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

МОБИЛЬНОЕ ПРИЛОЖЕНИЕ «ESCAPE BUILDING-9»

Студент 29.05.25 Кирмасов Иван Дмитриевич
(подпись, дата)

Руководитель 29.05.25 Зинкин С. А.
(подпись, дата)

Нормоконтролёр 29.05.25 А.С. Бычков
(подпись, дата)

Работа допущена к защите (протокол зас. кафедры от 28.05.2025 г. № 16)

Заведующий кафедрой _____ М.А. Митрохин
(подпись)

Работа защищена с отметкой отлично (протокол заседания ГЭК от 11 июня 2025 г. № 28)

Секретарь ГЭК _____ Е.И. Гурин
(подпись)

Пенза 2025

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

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
М.А. Митрохин
«06» 05 2025 г.

ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ

1. Студент Кирмасов Иван Дмитриевич гр.21ВВП2 факультета вычислительной техники.

2.Тема работы: Мобильное приложение «Escape Building-9»

Тема утверждена приказом ПГУ №601/о от "06" мая 2025г.

3. Руководитель бакалаврской работы д.т.н., профессор кафедры ВТ ПГУ Зинкин С. А.

4. Задание на работу (назначение разработки, исходные данные и т.п.)
Разработать мобильное приложение «Escape Building-9» в жанре survival-horror с элементами RPG, обеспечивающее погружение за счёт ретро-стилистики, адаптированных игровых механик и локаций, основанных на архитектуре 9-го корпуса ПГУ. Приложение должно включать: систему управления персонажем с адаптацией под сенсорный ввод; динамический инвентарь с механикой стратегического планирования ресурсов; внутриигровой интерфейс, оптимизированный для мобильных устройств; систему сохранения прогресса с шифрованием данных; интеграцию обучающей системы для новых игроков.
5. Перечень подлежащих разработке вопросов
Произвести анализ предметной области, включающий исследование игровых механик survival-horror и RPG, а также изучение особенностей ретро-стилистики на примере графики PlayStation 1 и требований к мобильным интерфейсам с учётом удобства сенсорного управления. Спроектировать игровые системы, для управления персонажем, инвентарём и сохранениями. Оптимизировать код на Godot Engine (GDScript) для снижения нагрузки на CPU и GPU.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Изучение предметной области	32	6.05.2025 - 28.05.2025	
2	Анализ требований	32	7.05.2025 - 30.05.2025	
3	Создание макета графического интерфейса	30	9.04.2025 - 3.05.2025	
4	Выбор технологий, применяемых при проектировании и разработке	40	11.05.2025 - 13.05.2025	
5	Реализация графического интерфейса	66	14.05.2025 - 17.05.2025	
6	Разработка основной части программы	70	16.05.2025 - 20.05.2025	
7	Оформление пояснительной записки	54	21.05.2025 - 23.05.2025	

Дата выдачи задания "06" мая 2025 г.

Руководитель бакалаврской работы _____ 6.05.25 Зинкин С. А.
(подпись, дата)

Задание к исполнению принял студент _____ 6.05.25 Кирмасов И. Д.
(подпись, дата)

Работу к защите допустить
Декан факультета _____ 09.06.2025 Фионова Л.Р.

(подпись, дата)

Реферат

Бакалаврская работа содержит 113 листов, 29 рисунков, 2 таблицы, 7 источников и 3 приложения.

МОБИЛЬНАЯ РАЗРАБОТКА, GODOT ENGINE, ОБРАЗОВАТЕЛЬНАЯ ГЕЙМИФИКАЦИЯ, АДАПТИВНЫЙ ИНТЕРФЕЙС.

Цель работы – разработать мобильное приложение «Escape Building-9» в жанре survival-horror с элементами RPG, обеспечивающее погружение за счёт ретро-стилистики, адаптированных механик и локаций на основе архитектуры девятого корпуса ПГУ.

Объектами разработки являются:

– игровое приложение для Android, включающее систему управления персонажем, инвентарём и диалогами;

– 3D-модели и текстуры в стиле PlayStation 1, оптимизированные для слабых устройств;

– внутриигровой интерфейс с адаптацией под сенсорное управление;

– система сохранения прогресса с шифрованием данных.

В результате проделанной работы разработано мобильное приложение, сочетающее жанровые механики survival-horror и RPG с ретро-графикой.

Реализованы:

– адаптивный интерфейс для сенсорных устройств;

– оптимизация производительности (стабильные 30 FPS на слабых Android-устройствах);

– интеграция обучающей системы.

Проект демонстрирует возможность создания качественных мобильных игр с ограниченными ресурсами, а также потенциал геймификации образовательной среды через знакомые локации.

Содержание

Введение ... 8
1 Мобильное приложение «Escape Building-9» ... 10
1.1 Анализ предметной области ... 10
1.1.1 Актуальность разработки ... 12
1.2 Анализ существующих аналогов ... 17
1.2.1 Five Nights at Freddy’s ... 17
1.2.2 Pixel Dungeon ... 19
1.2.3 Eyes ... 21
1.3 Постановка задачи ... 23
1.4 Выбор инструментов разработки ... 25
1.4.1 Выбор игрового движка ... 25
1.4.2 Выбор редактора 2D-графики и текстур ... 28
1.4.3 Выбор редактора 3D-графики ... 29
1.4.4 Выбор плагинов для движка ... 29
1.5 Проектирование приложения ... 30
1.5.1 Общий гейм-дизайн ... 31
1.5.2 Система управления персонажем ... 32
1.5.3 Система инвентаря ... 33
1.5.4 Система сохранений ... 36
1.6 Разработка приложения ... 38
1.6.1 Создание стилизованных текстур и 3D-моделей ... 38
1.6.2 Реализация внутриигрового пользовательского интерфейса ... 40
1.6.3 Разработка системы управления персонажем ... 43
1.6.4 Разработка системы управления инвентарём ... 49
1.6.5 Разработка системы сохранений ... 51
1.6.6 Тестирование и отладка ... 54
1.7 Описание приложения ... 56
1.7.1 Руководство пользователя ... 56
2 Охрана труда ... 59

2.1 Эргономические требования 59
2.2 Временные ограничения и профилактика переутомления 60
2.3 Психологическая безопасность 60
2.4 Требования к разработчикам 60
2.5 Правовое соответствие 61
Заключение 62
Список использованных источников 64
Приложение А – UML Диаграммы 65
Приложение Б – Листинг программы 73
Приложение С – Презентация 108

Перечень сокращений и обозначений

API (Application Programming Interface) – интерфейс программирования приложений, набор инструментов для взаимодействия между компонентами программного обеспечения.

FPS (Frames Per Second) – количество кадров в секунду; показатель производительности игры.

HUD (Heads-Up Display) – интерфейс, отображающий информацию поверх игрового мира (здоровье, инвентарь и т.д.).

LOD (Level of Detail) – уровень детализации 3D-моделей, адаптирующийся к расстоянию для оптимизации производительности.

NPC (Non-Playable Character) – неигровой персонаж, управляемый искусственным интеллектом.

RPG (Role-Playing Game) – жанр ролевой игры, где игрок управляет развитием персонажа.

UI (User Interface) – пользовательский интерфейс.

UX (User Experience) – пользовательский опыт, включающий удобство взаимодействия с интерфейсом.

UV-развёртка – процесс проекции 2D-текстуры на 3D-модель.

ГОСТ – региональный стандарт, принятый Межгосударственным советом по стандартизации, метрологии и сертификации Содружества Независимых Государств.

Введение

Современную реальность невозможно представить без технологий, а ещё сложнее без информации. Она преследует нас везде, что даже не всегда можно остаться на едине со своими собственными мыслями. Новости и реклама, полезные статьи и видео, музыка и подкасты, видеоигры и приложения – всего этого нам теперь хватит не на один десяток лет. В особенности это касается сферы мобильных игр. Рынок ощущает перенасыщение некачественными и непрезентабельными продуктами, повторяющими друг друга, поэтому человеку, который ценит своё время, необходимо тщательно их подбирать.

Разработчик, который по-настоящему вовлечён в успех своего проекта, должен осознавать вышеуказанную проблему и заботиться о продукте на благо конечного пользователя. Однако большинство компаний и отдельных разработчиков мобильных игр зачастую волнует только финансовая сторона вопроса, и они не боятся значительно экономить на игровой графике, музыке, визуальном стиле, они пытаются вставить как можно больше рекламы и внутриигровых покупок. Игровой процесс нередко выполнен в казуальном жанре, то есть не требующем погружения и завязанном на примитивных механиках. Более того, разработчики таких игр привыкли пользоваться психологическими уловками различного рода, заставляя пользователя бессмысленно проводить в их игре как можно больше времени. В приоритете обычно находится скорость разработки и количество выпускаемых продуктов, что плохо сказывается на кодовой базе, то есть напрямую влияет на стабильность работы приложения и игровой опыт.

Рынок мобильных игр – один из самых крупных в сфере программного обеспечения. Он охватывает внушительную демографию от самых малых детей до взрослых разных социальных групп и профессий, поэтому спрос на уникальные игры под конкретную целевую аудиторию очень велик. Но существует и особая довольно крупная группа людей: поклонники компьютерных игр – так называемые геймеры, которые предпочитают некаузальные жанры и недолюбливают мобильные игры из-за вышеизложенных проблем.

Таким образом, возникает запрос на реализацию качественного решения, способного заинтересовать как опытного геймера, так и любителя казуальных развлечений. Нужно разработать игру, обеспечивающую погружение, обладающую интересным визуальным стилем и сеттингом, продвинутыми механиками, но при этом сохраняющую простоту и удобство, которые обеспечат лёгкий старт непринуждённым игрокам. Мобильное приложение, разработанное в рамках выпускной работы, ориентировано не только на широкую аудиторию, но и на студентов ПГУ, которые смогут пережить опыт потерянности в уже знакомых локациях 9-го корпуса. Это подчёркивает его актуальность как инструмента, сочетающего игровые механики с элементами образовательной среды.

Целями выпускной работы являются:

– исследование предметной области и анализ существующих решений;

– проектирование игрового процесса, подходящего выбранному жанру, и адаптация механик под условности мобильного формата;

– разработка игровых ассетов в определённой стилистике для создания особого нарратива;

– реализовать игровые локации на основе архитектуры 9-го корпуса ПГУ, адаптированные под стилистику ретро-хоррора;

– разработка мобильного игрового приложения «Escape Building-9»;

– реализация внутриигрового пользовательского интерфейса;

– тестирование разработанного решения.

Полученное в результате разработки игровое приложение будет готово к выпуску на различных площадках, поддерживаемых ОС Android.

1 Мобильное приложение «Escape Building-9»

1.1 Анализ предметной области

Для успешного начала проектирования мобильной игры стоит определиться с базовыми направлениями развития идеи. В общем виде игру можно охарактеризовать игровым жанром и стилистикой.

Жанр определяет все геймплейные составляющие, то есть весь функционал, который будет доступен игроку во время игрового процесса, и который касается только этого процесса, не затрагивая технический аспект приложения (меню настроек, главное меню). Жанры часто пересекаются, образуя гибридные формы. Так, RPG может совмещать элементы стратегии с развитием персонажа и управлением ресурсами, а головоломка – интегрировать нарративные элементы, превращаясь в интерактивную историю. Важно учитывать, что каждый жанр должен действовать вокруг своих определённых правил: экшены и аркады фокусируются на скорости и реакции, симуляторы опираются на реализм процессов, а казуальные игры акцентируются на простоте и коротких сессиях.

Стилистика формирует визуальное и эмоциональное восприятие игры. Она охватывает художественное оформление, цветовую палитру, анимацию, звуковое сопровождение и общую атмосферу, создавая уникальное настроение проекта. Например, игры в жанре платформер могут быть выполнены с пиксельной графикой и чиптюн-музыкой для создания атмосферы игр былой эпохи, или в современном мультяшном дизайне с яркими красками и динамичными эффектами для более широкого спектра игроков. Стилистика также влияет на целевую аудиторию. Детские проекты часто выбирают яркие, добрые визуальные решения с упрощенной анимацией, тогда как игры для взрослых могут экспериментировать с абстракцией, сюрреализмом или кинематографичностью. Минимализм с геометрическими формами и лаконичным интерфейсом подходит для медитативных или логических игр, а детализированный фэнтезийный мир с богатым лором – для масштабных RPG. Звуковое оформление играет не меньшую

роль: электронная музыка создает футуристическую атмосферу, а оркестровые композиции – эпический размах.

При этом стилистика работает в тандеме с жанром, усиливая его идею. Например, хоррор-игра с механиками выживания может использовать мрачную рисовку в стиле низкополигональной 3D-графики, приглушенные тона и тревожный саундтрек, чтобы вызвать чувство напряжения. Важно сохранять согласованность между жанром и стилистикой. Даже оригинальная идея рискует провалиться, если, например, реалистичная графика конфликтует с упрощенным геймплеем, или если игровой процесс требует концентрации, но интерфейс перегружен элементами. Удачное сочетание жанра и стиля рождает целостный опыт, где механики, визуал и звук работают как единый организм, удерживая внимание игрока и передавая задуманные эмоции. Эксперименты возможны, но они должны быть осознанными: смешение аниме-стилистики с хардкорными стратегическими элементами или ретро-эстетики с современными механиками может стать положительной особенностью, если баланс соблюден. В конечном итоге, именно гармония жанра и стилистики превращает набор функций в резонирующий игровой опыт.

Одним из ключевых аспектов успешной мобильной игры является адаптация управления и взаимодействия с интерфейсом. Мобильные платформы обладают уникальными особенностями, игнорирование которых приводит к ухудшению пользовательского опыта, снижению вовлеченности и, как следствие, потере аудитории. Мобильные устройства, в отличие от ПК или консолей, имеют ограниченное пространство для отображения элементов интерфейса. Попытка перенести на компактный дисплей сложные схемы управления, разработанные для мыши и клавиатуры, делает элементы слишком мелкими и неудобными для касания. Например, кнопки действий, которые на ПК легко активируются клавишами, на смартфоне требуют крупных зон нажатия, чтобы избежать ошибок. Кроме того, избыток информации на экране (например, инвентарь, карты, статусы) может визуально перегружать игрока, что особенно критично в условиях мобильного геймплея, где важна мгновенная читаемость. На мобильных

устройствах основным инструментом взаимодействия становятся пальцы, а не точный курсор мыши или джойстик. Это накладывает ограничения на количество и расположение элементов. Игнорирование этих принципов приводит к типичным ошибкам: нечитаемому тексту, случайным нажатиям, усталости рук и, как итог, низким оценкам в магазинах приложений. Например, ранние мобильные порты PC-игр критиковали за неудобный интерфейс, что заставило разработчиков полностью пересмотреть подход к сенсорному вводу. В то же время проекты, изначально созданные для мобильных платформ, показывают, как тщательная адаптация управления и интерфейса превращает ограничения устройства в преимущества, создавая интуитивный и приятный опыт.

Адаптация – это переосмысление взаимодействия игрока с продуктом. Она позволяет превратить мобильное устройство из «урезанной версии ПК» в самостоятельную платформу с уникальными возможностями, что в итоге определяет лояльность аудитории и коммерческий успех проекта.

1.1.1 Актуальность разработки

На данный момент рынок мобильных игр представляет собой противоречивое пространство. С одной стороны, он демонстрирует невероятный рост: по данным MordorIntelligence, в 2024 году объём рынка превысил $100 млрд, а число активных игроков достигло 2.5 млрд человек. С другой – большая часть этих игр создаётся по шаблонам, где примитивный геймплей служит лишь оболочкой для агрессивной монетизации. Казуальные головоломки, бесконечные раннеры и гиперказуальные проекты с минималистичной графикой заполонили площадки, оставляя мало места для игр с глубоким нарративом или сложными механиками. Это приводит к разочарованию аудитории, которая ищет не просто «убийцу времени», а полноценный игровой опыт, способный удивить и вовлечь.

На этом фоне выделяется запрос на проекты, нацеленные на оригинальность и качество. Игроки устали от однотипных механик и навязчивой рекламы. Они хотят видеть истории, которые заставляют задуматься, визуальные стили, вызывающие эмоции, и геймплей, требующий

стратегии, вместо бесконечных повторов. Особенно остро этот запрос ощущается среди поклонников PC/консольных игр, которые нередко избегают мобильных платформ из-за их репутации «второсортного» формата. Однако именно мобильные устройства, благодаря своей доступности, могут стать идеальной площадкой для экспериментов – при условии, что разработчик готов вкладываться в уникальность, а не в шаблонность.

Также стоит обратить внимание на выбор жанра. Современная игровая сфера демонстрирует устойчивый интерес к гибридным жанрам, например, сочетающим элементы survival-horror, RPG и квестов. Квестовые механики делают упор на решение головоломок и исследование окружения. В свою очередь, RPG-элементы (прокачка навыков, инвентарь, ветвящийся сюжет) добавляют глубину и реиграбельность. Survival-horror, как направление, фокусируется на создании напряжения через ограниченные ресурсы, атмосферу опасности и необходимость выживания. Однако классические представители жанра ориентированы на консольные и ПК-платформы, что оставляет нишу для мобильных проектов с адаптированным геймплеем.

Касаясь стилистики и нарратива, можно рассмотреть набирающий популярность стиль ретро-графики PlayStation 1 (рисунок 1.1). Технические ограничения PS1 – низкое разрешение текстур, угловатые модели, отсутствие сглаживания – переосмыслены как художественные приёмы. Например, «дрожащие» тени и примитивная анимация создают эффект «цифровой неустойчивости», усиливая ощущение сюрреализма, что играет на руку жанру survival-horror. Такая графика контрастирует с гиперреализмом (рисунок 1.3) современных проектов крупных студий, но при этом требует меньше ресурсов, что критично для мобильных устройств. Существуют и другие варианты стилизации, такие как пиксельарт (рисунок 1.2) и минималистичный 3D стиль (рисунок 1.4). Сравнение с другими стилями подчёркивает её преимущества (таблица 1).

Рисунок 1.1 – Стилистика ретро-графики PS1 (Crash Bandicoot)

Таблица 1 – Недостатки стилистик в игровой графике в сравнении со стилистикой ретро-графики PS1

Стилистика	Недостаток в сравнении со стилистикой ретро-графики PS1
Пиксель арт	Ограничивает возможности 3D-исследования локации.
Гиперрреализм	Требует высокопроизводительного стройства, сужая целевую аудиторию.
Современная низкополигональная графика	Часто воспринимается как «милая», что не подходит для создания ряда настроений.

Рисунок 2.2 – Стилистика пиксель арт (Stardew Valley)

Рисунок 1.3 – Стилистика гиперреализма (Red Dead Redemption 2)

Рисунок 1.4 – Стилистика современной низкополигональной графики (For the King)

Стиль PS1 балансирует между узнаваемостью и гибкостью. Кроме того, такая графика стимулирует воображение игрока, заставляя домысливать детали, что усиливает погружение. Для мобильных платформ подход с такой стилистикой особенно актуален. Большой процент игр с «тяжёлой» графикой на Android сталкиваются с падением FPS на устройствах среднего класса. Стилизация под PS1 решает эту проблему: даже на слабых смартфонах игра может сохранять стабильные 30 кадров/с. Говоря в общем, ретро-графика становится не просто эстетическим выбором, а прагматичным решением, объединяющим художественное виденье, техническую оптимизацию и эмоциональный резонанс.

Итак, на рынке определённо существует перспективная ниша для качественных игровых решений со смешением жанров и в оригинальной стилистике, например, ретро-графики PS1.

