem, с реализованным методом OnMouseDown. При клике на такой объект будет вызван метод AddItemToInventory из InventoryManager, который, при наличии свободных мест в инвентаре, удалит объект со сцены, и отобразит его в новой ячейке, для последующего взаимодействия[5]. Пример использования механики показан на рисунках 1.27, 1.28. Демонстрация описания классов данной механики отображена на рисунке 1.29.

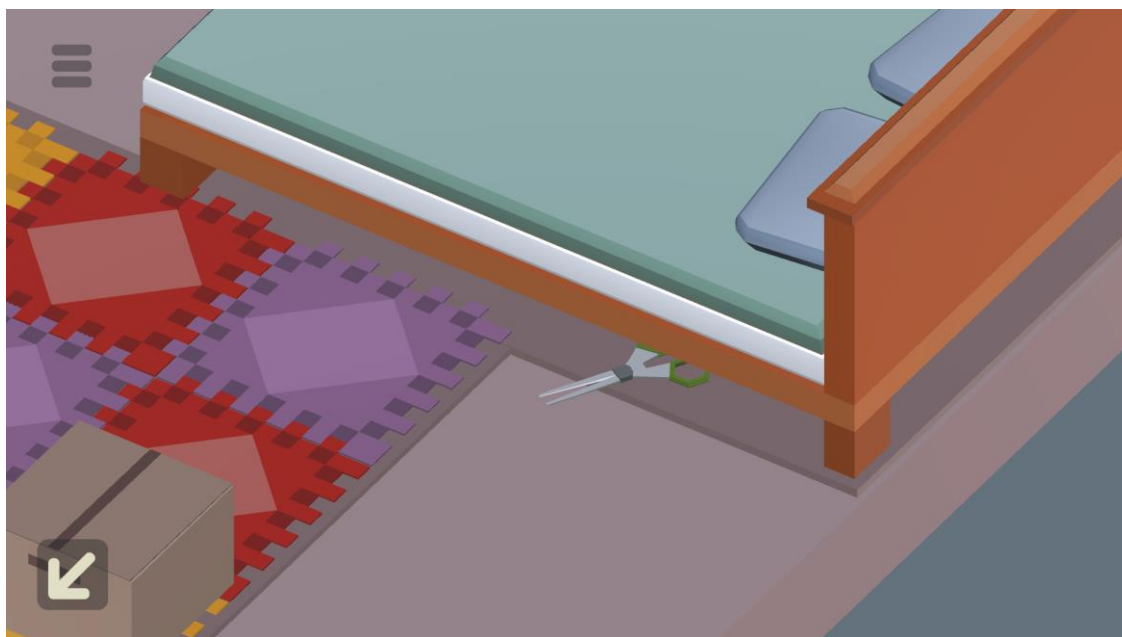


Рисунок 1.27 – Предмет, до подбора в инвентарь

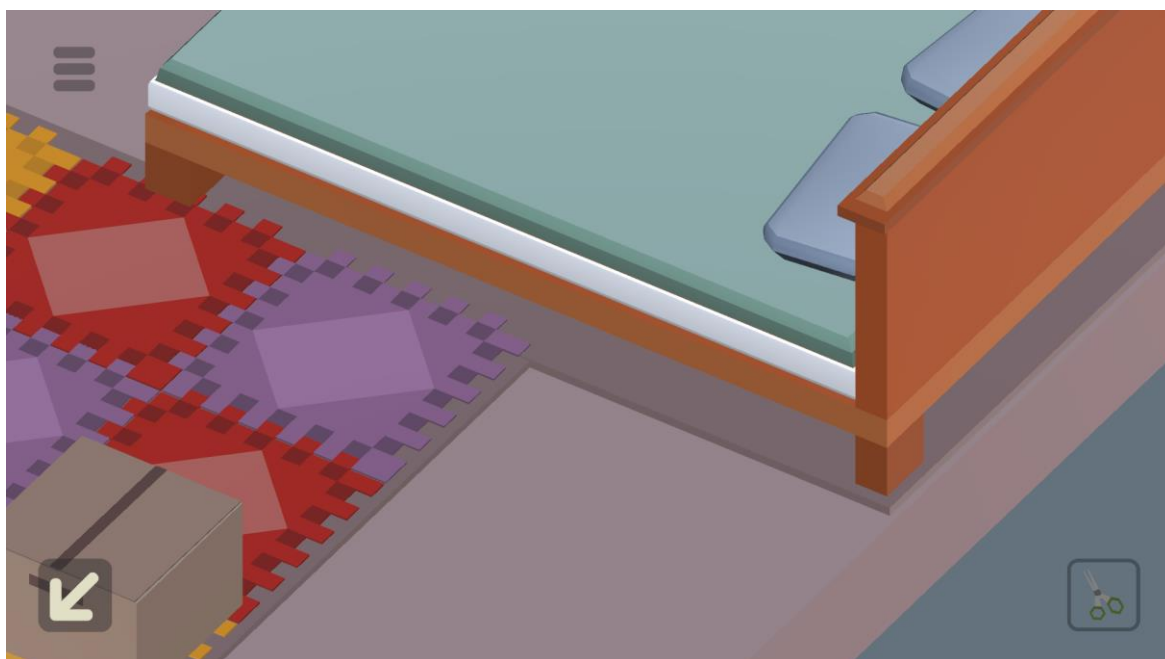


Рисунок 1.28 – Предмет в ячейке инвентаря после подбора

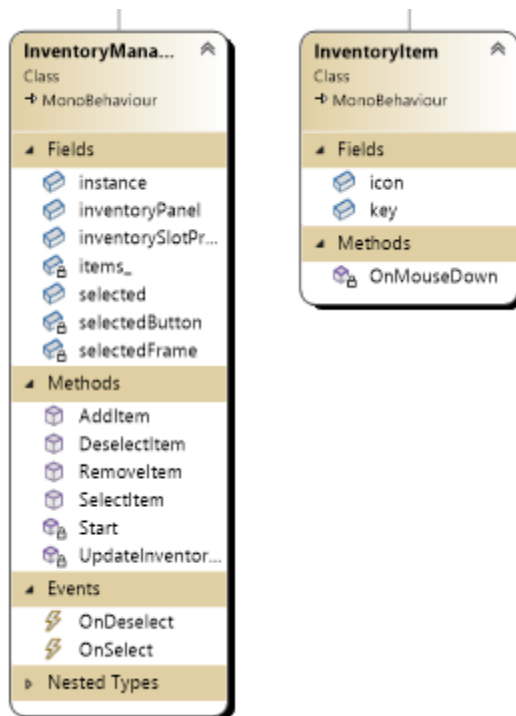


Рисунок 1.29 – Классы механики инвентаря

1.6.3 Паттерны проектирования

За счет своей наполненности различными объектами и их частым взаимодействиям, при разработке игровых приложений часто используются паттерны проектирования, так как, при правильном применении, они могут обеспечить большую гибкость кода под конкретные задачи, и предоставляют проверенные решения для распространенных проблем проектирования[6].

1.6.3.1 Одиночка

Паттерн «Одиночка» (Singleton) гарантирует, что класс имеет только один экземпляр и предоставляет глобальную точку доступа к этому экземпляру. В контексте данного приложения, «Одиночка» используется для создания объектов – менеджеров, описанных выше. Это обеспечивает возможность обращаться к этим объектам из любой точки программы, без необходимости создавать экземпляр. А также позволяет такому менеджеру эффективно контролировать глобальные состояния для различных объектов, как например это делает RoomManager с состоянием поворота комнаты.

1.6.3.2 Команда

Паттерн «Команда» (Command) – это поведенческий шаблон проектирования, который инкапсулирует запрос как объект. Основная идея паттерна заключается в том, чтобы отделить отправителя запроса от его получателя. Это достигается созданием классов – команд, которые реализуют общий интерфейс с методами для выполнения и отмены действия. Каждая команда содержит всю необходимую информацию для выполнения конкретного действия: ссылку на объект – исполнитель и параметры действия. Использование паттерна Команда упрощает добавление новых операций, делает код более гибким и позволяет легко интегрировать новые команды без изменения существующего кода. В контексте данного приложения паттерн команда используется для реализации реакции различных объектов на нажатие мыши. И для изменение этой реакции в любой нужный для игры момент. Интерфейс и классы, реализованные с использованием паттерна команда, отображены на рисунке 1.30.

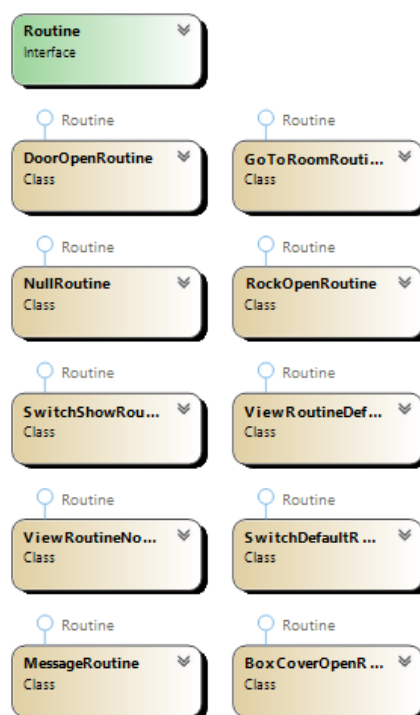


Рисунок 1.30 – Паттерн «Команда»

1.6.4 Пользовательский интерфейс

Создание пользовательского интерфейса в Unity и обеспечение его функциональности является довольно простой задачей, используя современные средства Unity.

Для создания любого интерфейса в Unity, на сцену должен быть добавлен специальный объект Canvas обеспечивающий корректное отображение и отрисовку элементов.

В данном игровом приложении присутствует пять видов пользовательского интерфейса:

- интерфейс инвентаря;
- кнопки для управления положением комнаты;
- кнопка для отдаления камеры;
- интерфейс всплывающих сообщений;
- меню настроек игры и выхода из приложения.

Для реализации интерфейса, при помощи которого происходит управление комнатой и приближением, были созданы два объекта типа Canvas – RoomCanvas и ViewCanvas, каждый из которых содержит необходимые для его реализации кнопки. Иерархия объектов интерфейса представлена на рисунке 1.31.



Рисунок 1.31 – Пользовательский интерфейс кнопок

Каждая кнопка по умолчанию обладает методом OnClick, который вызывается при нажатии на объект интерфейса. Этот метод может принимать как параметр любую публичную функцию, и вызывать ее. В данном случае Left_Rotate_Btn и Right_Rotate_Btn вызывают методы RoomManager – TryRotate,

для обеспечения ротации комнаты, а UnZoom_Btn вызывает метод UnZoom у скрипта ViewManager, для возврата камеры в своё изначальное положение после приближения. Стоит также отметить, что логика отображения данных объектов интерфейса реализована в ViewContoller, так как кнопки должны отображаться и скрываться, в момент вызова методов приближения или отдаления у этого контроллера. Демонстрация пользовательского интерфейса показана на рисунках 1.32 - 1.36.