Особое внимание уделено аудитории студентов ПГУ: локации игры воссоздают реальные пространства 9-го корпуса университета, что добавляет уникальный опыт взаимодействия с привычной средой в игровых условиях. Это не только усиливает эмоциональное погружение, но и создаёт дополнительную

связь между игрой и её пользователями, превращая знакомые аудитории места в пространство для исследования и преодоления страхов.

1.2 Анализ существующих аналогов

Несмотря на серьёзность проблемы заполнения рынка мобильных игр казуальными проектами, всё-таки существуют интересные решения. Из них можно выделить:

–Five Nights at Freddy’s;

–Pixel Dungeon;

–Eyes.

Проекты подходят под критерии качественных стилизованных игр в интересных жанрах и послужат хорошим материалом для дальнейшего анализа.

1.2.1 Five Nights at Freddy’s

Игра относится к жанру survival-horror, делая акцент на создании напряжённой атмосферы и управлении ограниченными ресурсами. В проекте используются статичные камеры и звуковые подсказки для усиления тревожности: в Five Nights at Freddy’s игрок следит за аниматрониками через систему видеонаблюдения (рисунок 1.5). Однако игра фокусируется на чистом выживании в замкнутом пространстве (рисунок 1.6), исключая элементы исследования или сюжетной вариативности. Это ограничивает игровой опыт линейностью, где действия сводятся к реактивным решениям, а не стратегическому планированию. Кроме того, мрачная тональность Five Nights at Freddy’s, сужает аудиторию, делая игру менее доступной для тех, кто избегает «чистого» хоррора.

Рисунок 1.5 – Экран видеонаблюдения в Five Nights at Freddy’s

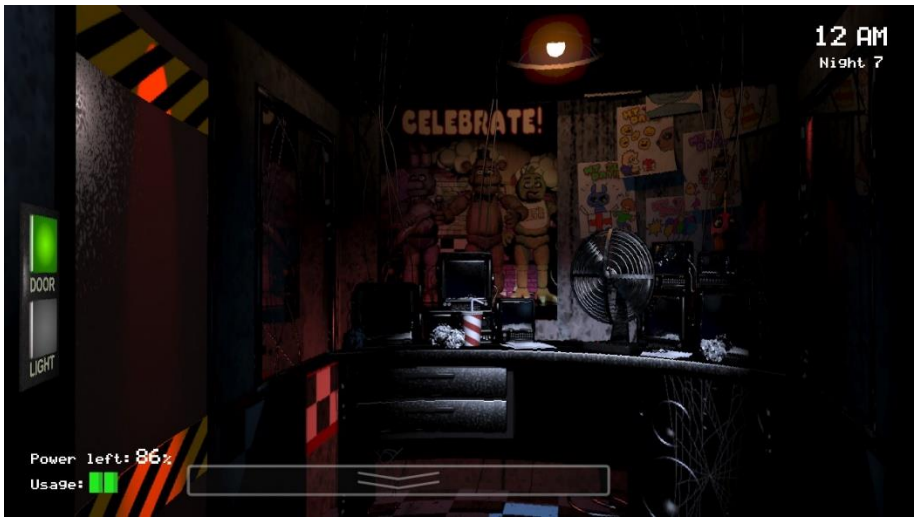


Рисунок 1.6 – Основной игровой экран в Five Nights at Freddy’s

Преимущество Five Nights at Freddy’s заключается в способности вызывать интенсивный эмоциональный отклик, используя минимальный набор механик. В игре нельзя передвигаться персонажем, это создаёт дополнительное напряженние. Атмосфера, воссозданная всеми этими ограничениями, играет на руку жанру и генерирует уникальный опыт для игрока. Но есть и недостаток. Отсутствует прогрессия с RPG-элементами, которые могли бы добавить глубину и реиграбельность. Также к недочётам можно отнести отсутствие меню настроек (рисунок 1.7).

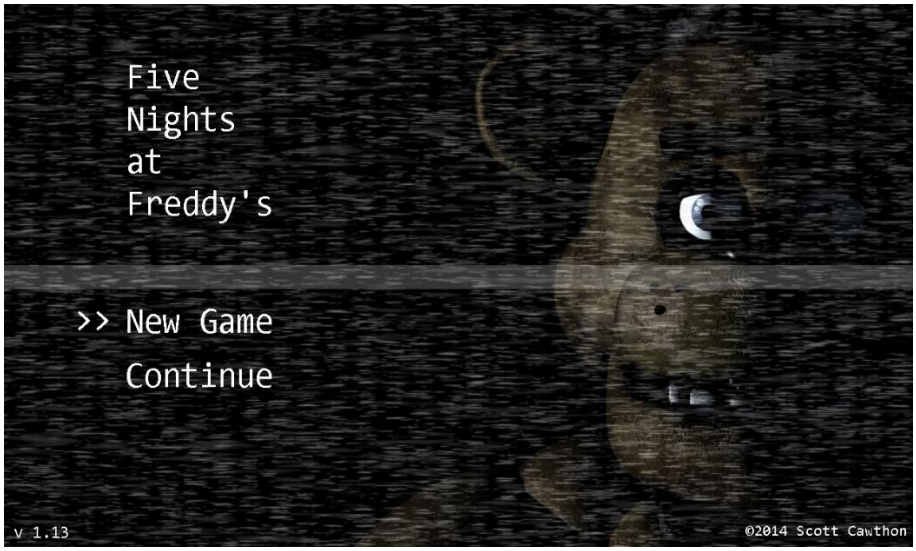


Рисунок 1.7 – Экран главного меню в Five Nights at Freddy’s

Five Nights at Freddy’s отлично справляется с задачей объединения казуальных игровых механик и красочной стилизованной реализацией, свойственной серьёзным проектам.

1.2.2 Pixel Dungeon

Игра представляет собой пошаговую ролевую стратегию, с возможностью полноценной настройки своего приключения (рисунок 1.9). Проект развился до полноценного центра огромного сообщества с рейтингом и новостями (рисунок 1.8). Игровой процесс специально сделан для мобильных платформ, адаптируя управление свайпами и делая элементы интерфейса удобными для маленьких сенсорных дисплеев.

Рисунок 1.8 – Экран главного меню в Pixel Dungeon

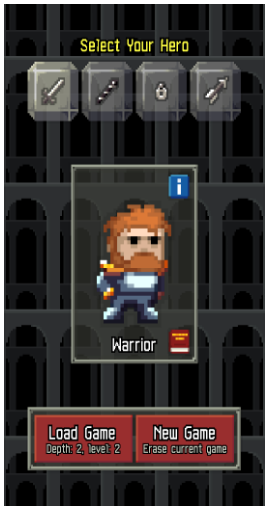


Рисунок 1.9 – Экран начала приключения в Pixel Dungeon

Pixel Dungeon, будучи roguelike-RPG делает акцент на управлении ресурсами и стратегическом мышлении. Игра требует от игрока анализа окружения, распределения ограниченных предметов и принятия решений, влияющих на исход. Например, в Pixel Dungeon выбор между использованием зелья для лечения или атаки (рисунок 1.10). К главным преимуществам игры можно отнести высокую вариативность действий. В проекте реализовано много продвинутых механик, таких как: полноценный инвентарь с различными типами предметов, обилие врагов и боссов с уникальным поведением, сложная система параметров игрока – всё это хорошо способствует погружению в игровой процесс.

Рисунок 1.10 – Основной игровой экран в Pixel Dungeon

Однако Pixel Dungeon ориентирован на хардкорную аудиторию: процедурная генерация уровней и перманентная смерть создают высокий порог вхождения, что может отпугнуть казуальных игроков. Кроме того, пиксельная графика, хотя и стильная, ограничивает визуальную выразительность, делая окружение менее запоминающимся.

Pixel Dungeon, несмотря на высокий порог вхождения, всё же является хорошим примером адаптации сложных игровых механик и систем в компактный мобильный интерфейс.

1.2.3 Eyes

Игра относится к жанру survival-horror с акцентом на исследование мрачных локаций и управление ресурсами. Проект использует эмоциональное давление через ограниченную видимость и звуковые подсказки. В Eyes игрок полагается на «глаза», которые он собирает со стен (рисунок 1.12). С помощью «глаз» у игрока появляется возможность смотреть от лица монстра (рисунок 1.13), от которого игрок должен скрываться. Механика сбора предметов (рисунок 1.11) стимулирует игрока к тщательному изучению окружения.

Рисунок 1.11 – Процесс сбора ресурсов в виде мешочков в Eyes

Стоит отметить хорошую адаптацию управления под сенсорный экран, кнопки и экранный стик расположены удобно, а реализация взаимодействия с окружением с помощью прикосновений к внутриигровым объектам способствует погружению и организует солидный игровой опыт.

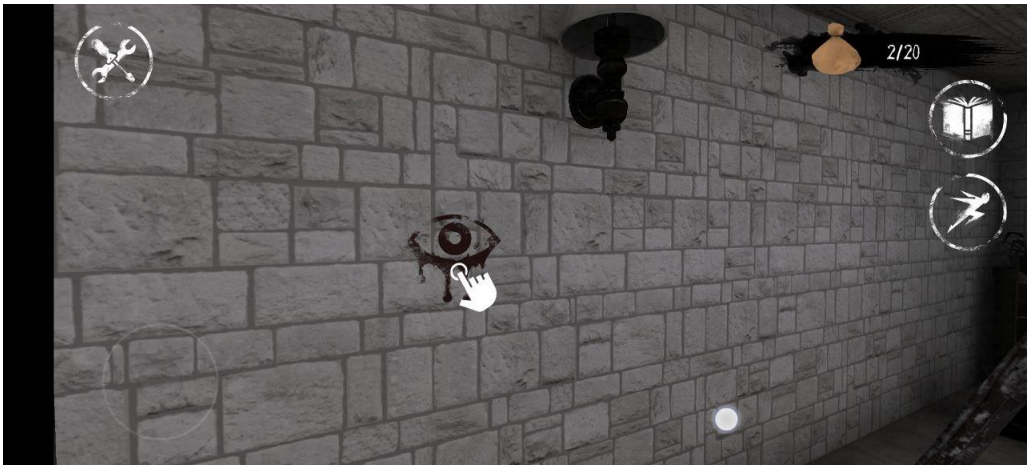


Рисунок 1.12 – Процесс сбора ресурсов в виде «глаз» в Eyes

Eyes успешно создаёт напряжение через минималистичные средства: отсутствие боевой системы вынуждает игрока полагаться на скрытность и планирование маршрутов. Звуковой дизайн, включая шаги преследователей и скрипы дверей, усиливает погружение. Кроме того, игра оптимизирована для мобильных устройств, демонстрируя стабильную работу даже на слабых смартфонах, что критически важно для широкой аудитории.

Рисунок 1.13 – Вид от лица монстра в Eyes

Основной минус Eyes – линейность и отсутствие нарративной глубины. Игрок следует по заранее заданному пути, собирая предметы и избегая врагов, но его действия почти не влияют на сюжет или концовку. Это ограничивает реиграбельность и эмоциональную вовлечённость.

Кроме того, визуальный стиль игры, хотя и атмосферный, но лишён уникальности: тёмные коридоры и устаревшая графика в стиле гиперреализма не выделяются на фоне других мобильных приложений жанра survival-horror.

1.3 Постановка задачи

Требуется разработать мобильное приложение «Escape Building-9», которое будет следовать целям:

– разработать систему управления инвентарём, требующую стратегического планирования ресурсов;

– реализовать интерактивное взаимодействие с окружением (открытие дверей, использование предметов, решение головоломок);

– интегрировать диалоговую систему с ветвящимся сюжетом, влияющим на концовку;

– создать низкополигональные 3D-модели в стиле PS1, сохраняющие атмосферу ретро-хоррора;

– подобрать звуковое сопровождение, включая эмбиентные шумы, шаги врагов и тематический саундтрек;

– адаптировать управление для сенсорных экранов: контекстные кнопки взаимодействия, интуитивный интерфейс инвентаря;

– внедрить систему сохранения прогресса с шифрованием данных для защиты от несанкционированного доступа;

– оптимизировать производительность под слабые устройства, обеспечив стабильные 30 FPS.

На рисунке 1.14 изображена диаграмма вариантов использования приложения при запуске.

Разрабатываемое приложение должно соответствовать следующим техническим требованиям:

– операционная система: Android не ниже версии 7;

– оперативная память: от 1ГБ;

– центральный процессор: от 4 ядер;

– графический процессор: с поддержкой API Vulkan.

Рисунок 1.14 – Диаграмма вариантов использования приложения

1.4 Выбор инструментов разработки

Как и для любого другого приложения, разработчику игры требуется основательно подойти к набору средств, которыми он воспользуется. Процесс разработки такого проекта обычно является довольно комплексным предметом, который может включать работу из разных сфер. Поэтому каждому аспекту нужно уделить не меньше внимания, чем остальным.

Игры состоят из технической части и художественной, требуется определиться с базовыми инструментами, которые в полной мере обеспечат эти базовые элементы.

1.4.1 Выбор игрового движка

В настоящее время, сфера разработки игр большими скачками прогрессировала вперёд. Теперь для создания полноценной игры не нужно: разрабатывать дополнительные «прослойки» для работы с графическими API. Благодаря этому больше не нужно так тщательно подбирать и комбинировать различные IDE для управления кодом, а также следить за ресурсами, которые использовались в проекте. Все эти трудности создавали неприятности для разработчиков былых времён. Но сейчас существует ряд универсальных решений (игровых движков), которые предоставят удобства организации для такого рода проектов. Многие игровые движки могут полноценно заменить среду разработки, и однако могут ограничивать в выборе отдельных инструментов, например языка программирования. Поэтому к выбору движка стоит подойти с ответственностью. Из популярных развитых решений можно выделить Unity, Unreal Engine и Godot Engine.

Для реализации игрового приложения «Escape Building-9» в качестве основного инструмента выбран Godot Engine 4.4. Движок является открытым и бесплатным ПО, в отличие от Unity и Unreal Engine. Это делает Godot оптимальным выбором для инди-разработки, то есть для проекта с единственным разработчиком без бюджета.

Лицензия MIT обеспечивает полную свободу использования, модификации и распространения движка без юридических или финансовых ограничений. В

случае для проекта, ориентированного на образовательные цели и локальную аудиторию, где коммерческие движки с подпиской или роялти могут стать препятствием, эта лицензия позволит избежать неприятностей. Открытый исходный код хоть и является очень специфическим достоинством, доступным лишь профессионалам, но достаточно влиятельным на развитие движка как инструмента. Открытость позволяет движку развиваться и улучшаться независимо от больших компаний и их вложений.

Кроссплатформенность Godot обеспечивает не только разработку под Android, но и потенциальное портирование на другие системы, например ПК. Так что перспектива расширения аудитории за счёт пользователей разных платформ внушительна.

Для мобильной версии важна возможность нативной работы с сенсорным вводом. Мультитач, жесты и виртуальные джойстики - Godot предоставляет все эти инструменты, что облегчает адаптацию на стороне разработчика. Такие возможности обеспечат качественный функционал и для конечного пользователя.

Godot, как и многие современные игровые движки, разрешает и 2D, и 3D-разработку. Для 3D проекта с низкополигональными моделями и атмосферным дизайном, необходима поддержка современных графических технологий. Встроенный рендерер движка, который и включает современные технологии на базе Vulkan, позволяет работать с динамическим освещением, постобработкой и частицами. При этом совместимость со слабыми устройствами может обеспечить отдельный режим, «Compatability». Он позволяет собирать итоговое приложение для старых или слабых устройств, которые, может, не способны работать с API Vulkan. Инструменты для 2D, такие как спрайтовые анимации и тайлмапы, используются для создания интерфейсов и элементов HUD, сохраняя единый рабочий процесс внутри движка.

Так же универсальным и мощным элементом движка является узел. В Godot Engine узел представляет собой фундаментальный строительный блок, на котором основывается вся структура проекта. Каждый узел – это объект, обладающий определёнными свойствами, методами и поведением, и он существует в рамках

древовидной иерархии, называемой сценой. Узлы могут быть как самостоятельными элементами, так и частью более сложных композиций, где родительские узлы управляют дочерними, формируя логические группы. Эта иерархия позволяет организовывать игровые объекты, интерфейсы, системы частиц, звуки и другие элементы в модульную структуру, где изменения в родительском узле автоматически влияют на его потомков. Одной из ключевых возможностей узлов является их наследование от базового класса Node. Каждый узел можно настраивать через инспектор свойств в редакторе Godot, где изменяются числовые значения, текстуры, скрипты или другие атрибуты, что делает процесс визуальным и интуитивным. Узлы могут быть объединены в группы, что позволяет управлять множеством объектов одновременно. Присоединение скриптов к узлам расширяет их функционал до бесконечности. Используя языки GDScript, C# или VisualScript, разработчик может добавлять пользовательскую логику, обрабатывать данные, управлять состоянием игры или взаимодействовать с другими узлами. Скрипты наследуют все свойства и методы родительского узла. Узлы также участвуют в цикле жизненных методов движка, таких как _ready() (вызывается при инициализации), _process(delta) (обновление каждый кадр) или _physics_process(delta) (обновление с фиксированным шагом для физики), это делает их центральными точками для программирования игровой логики. Важной чертой узлов является их модульность и переиспользуемость. Сцену, состоящую из множества узлов, можно сохранить как отдельный ресурс (.tscn или .scn) и инстанциировать её в других сценах, что способствует организации проекта и сокращению дублирования кода. Узлы могут динамически создаваться, удаляться или модифицироваться во время выполнения игры через код, позволяя генерировать уровни, спавнить противников или изменять окружение в реальном времени. Всё это делает узлы в Godot Engine универсальным инструментом, который сочетает простоту визуального редактирования с глубиной программируемого поведения, обеспечивая разработчику полный контроль над каждым аспектом игры.

Активное сообщество и документация Godot – ключевой аргумет в пользу выбора данного решения. Форумы и GitHub-репозитории обеспечивают доступ к лучшим практикам, например, по снижению потребления памяти или настройке LOD-моделей. Godot поддерживает адаптивное разрешение, динамическую загрузку текстур и упрощённые физические вычисления, это позволяет достичь стабильной частоты кадров. Модульная архитектура движка даёт возможность отключить неиспользуемые компоненты, такие как 3D-физика или продвинутые эффекты.

Интеграция инструментов быстрой итерации, редактора сцен, инспектора объектов и отладчика ускоряет процесс тестирования. Например, изменения в расположении объектов на сцене или настройки освещения можно сразу увидеть в предпросмотре, без необходимости пересборки проекта. Встроенный профилировщик помогает отслеживать утечки памяти или нагрузку на устройство.

1.4.2 Выбор редактора 2D-графики и текстур

Для работы над текстурами и элементами интерфейса проекта выбран Adobe Photoshop, так как он обеспечивает необходимую гибкость для стилизации. Инструмент позволяет создавать низкодетализированные текстуры (256x256 пикселей и ниже) с использованием определённых шаблонов экспорта в «низкокачественный» формат PNG, имитируя технические ограничения 90-х. Режим индексированных цветов сокращает палитру до 256 оттенков, что воссоздает характерные для PS1 цветовые переходы в виде дизеринга.

Photoshop также применяется для разработки интерфейсов: инструменты векторной графики и стилей слоя помогают создавать чёткие кнопки управления и элементы HUD, сохраняя единый визуальный стиль. Несмотря на наличие бесплатных аналогов, выбор обусловлен стабильностью, плагинами и возможностью тонкой настройки. Это позволяет не только работать с текстурами, но и интегрировать их в Godot без потери качества, сохраняя атмосферу ретро-хоррора через ручную проработку деталей – царапин, потёртостей и неравномерного освещения.