Рисунок 1.32 – Кнопка поворота комнаты

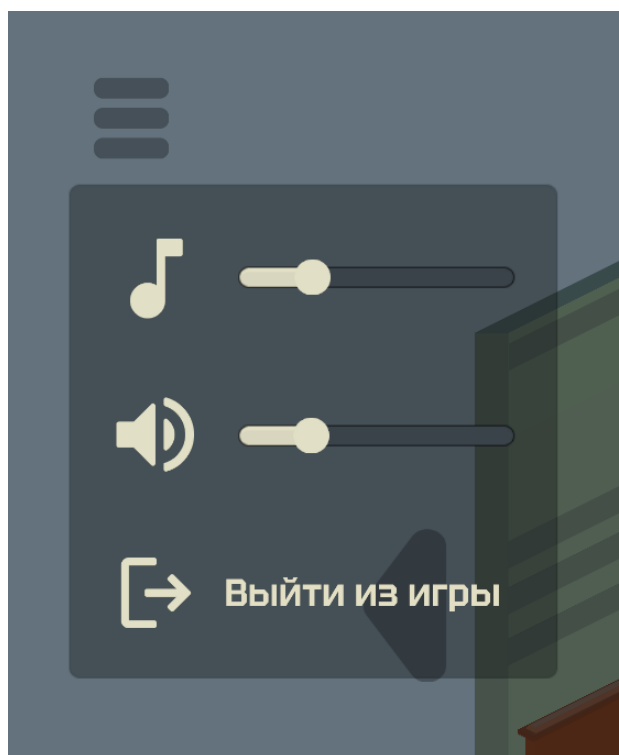


Рисунок 1.33 – Панель меню

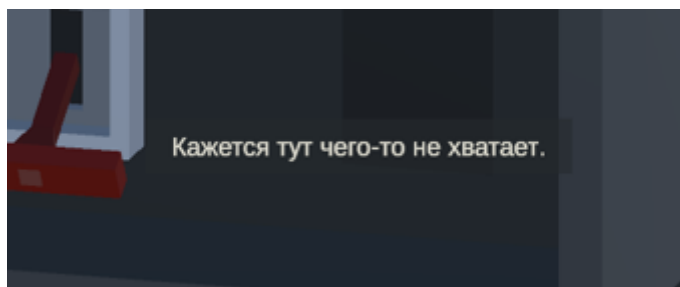


Рисунок 1.34 – Вывод сообщения



Рисунок 1.35 – Элемент панели инвентаря



Рисунок 1.36 – Кнопка отдаления камеры от объекта

2 Охрана труда

2.1 Организация рабочего места при работе на персональном компьютере

В современном мире информационных технологий вопросы безопасности жизнедеятельности становятся все более актуальными. При разработке программного обеспечения, как и в любой другой области деятельности, необходимо учитывать требования охраны труда и обеспечения безопасности. В данном разделе рассматриваются аспекты безопасности жизнедеятельности, связанные с профессиональной деятельностью программиста.

Согласно ГОСТ 12.2.032-78 "ССБТ. Рабочее место при выполнении работ сидя. Общие эргономические требования", рабочее место программиста должно быть организовано таким образом, чтобы минимизировать вредное воздействие на здоровье.

2.1.1 Освещение

Рабочее место должно быть оборудовано достаточным освещением. Согласно СНиП 23-05-95 "Естественное и искусственное освещение", уровень освещенности на рабочем месте программиста должен составлять не менее 300-500 люкс. Рекомендуется использовать комбинацию естественного и искусственного освещения, чтобы избежать чрезмерной нагрузки на зрение.

Необходимо учитывать:

- расположение источников света, чтобы избежать бликов на экране монитора.
- использование регулируемых ламп для индивидуальной настройки освещения.

2.1.2 Общие эргономические требования к рабочему месту оператора

Рабочее место с экраном должно обеспечивать комфортные условия для работы в положении сидя, не вызывая перегрузок костно-мышечной системы оператора.

Основные элементы рабочего места оператора включают рабочий стол, рабочий стул (кресло), дисплей и клавиатуру; дополнительно могут использоваться пюпитр и подставка для ног. Общая организация рабочего места показана на рисунке 2.1.

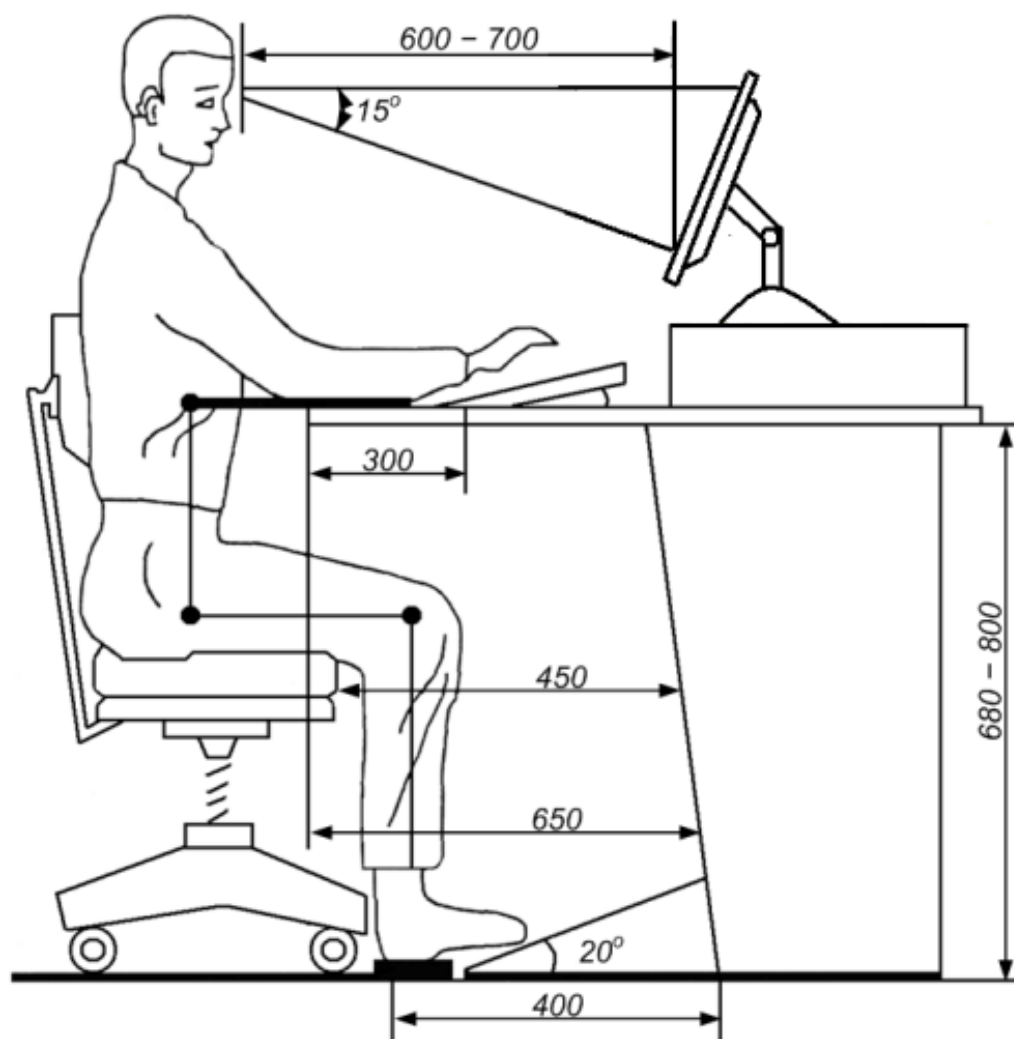


Рисунок 2.1 – Общая организация рабочего места

Изм.	Лист	№ докум.	Подпись	Дата

Рабочий стол должен быть спроектирован таким образом, чтобы на его поверхности можно было разместить необходимое оборудование и документы, учитывая характер работы оператора.

Высота рабочей поверхности стола может быть регулируемой (680 -- 800 мм) или фиксированной (725 мм), в зависимости от конструкции. Размеры стола должны обеспечивать глубину не менее 600 мм и ширину не менее 1200 мм.

Рабочая поверхность стола должна быть без острых углов, покрыта материалом с коэффициентом отражения 0,45-0,50.

Рабочее кресло должно поддерживать физиологически правильное положение тела оператора, позволяя менять позу для снижения напряжения мышц спины и шеи.

Кресло должно быть регулируемым по высоте и углам наклона сиденья и спинки, а также по расстоянию спинки от переднего края сиденья. Подлокотники должны быть регулируемыми по высоте и ширине.

Подставка для ног должна регулироваться по высоте и углу наклона, обеспечивая комфортную поддержку ног оператора.

Поверхность подставки должна быть рифленой с бортиком для предотвращения скольжения.

Дисплей должен быть установлен на рабочем месте таким образом, чтобы оператор мог четко видеть изображение без лишнего напряжения глаз.

Угол наклона экрана относительно горизонтальной линии взгляда оператора не должен превышать 60°, показано на рисунке 2.1.

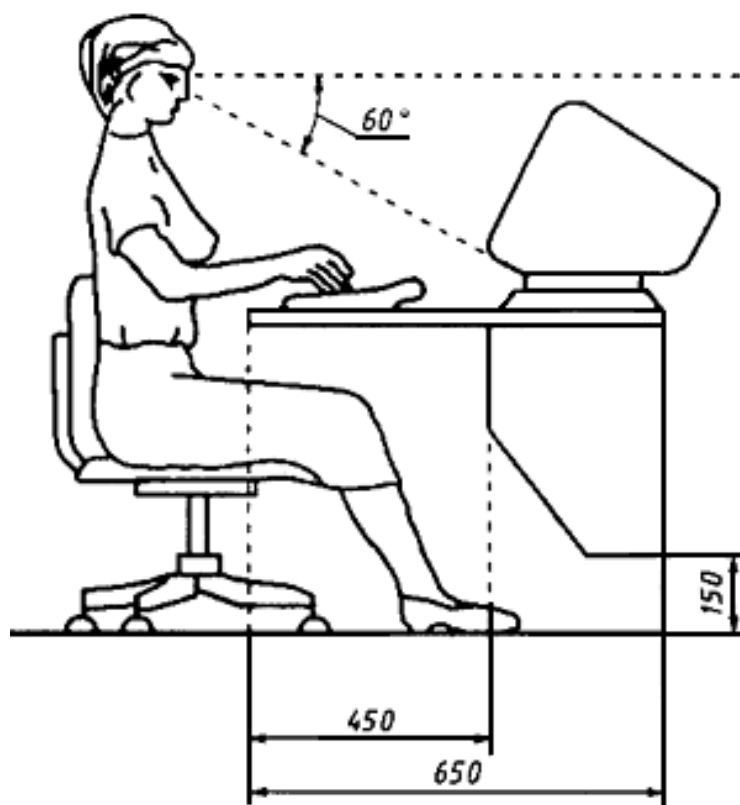


Рисунок 2.2 – Расположение дисплея на столе

Клавиатура должна быть расположена на достаточном расстоянии от оператора, чтобы обеспечивать оптимальную видимость экрана.

Клавиатура должна иметь возможность свободного перемещения или быть установлена на регулируемой по высоте поверхности.

2.1.3 Климатические условия

В помещении, где работает программист, должны поддерживаться оптимальные климатические условия. Согласно ГОСТ 30494-2011 "Здания жилые и общественные. Параметры микроклимата в помещениях", температура воздуха должна составлять 18-22°C, относительная влажность – 40-60%. Также необходимо обеспечить:

- регулярное проветривание помещения.
- использование систем кондиционирования и увлажнения воздуха.

2.2 Психофизиологические аспекты труда программиста

Работа программиста связана с высокой умственной нагрузкой и длительным пребыванием в одной позе, что может приводить к утомлению и снижению работоспособности.

В соответствии с СанПиН 2.2.2/2.4.1340-03 "Гигиенические требования к персональным электронно-вычислительным машинам и организации работы", необходимо соблюдать режим труда и отдыха. Рекомендуется:

- делать перерывы через каждые 1,5-2 часа работы по 10-15 минут.
- выполнять физические упражнения для снятия напряжения с мышц и улучшения кровообращения.

2.3 Электробезопасность

При работе с компьютером программист подвергается риску поражения электрическим током. Для обеспечения электробезопасности необходимо соблюдать требования ГОСТ 12.1.019-79 "Электробезопасность. Общие требования и номенклатура видов защиты".

2.3.1 Использование исправного оборудования

Все электронные устройства должны быть исправны и иметь сертификацию соответствия стандартам безопасности. Важно регулярно проверять оборудование на предмет повреждений и своевременно устранять выявленные дефекты.

2.3.2 Заземление и защитные устройства

Рабочее место должно быть оснащено розетками с заземлением и устройствами защитного отключения (УЗО), которые предотвращают поражение электрическим током в случае короткого замыкания или утечки тока. Необходимо также соблюдать правила использования удлинителей и сетевых фильтров.

2.4 Пожарная безопасность

Для обеспечения пожарной безопасности в помещении, где работают программисты, должны быть соблюдены требования ГОСТ 12.1.004-91 "Пожарная безопасность. Общие требования".

В помещении должны быть установлены средства пожаротушения (огнетушители), система оповещения о пожаре и пути эвакуации, обозначенные согласно ГОСТ Р 12.4.026-2015 "ССБТ. Цвета сигнальные и знаки безопасности". Важно регулярно проверять исправность средств пожаротушения и наличие свободных путей эвакуации.

Все сотрудники должны проходить регулярные инструктажи по пожарной безопасности, включая действия в случае возникновения пожара и использование средств пожаротушения. Важно также проводить тренировки по эвакуации для отработки действий в чрезвычайных ситуациях.

					<i>ПГУ1.090301.19.001 ПЗ</i>	Лист
						45
Изм.	Лист	№ докум.	Подпись	Дата		

Заключение

Разработка игрового приложения в жанре изометрической головоломки на платформе Unity представила собой увлекательный и многогранный процесс, включающий все этапы от концепции до реализации. Мой проект продемонстрировал, как современные инструменты и технологии позволяют создавать сложные и захватывающие игровые механики, обеспечивая богатый опыт для игроков.

Основное внимание в проекте было уделено функциональной части игры, включающей разработку игровых механик, алгоритмов и систем управления. Каждая из этих составляющих требовала детального планирования и реализации. Начальный этап включал в себя разработку концепции игры и создание подробного технического задания, что стало основой для последующей работы над функционалом.

Работа над проектом включала создание и оптимизацию основных игровых механик. Например, была разработана система управления комнатами, алгоритмы поведения игровых объектов и логика решения головоломок. Использование языка программирования C# в среде Unity позволило эффективно реализовать и интегрировать все необходимые функции. Особое внимание было уделено разработке кода, который обеспечивал бы надежную и стабильную работу игры, а также поддерживал бы гибкость для дальнейшего расширения и модификации.

Подводя итог, можно сказать, что проделанная работа открывает перспективы для дальнейшего развития и улучшения проекта. В будущем можно добавить новые уровни и игровые механики, усовершенствовать существующие алгоритмы и расширить функционал игры. Моя работа подчеркивает, что современные технологии и инструменты предоставляют большие возможности для создания инновационных и увлекательных игровых решений, которые могут удовлетворить потребности широкой аудитории.

Список использованных источников

1. Хокинг Д. С. Unity в действии. Мультиплатформенная разработка на C# / Д. С. Хокинг. – Питер : Питер, 2019. – 352 с.
2. docs.unity.com: [Электронный ресурс]. URL: <https://docs.unity3d.com/ScriptReference/MonoBehaviour.html> (дата обращения: 7.05.2023)
3. <https://learn.microsoft.com>: [Электронный ресурс] URL: <https://learn.microsoft.com/ru-ru/dotnet/csharp/> (дата обращения: 10.04.2023)
4. Дж. Рамбо, М. Блаха UML 2.0. Объектно-ориентированное моделирование и разработка, 2-е издание. – СПб.: Питер, 2007 — 544 с.
5. Язык программирования C# 7 и платформы .NET и .NET Core, 8-е изд. : Пер. с англ. — СПб. : ООО “Диалектика”, 2018 — 1328 с.
6. Найстром, Роберт. Паттерны программирования игр / Роберт. Найстром. – Москва : Бомбора, 2022. – 432 с. – ISBN 978-5-04-102290-7.

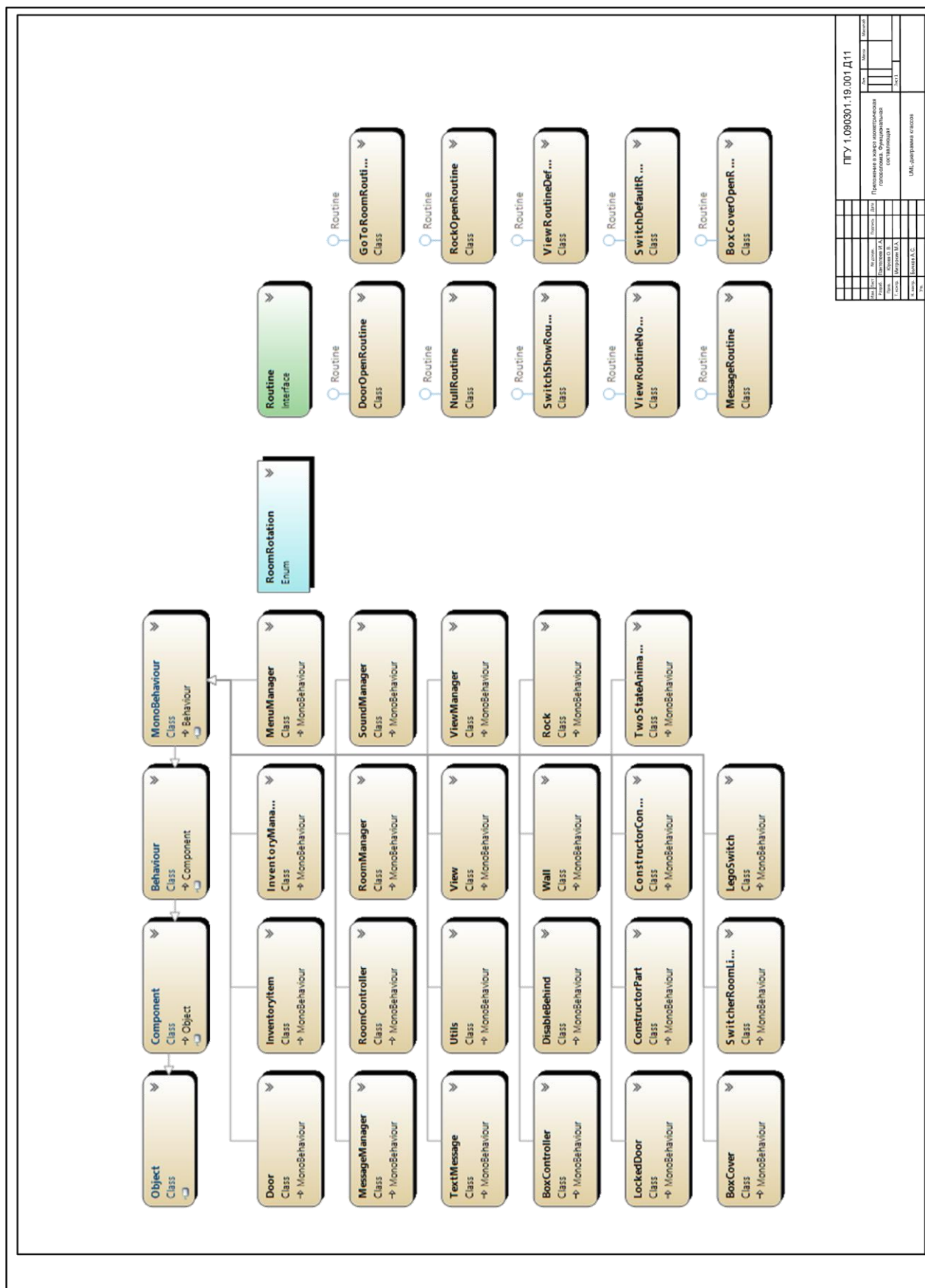
Приложение А

(обязательное)

Приложение в жанре изометрическая головоломка. Функциональная
составляющая
UML – диаграммы

					<i>ПГУ1.090301.19.001 ПЗ</i>	Лист
						48
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение А. 1 –UML - диаграмма классов

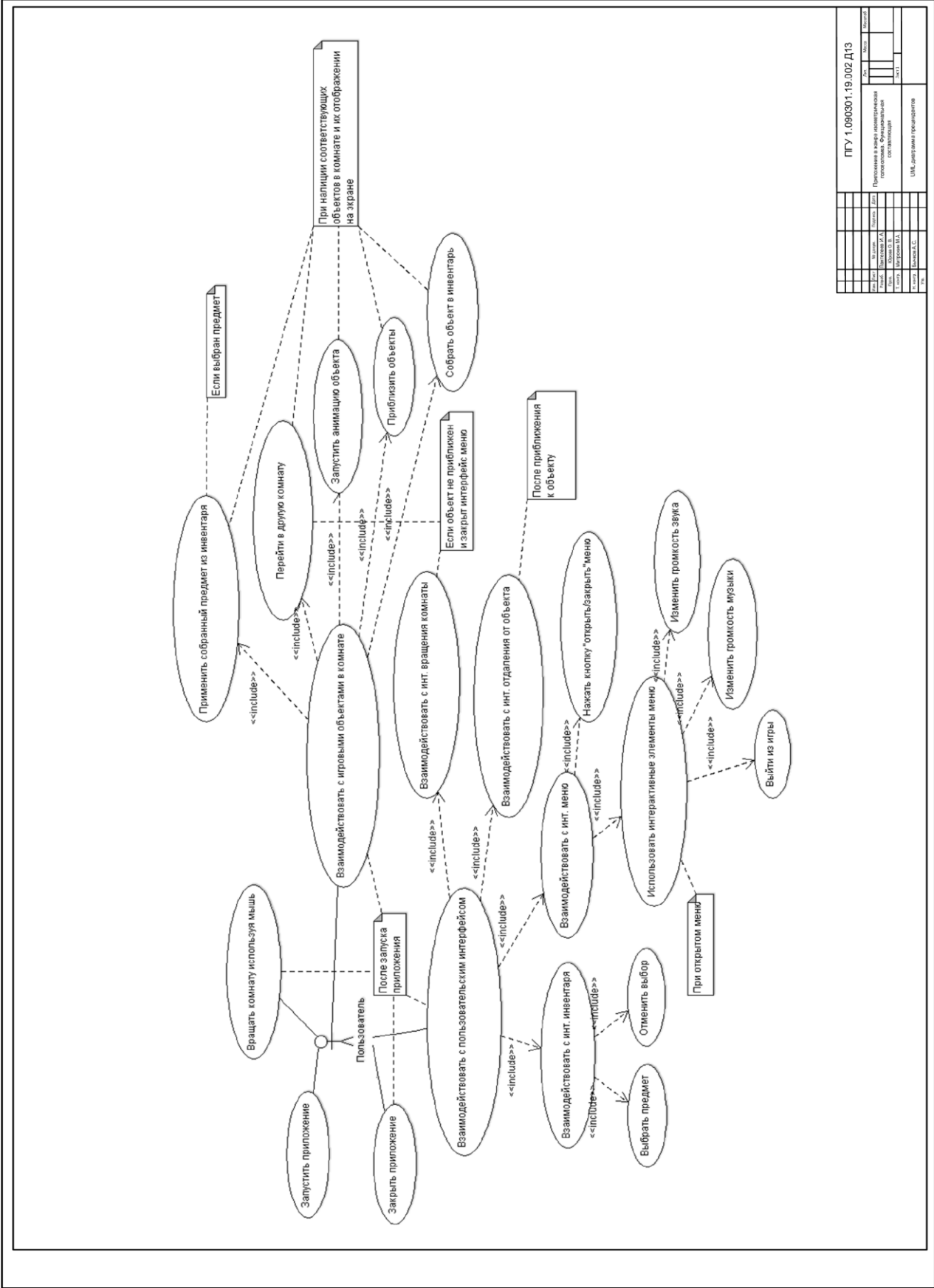


ПУУ 1.090301.19.001 Д11									
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия

Изм.	Лист	№ докум.	Подпись	Дата

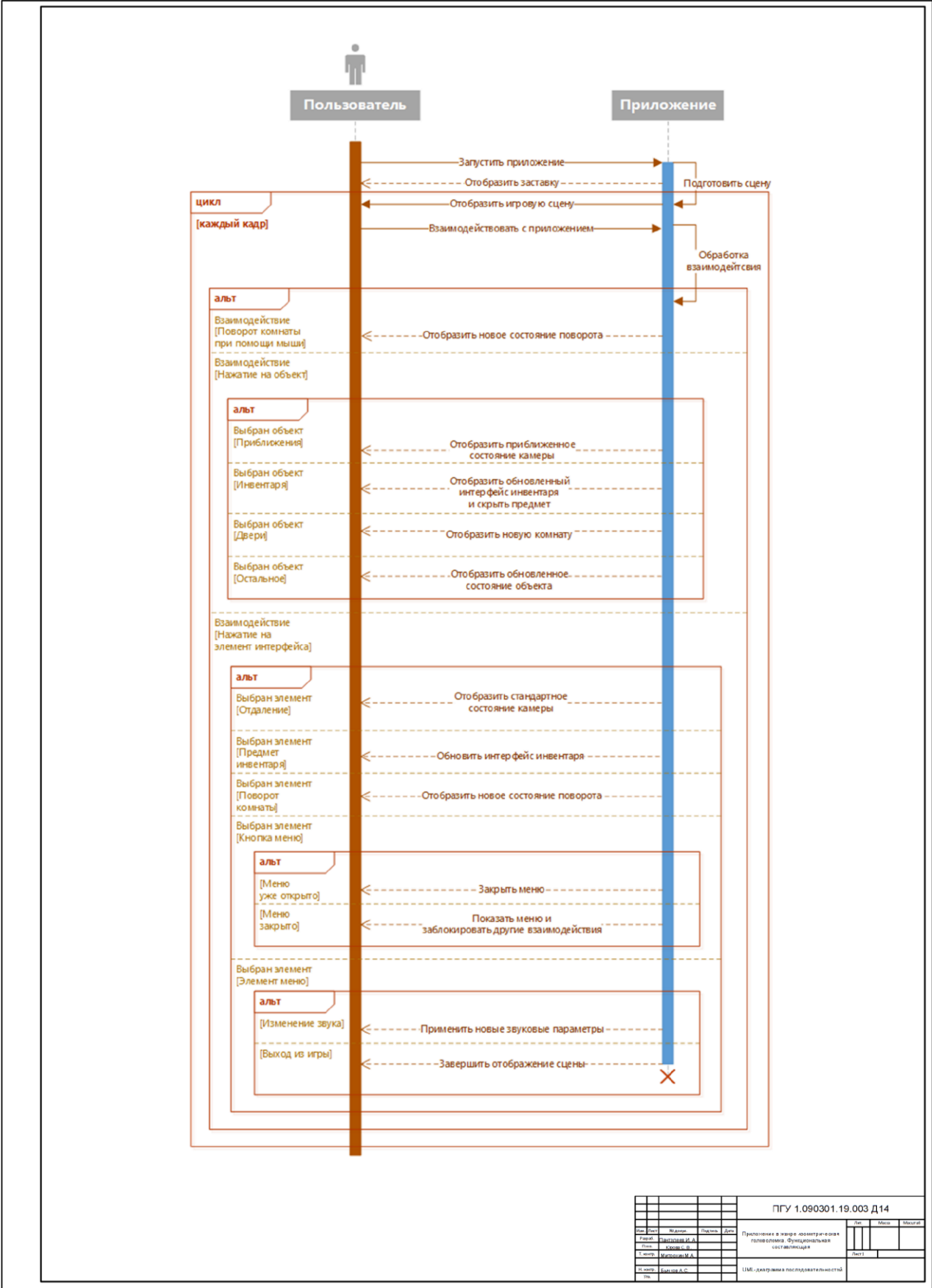
ПУУ1.090301.19.001 ПЗ

Приложение А. 2 –UML - диаграмма прецедентов



ПГУ 1.090301.19.002 Д13									
№ п/п	№ докум.	Имя	Долг.	Дата	Стр.	Всего стр.	Присвоение в сфере образовательной деятельности. Функциональная спецификация		
1	1	Васильев А.А.	Студент	2019.09.01	1	1			
2	2	Васильев А.А.	Студент	2019.09.01	2	2			
3	3	Васильев А.А.	Студент	2019.09.01	3	3			
4	4	Васильев А.А.	Студент	2019.09.01	4	4			
5	5	Васильев А.А.	Студент	2019.09.01	5	5			
6	6	Васильев А.А.	Студент	2019.09.01	6	6			
7	7	Васильев А.А.	Студент	2019.09.01	7	7			
8	8	Васильев А.А.	Студент	2019.09.01	8	8			
9	9	Васильев А.А.	Студент	2019.09.01	9	9			
10	10	Васильев А.А.	Студент	2019.09.01	10	10			
11	11	Васильев А.А.	Студент	2019.09.01	11	11			
12	12	Васильев А.А.	Студент	2019.09.01	12	12			
13	13	Васильев А.А.	Студент	2019.09.01	13	13			
14	14	Васильев А.А.	Студент	2019.09.01	14	14			
15	15	Васильев А.А.	Студент	2019.09.01	15	15			
16	16	Васильев А.А.	Студент	2019.09.01	16	16			
17	17	Васильев А.А.	Студент	2019.09.01	17	17			
18	18	Васильев А.А.	Студент	2019.09.01	18	18			
19	19	Васильев А.А.	Студент	2019.09.01	19	19			
20	20	Васильев А.А.	Студент	2019.09.01	20	20			
21	21	Васильев А.А.	Студент	2019.09.01	21	21			
22	22	Васильев А.А.	Студент	2019.09.01	22	22			
23	23	Васильев А.А.	Студент	2019.09.01	23	23			
24	24	Васильев А.А.	Студент	2019.09.01	24	24			
25	25	Васильев А.А.	Студент	2019.09.01	25	25			
26	26	Васильев А.А.	Студент	2019.09.01	26	26			
27	27	Васильев А.А.	Студент	2019.09.01	27	27			
28	28	Васильев А.А.	Студент	2019.09.01	28	28			
29	29	Васильев А.А.	Студент	2019.09.01	29	29			
30	30	Васильев А.А.	Студент	2019.09.01	30	30			
31	31	Васильев А.А.	Студент	2019.09.01	31	31			
32	32	Васильев А.А.	Студент	2019.09.01	32	32			
33	33	Васильев А.А.	Студент	2019.09.01	33	33			
34	34	Васильев А.А.	Студент	2019.09.01	34	34			
35	35	Васильев А.А.	Студент	2019.09.01	35	35			
36	36	Васильев А.А.	Студент	2019.09.01	36	36			
37	37	Васильев А.А.	Студент	2019.09.01	37	37			
38	38	Васильев А.А.	Студент	2019.09.01	38	38			
39	39	Васильев А.А.	Студент	2019.09.01	39	39			
40	40	Васильев А.А.	Студент	2019.09.01	40	40			
41	41	Васильев А.А.	Студент	2019.09.01	41	41			
42	42	Васильев А.А.	Студент	2019.09.01	42	42			
43	43	Васильев А.А.	Студент	2019.09.01	43	43			
44	44	Васильев А.А.	Студент	2019.09.01	44	44			
45	45	Васильев А.А.	Студент	2019.09.01	45	45			
46	46	Васильев А.А.	Студент	2019.09.01	46	46			
47	47	Васильев А.А.	Студент	2019.09.01	47	47			
48	48	Васильев А.А.	Студент	2019.09.01	48	48			
49	49	Васильев А.А.	Студент	2019.09.01	49	49			
50	50	Васильев А.А.	Студент	2019.09.01	50	50			
51	51	Васильев А.А.	Студент	2019.09.01	51	51			
52	52	Васильев А.А.	Студент	2019.09.01	52	52			
53	53	Васильев А.А.	Студент	2019.09.01	53	53			
54	54	Васильев А.А.	Студент	2019.09.01	54	54			
55	55	Васильев А.А.	Студент	2019.09.01	55	55			
56	56	Васильев А.А.	Студент	2019.09.01	56	56			
57	57	Васильев А.А.	Студент	2019.09.01	57	57			
58	58	Васильев А.А.	Студент	2019.09.01	58	58			
59	59	Васильев А.А.	Студент	2019.09.01	59	59			
60	60	Васильев А.А.	Студент	2019.09.01	60	60			
61	61	Васильев А.А.	Студент	2019.09.01	61	61			
62	62	Васильев А.А.	Студент	2019.09.01	62	62			
63	63	Васильев А.А.	Студент	2019.09.01	63	63			
64	64	Васильев А.А.	Студент	2019.09.01	64	64			
65	65	Васильев А.А.	Студент	2019.09.01	65	65			
66	66	Васильев А.А.	Студент	2019.09.01	66	66			
67	67	Васильев А.А.	Студент	2019.09.01	67	67			
68	68	Васильев А.А.	Студент	2019.09.01	68	68			
69	69	Васильев А.А.	Студент	2019.09.01	69	69			
70	70	Васильев А.А.	Студент	2019.09.01	70	70			
71	71	Васильев А.А.	Студент	2019.09.01	71	71			
72	72	Васильев А.А.	Студент	2019.09.01	72	72			
73	73	Васильев А.А.	Студент	2019.09.01	73	73			
74	74	Васильев А.А.	Студент	2019.09.01	74	74			
75	75	Васильев А.А.	Студент	2019.09.01	75	75			
76	76	Васильев А.А.	Студент	2019.09.01	76	76			
77	77	Васильев А.А.	Студент	2019.09.01	77	77			
78	78	Васильев А.А.	Студент	2019.09.01	78	78			
79	79	Васильев А.А.	Студент	2019.09.01	79	79			
80	80	Васильев А.А.	Студент	2019.09.01	80	80			
81	81	Васильев А.А.	Студент	2019.09.01	81	81			
82	82	Васильев А.А.	Студент	2019.09.01	82	82			
83	83	Васильев А.А.	Студент	2019.09.01	83	83			
84	84	Васильев А.А.	Студент	2019.09.01	84	84			
85	85	Васильев А.А.	Студент	2019.09.01	85	85			
86	86	Васильев А.А.	Студент	2019.09.01	86	86			
87	87	Васильев А.А.	Студент	2019.09.01	87	87			
88	88	Васильев А.А.	Студент	2019.09.01	88	88			
89	89	Васильев А.А.	Студент	2019.09.01	89	89			
90	90	Васильев А.А.	Студент	2019.09.01	90	90			
91	91	Васильев А.А.	Студент	2019.09.01	91	91			
92	92	Васильев А.А.	Студент	2019.09.01	92	92			
93	93	Васильев А.А.	Студент	2019.09.01	93	93			
94	94	Васильев А.А.	Студент	2019.09.01	94	94			
95	95	Васильев А.А.	Студент	2019.09.01	95	95			
96	96	Васильев А.А.	Студент	2019.09.01	96	96			
97	97	Васильев А.А.	Студент	2019.09.01	97	97			
98	98	Васильев А.А.	Студент	2019.09.01	98	98			
99	99	Васильев А.А.	Студент	2019.09.01	99	99			
100	100	Васильев А.А.	Студент	2019.09.01	100	100			

Приложение А. 3 –UML - диаграмма последовательности



ПГУ 1.090301.19.003 Д14									
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Роль	Заказчик	И.А.	Разработчик	И.А.	Программист	И.А.	Тестировщик	И.А.	Модератор
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
UML-диаграмма последовательности									

Изм.	Лист	№ докум.	Подпись	Дата

Приложение Б

(обязательное)

Приложение в жанре изометрическая головоломка. Функциональная
составляющая
Листинг программы

					ПГУ1.090301.19.001 ПЗ	Лист
						52
Изм.	Лист	№ докум.	Подпись	Дата		

Файл Wall.cs

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

public class Wall : MonoBehaviour
{
    public List<GameObject> nearObjects; //Массив связанных со
    стеной объектов
    public void ShowWall(){
        foreach(GameObject obj in nearObjects){
            obj.SetActive(true);
        }
        gameObject.SetActive(true);
    }
    public void HideWall(){
        foreach(GameObject obj in nearObjects){
            obj.SetActive(false);
        }
        gameObject.SetActive(false);
    }
}
```

Файл ViewManager.cs