1.4.3 Выбор редактора 3D-графики

Для создания 3D-моделей и анимаций в проекте используется Blender – бесплатный и открытый инструмент, который сочетает мощный функционал с низким порогом входа. Его выбор обусловлен необходимостью работы с низкополигональной графикой, где важны простота форм, оптимизация сеток и ручная настройка материалов. Blender предоставляет полный цикл инструментов: от моделирования и скульптинга до UV-развёртки и риггинга, что позволяет создавать статичные объекты и анимированных персонажей без переключения между программами.

Blender также обеспечивает бесшовную интеграцию с Godot Engine через экспорт моделей в форматы FBX и glTF, поддерживающие текстуры, скелетную анимацию и LOD-группы. Открытая экосистема движка позволяет использовать аддоны для автоматизации задач – например, Batch Export для массовой конвертации ассетов. Важным аргументом стала активная поддержка сообщества: готовые уроки по созданию низкополигональных моделей, шейдеров с эффектом «дрожания» вертексов и оптимизации сцен под мобильные устройства ускорили разработку. В отличие от платных аналогов, Blender не требует лицензионных отчислений, что отлично подходит для студенческого проекта, сохраняя при этом профессиональный уровень качества.

1.4.4 Выбор плагинов для движка

Разработка игрового проекта требует не только базовых инструментов, но и гибкости в решении специфических задач, которые могут выходить за рамки возможностей движка «из коробки». Godot Engine, несмотря на свою универсальность, часто дополняется сторонними плагинами – модулями, расширяющими функционал и ускоряющими реализацию сложных систем. В рамках «Escape Building-9» интеграция плагинов позволила сосредоточиться на ключевых механиках, делегируя рутинные или узкоспециализированные задачи проверенным инструментам сообщества.

Одной из таких задач стала реализация диалоговой системы с ветвящимся сюжетом – элемента, напрямую влияющего на игровой опыт и нарративную

глубину. Для её воплощения потребовался инструмент, сочетающий простоту интеграции с поддержкой модульности, что привело к анализу доступных решений. В итоге выбор остановился на плагине Dialogue Manager, который не только соответствует техническим требованиям проекта, но и органично встраивается в рабочий процесс, дополняя выбранную экосистему разработки. Плагин позволяет управлять внутриигровыми диалоговыми окнами, как и любой другой системой движка, например, деревом анимаций. С ним появляется возможность редактировать реплики для NPC в соответствующих им файлах формата JSON.

1.5 Проектирование приложения

Игра как приложение имеет ряд особенностей, отличающих её от классического понимания прикладной программы. Приложение выполняет задачи пользователя, тогда как игра симулирует различные процессы, передавая игровой опыт и эмоции. Правдоподобность этих процессов обеспечивают так называемые системы игровых механик. В совокупности эти системы и определяют архитектуру игрового приложения. Набор игровых механик и их систем образуют структуру игры, как развлечения, её общий гейм-дизайн.

С точки зрения технической стороны вопроса проект в Godot Engine состоит из набора главных узлов, которые представляют собой сцены. Сцены содержат все графические и абстрактные узлы объектов, образуя иерархию. В разрабатываемом приложении основным состояниям игры соответствуют сцены (рисунок 1.15). Узел Global.gd присутствует на всех сценах и отвечает за хранение настроек приложения. К нему может обращаться любой узел сцены, например для регулирования звука. Основополагающими являются сцены главного меню, меню сохранений и сцена игрового мира. В иерархии узлов они находятся на одном уровне с Global.gd, тогда как инвентарь и другие меню игрока помещаются внутри узла игрового мира, то есть находятся в памяти во время игрового процесса.

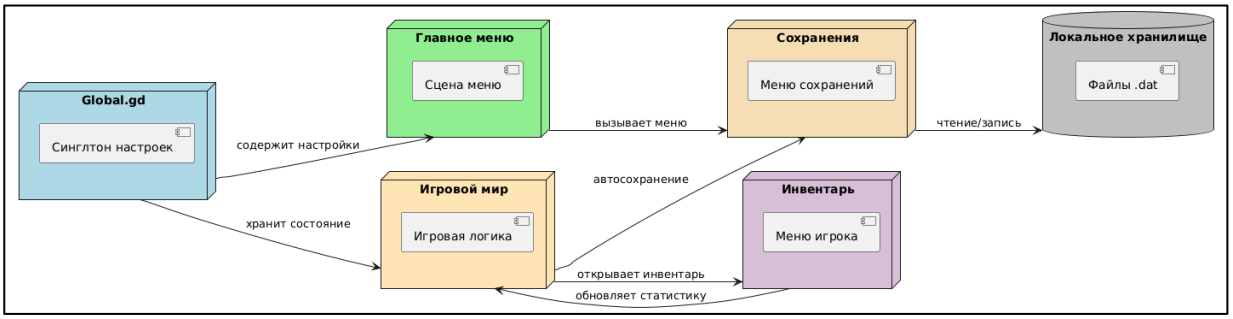


Рисунок 1.15 – Архитектура приложения

1.5.1 Общий гейм-дизайн

Игра «Escape Building-9» должна быть реализована в жанре survival-horror с элементами RPG, где ключевой акцент сделан на создание атмосферы напряжённого исследования, стратегического управления ресурсами и нелинейного взаимодействия с игровым миром. Жанровый аспект играет важнейшую роль в общем гейм-дизайне, так как определяет область соответствующих игровых механик.

Стоит четко задать условия, в которых игрок будет осуществлять взаимодействие при помощи игровых механик, то есть игровую локацию. Жанр survival-horror предполагает локации с ограниченной видимостью, для создания напряжения, поэтому большую часть игры основным источником света будет область вокруг игрока. Лабиринтообразная структура 9-го корпуса должна быть спроектирована для стимуляции тактического мышления и чувства дезориентации. Локации сочетают линейные «узкие» зоны (коридоры, аудитории) с полуоткрытыми пространствами (холлы, подвалы), где игроку приходится запоминать маршруты, искать скрытые проходы и возвращаться к ранее посещённым областям после получения новых предметов.

Элементом RPG, в свою очередь, служит продвинутая система инвентаря с различными типами предметов в комплексе с системой диалогов и обширный набор персонажей для взаимодействия (NPC). С персонажами можно обмениваться предметами или игровой валютой, если конкретный NPC позволит это сделать.

Так же стоит отметить, что каждому игроку будет принципиально иметь возможность сохранить свой прогресс. Для погружения и для нерушимости

стилистической целостности данная система должна быть аккуратно внедрена в игровой опыт: переход между меню сохранений и игрой следует реализовать бесшовно.

1.5.2 Система управления персонажем

Управление игровым персонажем в игре реализуется через экранные элементы интерфейса, адаптированные под тактильное взаимодействие. Основой навигации служат четыре кнопки с направлением, которые обеспечивают плавное перемещение в трёхмерном пространстве. Скорость передвижения фиксирована, но динамически корректируется при активации режима приседания – отдельной кнопкой в левой зоне интерфейса. В этом состоянии скорость снижается на 35%, а коллизия персонажа уменьшается, позволяя игроку избегать обнаружения или проходить через узкие участки среды. Вертикальная мобильность ограничена: прыжок исключён из механик, что акцентирует фокус на горизонтальном планировании маршрутов и анализе препятствий.

Контекстное взаимодействие с объектами и NPC осуществляется через универсальную кнопку «Взаимодействие», расположенную левее от середины экрана. Её функциональность варьируется в зависимости от цели: открытие контекстных меню, активация механизмов или открытие двери.

Диалоговая система активируется либо произвольно в ключевые моменты сюжета, либо на экранную кнопку «Говорить» правее от центра экрана.

Доступ к инвентарю предоставляет отдельная кнопка, активация которой вызывает меню без прерывания игрового процесса.

Система управления проектируется как минималистичная, но функциональная. Отсутствие избыточных элементов (например, прыжка) и фиксированные параметры скорости направлены на упрощение освоения механик, сохраняя при этом тактическую глубину за счёт анализа окружения и стратегического использования инвентаря. Интеграция RPG-элементов, таких как диалоги и торговля, обеспечивает нелинейность, не перегружая интерфейс.

На рисунке 1.16 представлена диаграмма вариантов использования системы, показывающая все возможности данного элемента.

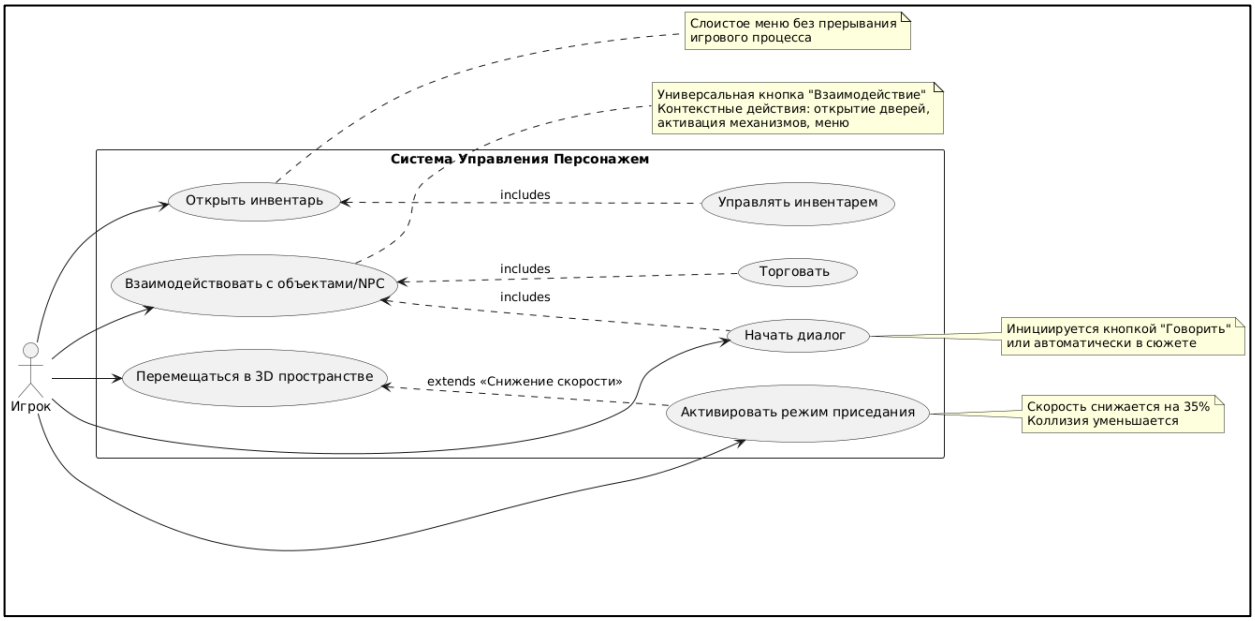


Рисунок 1.16 – Диаграмма вариантов использования системы управления персонажем

1.5.3 Система инвентаря

Основу системы составляют три класса, взаимодействующих через наследование и ссылки (рисунок 1.18).

Класс InventoryData служит контейнером для хранения данных игрока. Он содержит массив slots, каждый из которых является экземпляром класса SlotData, объединяющего объект ItemData и его количество (quantity). При добавлении нового предмета система автоматически создаёт слот, если аналогичный объект отсутствует, или увеличивает счётчик в существующем слоте. Динамическое расширение инвентаря достигается за счёт изменения размера массива slots в реальном времени, что позволяет игроку не беспокоиться о заполненности хранилища.

Класс ItemData определяет свойства предметов. Каждый объект имеет уникальный строковый идентификатор (ID), название (Name), описание (Description) и флаг Consumable, разрешающий использование. Тип предмета (Type) задаётся перечислением, включающим категории для еды, расходников, ключевых объектов и валюты (CARD). Поля Payment и Price используются в торговых операциях: первое указывает на тип валюты (например, CARD), второе – на её требуемое количество.

Метод add_item в классе InventoryData отвечает за добавление предметов. Он проверяет наличие объекта в существующих слотах по имени (itemData.Name). При обнаружении совпадения счётчик quantity увеличивается, в противном случае создаётся новый слот. Для упрощения интеграции с игровыми событиями реализован метод add_item_by_ID, который добавляет предмет по его ID, сверяясь с внешней базой данных.

Использование предметов обрабатывается методом use_item. Если объект помечен как расходуемый (Consumable), его количество уменьшается. При достижении нуля слот удаляется из массива. Торговые операции используют методы check_if_affordable и buy_item. Первый проверяет, достаточно ли у игрока валюты (CARD), сравнивая её количество с ценой товара. Если условие выполнено, buy_item списывает валюту и передаёт игроку приобретённый предмет (рисунок 1.17).

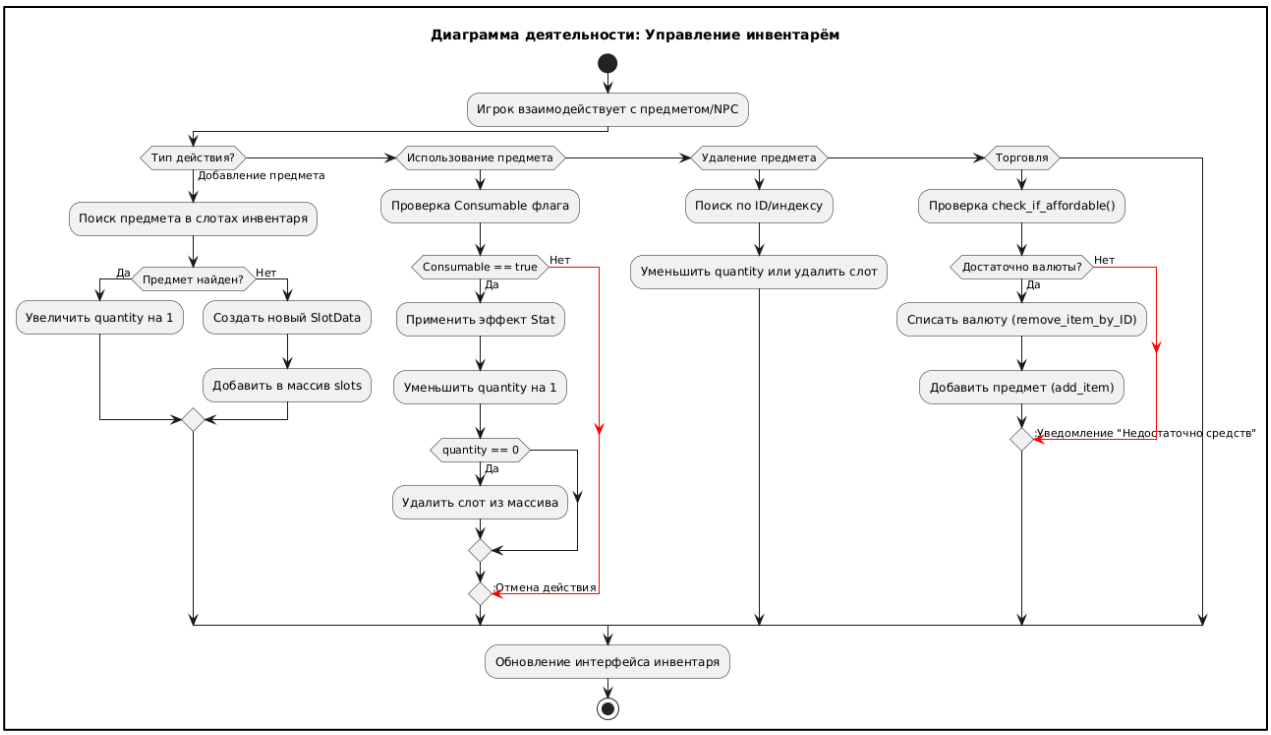


Рисунок 1.17 – Диаграмма деятельности системы инвентаря

Интерфейс инвентаря реализован как вертикальный список слотов, отображающий иконки предметов и их количество. Отсутствие категорий и сложных элементов управления (вроде перетаскивания) делает взаимодействие интуитивным. Игрок может использовать предметы через контекстное меню,

активируемое выбором слота. Предметы типа CARD отображаются в отдельном счётчике валюты, расположенном в названии предмета Карты, что упрощает оценку ресурсов при торговле с NPC.

Динамическое расширение инвентаря исключает необходимость ручной сортировки или удаления предметов. Новые слоты автоматически добавляются в конец списка, а их прокрутка реализована через скролл-панель. Визуально интерфейс сохраняет минимализм: иконки предметов стилизованы под схематичные рисунки, а фон меню полупрозрачен, чтобы не перекрывать игровой мир.

Использование ресурсов Godot (Resource) позволяет сериализовать данные инвентаря, упрощая сохранение и загрузку прогресса. Например, при сохранении игры состояние массива slots записывается в файл, а при загрузке – восстанавливается без потери данных. Метод remove_item_by_ID обеспечивает точное удаление предметов по их ID.

Класс InventoryData поддерживает модификацию через редактор Godot, где можно настраивать начальные предметы игрока, что полезно для тестирования. Автоматическое объединение стеков снижает нагрузку на память и предотвращает фрагментацию данных. Система не требует жёсткой привязки к конкретному типу предметов, что позволяет легко расширять её новыми категориями или механиками без изменения базовой логики.

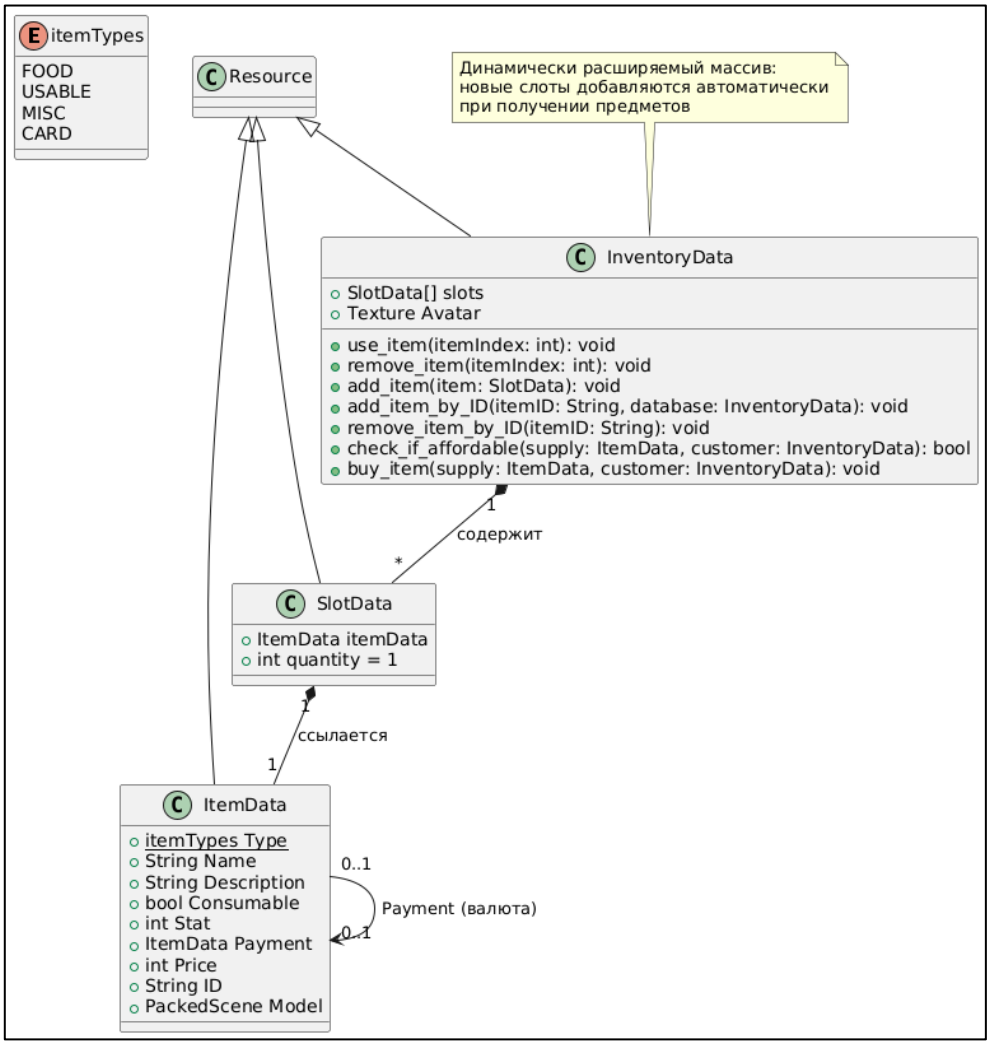


Рисунок 1.18 – Диаграмма классов системы инвентаря

1.5.4 Система сохранений

Сохранение данных осуществляется через сериализацию игровых объектов в бинарные файлы формата .dat, расположенные в директории user://save/. Основной файл tempSAVE.dat выступает временным хранилищем текущего прогресса и автоматически обновляется при критических событиях: переходе между локациями, взаимодействии с NPC или активации контрольных точек («Зон-банкоматов»). При ручном сохранении через меню создаётся копия tempSAVE.dat с уникальным именем, формируемым на основе профиля игрока и временной метки. Диаграмма развёртывания системы представлена на рисунке 1.19.

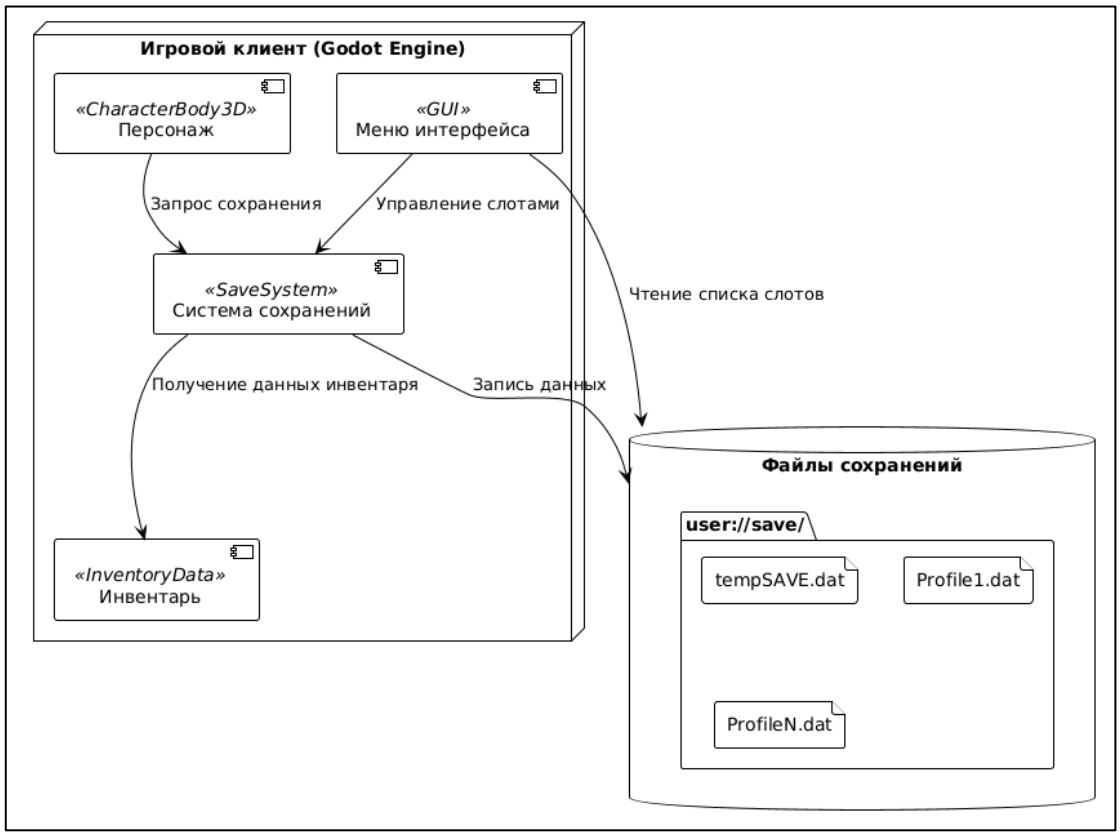


Рисунок 1.19 – Диаграмма развёртывания системы сохранений

Узел CharacterBody3D (управление персонажем) содержит переменные для отслеживания состояния:

– elapsed_time – накопленное игровое время;

– playerItemsList и collectableItemsList – строковые представления инвентаря и коллекционных объектов;

– inSaveZone – флаг нахождения в зоне сохранения.

Методы save_savefile() и load_savefile() отвечают за кодирование и восстановление данных. При сохранении инвентарь преобразуется в строку формата [ID][Количество]!, а позиции коллекционных объектов – в [ID][X][Y][Z]!. Это позволяет компактно хранить динамические данные и воссоздавать их при загрузке через методы process_items() и process_collectables(). Подробнее процессы загрузки сохранения можно рассмотреть на рисунках 1.20 и 1.21 соответственно.

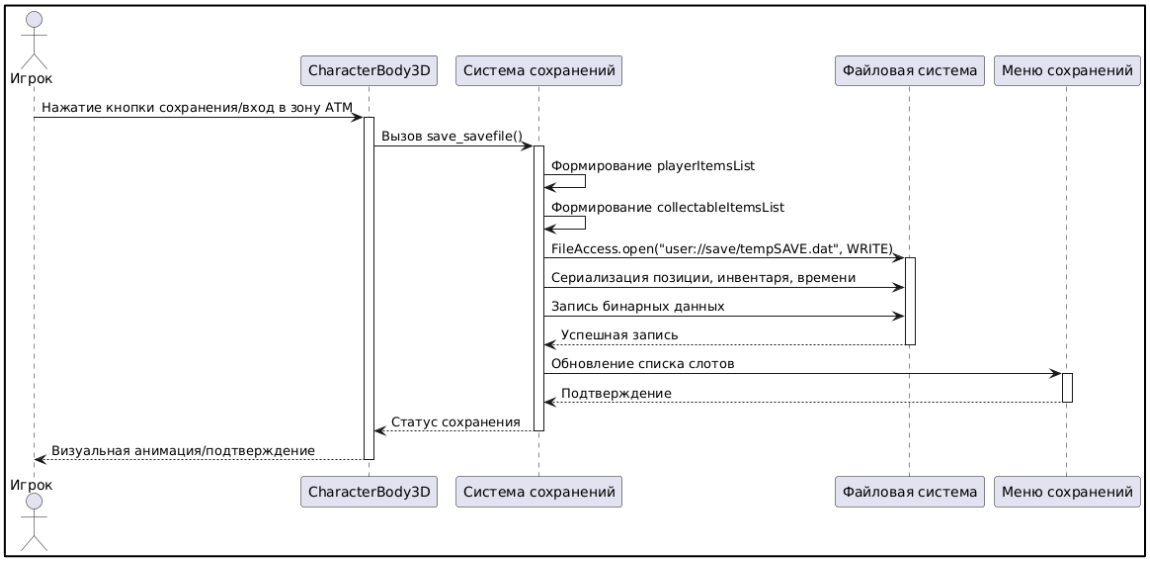


Рисунок 1.20 – Диаграмма последовательности сохранения

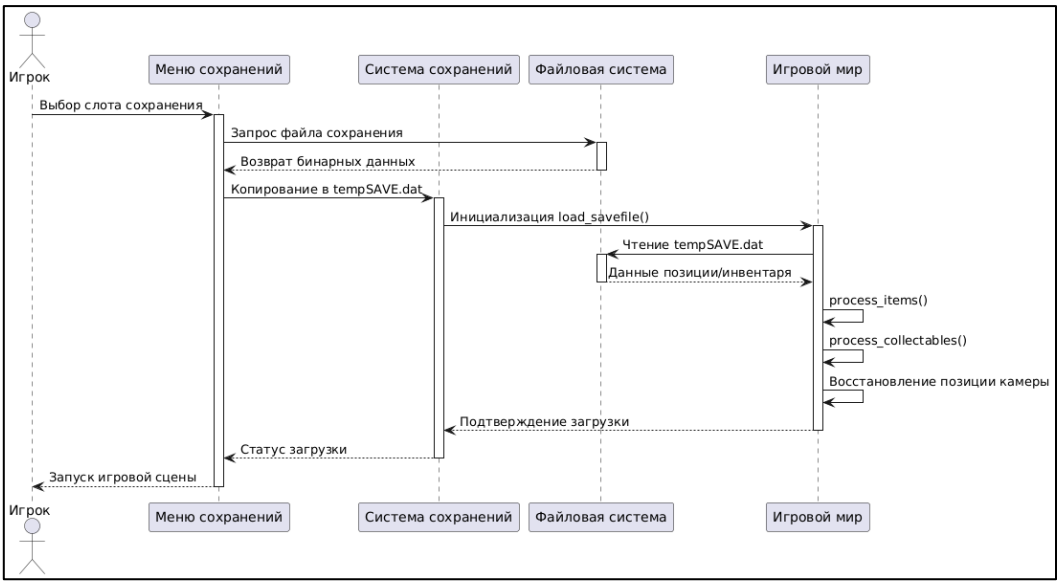


Рисунок 1.21 – Диаграмма последовательности загрузки

1.6 Разработка приложения

1.6.1 Создание стилизованных текстур и 3D-моделей

Процесс создания текстур и моделей для проекта «Escape Building-9» направлен на воссоздание эстетики PlayStation 1, где технические ограничения эпохи становятся художественным приёмом. Работа начинается с подготовки текстур, для чего используются реальные фото-референсы – например, стены корпуса ПГУ, двери или элементы интерьера. В Adobe Photoshop создаётся квадратный шаблон с низким разрешением (часто 256x256 пикселей), куда

копируются фрагменты референсов. Эти фрагменты вручную комбинируются и корректируются: инструмент «Заплатка» устраняет швы и артефакты, а дублирование участков позволяет создать иллюзию повторяющихся паттернов, таких как кирпичная кладка или плитка. Для сохранения аутентичности применяется дизеринг – метод, имитирующий ограниченную цветовую палитру PS1. Текстура экспортируется в формате PNG-8 с включённым дизерингом, что обеспечивает плавные переходы между цветами даже при минимальном количестве оттенков, сохраняя «шум» и зернистость, характерные для ретро-графики (рисунок 1.22).

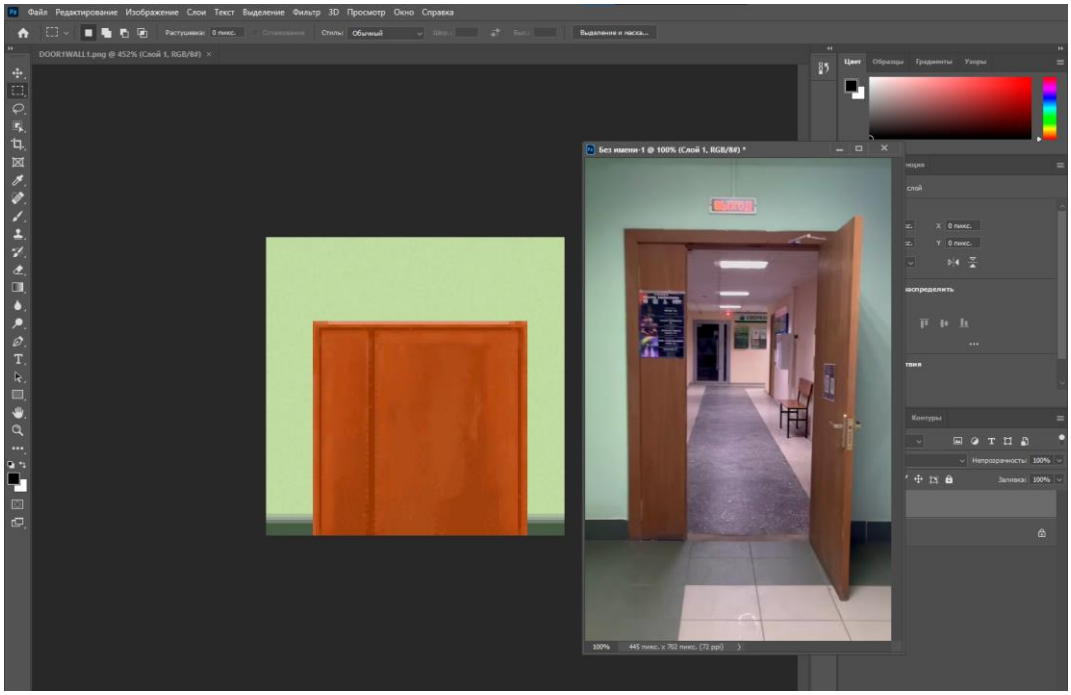


Рисунок 1.22 – Создание текстуры по фото-референсу

3D-моделирование в Blender начинается с построения низкополигональных объектов, где каждая грань имеет выраженные углы. Модели создаются методом экструдирования базовых форм (кубов, цилиндров), а количество полигонов строго контролируется, чтобы избежать избыточной детализации. В свойствах сетки активируется режим Flat Shading, убирающий сглаживание и подчёркивающий «блочность» силуэтов, что соответствует визуальному стилю PS1. Для персонажей и сложных объектов, таких как двери или мебель, важна работа с UV-развёрткой. Референсы помогают корректно расположить текстуру: например, лицо персонажа или элементы одежды вручную редактируются на UV-

карте, чтобы избежать искажений. Далее выполняется запекание текстуры с целью переноса модифицированной текстуры на новую компактную UV-карту, которая корректно обеспечит расположение всех деталей с учетом внесённых нами модификаций. Результат моделирования можно увидеть на рисунке 1.23.

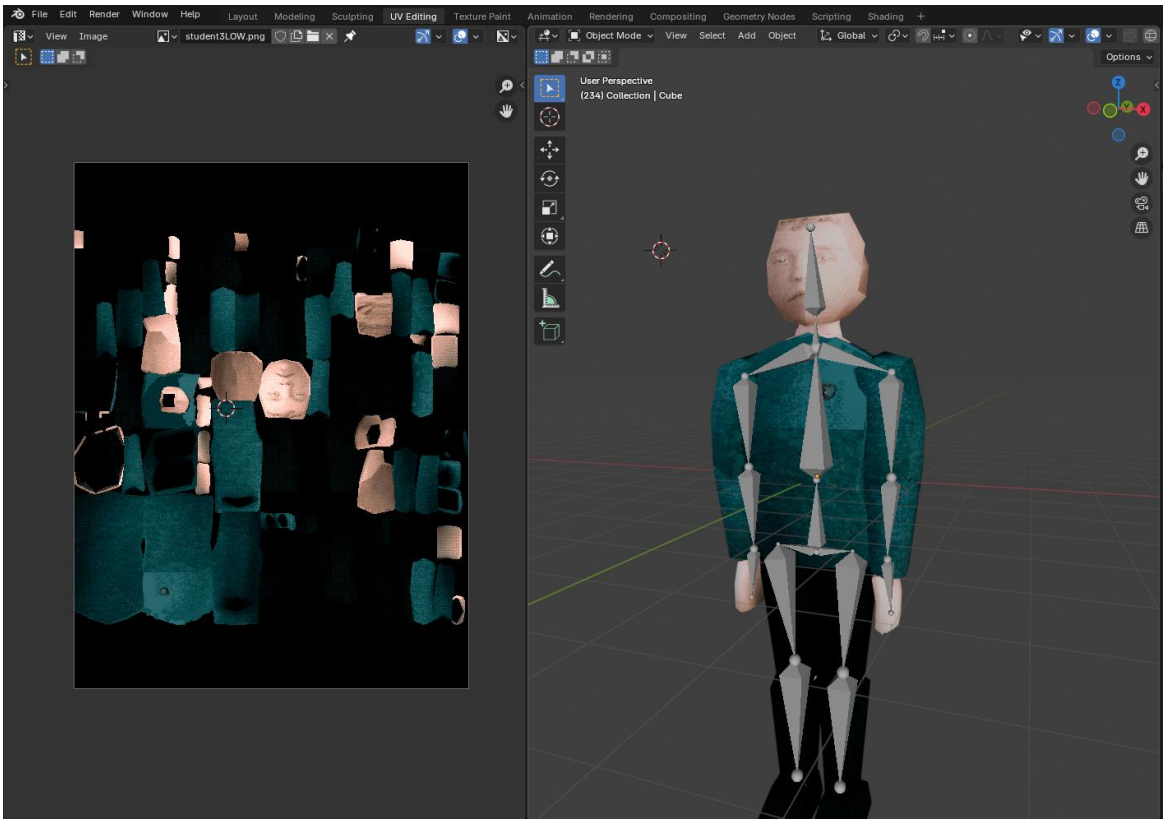


Рисунок 1.23 – Создание стилизованной 3D-модели персонажа

1.6.2 Реализация внутриигрового пользовательского интерфейса

Создание пользовательского интерфейса (UI) в Godot Engine требует тщательной работы с контейнерами, якорями и иерархией элементов, наследующихся от узла Control. Каждый компонент UI – кнопки, панели, текстовые поля – настраивается через свойства позиционирования, привязки к краям экрана и стили, загружаемые из специальных ресурсов (*.tres). Это обеспечивает адаптацию интерфейса под разные разрешения и сохраняет целостность визуала при масштабировании. Пример реализации интерфейса в редакторе Godot можно увидеть на рисунке 1.24.

Для мобильной версии выбрана портретная ориентация, позволяющая управлять игрой одной рукой: основные действия сосредоточены в нижней трети экрана, где расположены крупные сенсорные зоны.

Ключевой элемент UI – панель управления с полупрозрачными кнопками, которые не перекрывают обзор. Кнопка перемещения вперёд занимает центральное положение и имеет увеличенный размер, подчеркивая её приоритет (рисунок 1.25). Многие кнопки (как на рисунке 1.26) используют текстуры с пиксельным градиентом, созданным на основе чёрно-белых изображений. Цвет задаётся динамически через свойство self_modulate, что позволяет перекрашивать элементы интерфейса в зависимости от контекста (например, красный для опасных действий). В меню (главное, настройки, сохранения) применяются кнопки со скруглёнными рамками и прозрачным фоном, сохраняя минималистичный стиль.

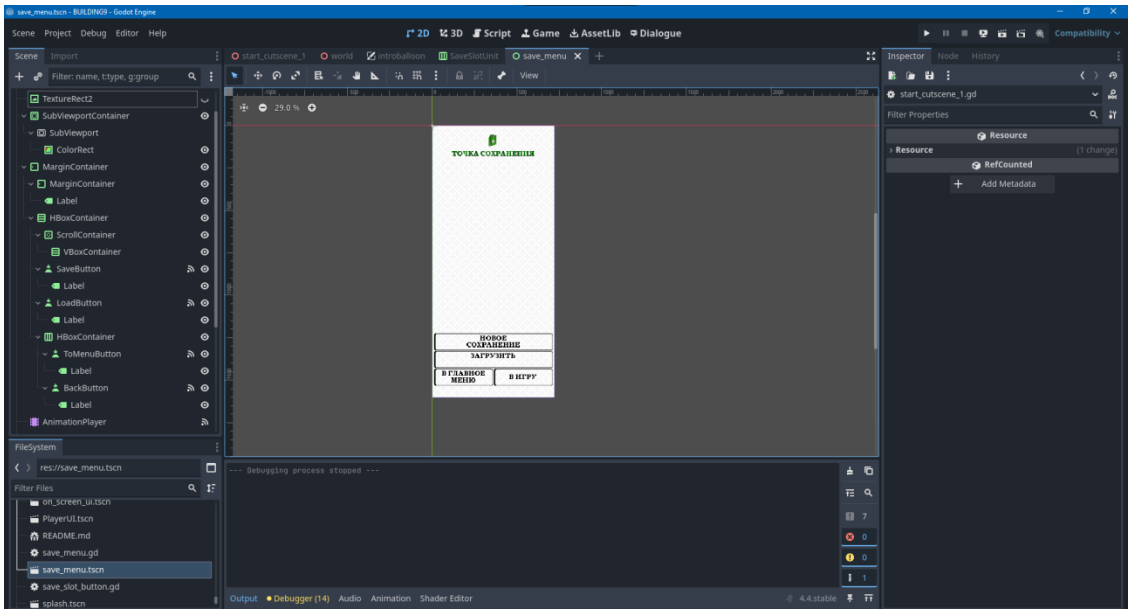


Рисунок 1.24 – Вид редактора графического интерфейса в Godot

Шрифт Times New Roman, пикселизированный до размера 8x8, напоминает «студенческие распечатки» и сочетается с ретро-стилистикой игры.