```
using UnityEngine;
using System.Collections;
using System;

public class ViewManager : MonoBehaviour
{
    public delegate void ZoomHandler(View view);
    public static event ZoomHandler OnZoom;
    public static event ZoomHandler OnUnZoom;
    private View lastView;
    public static ViewManager instance { get; private set; }
    public Camera currentCamera;
    public float zoomDuration = 1.0f;
    private Vector3 defaultPosition;
    private Quaternion defaultRotation;
    private float defaultSize;
    private Vector3 lastPosition;
    private Quaternion lastRotation;
    private float lastSize;
    public Canvas viewCanvas;
    public Canvas roomCanvas;
    public bool isZoomed;

    void Start()
    {

```

```

        instance = this;

        if (currentCamera == null)
        {
            Destroy(this);
            return;
        }

        roomCanvas.gameObject.SetActive(true);
        viewCanvas.gameObject.SetActive(false);

        defaultPosition = currentCamera.transform.position;
        defaultRotation = currentCamera.transform.rotation;
        defaultSize = currentCamera.orthographicSize;
    }

    public void Zoom(View view)
    {
        RoomRotation state =
        RoomManager.instance.GetRoomState();
        if (!view.utils.Has(state))
            return;
        MessageManager.instance.HideMessage();
        isZoomed = true;

        if(lastView!=null){
            lastView.BlockChildColliders();
            lastView.UnblockParentCollider();
        }

        lastView = view;

        OnZoom?.Invoke(lastView);

        lastPosition = currentCamera.transform.position;
        lastRotation = currentCamera.transform.rotation;
        lastSize = currentCamera.orthographicSize;

        RoomManager.instance.Rotatable = false;

        roomCanvas.gameObject.SetActive(false);
        lastView.BlockParentCollider();
        StartCoroutine(ZoomCoroutine(lastView.utils, state));
        lastView.UnblockChildColliders();
    }

    public void UnZoom()
    {
        isZoomed = false;
        MessageManager.instance.HideMessage();
    }

```

```

OnUnZoom?.Invoke(lastView);

lastView.BlockChildColliders();
viewCanvas.gameObject.SetActive(false);
StartCoroutine(UnZoomCoroutine());
lastView.UnBlockParentCollider();

lastView = null;
RoomManager.instance.Rotatable = true;
}

private IEnumerator ZoomCoroutine(Utils utils, RoomRotation
state)
{
    Vector3 targetPosition = utils.GetPosition(state);
    float targetSize = utils.GetSize();
    Vector3 objectPosition = lastView.transform.position;

    float elapsedTime = 0.0f;

    while (elapsedTime < zoomDuration)
    {
        float t = EaseInOutQuad(elapsedTime / zoomDuration);

        currentCamera.transform.position =
Vector3.Lerp(lastPosition, targetPosition, t);
        currentCamera.orthographicSize =
Mathf.Lerp(lastSize, targetSize, t);

        Quaternion targetRotation =
Quaternion.LookRotation(objectPosition -
currentCamera.transform.position);
        currentCamera.transform.rotation =
Quaternion.Slerp(lastRotation, targetRotation, t);

        elapsedTime += Time.deltaTime;
        yield return null;
    }

    currentCamera.transform.position = targetPosition;
    currentCamera.orthographicSize = targetSize;
    currentCamera.transform.rotation =
Quaternion.LookRotation(objectPosition -
currentCamera.transform.position);
    viewCanvas.gameObject.SetActive(true);
}
private IEnumerator UnZoomCoroutine()
{

```

```

        Vector3 currentPosition =
currentCamera.transform.position;
        float currentSize = currentCamera.orthographicSize;

        Vector3 objectPosition = lastView.transform.position;

        float elapsedTime = 0.0f;

        while (elapsedTime < zoomDuration)
        {
            float t = EaseInOutQuad(elapsedTime / zoomDuration);

            currentCamera.transform.position =
Vector3.Lerp(currentPosition, defaultPosition, t);
            currentCamera.orthographicSize =
Mathf.Lerp(currentSize, defaultSize, t);

            Quaternion targetRotation =
Quaternion.LookRotation(objectPosition -
currentCamera.transform.position);
            Quaternion finalRotation =
Quaternion.Slerp(targetRotation, defaultRotation, t);
            currentCamera.transform.rotation = finalRotation;

            elapsedTime += Time.deltaTime;
            yield return null;
        }

        currentCamera.transform.position = defaultPosition;
        currentCamera.orthographicSize = defaultSize;
        currentCamera.transform.rotation = defaultRotation;
        roomCanvas.gameObject.SetActive(true);
    }

    private float EaseInOutQuad(float t)
    {
        return t < 0.5f ? 2 * t * t : 1 - Mathf.Pow(-2 * t + 2,
2) / 2;
    }
}

```

Файл View.cs

```

using System;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.EventSystems;

public class View : MonoBehaviour
{
    private List<Collider> child_colliders;
    public Utils utils {get; private set;}
}

```

```

private Collider view_collider;
public Routine viewRoutine;
public bool AlwaysWork = false;
void Start(){
    viewRoutine = new ViewRoutineNotSee(this);
    view_collider = gameObject.GetComponent<Collider>();
    utils = gameObject.GetComponent<Utils>();

    child_colliders = new List<Collider>();
    foreach (Transform child in transform){
        Collider collider = child.GetComponent<Collider>();
        if (collider != null)
        {
            child_colliders.Add(collider);
        }
    }
    BlockChildColliders();
}
private void OnMouseDown()
{
    if (EventSystem.current.IsPointerOverGameObject())
        return;
    if(AlwaysWork){
        new ViewRoutineDefault(this).Execute();
        return;
    }
    viewRoutine.Execute();

}
public void UnblockChildColliders(){
    foreach (Collider child in child_colliders){
        if(child == null)
            continue;
        child.enabled = true;
    }
}
public void BlockParentCollider(){
    view_collider.enabled = false;
}
public void UnblockParentCollider(){
    view_collider.enabled = true;
}
public void BlockChildColliders(){
    foreach (Collider child in child_colliders){
        if(child == null)
            continue;
        child.enabled = false;
    }
}
}

```

```

public class ViewRoutineNotSee : Routine
{
    View view;
    public ViewRoutineNotSee(View view){
        this.view = view;
    }
    public virtual void Execute(){
        MessageManager.instance.ShowMessage("Ничего не видно,
        нужно включить свет.",view.transform.position);
    }
}
public class ViewRoutineDefault : Routine
{
    View view;
    public ViewRoutineDefault(View view){
        this.view = view;
    }
    public virtual void Execute(){
        ViewManager.instance.Zoom(view);
    }
}

```

Файл Utils.cs

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Utils : MonoBehaviour
{
    public Vector3 StateDownPosition;
    public Vector3 StateLeftPosition;
    public Vector3 StateUpPosition;
    public Vector3 StateRightPosition;
    public float Size;

    private Dictionary<RoomRotation,ZoomInfo> points = new
    Dictionary<RoomRotation, ZoomInfo>();
    void Start(){
        if(StateDownPosition!=Vector3.zero&&Size!=0){
            points.Add(RoomRotation.Rotation0,new
            ZoomInfo(StateDownPosition,Size));
        }
        if(StateLeftPosition!=Vector3.zero&&Size!=0){
            points.Add(RoomRotation.Rotation90,new
            ZoomInfo(StateLeftPosition,Size));
        }
        if(StateUpPosition!=Vector3.zero&&Size!=0){
            points.Add(RoomRotation.Rotation180,new
            ZoomInfo(StateUpPosition,Size));
        }
        if(StateRightPosition!=Vector3.zero&&Size!=0){

```

```

        points.Add(RoomRotation.Rotation270,new
ZoomInfo(StateRightPosition,Size));
    }
}
public bool Has(RoomRotation state){
    if (points.ContainsKey(state))
    {
        return true;
    }
    return false;
}

public Vector3 GetPosition(RoomRotation state)
{
    return points[state].pos;
}

public float GetSize(){
    return Size;
}
public class ZoomInfo
{
    public ZoomInfo(Vector3 pos,float size){
        this.pos = pos;
        this.size = size;
    }
    public Vector3 pos;
    public float size;
}
}

```

Файл TextMessage.cs

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class TextMessage : MonoBehaviour
{
    public string msg;
    private void OnMouseDown()
    {
        if(msg == null){
            return;
        }

        GameManager.instance.ShowMessage(msg,transform.position);
    }
}

```

Файл SoundManager.cs

```

using UnityEngine;

```

```

using System.Collections.Generic;

public class SoundManager : MonoBehaviour
{
    private static SoundManager instance_;
    public static SoundManager instance
    {
        get
        {
            instance_ = FindObjectOfType<SoundManager>();
            return instance_;
        }
    }

    public string mainSoundName = "main";
    public AudioSource soundSource;
    public AudioSource musicSource;
    private Dictionary<string, AudioClip> audioClips;
    public bool isSoundEnabled { get; private set; } = true;
    public bool isMusicEnabled { get; private set; } = true;

    [Range(0.0f, 1.0f)]
    public float startSoundVolume = 1.0f;
    [Range(0.0f, 1.0f)]
    public float startMusicVolume = 1.0f;

    private void Start()
    {
        if(soundSource == null || musicSource == null){
            Destroy(this);
            return;
        }

        musicSource.loop = true;
        audioClips = new Dictionary<string, AudioClip>();
        LoadAudioClips();

        musicSource.volume = startMusicVolume;
        soundSource.volume = startSoundVolume;
        PlayMusic(mainSoundName);
    }

    private void LoadAudioClips()
    {
        AudioClip[] clips =
Resources.LoadAll<AudioClip>("Audio");
        foreach (var clip in clips)
        {
            if (audioClips.ContainsKey(clip.name))
                continue;

```

```

        audioClips.Add(clip.name, clip);
    }
}

public void PlaySound(string key)
{
    if(!isSoundEnabled)
        return;

    if (!audioClips.ContainsKey(key))
        return;

    soundSource.PlayOneShot(audioClips[key]);
}

public void PlayMusic(string key)
{
    if (!isMusicEnabled)
        return;

    if(!audioClips.ContainsKey(key))
        return;

    musicSource.clip = audioClips[key];
    musicSource.Play();
}

public void StopMusic()
{
    if (musicSource.isPlaying)
        musicSource.Stop();
}

public void SetSoundEnabled(bool flag)
{
    isSoundEnabled = flag;
}

public void SetMusicEnabled(bool flag)
{
    isMusicEnabled = flag;
    if (!isMusicEnabled && musicSource.isPlaying)
    {
        musicSource.Stop();
    }
    else if (isMusicEnabled && !musicSource.isPlaying)
    {
        musicSource.Play();
    }
}

```

```

public void SetSoundVolume(float volume)
{
    soundSource.volume = volume;
}

public void SetMusicVolume(float volume)
{
    musicSource.volume = volume;
}
}

```

Файл Routine.cs

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public interface Routine{
    public virtual void Execute(){
        return;
    }
}

```

Файл RoomManager.cs

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class RoomManager : MonoBehaviour
{
    public string doorOpenSound = "door";
    public static RoomManager instance {get; private set;}
    public RoomController startRoom;
    private RoomController room;
    public bool Rotatable = true;
    private bool isDragging = false;
    private float rotationThreshold = 100f; // Порог для начала
    поворота
    private Vector3 initialMousePosition;

    public Light light_;

    void Start()
    {
        instance = this;

        if(startRoom==null){
            Destroy(this);
            return;
        }
    }
}

```

```

        RoomController[] rooms =
GameObject.FindObjectsOfType<RoomController>();
        foreach (RoomController item in rooms)
        {
            item.Hide();
        }

        room = startRoom;
        room.Show();
    }
    void Update()
    {
        if(!Rotatable)
            return;
        // Начало поворота
        if (Input.GetMouseButtonDown(0))
        {
            isDragging = true;
            initialMousePosition = Input.mousePosition;
        }