В инвентаре (рисунок 1.25) и меню взаимодействия с NPC используется аватар в пиксель-арте, отражающий характер персонажа или объекта. Рядом с аватаром расположено текстовое описание, стилизованное под реплики персонажей, или индикатор голода. Окно инвентаря включает 3D-модель текущего предмета, отрендеренную в реальном времени, и текстовое описание его свойств. Кнопка действия (например, «Использовать» или «Съесть») размещена в нижней части экрана для удобства нажатия, а под ней – кнопка «В игру»,

обеспечивающая мгновенный возврат в игровой процесс. Переключение между предметами осуществляется стрелками по бокам.

Рисунок 1.25 – Панель управления персонажем

Рисунок 1.26 – Меню инвентаря

1.6.3 Разработка системы управления персонажем

Теперь нужно подробно рассмотреть реализацию системы управления персонажем и взаимодействиями в игровом мире. Система охватывает множество

аспектов – от передвижения и управления камерой до сохранения/загрузки прогресса и работы с инвентарём.

Скрипт начинается с наследования от класса `CharacterBody3D`, что позволяет использовать встроенные функции для управления 3D персонажем. Далее объявляются сигналы, которые используются для оповещения других компонентов о состоянии меню и зоне магазина:

```
extends CharacterBody3D


signal menu_opened
signal menu_closed
signal entered_shop_zone(meta: Resource)
signal exited_shop_zone
```

Эти сигналы позволяют отделить логику взаимодействия с меню и магазинами от основной логики персонажа.

Далее происходит инициализация аудиоплееров и элементов интерфейса с помощью `@onready var`, что гарантирует, что узлы уже присутствуют в сцене к моменту использования:

```
@onready var track: AudioStreamPlayer2D = $"../../../Track"
@onready var door_open: AudioStreamPlayer2D =
$"../../../DoorOpen"
@onready var door_close: AudioStreamPlayer2D =
$"../../../DoorClose"
@onready var debugging: Label =
$"../../../OnScreenUi/Debugging"
```

Это позволяет воспроизводить звуки, связанные с передвижением, открытием дверей и отладочной информацией.

Для управления движением персонажа используются переменные скорости и направления, а также ускорения, что даёт более плавное и реалистичное передвижение:

```
var Speed: float = 0
var correctedSpeed: float
var acceleration = 18.0
@onready var player: CharacterBody3D = $"."
@onready var collider: CollisionShape3D = $CollisionShape3D
var input_dir
var direction
```

Переменная `acceleration` отвечает за плавность изменения скорости, а `direction` вычисляется на основе ввода пользователя.

В части, касающейся инвентаря и меню, определены переменные для хранения состояния отображения меню, ссылки на ресурсы инвентаря и базы данных предметов, а также элементы интерфейса и анимации:

```
var Viewing: bool = true
var inMenu: bool = false
var tempMeta: Resource
var playerInventory: Resource =
preload("res://data/inventory/PlayerInventory.tres")
var itemDatabase: Resource =
preload("res://data/inventory/Database.tres")
@onready var playerAnimations: AnimationPlayer =
$PlayerAnimations
@onready var player_menu: Panel =
$"../../../Control/PlayerMenu"
@onready var on_screen_ui: Node2D = $"../../../OnScreenUi"
```

Здесь `Viewing` отвечает за возможность осмотра мира, а `inMenu` – за состояние открытия меню. Использование `preload` позволяет загружать ресурсы заранее для быстрого доступа.

Для сохранения прогресса введены переменные, хранящие информацию о времени, позиции, инвентаре и состоянии игрока:

```
var tempSAVE
var newGame: bool
var inSaveZone: bool = false
var lastATM
var profileName: String
var elapsed_time: float
var playerItemsList: String
var collectableItemsList: String

var time_scale: float = 120.0  # 1 реальная секунда = 1 игровая
минута
var current_period: String = ""
var current_location: String = ""
  @onready    var    world_environment:    WorldEnvironment    =
  $"../../../WorldEnvironment"
```

`time_scale` задаёт соотношение реального времени к игровому, что влияет на смену времени суток и связанные с этим эффекты.

Для реализации приседания и изменения коллайдера персонажа используются отдельные переменные и константы, а также отдельный коллайдер для положения в приседе:

```
@onready var colliderCrouch: CollisionShape3D =
$CollisionShape3DCrouch
const standHeight: float = 0.054 # Высота персонажа в стоячем
положении
const crouchHeight: float = -0.6  # Высота персонажа в
присевшем положении
var isCrouching: bool = false
```

Это позволяет переключаться между двумя состояниями коллайдера, обеспечивая корректное взаимодействие с окружением.

Управление камерой реализовано через узлы головы и камеры, а также с помощью переменных для плавного вращения по осям X и Y, учитывая чувствительность мыши или гироскопа:

```
@onready var head: Node3D = $Head
@onready var raycast: RayCast3D = $Head/Camera3D/RayCast3D
@onready var camera: Camera3D = $Head/Camera3D
var x_rotation: float = 0.0
var mouseSensitivity: float
var target_y_degrees = 0.0
var current_y_degrees = 0.0
var target_x_degrees = 0.0
var current_x_degrees = 0.0
var smoothness = 20.0
```

Плавное вращение реализуется с помощью интерполяции текущих углов к целевым, что обеспечивает приятный пользовательский опыт.

В функции `_ready` происходит начальная настройка переменных, загрузка сохранений и инициализация углов поворота персонажа:

```
func _ready() -> void:
    current_location = "Hub"

    if Global.sceneHistory[-1] == "res://start_cutscene_1.tscn":
            for npc in get_tree().get_nodes_in_group("npcs"):
                    for item in npc.get_meta("StarterInventory").slots:
                            npc.get_meta("ShopData").attachedInventory.slots.append(item)

    load_savefile()
    current_y_degrees = rad_to_deg(rotation.y)
    target_y_degrees = current_y_degrees
    current_x_degrees = rad_to_deg($Head.rotation.x)
    target_x_degrees = current_x_degrees
```

Здесь также реализована логика для инициализации инвентаря NPC после определённой сцены.

Основная игровая логика передвижения и обновления состояния происходит в функции `_physics_process`. В ней учитывается прошедшее время, вычисляется игровое время, обновляется окружение и музыка в зависимости от времени суток, а также обрабатываются входные данные и физика персонажа:

```
func _physics_process(delta) -> void:
  elapsed_time += delta

  var game_time = elapsed_time * time_scale
  var hours = int(game_time / 3600) % 24
  var new_period = get_time_period(hours)

  if new_period != current_period and current_location ==
"Hub":
       update_environment_and_music(new_period)
       current_period = new_period

  if not is_on_floor():
       velocity += get_gravity() * delta
  if not inMenu:
       # Плавное вращение по горизонтали (Y-axis)
       var delta_y = (target_y_degrees - current_y_degrees) *
smoothness * delta
       current_y_degrees += delta_y
       rotate_y(deg_to_rad(delta_y))
       # Плавное вращение по вертикали (X-axis)
       var delta_x = (target_x_degrees - current_x_degrees) *
smoothness * delta
       current_x_degrees += delta_x
       current_x_degrees = clamp(current_x_degrees, -60, 60)
       $Head.rotation.x = deg_to_rad(current_x_degrees)
       if direction:
            if isCrouching:
                 correctedSpeed = ((float(player_menu.hunger) /
500))
  else:
                 correctedSpeed = ((float(player_menu.hunger) /
250))

            var target_velocity = direction * correctedSpeed
            velocity.x = move_toward(velocity.x,
target_velocity.x, acceleration * delta)
            velocity.z = move_toward(velocity.z,
target_velocity.z, acceleration * delta)
            debugging.text = str(target_velocity)

  else:
            velocity.x = move_toward(velocity.x, 0, 1)
```

```
            velocity.z = move_toward(velocity.z, 0, 1)

    move_and_slide()
```

Здесь реализована физика гравитации, плавное вращение камеры, а также корректировка скорости персонажа в зависимости от состояния голода и положения (приседание или стоя).

В функции `_input` обрабатываются различные события ввода – движение, взаимодействия с объектами, переключение меню, а также управление камерой через мышь или сенсорный экран:

```
func _input(event) -> void:
  if not inMenu:
       input_dir = Input.get_vector("ui_left", "ui_right",
"ui_up", "ui_down")
       direction = (transform.basis * Vector3(input_dir.x, 0,
input_dir.y)).normalized()

  if event is InputEventScreenDrag and Viewing:
       mouseSensitivity = Global.sensitivity
       target_y_degrees += -event.relative.x * mouseSensitivity
       target_x_degrees -= event.relative.y * mouseSensitivity
       target_x_degrees = clamp(target_x_degrees, -60, 60)

  if event.is_action_pressed("ui_cancel"):
       Viewing = not Viewing
       Input.set_mouse_mode(Input.MOUSE_MODE_VISIBLE)

  if event.is_action_pressed("ui_text_indent"):
       menu_open_action()
```

Функция `toggle_crouch` переключает состояние приседания персонажа, изменяя коллайдер и высоту головы с плавной анимацией:

```
func toggle_crouch() -> void:
  isCrouching = not isCrouching
  var height
  if isCrouching:
       height = crouchHeight
  else:
       height = standHeight
  collider.disabled =  not collider.disabled
  start_height_change(head.position.y, height)  # Изменяет
высоту персонажа

func start_height_change(start_height: float, target_height:
float) -> void:
  var elapsed_time = 0.0

  while elapsed_time < 0.2:
```

```
        elapsed_time += get_process_delta_time()
        var t = elapsed_time / 0.2
        head.position.y = lerp(start_height, target_height, t)
        await get_tree().create_timer(0.01).timeout
    head.position.y = target_height
```

1.6.4 Разработка системы управления инвентарём

Основой системы инвентаря является иерархия классов данных, реализованных через ресурсы Godot. Класс ItemData определяет структуру предметов, храня мета-информацию: тип (еда, расходуемое, карта), название, описание, цену, 3D-модель и уникальный идентификатор. Это позволяет централизованно управлять параметрами всех игровых объектов через редактор Godot.

```
extends Resource
class_name ItemData
enum itemTypes {FOOD, USABLE, MISC, CARD}

@export var Type: itemTypes
@export var Name: String = ""
@export_multiline var Description: String = ""
@export var Consumable: bool
@export var Stat: int = 0
@export var Payment: ItemData
@export var Price: int
@export var Model: PackedScene
@export var ID: String = "Sosiska"
```

Для работы с инвентарями используется композитная структура. Класс SlotData инкапсулирует предмет и его количество, обеспечивая stackable-логику. InventoryData управляет коллекцией слотов, реализуя базовые операции: добавление/удаление предметов, проверку условий транзакций. Особенность системы - поддержка как инвентаря игрока, так и NPC через единый интерфейс.

```
extends Resource
class_name InventoryData

@export var slots: Array[SlotData]
@export var Avatar: Texture

func add_item(item: SlotData) -> void:
    var found: int = 0
    if slots.size() > 0:
        for i in range(0, slots.size()):
            if item.itemData.Name == slots[i].itemData.Name:
                slots[i].quantity += 1
                found += 1
```

```
        if found == 0:
            var tempItem:SlotData = SlotData.new()
            tempItem.itemData = item.itemData
            slots.append(tempItem)
```

Взаимодействие игрока с предметами реализовано через физические коллайдеры и raycast. При обнаружении объекта с меткой "Collectable" запускается анимация подбора с плавным перемещением предмета к игроку с использованием Tween. После завершения анимации предмет добавляется в инвентарь через сигнальную систему.

```
func start_pickup_animation(item: Node3D):
    is_picking_up = true
    picked_item = item
    var target_position = global_transform.origin +
global_transform.basis.z

    tween = create_tween().set_parallel(true)
    tween.tween_property(picked_item, "global_position",
target_position, 0.8)

    tween.finished.connect(func():

playerInventory.add_item(picked_item.get_meta("inv").slots[0])
        picked_item.queue_free()
        is_picking_up = false
    )
```

Система сохранения использует бинарную сериализацию через FileAccess. При сохранении происходит преобразование инвентарей игрока и NPC в строковые представления с парсингом координат коллекционных предметов. Особое внимание уделено атомарности операций - данные записываются только после полной подготовки всех структур.

```
func save_savefile() -> bool:
    playerItemsList = ""
    for item in playerInventory.slots:
        playerItemsList += "[%s][%d]!" % [item.itemData.ID,
item.quantity]

    collectableItemsList = ""
    for collectable in map.get_children():
        if "Collectable" in collectable.name:
            collectableItemsList += "[%s][%.2f][%.2f][%.2f]!" %
[collectable.name.replace("Collectable", ""),
                collectable.global_position.x,
collectable.global_position.y, collectable.global_position.z]
```

```
    tempSAVE = FileAccess.open("user://save/tempSAVE.dat",
FileAccess.WRITE)
    tempSAVE.store_var(global_position)
    tempSAVE.store_var(player_menu.Bill)
    tempSAVE.store_var(collectableItemsList)
    tempSAVE.close()
```

Интерфейс инвентаря построен на паттерне State Machine, где каждый режим (инвентарь, магазин, обмен) представлен отдельным обработчиком. Классы-наследники MenuModeHandler переопределяют логику отображения и взаимодействия, обеспечивая полиморфное поведение без условных проверок. Динамическая подгрузка 3D-моделей в подокно просмотра реализована через ViewportContainer.

```
class ShopHandler extends MenuModeHandler:

    func update_display() -> void:

        if
menu.currentInventory.check_if_affordable(menu.currentInventory
.slots[menu.currentSlot].itemData, menu.playerInventory):

            menu.buttonlabel.text = "Купить %d₽" %
menu.currentInventory.slots[menu.currentSlot].itemData.Price

            menu.texture_button.self_modulate =
Color(0.61,0.68,0.72,1)

        else:

            menu.buttonlabel.text = "Нужно %d₽" %
menu.currentInventory.slots[menu.currentSlot].itemData.Price
```

Для взаимодействия с игровым миром используются специализированные зоны: торговые зоны изменяют состояние интерфейса через сигналы, зоны сохранения активируют анимацию перехода в меню сохранения. Динамическое обновление окружения (освещение, туман, музыка) привязано к игровому времени, рассчитываемому через коэффициент масштабирования реального времени.

1.6.5 Разработка системы сохранений

Система сохранения в проекте реализована через сериализацию ключевых игровых данных в бинарный файл с последующей возможностью загрузки из него. Основная логика работы с сохранениями сосредоточена в методах

`load_savefile()` и `save_savefile()`, которые взаимодействуют с файловой системой через класс FileAccess.

```
var tempSAVE
var newGame: bool
var inSaveZone: bool = false
var lastATM
var profileName: String
var elapsed_time: float
var playerItemsList: String
var collectableItemsList: String
```

При инициализации персонажа в методе `_ready()` автоматически запускается процедура загрузки сохранения. Это обеспечивает восстановление игрового состояния сразу после загрузки сцены. Система использует временный файл `tempSAVE.dat` для хранения актуального состояния игры, тогда как постоянные сохранения создаются через отдельный интерфейс.

```
func _ready() -> void:
    current_location = "Hub"
    load_savefile()
```

Метод `load_savefile()` выполняет десериализацию данных из файла, используя цепочку вызовов `get_var()`. Особое внимание уделяется восстановлению позиции и ротации персонажа, что гарантирует точное возобновление игры с места последнего сохранения.

```
func load_savefile() -> bool:
    tempSAVE = FileAccess.open("user://save/tempSAVE.dat",
FileAccess.READ)
    if tempSAVE != null:
        global_position = tempSAVE.get_var()
        global_rotation = tempSAVE.get_var()
        profileName = tempSAVE.get_var()
        elapsed_time = tempSAVE.get_var()
        playerItemsList = tempSAVE.get_var()
        player_menu.Bill = tempSAVE.get_var()
        collectableItemsList = tempSAVE.get_var()
        player_menu.hunger = tempSAVE.get_var()
        var npcInventories = tempSAVE.get_var()
```

Для обработки инвентарей используется строковое представление с специальным форматом. Метод `process_items()` разбирает строку вида `[ItemID][Quantity]!`, динамически наполняя инвентарь предметами через вызов

`add_item_by_ID()`. Аналогично `process_collectables()` восстанавливает позиции коллекционных объектов на карте с помощью регулярных выражений.

При вызове `save_savefile()` происходит сбор данных со всей игровой сессии. Инвентарь игрока сериализуется в строку с перечислением ID предметов и их количества. Для коллекционных объектов записываются их координаты в формате `[ItemID][X][Y][Z]`.

```
func save_savefile() -> bool:
    for item in playerInventory.slots:
        playerItemsList += "[%s][%d]!" % [item.itemData.ID,
item.quantity]

    for collectable in map.get_children():
        collectableItemsList += "[%s][%.2f][%.2f][%.2f]!" %
[newName, collectable.global_position.x, ...]

    var npcInventories = {}
    for npc in get_tree().get_nodes_in_group("npcs"):
        npcInventories[npc.get_meta("NPC_ID")] =
форматированный_строковый_инвентарь
```

Отдельная сцена `SaveMenu` реализует интерфейс для работы со слотами через динамическое создание элементов интерфейса. При сохранении используется механизм копирования временного файла в постоянный слот через `copy_tempSave_to()`, что предотвращает повреждение данных при аварийном завершении.

```
func copy_tempSave_to(file_name: String) -> void:
    var tempSAVE = FileAccess.open("user://save/tempSAVE.dat",
FileAccess.READ)
    var newSAVE = FileAccess.open(file_name,
FileAccess.WRITE_READ)
    newSAVE.store_buffer(tempSAVE.get_file_as_bytes())
```

Функция `get_max_save_index()` обеспечивает уникальность имен файлов через поиск максимального номера в существующих слотах. Это предотвращает перезапись сохранений и позволяет создавать неограниченное количество слотов.

```
func get_max_save_index(base_name: String) -> int:
    var max_counter = 0
    var dir = DirAccess.open(pathPrefix)
    while file = dir.get_next():
        if file.begins_with(base_name + " - "):
            max_counter = max(number, max_counter)
    return max_counter +
```

При загрузке система не только восстанавливает позицию персонажа, но и обновляет состояние NPC, перезаписывая их инвентари из сохраненных данных. Это реализовано через группу "npcs" и мета-данные каждого персонажа, содержащие их уникальные идентификаторы.

```
for npc in get_tree().get_nodes_in_group("npcs"):
    if npcInventories.has(npc.get_meta("NPC_ID")):
        process_items(npcInventories[npc_id], npc_инвентарь)
```

Специальный формат хранения игрового времени `elapsed_time` позволяет точно воспроизводить циклы дня и ночи. При загрузке значение дельта-времени умножается на коэффициент `time_scale`, что обеспечивает синхронизацию внутриигрового таймера с реальным.

```
var game_time = elapsed_time * time_scale
var hours = int(game_time / 3600) % 24
current_period = get_time_period(hours)
```

Интеграция системы сохранения с игровыми механиками достигается через обработку зон сохранения и АТМ-терминалов. При взаимодействии с ними вызывается анимация использования и блокируется ввод, что предотвращает возникновение конфликтов при записи данных.

1.6.6 Тестирование и отладка

Когда игроки взаимодействуют с игрой, они иногда могут делать что-то, что приводит к сбоям в работе системы. Например, это может быть попытка сохранить игру в месте, где это сделать нельзя, или ввод отрицательной громкости для музыки в настройках. Также бывает так, что игроки могут пытаться использовать предмет, которого у них нет в инвентаре, или разговаривать с персонажем игры без выполнения каких-то условий.

Чтобы избежать этих проблем, в игру встроены механизмы, которые проверяют правильность вводимых данных и помогают обрабатывать ошибки. Это включает в себя проверки файлов сохранений, чтобы удостовериться, что они корректны, контроль за тем, какие параметры вводятся, и блокировку действий, которые игроки не могут выполнять.

Работа этих механизмов нуждается в проверке с помощью специальных тестов, которые моделируют ситуации, когда что-то идет не так. Такой подход

помогает удостовериться, что игра работает как надо и игроки могут наслаждаться процессом без неожиданных сбоев. Это важно, потому что никто не хочет, чтобы игра ломалась в самый неожиданный момент.

Таблица 2 – Тест-кейсы

Наименование	Описание	Шаги для воспроизведения	Ожидаемый результат
1.Проверка сохранения игры	Убедиться, что игра корректно сохраняет прогресс	1. Запустить игру. 2. Взаимодействовать с зоной сохранения. 3. Перезапустить игру и загрузить сохранение.	Прогресс (позиция, инвентарь, время) соответствует сохраненным данным.
2.Загрузка игры при отсутствии сохранений	Проверка обработки отсутствия файлов сохранений	1. Удалить файлы в папке user//:save. 2. Нажать "Загрузить игру".	Отображается сообщение: *«Нет доступных сохранений»*.
3.Включение/выключение шейдеров	Проверка графических настроек	1. В меню настроек переключить опцию *«Режим низкой графики»*. 2. Вернуться в игру.	Шейдеры отключаются/включаются.
4.Настройка громкости музыки	Проверка регулировки звука	1. Изменить ползунок громкости в меню. 2. Проверить звук в игре.	Громкость музыки изменяется согласно настройкам.
5.Взаимодействие с NPC	Проверка диалогов и торговли	1. Подойти к NPC. 2. Нажать кнопку взаимодействия. 3. Купить предмет.	Предмет добавляется в инвентарь, интерфейс торговли работает корректно.
6.Смена времени суток	Проверка динамического окружения	1. Играть 6 игровых часов. 2. Наблюдать за изменениями.	Освещение, туман и музыка плавно меняются.
7.Использование предметов инвентаря	Проверка механики потребления еды	1. Открыть инвентарь. 2. Выбрать еду. 3. Нажать *«Съесть»*.	Голод уменьшается, предмет исчезает из инвентаря.
8.Обработка ввода на мобильных устройствах	Проверка сенсорного	1. Запустить игру на Android.	Персонаж двигается плавно, интерфейс

Наименование	Описание	Шаги для воспроизведения	Ожидаемый результат
	управления	2. Использовать сенсорный ввод.	реагирует без задержек.
9.Ошибка при некорректном имени профиля	Проверка валидации ввода	1. Ввести пустое имя. 2. Нажать *«Подтвердить»*.	Создаётся профиль с названием « ».
10.Переход между локациями	Проверка загрузки новых зон	1. Взаимодействовать с дверью. 2. Дождаться анимации.	Персонаж перемещается в новую локацию, музыка изменяется.

1.7 Описание приложения

1.7.1 Руководство пользователя

Для того чтобы разобраться в механиках игры, в приложении есть обучающая система, которая интегрирована в интерфейс инвентаря. Чтобы начать обучение, можно открыть инвентарь – то есть нажать на иконку с надписью «инв.» в правом нижнем углу экрана. В верхней части экрана ты увидишь кнопку «Обуч.». Чтобы увидеть подсказки, следует нажать на неё.

Вместо обычных предметов здесь будут учебные материалы, которые расскажут о том, как управлять игрой, продавать и использовать предметы. Чтобы переключаться между подсказками, следует нажимать на жёлтые стрелки по обе стороны «предмета». Текущая подсказка будет показываться в формате «ПОДСКАЗКИ (X/Y)», где X – это номер текущей подсказки, а Y – общее количество доступных подсказок (Рисунок 1.27). Если нужно посмотреть обучающее видео, существует кнопка «Посмотреть» – она выделена желтым. Видео откроется поверх игрового интерфейса. Стоит учесть, что видео в формате .ogv, и их названия совпадают с ID подсказок. Например, подсказка «Базовая навигация» будет использовать файл BasicNav.ogv.

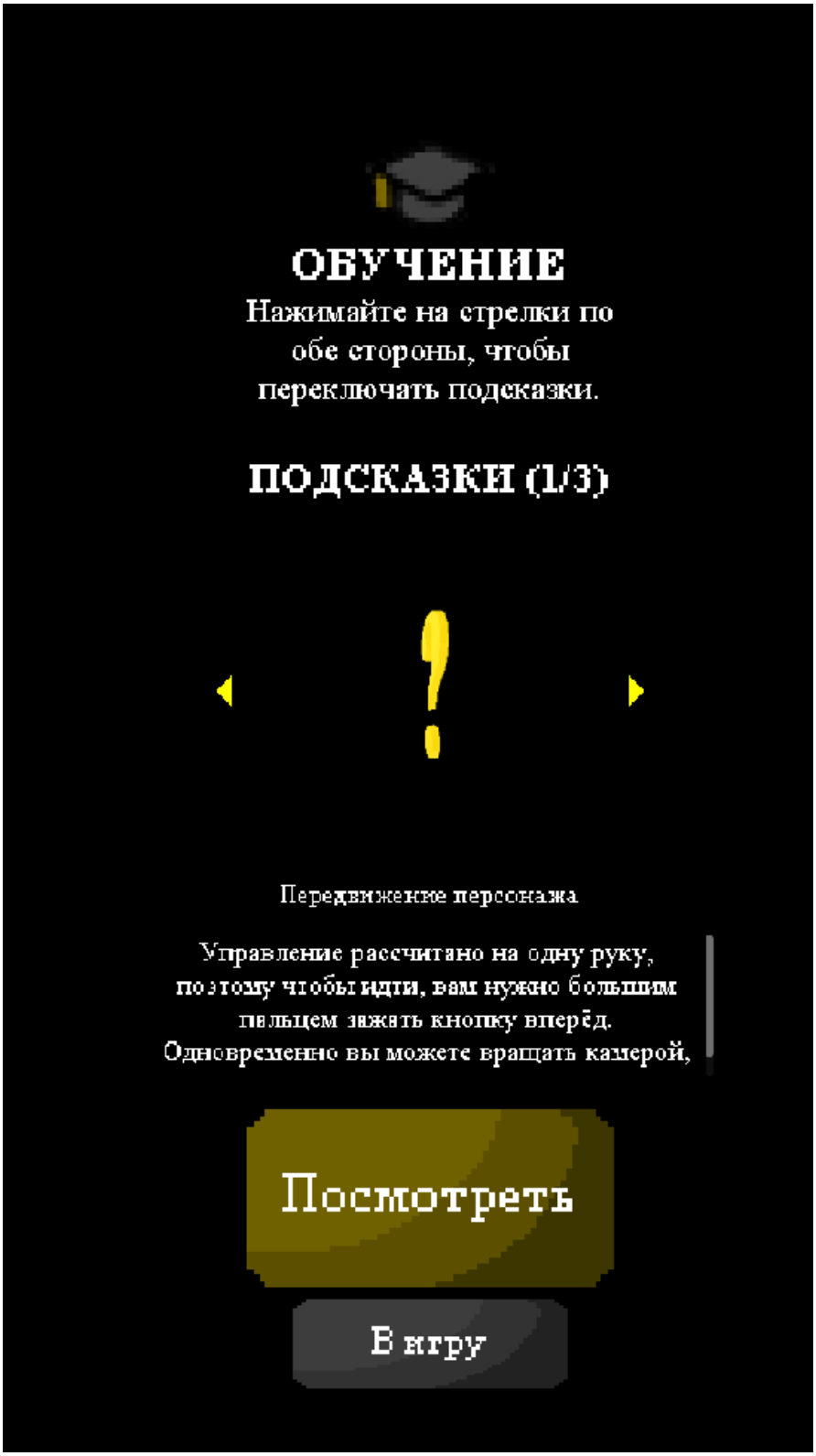


Рисунок 1.27 – Меню подсказок

Если нужно будет вернуться в «инвентарь» подсказок, можно нажать на кнопку «Назад», которая выглядит как элемент главного меню (рисунок 1.28). Некоторые видеоматериалы доступны только после выполнения определенных условий, например, после прохождения первой миссии.

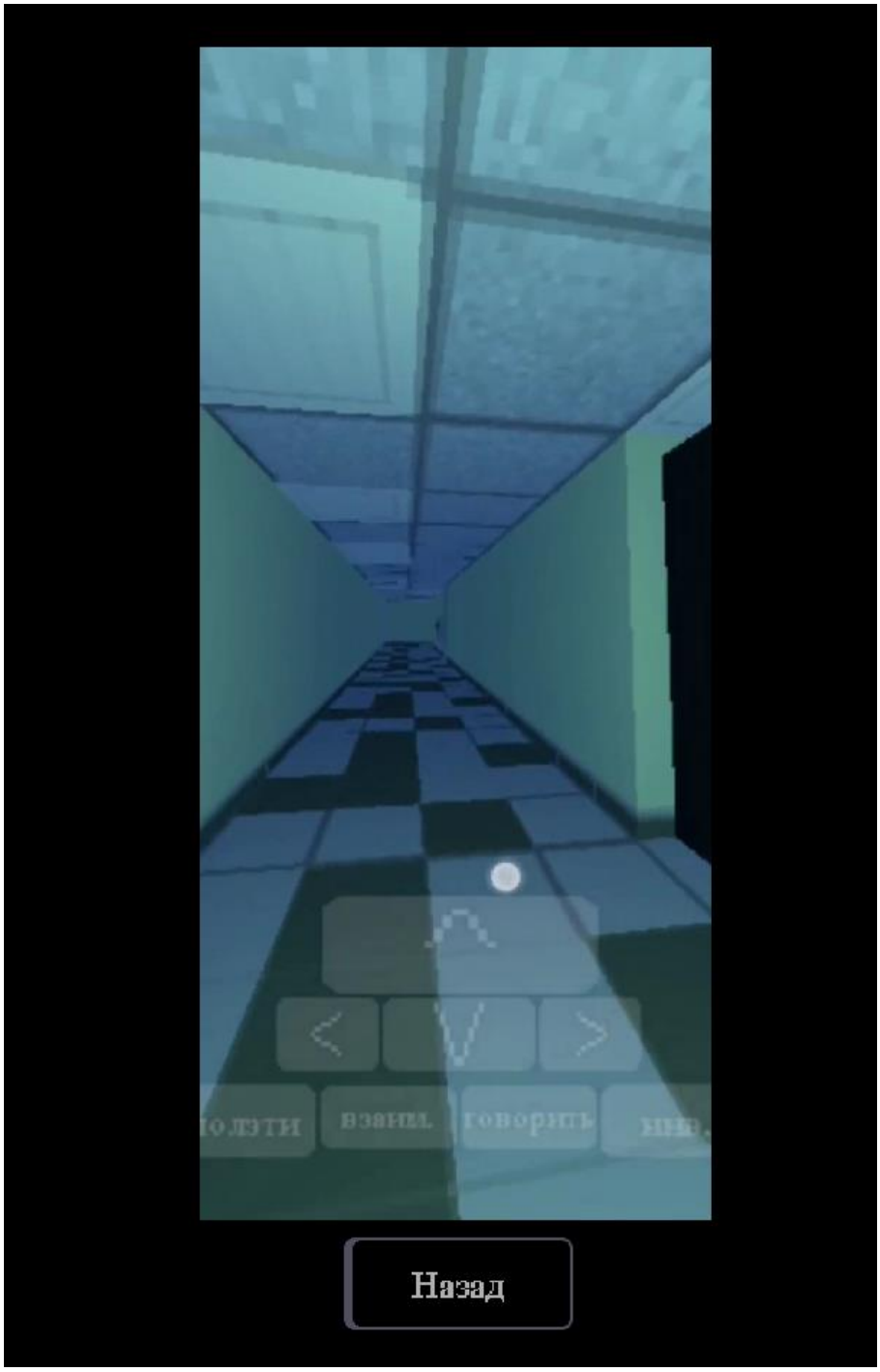


Рисунок 1.28 – Обучающий ролик

Пересматривать обучающие можно материалы столько раз, сколько потребуется. Система сделана так, чтобы удобно было играть даже новичкам, и хорошо адаптирована для управления с помощью сенсора. Это действительно помогает освоить игру и не чувствовать себя абсолютно потерянным.

2 Охрана труда

Разработка мобильной игры требует строгого соблюдения требований охраны труда, регламентированных действующими нормативными документами Российской Федерации, включая ГОСТ Р 12.0.230-2007 «Система управления охраной труда» и ГОСТ Р 50923-96 «Безопасность информационных технологий». Основные аспекты обеспечения безопасности пользователей и разработчиков представлены ниже.

2.1 Эргономические требования

При проектировании интерфейса мобильной игры необходимо учитывать эргономические принципы, направленные на минимизацию физической нагрузки пользователя. Рекомендуется предусмотреть адаптацию элементов управления под различные размеры экранов устройств, чтобы исключить перенапряжение мышц кисти и запястья.

Важным аспектом является регулировка яркости и контрастности экрана. Согласно ГОСТ Р 50948-96, уровень освещенности должен соответствовать условиям внешней среды. Для снижения зрительной усталости целесообразно внедрение автоматического режима «Ночной свет», уменьшающего долю синего спектра.

На рисунке 2.1 представлен пример оптимального положения мобильного устройства при длительном использовании.

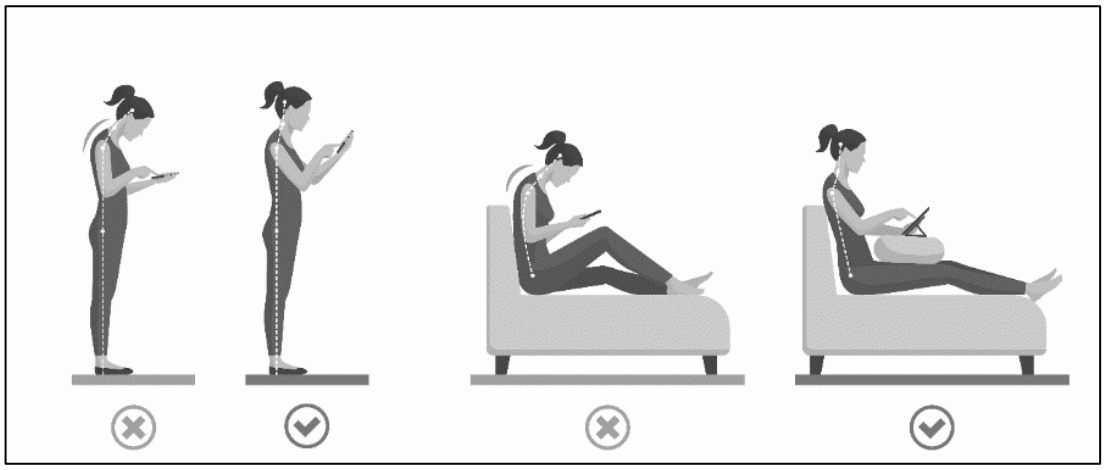


Рисунок 2.1 – Эргономика мобильного устройства

2.2 Временные ограничения и профилактика переутомления

Согласно СанПиН 2.2.2/2.4.1340-03, продолжительность непрерывного взаимодействия с мобильным устройством не должна превышать 30 минут для взрослых пользователей и 15 минут – для детей. В игре рекомендуется реализовать систему уведомлений, напоминающих о необходимости перерыва.

Для детской аудитории обязательна интеграция родительского контроля, ограничивающего время игровых сессий и блокирующего доступ в ночное время (с 22:00 до 06:00).

2.3 Психологическая безопасность

Игра должна соответствовать возрастному рейтингу, установленному в соответствии с Федеральным законом № 436-ФЗ «О защите детей от информации, причиняющей вред их здоровью и развитию». Запрещается использование контента, провоцирующего агрессию, тревожность или иные негативные эмоциональные состояния.

Для минимизации риска игровой зависимости следует исключить механики, основанные на бесконечных циклах вознаграждений. В многопользовательских режимах необходима модерация чатов и автоматическая фильтрация нецензурной лексики.

2.4 Требования к разработчикам

Процесс разработки должен включать этапы тестирования на различных устройствах для проверки соответствия эргономическим нормам. Особое внимание уделяется оптимизации энергопотребления, чтобы исключить перегрев батареи, что регламентировано ГОСТ Р МЭК 62368-1-2015.

Для пользователей с ограниченными возможностями рекомендуется внедрение функций адаптации: изменение размера шрифтов, озвучивание текста, настройка скорости анимации.

2.5 Правовое соответствие

Перед выпуском игры требуется прохождение процедуры сертификации в аккредитованных органах. В описании приложения необходимо разместить предупреждения о возможных рисках, таких как мигающие эффекты, которые могут вызывать дискомфорт у лиц с фотосенситивной эпилепсией.

Заключение

Выполненная выпускная работа представляет собой комплексное исследование и практическую реализацию мобильного игрового приложения «Escape Building-9», разработанного в жанре survival-horror с интеграцией элементов ролевой игры. Проект направлен на преодоление ключевых проблем современного рынка мобильных игр, таких как шаблонность механик, агрессивная монетизация и низкое качество визуального исполнения, предлагая пользователям уникальный опыт, сочетающий атмосферное погружение, стратегическую глубину и образовательную составляющую.

На первом этапе проведён детальный анализ текущего состояния игровой индустрии, выявивший устойчивый спрос на проекты, которые сочетают оригинальную стилистику с продуманным геймплеем. Особое внимание уделено изучению предпочтений целевой аудитории – студентов ПГУ, для которых знакомые локации 9-го корпуса университета стали основой игрового сеттинга. Это решение не только усилило эмоциональную вовлечённость пользователей, но и продемонстрировало потенциал интеграции образовательной среды в игровые механики.

Художественная составляющая проекта базируется на стилизации под ретро-графику эпохи PlayStation 1, что позволило достичь баланса между эстетической выразительностью и технической оптимизацией. Низкополигональные модели, созданные в Blender, и текстуры с ограниченной цветовой палитрой не только воссоздают атмосферу ретро-хоррора, но и обеспечивают стабильную работу приложения на устройствах с минимальными техническими характеристиками. Интеграция динамического освещения и эмбиентного звукового сопровождения дополнительно усилила immersion, трансформируя привычные университетские пространства в зловещие лабиринты.

Техническая реализация осуществлена в Godot Engine – открытом игровом движке, выбранном за гибкость, поддержку Vulkan API и возможность кросс-платформенной разработки. Ключевые системы, такие как инвентарь с динамическим расширением, механика сохранения прогресса с шифрованием

данных и адаптивная генерация игровых событий, были протестированы на различных устройствах под управлением Android. Результаты тестирования подтвердили соответствие приложения заявленным требованиям, включая стабильную частоту кадров и отсутствие критических ошибок.