        // Завершение поворота
        if (Input.GetMouseButtonUp(0))
        {
            isDragging = false;
        }

        // Обработка поворота
        if (isDragging)
        {
            Vector3 currentMousePosition = Input.mousePosition;
            Vector3 difference = currentMousePosition -
initialMousePosition;

            if (Mathf.Abs(difference.x) > rotationThreshold)
            {
                TryRotate((difference.x < 0));
                isDragging = false; // Сбрасываем флаг после
начала поворота, чтобы предотвратить повторные повороты
            }
        }
    }
    public RoomController GetCurrentRoom(){
        return room;
    }
    public void TryRotate(bool clockwise){
        room.SmoothRotate(clockwise);
    }
    public void SwitchRoom(RoomController to){
        MessageManager.instance.HideMessage();
        if(to == null){

```

```

        return;
    }
    SoundManager.instance.PlaySound(doorOpenSound);
    RoomRotation state = room.Hide();
    room = to;
    room.Show(state);
}
public RoomRotation GetRoomState() {
    return room.State;
}
}

```

Файл RoomController.cs

```

using UnityEngine;
using System.Collections;
using System.Collections.Generic;
using UnityEngine.Scripting;

public class RoomController : MonoBehaviour
{
    public delegate void RotateHandler(bool clockwise);
    public static event RotateHandler OnRotate;
    public List<Wall> walls;
    RoomRotation state;
    private float rotationDuration = 0.5f; // Продолжительность
поворота
    private Quaternion targetRotation;
    private Vector3 originalScale;
    private float scaleAmount = 0.96f; // Коэффициент уменьшения
размера при повороте
    bool isRotating = false;
    bool isLightEnabled = false;
    public bool AlwaysLight = false;

    private List<View> view_list = new List<View>();

    public RoomRotation State { get => state; }

    void Start() {
        originalScale = transform.localScale;

        foreach (Transform child in transform) {
            View temp = child.GetComponent<View>();
            if(temp != null) {
                view_list.Add(temp);
            }
        }
    }

    public void SwitchLight() {
        isLightEnabled = !isLightEnabled;
    }
}

```

```

        if(RoomManager.instance.GetCurrentRoom() == this){
            SetRoomLight();
        }
    }
    private void SetRoomLight(){
        if(AlwaysLight){
            RoomManager.instance.light_.enabled = true;
            return;
        }

        RoomManager.instance.light_.enabled = isLightEnabled;
        if(isLightEnabled){
            EnableViews();
            return;
        }
        DisableViews();
    }
    void EnableViews(){
        foreach(View view in view_list){
            view.viewRoutine = new ViewRoutineDefault(view);
        }
    }
    void DisableViews(){
        foreach(View view in view_list){
            view.viewRoutine = new ViewRoutineNotSee(view);
        }
    }

    private void FastRotate()
    {
        UpdateWalls();
        transform.rotation = Quaternion.Euler(0f, (float)state,
0f);
    }
    private void UpdateWalls(){
        if(walls.Count!=4)
            return;

        switch (state)
        {
            case RoomRotation.Rotation0:
                walls[0].HideWall();
                walls[1].HideWall();
                walls[2].ShowWall();
                walls[3].ShowWall();
                break;
            case RoomRotation.Rotation90:
                walls[0].HideWall();
                walls[1].ShowWall();
                walls[2].ShowWall();
                walls[3].HideWall();

```

```

        break;
    case RoomRotation.Rotation180:
        walls[0].ShowWall();
        walls[1].ShowWall();
        walls[2].HideWall();
        walls[3].HideWall();
        break;
    case RoomRotation.Rotation270:
        walls[0].ShowWall();
        walls[1].HideWall();
        walls[2].HideWall();
        walls[3].ShowWall();
        break;
    }
}

public void SmoothRotate(bool clockwise){
    if(isRotating)
        return;
    state = GetNext(clockwise);
    OnRotate?.Invoke(clockwise);
    if(clockwise){
        StartCoroutine(SmoothRotateCoroutine(90));
    }else{
        StartCoroutine(SmoothRotateCoroutine(-90));
    }
}

private IEnumerator SmoothRotateCoroutine(float angle)
{
    MessageManager.instance.HideMessage();
    isRotating = true;
    // Записываем начальную и целевую ротацию
    Quaternion startRotation = transform.rotation;
    targetRotation = Quaternion.Euler(transform.eulerAngles
+ new Vector3(0, angle, 0));

    float elapsedTime = 0f;

    // Сначала уменьшаем масштаб
    Vector3 targetScale = originalScale * scaleAmount;
    while (elapsedTime < rotationDuration / 4f)
    {
        float t = elapsedTime / (rotationDuration / 4f);
        transform.localScale = Vector3.Lerp(originalScale,
targetScale, t);
        elapsedTime += Time.deltaTime;
        yield return null;
    }

    elapsedTime = 0f;

    bool is_Updated = false;

```

```

// Плавное вращение с восстановлением масштаба
while (elapsedTime < rotationDuration)
{
    float t = elapsedTime / rotationDuration;
    t = Mathf.SmoothStep(0, 1, t);
    transform.rotation = Quaternion.Slerp(startRotation,
targetRotation, t);
    transform.localScale = Vector3.Lerp(targetScale,
originalScale, t);

    elapsedTime += Time.deltaTime;

    if (!is_Updated && t > rotationDuration/1.2f)
    {
        UpdateWalls();
        is_Updated = true;
    }
    yield return null;
}

transform.rotation = targetRotation;
transform.localScale = originalScale;
isRotating = false;
}

public void Show(RoomRotation state =
RoomRotation.Rotation0){
    this.state = state;
    FastRotate();
    SetRoomLight();
    gameObject.SetActive(true);
}

public RoomRotation Hide(){
    gameObject.SetActive(false);
    return state;
}

private RoomRotation GetNext(bool clockwise)
{
    if(clockwise)
    {
        switch (state)
        {
            case RoomRotation.Rotation0: return
RoomRotation.Rotation90;
            case RoomRotation.Rotation90: return
RoomRotation.Rotation180;
            case RoomRotation.Rotation180: return
RoomRotation.Rotation270;
            case RoomRotation.Rotation270: return
RoomRotation.Rotation0;
            default: return RoomRotation.Rotation0;
        }
    }
}

```



```

        messageObject.transform.position += new
Vector3(10,10,0);
        messageObject.SetActive(true);
        StartCoroutine(HideMessageAfterDelay(duration));
    }

    private IEnumerator HideMessageAfterDelay(float delay)
    {
        yield return new WaitForSeconds(delay);
        messageObject.SetActive(false);
    }
    public void HideMessage(){
        messageObject.SetActive(false);
    }
}

```

Файл MenuManager.cs

```

using UnityEngine;
using UnityEngine.UI;

public class MenuManager : MonoBehaviour
{
    public GameObject settingsPanel;
    public GameObject blockPanel;
    public Button menuButton;
    public Button soundButton;
    public Button musicButton;
    public Slider soundSlider;
    public Slider musicSlider;
    public Button exitButton;
    public Sprite soundOnIcon;
    public Sprite soundOffIcon;
    public Sprite musicOnIcon;
    public Sprite musicOffIcon;
    private bool isRoomRotatable;
    private bool isBoxRotatable;

    private void Start()
    {
        settingsPanel.SetActive(false);
        blockPanel.SetActive(false);

        menuButton.onClick.AddListener(ToggleSettingsPanel);
        soundButton.onClick.AddListener(ToggleSound);
        musicButton.onClick.AddListener(ToggleMusic);
        musicSlider.onValueChanged.AddListener(SetMusicVolume_);
        soundSlider.onValueChanged.AddListener(SetSoundVolume_);
        exitButton.onClick.AddListener(Application.Quit);

        soundSlider.value =
SoundManager.instance.startSoundVolume;
    }
}

```

```

        musicSlider.value =
SoundManager.instance.startMusicVolume;
    }

    private void ToggleSettingsPanel()
    {
        settingsPanel.SetActive(!settingsPanel.activeSelf);
        blockPanel.SetActive(settingsPanel.activeSelf);
        if(settingsPanel.activeSelf){
            isRoomRotatable = RoomManager.instance.Rotatable;
            isBoxRotatable = BoxController.instance.Rotatable;
            RoomManager.instance.Rotatable = false;
            BoxController.instance.Rotatable = false;
        } else{
            RoomManager.instance.Rotatable = isRoomRotatable;
            BoxController.instance.Rotatable = isBoxRotatable;
        }
    }
    private void SetSoundVolume_(float value){
        SoundManager.instance.SetSoundVolume(value);
    }
    private void SetMusicVolume_(float value){
        SoundManager.instance.SetMusicVolume(value);
    }

    private void ToggleSound()
    {
        SoundManager.instance.SetSoundEnabled(!SoundManager.instance.isSoundEnabled);

        if (SoundManager.instance.isSoundEnabled){
            soundButton.image.sprite = soundOnIcon;
            return;
        }
        soundButton.image.sprite = soundOffIcon;
    }

    private void ToggleMusic()
    {
        SoundManager.instance.SetMusicEnabled(!SoundManager.instance.isMusicEnabled);
        if (SoundManager.instance.isMusicEnabled){
            musicButton.image.sprite = musicOnIcon;
            return;
        }
        musicButton.image.sprite = musicOffIcon;
    }
}

```

Файл InventoryManager.cs

```
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class InventoryManager : MonoBehaviour
{
    public delegate void SelectHandler(string obj);
    public delegate void DeselectHandler();
    public static event SelectHandler OnSelect;
    public static event DeselectHandler OnDeselect;

    public static InventoryManager instance;
    public Transform inventoryPanel;
    public GameObject inventorySlotPrefab;
    private List<InventoryItem> items_ = new
List<InventoryItem>();
    public InventoryItem selected;
    private Button selectedButton;
    private GameObject selectedFrame;

    void Start(){
        instance = this;
    }
    public void AddItem(InventoryItem item)
    {
        items_.Add(item);
        UpdateInventoryUI();
    }

    public void RemoveItem(InventoryItem item)
    {
        items_.Remove(item);
        UpdateInventoryUI();
        DeselectItem();
    }

    public void SelectItem(InventoryItem item, Button button)
    {
        if (selected == item){
            DeselectItem();
            return;
        }
        OnSelect?.Invoke(item.key);

        if (selectedFrame != null)
        {
            selectedFrame.SetActive(false);
        }
    }
}
```

```

        selected = item;
        selectedButton = button;

        Transform frameTransform =
selectedButton.transform.Find("SelectionFrame");
        if (frameTransform != null)
        {
            selectedFrame = frameTransform.gameObject;
            selectedFrame.SetActive(true);
        }
    }

    public void DeselectItem()
    {
        OnDeselect?.Invoke();
        if (selectedFrame != null)
        {
            selectedFrame.SetActive(false);
            selectedFrame = null;
        }

        selected = null;
        selectedButton = null;
    }

    private void UpdateInventoryUI()
    {
        // Удаляем старые слоты
        foreach (Transform child in inventoryPanel)
        {
            Destroy(child.gameObject);
        }

        // Создаем новые слоты и настраиваем их
        foreach (var item in items_)
        {
            GameObject slot = Instantiate(inventorySlotPrefab,
inventoryPanel);
            slot.GetComponent<Image>().sprite = item.icon;

            Button button = slot.GetComponent<Button>();
            button.onClick.AddListener(() => SelectItem(item,
button));

            // Скрываем рамку выбора для всех новых слотов
            Transform frameTransform =
slot.transform.Find("SelectionFrame");
            if (frameTransform != null)
            {
                frameTransform.gameObject.SetActive(false);
            }
        }
    }

```

```

    }

    // Если был выбранный предмет, снова показываем его
    рамку выбора
    if (selected != null && selectedButton != null)
    {
        Transform frameTransform =
selectedButton.transform.Find("SelectionFrame");
        if (frameTransform != null)
        {
            selectedFrame = frameTransform.gameObject;
            selectedFrame.SetActive(true);
        }
    }
}

```

Файл InventoryItem.cs

```

using UnityEngine;
using System.Collections;
using System.Collections.Generic;

public class InventoryItem : MonoBehaviour
{
    public Sprite icon;
    public string key;
    void OnMouseDown()
    {
        gameObject.SetActive(false);
        InventoryManager.instance.AddItem(this);
    }
}

```

Файл Door.cs

```

using UnityEngine;

public class Door : MonoBehaviour
{
    public RoomController targetRoom;

    private void OnMouseDown()
    {
        if (ViewManager.instance.isZoomed)
            return;

        RoomManager.instance.SwitchRoom(targetRoom);
    }
}

```

Файл TwoStateAnimation.cs

```

using System.Collections;

```

```

using System.Collections.Generic;
using UnityEngine;

public class TwoStateAnimation : MonoBehaviour
{
    public delegate void SwitchStateDelegate();
    public static event SwitchStateDelegate OnSwitchState;
    private Animator animator;
    bool isAnimated = false;
    public AnimationClip firstAnimation;
    public AnimationClip secondAnimation;
    bool isInteractable = true;
    void Start()
    {
        animator = GetComponent<Animator>();
    }
    void OnMouseDown() {
        if(!isInteractable)
            return;

        if(isAnimated) {
            animator.Play(secondAnimation.name);
        } else {
            animator.Play(firstAnimation.name);
        }

    }
    public void EnableInteraction()
    {
        isAnimated=!isAnimated;
        OnSwitchState?.Invoke();
        isInteractable = true;
    }

    public void DisableInteraction()
    {
        isInteractable = false;
    }
    public bool GetState(){
        return isAnimated;
    }
}

```

Файл SwitcherRoomLight.cs

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class SwitcherRoomLight : MonoBehaviour
{
    private Animator animator;

```

```

public RoomController room;
bool isActive = false;

Routine obj_routine;

bool isInteractable = true;
void Start()
{
    animator = GetComponent<Animator>();
}
void OnMouseDown(){
    if(!isInteractable)
        return;

    if(isActive) {
        animator.Play("Off1");
    } else {
        animator.Play("On1");
    }

}

public void EnableInteraction()
{
    isInteractable = true;
}

public void DisableInteraction()
{
    isInteractable = false;
}

public void Switch(){
    room.SwitchLight();
    isActive=!isActive;
}
}

public class SwitchDefaultRoutine : Routine{

    public virtual void Execute(){

    }

}

```

Файл LegoSwitch.cs

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class LegoSwitch : MonoBehaviour
{
    public AnimationClip clip;
}

```

```

    public string cantMessage = "Кажется тут чего-то не
хватает.";
    public string needItemKey;
    Routine routine;
    void Start()
    {
        if(needItemKey == null){
            Destroy(this);
            return;
        }
        routine = new MessageRoutine(gameObject,cantMessage);
        InventoryManager.OnSelect+=OnInventorySelect;
        InventoryManager.OnDeselect+=OnInventoryDeselect;
        foreach(Transform child in transform){
            child.gameObject.SetActive(false);
        }
    }
    public void OnInventorySelect(string key){
        if(needItemKey == key){
            routine = new SwitchShowRoutine(this);
        } else {
            routine = new MessageRoutine(gameObject,"Нет, тут
должно быть что-то другое.");
        }
    }
    public void OnInventoryDeselect(){
        routine = new MessageRoutine(gameObject,cantMessage);
    }
    public void Show(){
        Collider col = GetComponent<Collider>();
        Destroy(col);

        InventoryManager.OnSelect-=OnInventorySelect;
        InventoryManager.OnDeselect-=OnInventoryDeselect;

        InventoryManager.instance.RemoveItem(InventoryManager.instance.s
elected);
        routine = new NullRoutine();
        Animator animator = GetComponent<Animator>();

        foreach(Transform child in transform){
            child.gameObject.SetActive(true);
        }

        gameObject.transform.localScale = Vector3.zero;
        animator.Play(clip.name);
    }

    void OnMouseDown(){
        routine.Execute();
    }

```

```

}

public class SwitchShowRoutine : Routine
{
    LegoSwitch swtch;
    public SwitchShowRoutine(LegoSwitch swtch){
        this.swtch = swtch;
    }
    public virtual void Execute(){
        swtch.Show();
    }
}

```

Файл DisableBehind.cs

```

using System.Collections;
using System.Collections.Generic;
using Unity.VisualScripting;
using UnityEngine;

public class DisableBehind : MonoBehaviour
{
    public GameObject obj_behind;
    private Collider collider_behind;
    bool state = false;
    void Start(){
        if(obj_behind == null){
            Destroy(this);
            return;
        }
        collider_behind = obj_behind.GetComponent<Collider>();
        if(collider_behind == null)
            return;
        state =
gameObject.GetComponent<TwoStateAnimation>().GetState();
        collider_behind.enabled = state;
        TwoStateAnimation.OnSwitchState+=ChangeState;
    }
    public void ChangeState(){
        state =
gameObject.GetComponent<TwoStateAnimation>().GetState();
        if(!obj_behind.activeSelf || collider_behind == null){
            return;
        }
        collider_behind.enabled = state;
    }
}

```

Файл ConstructorController.cs

```

using System.Collections;
using System.Collections.Generic;

```

```

using System.Linq;
using UnityEngine;

public class ConstructorController : MonoBehaviour
{
    public static ConstructorController instance;
    private Dictionary<uint, ConstructorPart> parts_ = new
Dictionary<uint, ConstructorPart>();
    private List<uint> history_parts_ = new List<uint>();
    private uint num = 2;
    public uint end_number = 60;

    public GameObject prize;

    void Start()
    {
        prize.SetActive(false);
        foreach(Transform child in transform){
            ConstructorPart temp =
child.GetComponent<ConstructorPart>();
            if (temp != null){
                temp.gameObject.SetActive(true);
                parts_.Add(temp.number, temp);
            }
        }
        if (parts_.Count != 29){
            Destroy(this);
            return;
        }
        instance = this;
    }

    public void CheckPartNumber(uint new_num){
        if(new_num-num!=2){
            if(num==2 || history_parts_.Count==0)
                return;

            parts_[history_parts_.Last()].Show();
            history_parts_.RemoveAt(history_parts_.Count-1);
            num-=2;
            return;
        }
        num=new_num;
        parts_[num].Hide();
        history_parts_.Add(num);
        if(num != end_number)
            return;
        //Win
        foreach(ConstructorPart item in parts_.Values){
            item.Hide();
        }
    }
}

```

```

        Animator prize_animator =
prize.GetComponent<Animator>();
        if(prize_animator==null){
            prize.SetActive(true);
            return;
        }

        prize.transform.localScale = Vector3.zero;
        prize.SetActive(true);
        prize_animator.Play("PrizeShow");
    }
}

```

Файл ConstructorPart.cs

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class ConstructorPart : MonoBehaviour
{
    private Animator animator;
    public uint number;

    private Collider col;

    void Start(){
        col = GetComponent<Collider>();
        animator = GetComponent<Animator>();
        if(animator == null || col ==null){
            Destroy(this);
            return;
        }
    }
    void OnMouseDown(){
        ConstructorController.instance.CheckPartNumber(number);
    }
    public void Show(){
        gameObject.transform.localScale = Vector3.zero;
        animator.Play("BrickShow");
        col.enabled = true;
    }
    public void Hide(){
        animator.Play("BrickHide");
        col.enabled = false;
    }
}

```

Файл BoxCover.cs

```

using System.Collections;

```

```

using System.Collections.Generic;
using UnityEngine;

public class BoxCover : MonoBehaviour
{
    public AnimationClip clip;
    public string cantMessage = "Не могу открыть руками.";
    public string needItemKey;
    Routine routine;
    void Start()
    {
        if(needItemKey == null){
            Destroy(this);
            return;
        }
        routine = new MessageRoutine(gameObject, cantMessage);
        InventoryManager.OnSelect+=OnInventorySelect;
        InventoryManager.OnDeselect+=OnInventoryDeselect;

    }
    public void OnInventorySelect(string key){
        if(needItemKey == key){
            routine = new BoxCoverOpenRoutine(this);
        } else {
            routine = new MessageRoutine(gameObject, "Кажется это
не подходит");
        }
    }
    public void OnInventoryDeselect(){
        routine = new MessageRoutine(gameObject, cantMessage);
    }
    public void OpenCover(){
        Collider col = GetComponent<Collider>();
        Destroy(col);

        InventoryManager.OnSelect-=OnInventorySelect;
        InventoryManager.OnDeselect-=OnInventoryDeselect;

        InventoryManager.instance.RemoveItem(InventoryManager.instance.s
elected);
        routine = new NullRoutine();

        Animator animator = GetComponent<Animator>();
        animator.Play(clip.name);
    }
    public void SendOpenBox(){
        BoxController.instance.OpenBox();
    }
    void OnMouseDown(){

        routine.Execute();
    }
}

```

```

    }
}
public class NullRoutine : Routine {
    public virtual void Execute(){
        //NoCommand
    }
}
public class BoxCoverOpenRoutine : Routine
{
    BoxCover cover;
    public BoxCoverOpenRoutine(BoxCover cover){
        this.cover = cover;
    }
    public virtual void Execute(){
        cover.OpenCover();
    }
}

public class MessageRoutine : Routine{

    GameObject obj;
    string message;
    public MessageRoutine(GameObject obj, string message){
        this.obj = obj;
        this.message = message;
    }
    public virtual void Execute(){

        GameManager.instance.ShowMessage(message,obj.transform.position);
        InventoryManager.instance.DeselectItem();
    }
}

```

Файл BoxController.cs

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class BoxController : MonoBehaviour
{
    public static BoxController instance;
    public bool isOpened = false;
    private float rotationDuration = 0.5f; // Продолжительность
поворота
    private Quaternion targetRotation;
    bool isRotating = false;

    public bool Rotatable = false;
    private bool isDragging = false;
    private float rotationThreshold = 100f;
}

```

```

private Vector3 initialMousePosition;
public List<Wall> walls;
private View parent_view;
RoomRotation rotation_state = RoomRotation.Rotation0;
RoomRotation walls_state;

void Start() {
    parent_view = GetComponentInParent<View>();
    if(parent_view == null){
        Destroy(this);
        return;
    }
    instance = this;

    ViewManager.OnZoom+=OnZoomRoutine;
    ViewManager.OnUnZoom+=OnUnZoomRoutine;
    RoomController.OnRotate+=OnRoomRotate;
}
void OnZoomRoutine(View view)
{
    if(view != parent_view || !isOpen) {
        return;
    }

    //walls_state = GetActualRoomState();
    SetWallsState();

    Rotatable = true;
}
public void OpenBox() {
    isOpen = true;
    OnZoomRoutine(parent_view);
}
void OnUnZoomRoutine(View view) {
    if(view != parent_view || !isOpen) {
        return;
    }
    Rotatable = false;

    ReturnWalls();
}
void OnRoomRotate(bool clockwise) {
    NextState(clockwise, ref walls_state);
}

void Update()
{
    if(!Rotatable)
        return;

    if (Input.GetMouseButtonDown(0))

```

```

    {
        isDragging = true;
        initialMousePosition = Input.mousePosition;
    }

    if (Input.GetMouseButtonUp(0))
    {
        isDragging = false;
    }

    if (isDragging)
    {
        Vector3 currentMousePosition = Input.mousePosition;
        Vector3 difference = currentMousePosition -
initialMousePosition;

        if (Mathf.Abs(difference.x) > rotationThreshold)
        {
            TryRotate((difference.x < 0));
            isDragging = false;
        }
    }

    public void TryRotate(bool clockwise){
        SmoothRotate(clockwise);
    }

    public void SmoothRotate(bool clockwise){
        if(isRotating)
            return;
        NextState(clockwise, ref rotation_state);
        StartCoroutine(BoxRotateCoroutine(clockwise));
    }
    private IEnumerator BoxRotateCoroutine(bool clockwise)
    {
        float angle;
        if (clockwise){
            angle = 90;
        } else {
            angle = -90;
        }

        isRotating = true;
        // Записываем начальную и целевую ротацию
        Quaternion startRotation = transform.rotation;
        targetRotation = Quaternion.Euler(transform.eulerAngles
+ new Vector3(0, angle, 0));

```