Перспективы развития проекта включают расширение игрового контента за счёт добавления новых сюжетных веток и локаций, внедрение мультиплеерного режима для кооперативного прохождения, а также публикацию на международных площадках, таких как Google Play и Steam. Дальнейшая работа может быть направлена на адаптацию приложения для iOS и ПК-платформ, что увеличит охват аудитории и усилит образовательный потенциал за счёт интеграции дополнительных материалов об истории университета.

В заключение следует отметить, что разработанное приложение «Escape Building-9» не только соответствует современным стандартам качества игровой индустрии, но и служит примером успешного синтеза технологических инноваций, художественного творчества и образовательных задач. Проект подтвердил возможность создания конкурентоспособного продукта в условиях ограниченных ресурсов, опираясь на открытые инструменты и методологии инди-разработки, что открывает новые горизонты для дальнейших исследований в области геймификации образовательных процессов.

Список использованных источников

1. Godot Engine Documentation. – Режим доступа: https://docs.godotengine.org (дата обращения: 10.04.2025).
2. Кузьминов, А. Н. Геймдизайн: как создать игру, в которую будут играть все. – М.: Питер, 2021. – 320 с.
3. Godot Engine Community Forum. – Режим доступа: https://godotforums.org. (дата обращения: 22.04.2025).
4. Gregory, J. Game Engine Architecture. – CRC Press, 2018. – 1040 p.
5. Introduction to Game Design (Coursera, University of California). – Режим доступа: https://www.coursera.org. (дата обращения: 25.04.2025).
6. Blender Guru Tutorials (YouTube-канал). – Режим доступа: https://www.youtube.com/@BlenderGuru. (дата обращения: 16.04.2025).

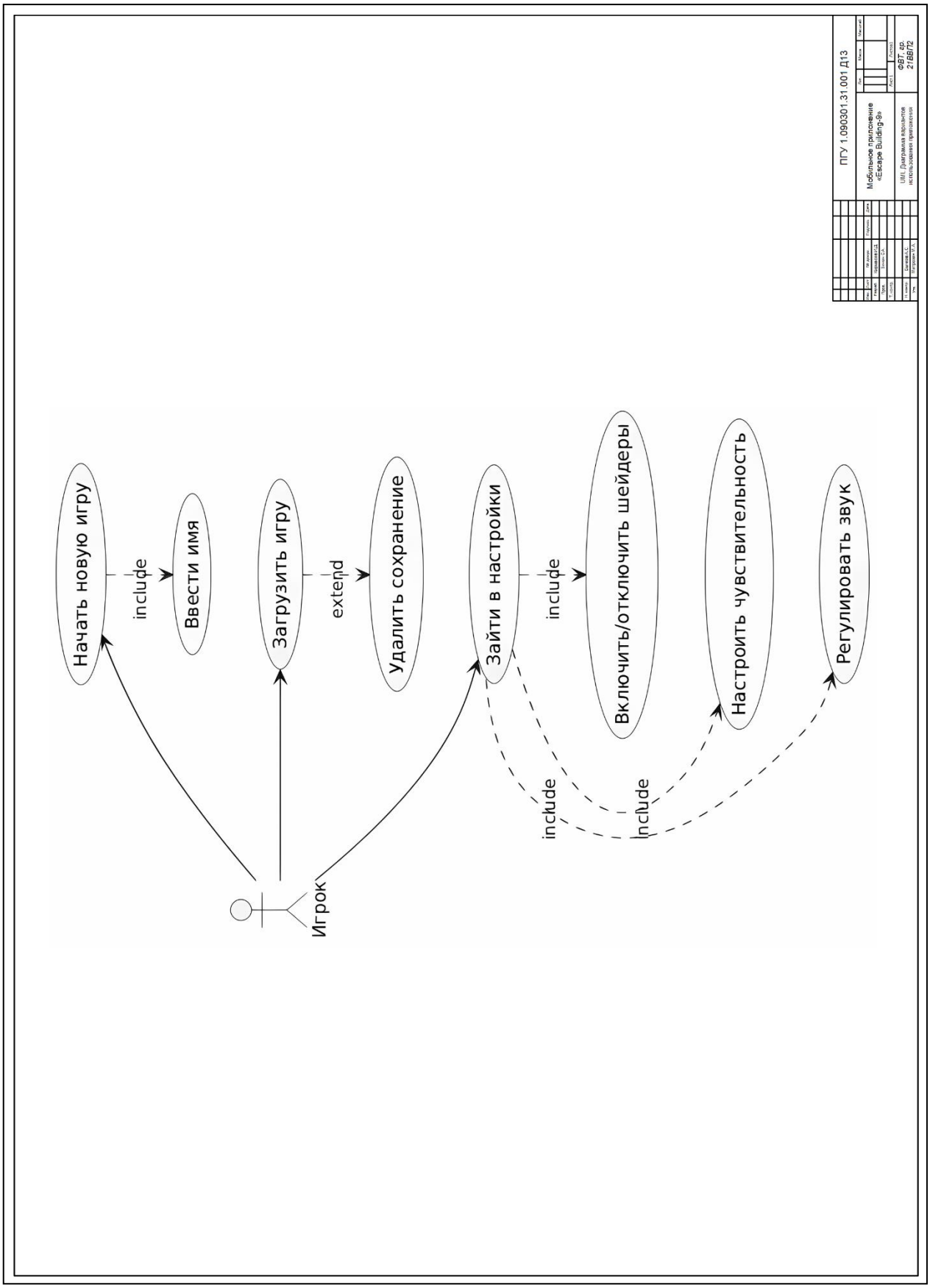
Приложение А

(Обязательное)

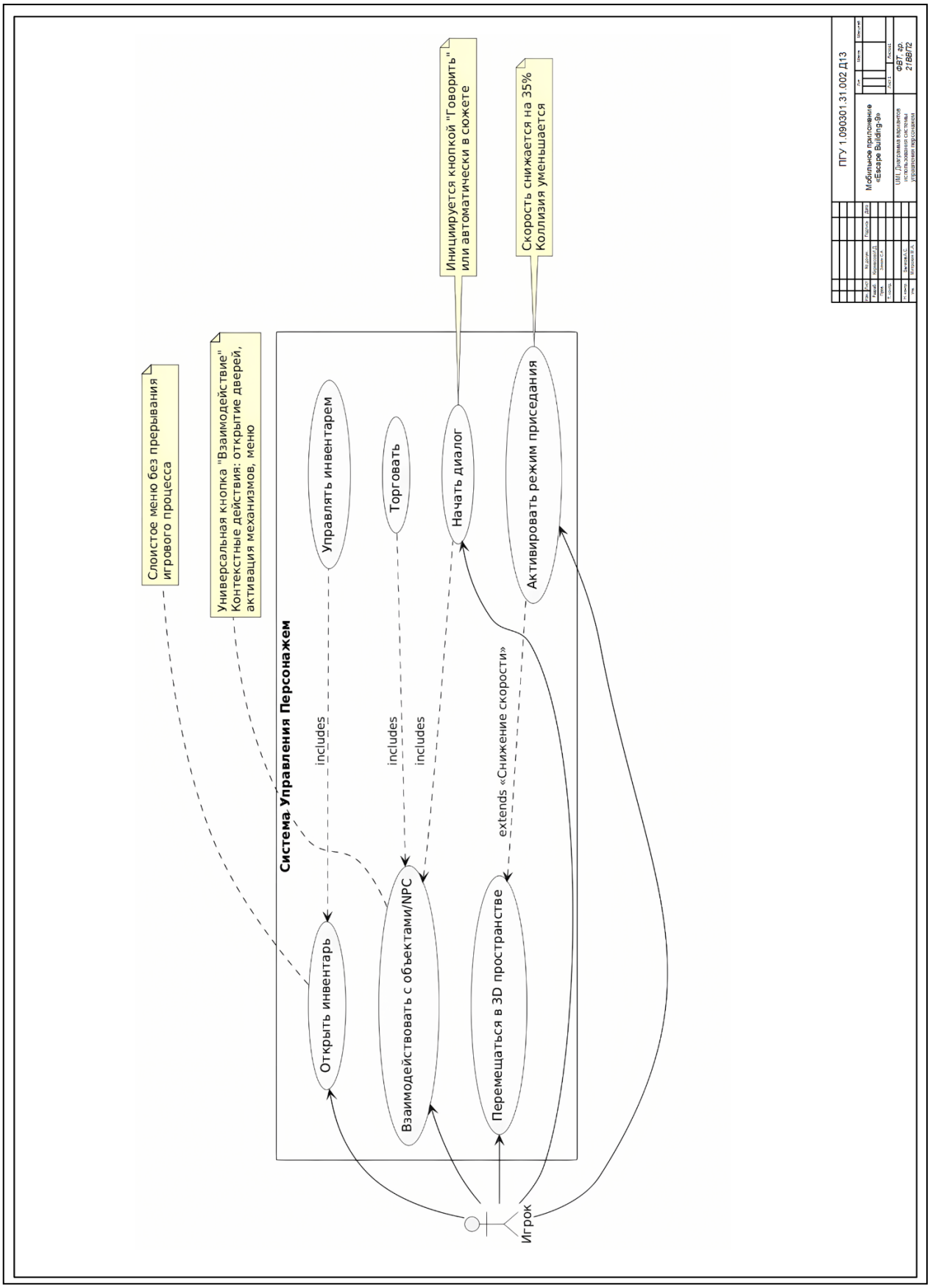
Игровое приложение «Escape Building-9»

UML-диаграммы

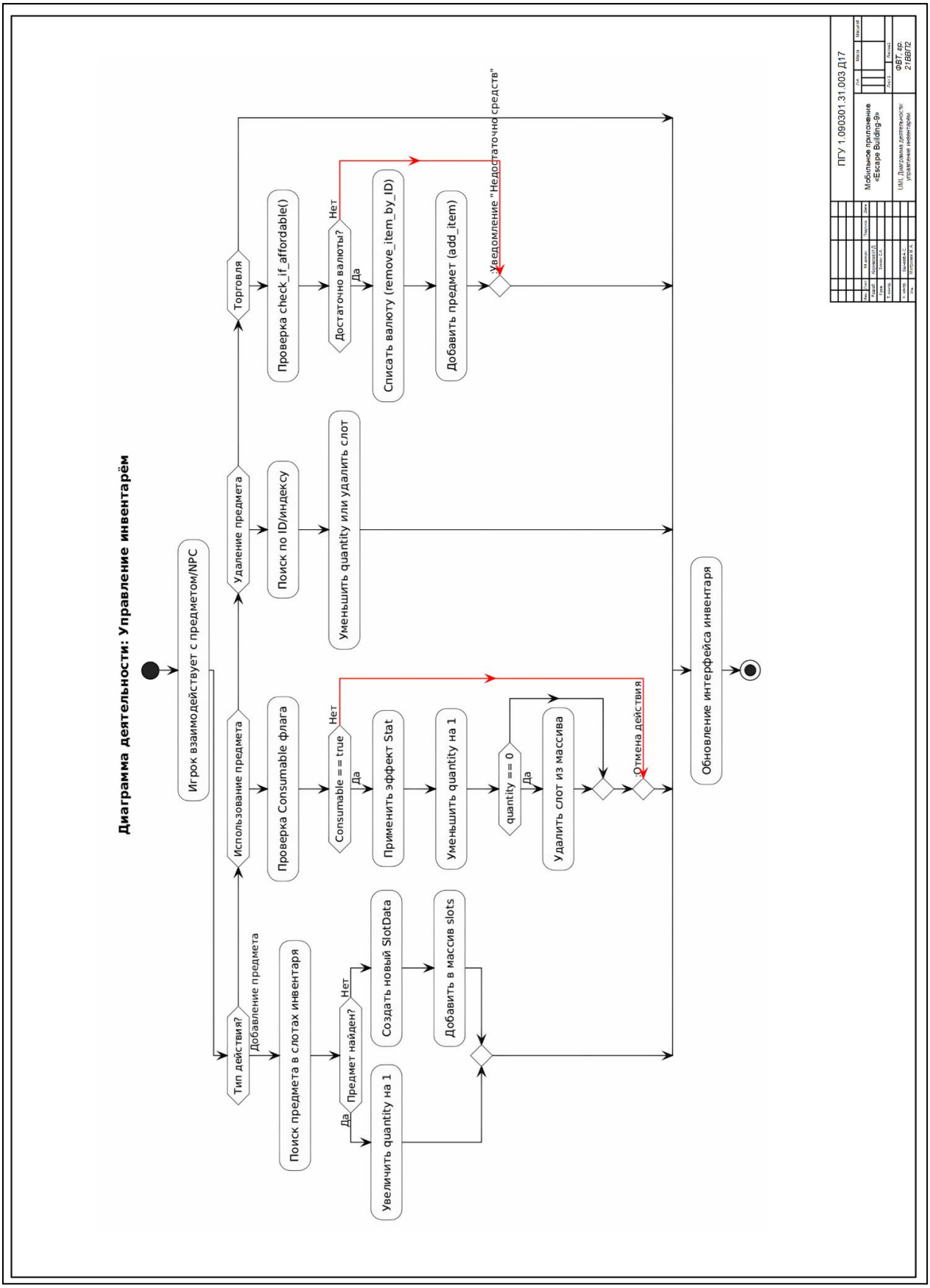
Приложение А.1 – UML – диаграмма прецедентов приложения



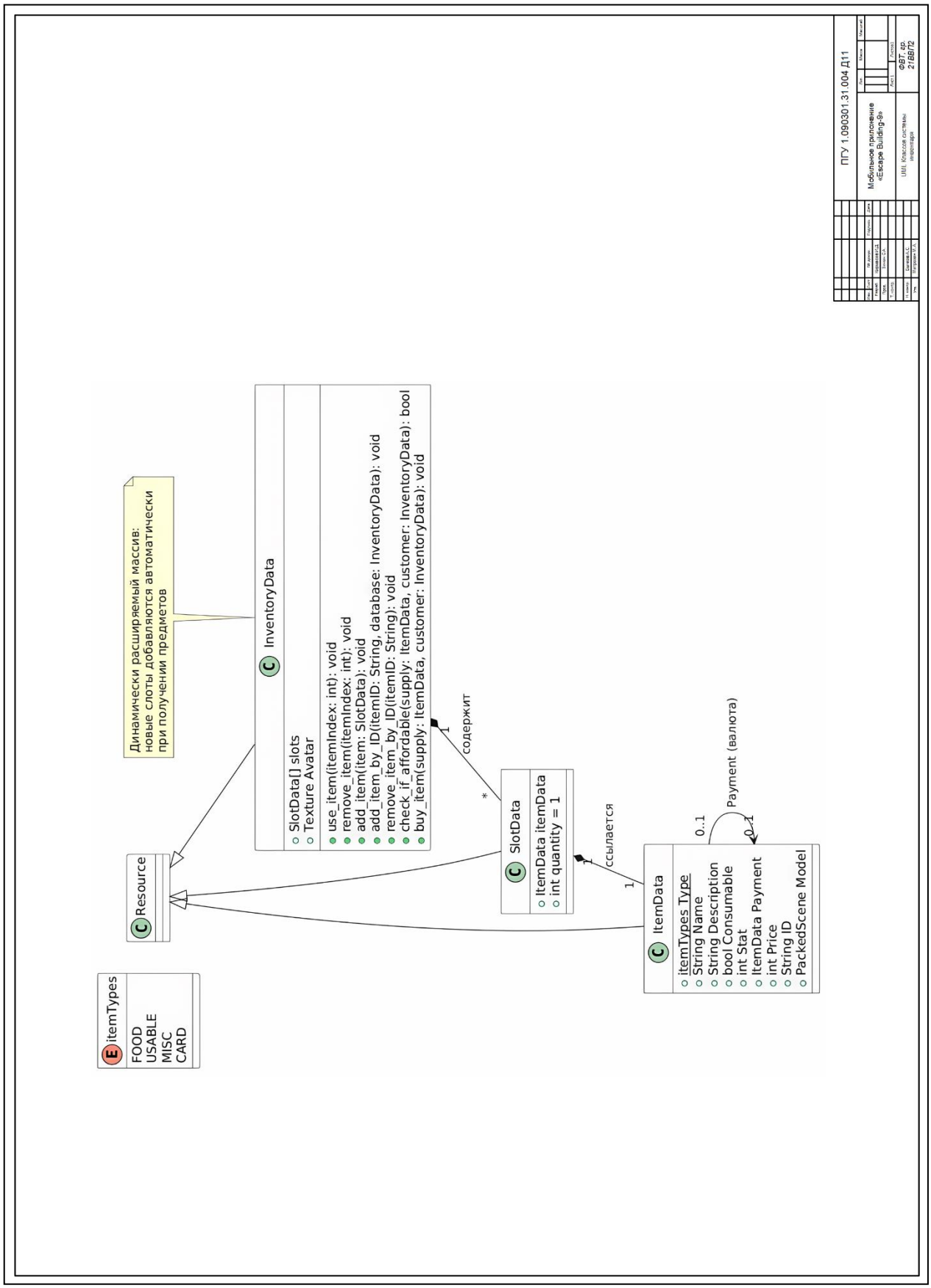
Приложение А.2 – UML – диаграмма прецедентов системы управления персонажем



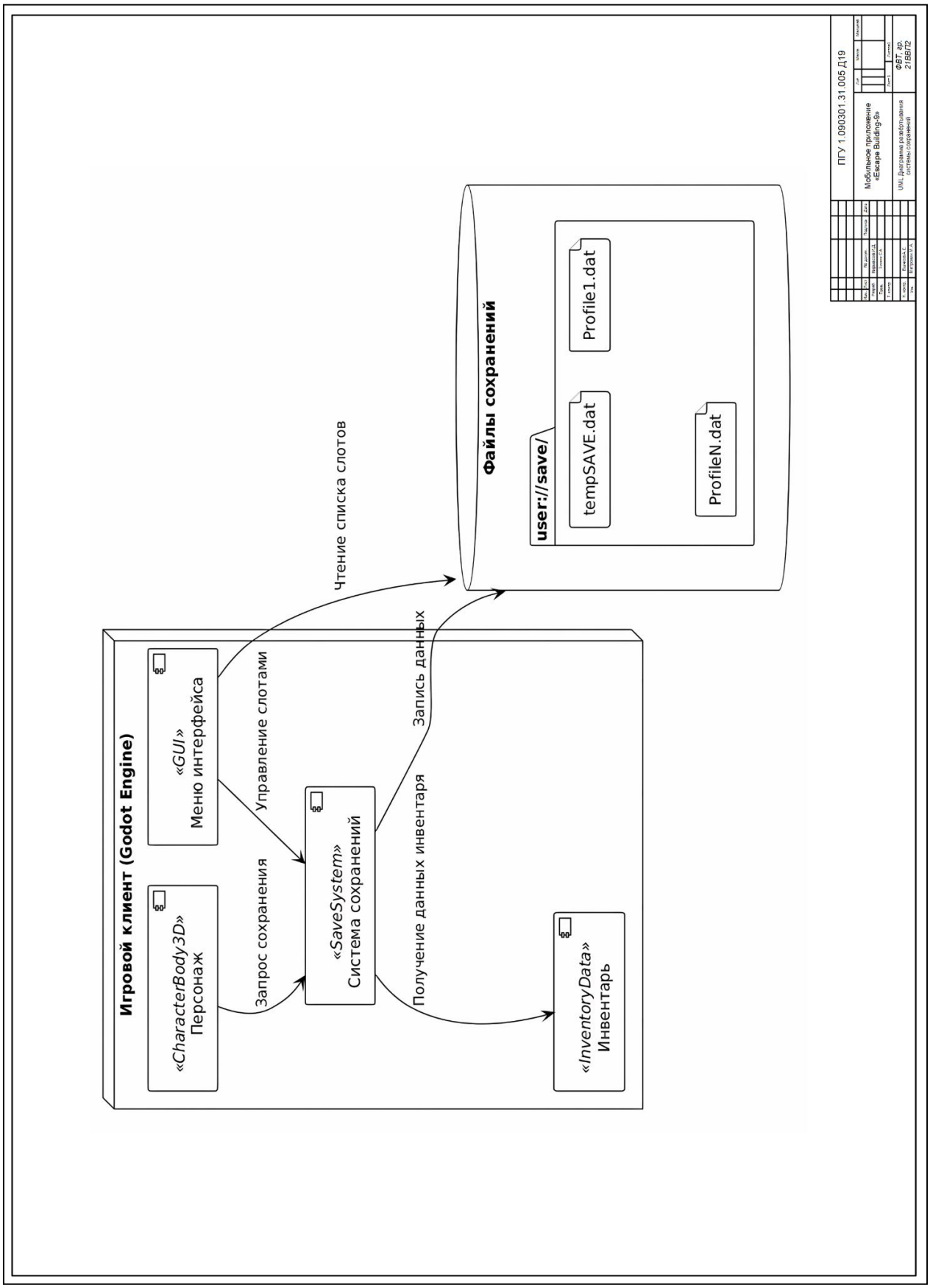
Приложение А.3 – UML – диаграмма деятельности системы инвентаря