```

float elapsedTime = 0f;

bool is_Updated = false;

// Плавное вращение с восстановлением масштаба
while (elapsedTime < rotationDuration)
{
    float t = elapsedTime / rotationDuration;
    t = Mathf.SmoothStep(0, 1, t);
    transform.rotation = Quaternion.Slerp(startRotation,
targetRotation, t);
    elapsedTime += Time.deltaTime;

    if (!is_Updated && t > rotationDuration/1.2f)
    {
        NextState(clockwise, ref walls_state);
        SetWallsState();
        is_Updated = true;
    }
    yield return null;
}

transform.rotation = targetRotation;
isRotating = false;
}

private void ReturnWalls(){
    foreach(Wall wall in walls){
        wall.ShowWall();
    }
}

private void SetWallsState(){
    if(walls.Count!=4)
        return;

    switch (walls_state)
    {
        case RoomRotation.Rotation0:
            walls[0].HideWall();
            walls[1].HideWall();
            walls[2].ShowWall();
            walls[3].ShowWall();
            break;
        case RoomRotation.Rotation90:
            walls[0].HideWall();
            walls[1].ShowWall();
            walls[2].ShowWall();
            walls[3].HideWall();
            break;
        case RoomRotation.Rotation180:
            walls[0].ShowWall();

```

```

        walls[1].ShowWall();
        walls[2].HideWall();
        walls[3].HideWall();
        break;
    case RoomRotation.Rotation270:
        walls[0].ShowWall();
        walls[1].HideWall();
        walls[2].HideWall();
        walls[3].ShowWall();
        break;
    }
}

private void NextState(bool clockwise, ref RoomRotation
state)
{
    if(clockwise)
    {
        switch (state)
        {
            case RoomRotation.Rotation0:
                state = RoomRotation.Rotation90;
                return;
            case RoomRotation.Rotation90:
                state = RoomRotation.Rotation180;
                return;
            case RoomRotation.Rotation180:
                state = RoomRotation.Rotation270;
                return;
            case RoomRotation.Rotation270:
                state = RoomRotation.Rotation0;
                return;
            default:
                state = RoomRotation.Rotation0;
                return;
        }
    } else{
        switch (state)
        {
            case RoomRotation.Rotation0:
                state = RoomRotation.Rotation270;
                return;
            case RoomRotation.Rotation90:
                state = RoomRotation.Rotation0;
                return;
            case RoomRotation.Rotation180:
                state = RoomRotation.Rotation90;
                return;
            case RoomRotation.Rotation270:
                state = RoomRotation.Rotation180;
                return;
        }
    }
}

```

```

        default:
            state = RoomRotation.Rotation0;
            return;
    }
}
}
}

```

Файл Rock.cs

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Rock : MonoBehaviour
{
    public AnimationClip clip;
    public string cantMessage = "Не получается поднять.";
    public string needItemKey;
    Routine routine;
    void Start()
    {
        if(needItemKey == null){
            Destroy(this);
            return;
        }
        routine = new MessageRoutine(gameObject, cantMessage);
        InventoryManager.OnSelect+=OnInventorySelect;
        InventoryManager.OnDeselect+=OnInventoryDeselect;

    }
    public void OnInventorySelect(string key){
        if(needItemKey == key){
            routine = new RockOpenRoutine(this);
        } else {
            routine = new MessageRoutine(gameObject, "Нужно что-
то другое.");
        }
    }
    public void OnInventoryDeselect(){
        routine = new MessageRoutine(gameObject, cantMessage);
    }
    public void OpenWithShovel(){
        Collider col = GetComponent<Collider>();
        Destroy(col);

        InventoryManager.OnSelect-=OnInventorySelect;
        InventoryManager.OnDeselect-=OnInventoryDeselect;

        InventoryManager.instance.RemoveItem(InventoryManager.instance.s
elected);
        routine = new NullRoutine();
    }
}

```

```

        Animator animator = GetComponentInParent<Animator>();
        if(animator == null){
            return;
        }
        animator.Play(clip.name);
    }
    void OnMouseDown(){
        routine.Execute();
    }
}

public class RockOpenRoutine : Routine
{
    Rock rock;
    public RockOpenRoutine(Rock rock){
        this.rock = rock;
    }
    public virtual void Execute(){
        rock.OpenWithShovel();
    }
}

```

Файл LockedDoor.cs

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class LockedDoor : MonoBehaviour
{
    public RoomController to;
    public AnimationClip clip;
    public string cantMesage = "Заперто.";
    public string needItemKey;
    Routine routine;
    void Start()
    {
        if(needItemKey == null){
            Destroy(this);
            return;
        }
        routine = new MessageRoutine(gameObject, cantMesage);
        InventoryManager.OnSelect+=OnInventorySelect;
        InventoryManager.OnDeselect+=OnInventoryDeselect;
    }
    public void OnInventorySelect(string key){
        if(needItemKey == key){
            routine = new DoorOpenRoutine(this);
        } else {

```

```

        routine = new MessageRoutine(gameObject, "Думаю, это
не подойдет.");
    }
}
public void OnInventoryDeselect() {
    routine = new MessageRoutine(gameObject, cantMesage);
}
public void GoToRoom() {
    InventoryManager.OnSelect==OnInventorySelect;
    InventoryManager.OnDeselect==OnInventoryDeselect;

InventoryManager.instance.RemoveItem(InventoryManager.instance.s
elected);
    routine = new GoToRoomRoutine(to);

    Animator animator = GetComponentInParent<Animator>();
    if(animator == null){
        return;
    }
    animator.Play(clip.name);
}
void OnMouseDown() {
    routine.Execute();
}
}

public class DoorOpenRoutine : Routine
{
    LockedDoor door;
    public DoorOpenRoutine(LockedDoor door){
        this.door = door;
    }
    public virtual void Execute(){
        door.GoToRoom();
    }
}

public class GoToRoomRoutine : Routine
{
    RoomController to;
    public GoToRoomRoutine(RoomController to){
        this.to = to;
    }
    public virtual void Execute(){
        if(ViewManager.instance.isZoomed)
            return;
        RoomManager.instance.SwitchRoom(to);
    }
}

```

Приложение В

(обязательное)

Приложение в жанре изометрическая головоломка. Функциональная
составляющая
Презентация

					ПГУ1.090301.19.001 ПЗ	Лист
						89
Изм.	Лист	№ докум.	Подпись	Дата		

ПРИЛОЖЕНИЕ В ЖАНРЕ ИЗОМЕТРИЧЕСКАЯ ГОЛОВОЛОМКА. ФУНКЦИОНАЛЬНАЯ СОСТАВЛЯЮЩАЯ

Автор работы:

Студент 20ВВП1 Пантелеев И. А.

Научный руководитель:

к.т.н. доцент кафедры ВТ Юрова О.В

Цели работы:

- 1) Реализация фундамента для создания приложения (написание игровых механик и скриптов отвечающих за поведение объектов)
- 2) Привязка написанных скриптов и механик к реальным игровым объектам (создание наполнения приложения)
- 3) Дополнение приложения визуальными и звуковыми эффектами (создание качественного игрового опыта)
- 4) Проверка багов и оптимизация

Аналоги приложения



Monument Valley



The Almost Gone



Tiny Room Stories Town Mystery

Игровой движок



Дополнительные инструменты для разработки



Игровые механики

- 1) Механика сообщений
- 2) Приближения к объектам и их группам
- 3) Вращение комнаты
- 4) Перемещение между комнатами

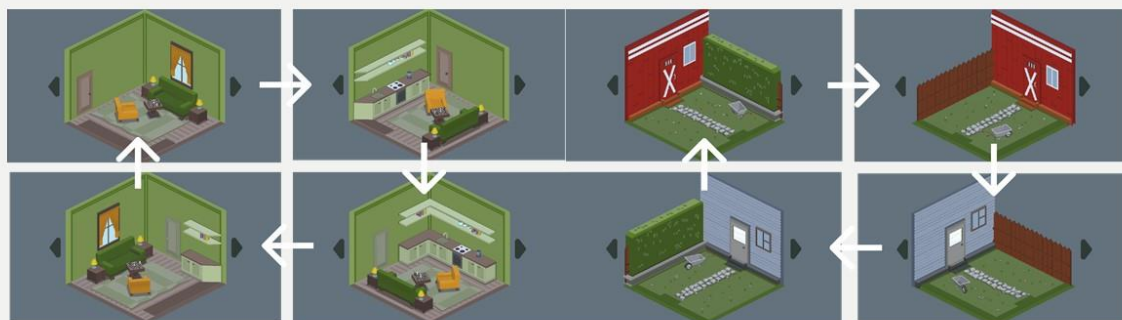
Вывод сообщений и подсказок



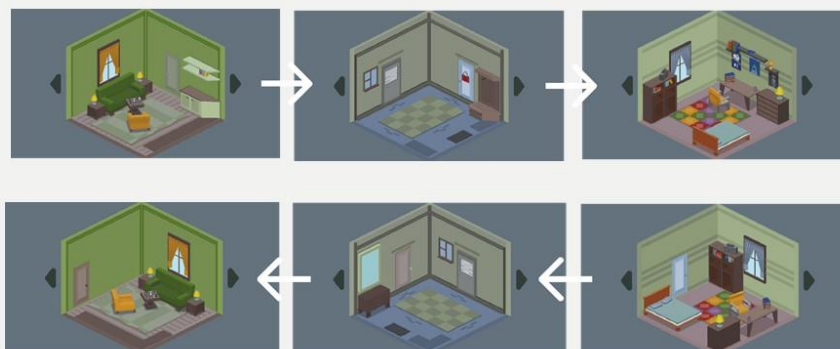
Приближение к объектам



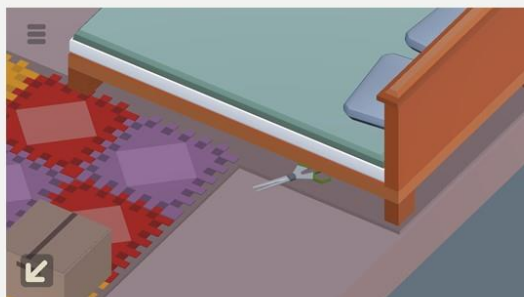
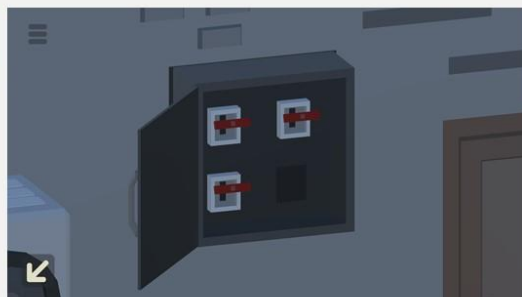
Вращение комнаты



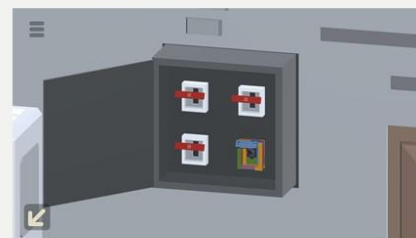
Перемещение между комнатами



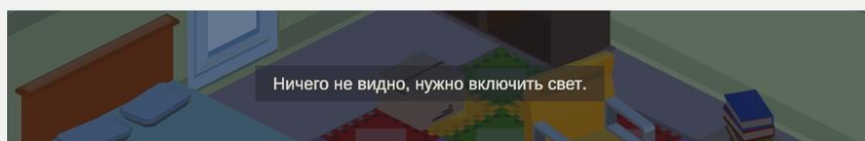
Наполнение игры



Головоломка «Конструктор»



Управление поведением объектов



Демонстрация игрового приложения



Спасибо за внимание