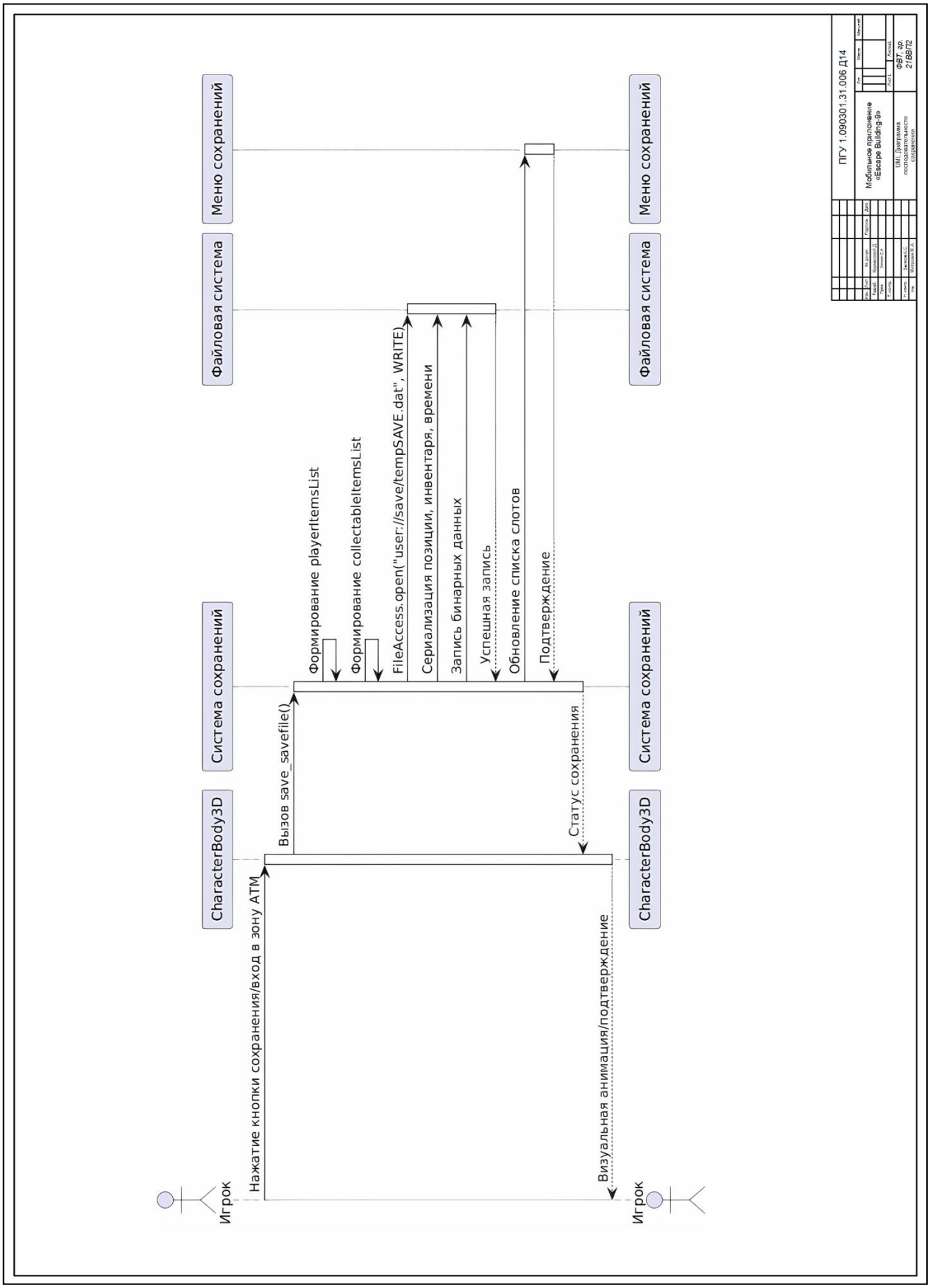
Приложение А.4 – UML – диаграмма классов системы инвентаря



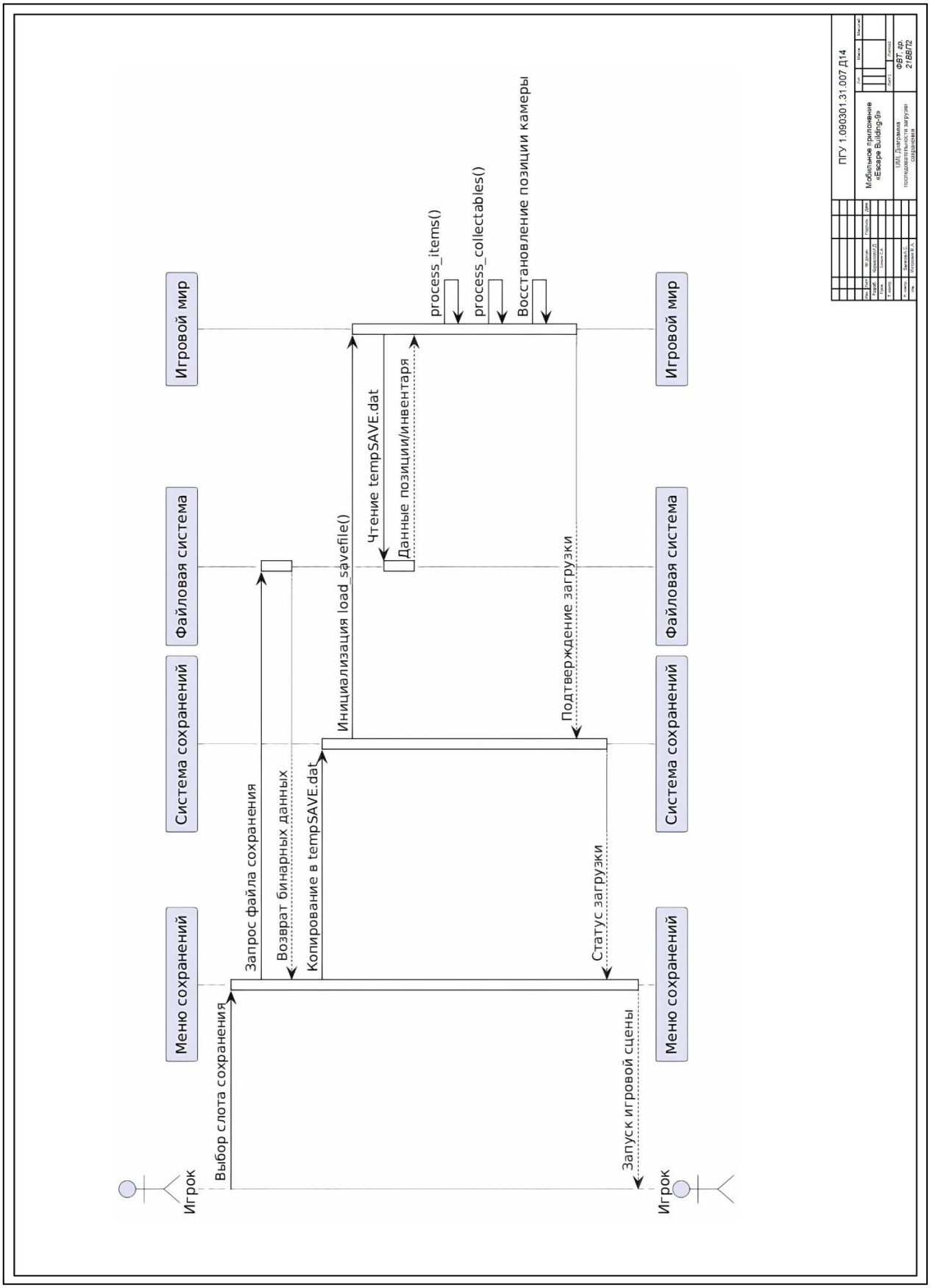
Приложение А.5 – UML – диаграмма развёртывания системы сохранений



Приложение А.6 – UML – диаграмма последовательности сохранения прогресса



Приложение А.7 – UML – диаграмма последовательности загрузки сохранения



Приложение Б

(Обязательное)

Игровое приложение «Escape Building-9»

Листинг программы

Файл «Global.gd»

```
extends Node
var sceneHistory = []
var shaderDisabled: bool = false
var musicEnabled: bool = true
var sensitivity:float = 0.6
var musicVolume: float = 12.0;
var PSXShader: ShaderMaterial =
preload("res://shaders/PSXshader.tres")

var yellowcardObtained = false

func update_music_object(node):
     if node.has_meta("Track"):
          node.volume_db = musicVolume
          if musicEnabled and not node.playing:
               node.playing = true
          elif not musicEnabled:
               node.stream_paused = true
     # Проходим по всем детям текущего узла
     for child in node.get_children():
          update_music_object(child)

func update_viewport_object(node):
     if node.has_meta("MainViewport"):
          if shaderDisabled:
               node.material = null
          else:
               node.material = PSXShader
     # Проходим по всем детям текущего узла
     for child in node.get_children():
          update_viewport_object(child)

func update_settings() -> void:
     update_viewport_object(get_tree().root)
     update_music_object(get_tree().root)
```

Файл «InventoryData.gd»

```
extends Resource
class_name InventoryData

@export var slots: Array[SlotData]
@export var Avatar: Texture

func use_item(itemIndex: int) -> void:
     if slots[itemIndex].itemData.Consumable && slots.size() > 0:
          if slots[itemIndex].quantity > 1:
               slots[itemIndex].quantity -= 1
          elif slots[itemIndex].quantity == 1:
               slots.remove_at(itemIndex)


func remove_item(itemIndex: int) -> void:
     if slots.size() > 0:
```

```
            if slots[itemIndex].quantity > 1:
                slots[itemIndex].quantity -= 1
            elif slots[itemIndex].quantity == 1:
                slots.remove_at(itemIndex)

func add_item(item: SlotData) -> void:
    var found: int = 0
    if slots.size() > 0:
        for i in range(0, slots.size()):
            if item.itemData.Name == slots[i].itemData.Name:
                slots[i].quantity += 1
                found += 1
    if found == 0:
        var tempItem:SlotData
        tempItem = SlotData.new()
        tempItem.itemData = item.itemData
        slots.append(tempItem)

func add_item_by_ID(itemID: String, database: InventoryData):
    for element in database.slots:
        if element.itemData.ID == itemID:
            add_item(element)

func remove_item_by_ID(itemID: String):
    if slots.size() > 0:
        var ct = 0
        for element in slots:
            ct += 1
            if element.itemData.ID == itemID:
                if element.quantity > 1:
                    element.quantity -= 1
                elif element.quantity == 1:
                    slots.remove_at(ct - 1)


func check_if_affordable(supply:ItemData,customer:InventoryData) -
> bool:
    for i in range(0, customer.slots.size()):
        if supply.Payment.Name ==
customer.slots[i].itemData.Name:
            if customer.slots[i].quantity >= supply.Price:
                return true
    return false


func buy_item(supply:ItemData,customer:InventoryData) -> void:
    for i in range(0, customer.slots.size()):
        if supply.Payment.Name ==
customer.slots[i].itemData.Name:
            if customer.slots[i].quantity >= supply.Price:
                customer.slots[i].quantity -= supply.Price
```

Файл «ItemData.gd»

```
extends Resource
class_name ItemData
```

```
enum itemTypes {FOOD, USABLE, MISC, CARD}

@export var Type: itemTypes
@export var Name: String = ""
@export_multiline var Description: String = ""
@export var Consumable: bool
@export var Stat: int = 0
@export var Payment: ItemData
@export var Price: int
@export var Model: PackedScene
@export var ID: String = "Sosiska"
```

Файл «SlotData.gd»

```
extends Resource
class_name SlotData

@export var itemData: ItemData
@export var quantity: int = 1
```

Файл «ShopData.gd»

```
extends Resource

class_name ShopData

enum shopTypes {SHOP, EXCHANGE}

@export var Type: shopTypes
@export var Name: String = ""
@export_multiline var Replica: String = ""
@export var attachedInventory: InventoryData
```

Файл «PlayerControls.gd»

```
extends CharacterBody3D

signal menu_opened
signal menu_closed
signal entered_shop_zone(meta: Resource)
signal exited_shop_zone
@onready var track: AudioStreamPlayer2D = $"../../../Track"
@onready var door_open: AudioStreamPlayer2D = $"../../../DoorOpen"
@onready var door_close: AudioStreamPlayer2D =
$"../../../DoorClose"
@onready var debugging: Label = $"../../../OnScreenUi/Debugging"

##Передвижение
var Speed: float = 0
var correctedSpeed: float
var acceleration = 18.0    # Скорость разгона
@onready var player: CharacterBody3D = $"."
@onready var collider: CollisionShape3D = $CollisionShape3D
var input_dir
var direction

##Инвентарь и меню
var Viewing: bool = true
var inMenu: bool = false
```

```
var tempMeta: Resource
var playerInventory: Resource =
preload("res://data/inventory/PlayerInventory.tres")
var itemDatabase: Resource =
preload("res://data/inventory/Database.tres")
@onready var playerAnimations: AnimationPlayer = $PlayerAnimations
@onready var player_menu: Panel = $"../../../Control/PlayerMenu"
@onready var on_screen_ui: Node2D = $"../../../OnScreenUi"


##Сохранения
var tempSAVE
var newGame: bool
var inSaveZone: bool = false
var lastATM
var profileName: String
var elapsed_time: float
var playerItemsList: String
var collectableItemsList: String

var time_scale: float = 120.0  # 1 реальная секунда = 1 игровая
минута
var current_period: String = ""
var current_location: String = ""
@onready var world_environment: WorldEnvironment =
$"../../../WorldEnvironment"

##Передвижение ползком
@onready var colliderCrouch: CollisionShape3D =
$CollisionShape3DCrouch
const standHeight: float = 0.054 # Высота персонажа в стоячем
положении
const crouchHeight: float = -0.6  # Высота персонажа в присевшем
положении
var isCrouching: bool = false

##Камера
@onready var head: Node3D = $Head
@onready var raycast: RayCast3D = $Head/Camera3D/RayCast3D
@onready var camera: Camera3D = $Head/Camera3D
var x_rotation: float = 0.0
var mouseSensitivity: float

#Взаимодействие с объектами
var collided_item: Node3D
var inShopZone: bool = false
var highlight_material: StandardMaterial3D =
preload("res://shaders/highlight.tres")
var currentPickup: Resource
var tempDialogue: DialogueResource
var nextRoomPos: Vector3
var lastDoor
var lastHeadPos: Vector3
var entering: bool
```

```
@onready var map: Node3D = $"../MAP"
var is_picking_up := false
var picked_item: Node3D = null
var tween: Tween = null

# Добавляем переменные для плавного вращения
var target_y_degrees = 0.0
var current_y_degrees = 0.0
var target_x_degrees = 0.0
var current_x_degrees = 0.0
var smoothness = 20.0  # Коэффициент плавности (можно настроить)

## Настройки гироскопа
var gyro_sensitivity: float = 1.0  # Чувствительность гироскопа
var is_android: bool = OS.get_name() == "Android"

func _ready() -> void:
     current_location = "Hub"

     if Global.sceneHistory[-1] == "res://start_cutscene_1.tscn":
          for npc in get_tree().get_nodes_in_group("npcs"):

               for item in npc.get_meta("StarterInventory").slots:

     npc.get_meta("ShopData").attachedInventory.slots.append(item)

     load_savefile()
     current_y_degrees = rad_to_deg(rotation.y)
     target_y_degrees = current_y_degrees
     current_x_degrees = rad_to_deg($Head.rotation.x)
     target_x_degrees = current_x_degrees

func _physics_process(delta) -> void:
     elapsed_time += delta

     var game_time = elapsed_time * time_scale
     var hours = int(game_time / 3600) % 24
     var new_period = get_time_period(hours)

     if new_period != current_period and current_location ==
"Hub":
          update_environment_and_music(new_period)
          current_period = new_period

     if not is_on_floor():
          velocity += get_gravity() * delta
     # Обработка гироскопа для Android
     if is_android and not inMenu and Viewing:
          var gyro = Input.get_gyroscope()
          # Корректировка осей в зависимости от ориентации экрана
          #target_y_degrees += rad_to_deg(gyro.y ) *
gyro_sensitivity * delta
          #target_x_degrees -= rad_to_deg(gyro.x * (-1)) *
gyro_sensitivity * delta
```

```
          #target_x_degrees = clamp(target_x_degrees, -60, 60)
     if not inMenu:
          # Плавное вращение по горизонтали (Y-axis)
          var delta_y = (target_y_degrees - current_y_degrees) *
smoothness * delta
          current_y_degrees += delta_y
          rotate_y(deg_to_rad(delta_y))
          # Плавное вращение по вертикали (X-axis)
          var delta_x = (target_x_degrees - current_x_degrees) *
smoothness * delta
          current_x_degrees += delta_x
          current_x_degrees = clamp(current_x_degrees, -60, 60)
          $Head.rotation.x = deg_to_rad(current_x_degrees)
          if direction:
               if isCrouching:
                    correctedSpeed = ((float(player_menu.hunger) /
500))
               else:
                    correctedSpeed = ((float(player_menu.hunger) /
250))

               var target_velocity = direction * correctedSpeed
               velocity.x = move_toward(velocity.x,
target_velocity.x, acceleration * delta)
               velocity.z = move_toward(velocity.z,
target_velocity.z, acceleration * delta)
               debugging.text = str(target_velocity)

          else:
               velocity.x = move_toward(velocity.x, 0, 1)
               velocity.z = move_toward(velocity.z, 0, 1)

          move_and_slide()



func _input(event) -> void:
     if not inMenu:
          input_dir = Input.get_vector("ui_left", "ui_right",
"ui_up", "ui_down")
          direction = (transform.basis * Vector3(input_dir.x, 0,
input_dir.y)).normalized()
          #Input.set_mouse_mode(Input.MOUSE_MODE_CAPTURED)

     if event is InputEventScreenDrag and Viewing:
          mouseSensitivity = Global.sensitivity
          # Обновляем целевые углы
          target_y_degrees += -event.relative.x * mouseSensitivity
          target_x_degrees -= event.relative.y * mouseSensitivity
          target_x_degrees = clamp(target_x_degrees, -60, 60)

          #Input.set_mouse_mode(Input.MOUSE_MODE_CAPTURED)

     if event.is_action_pressed("ui_cancel"):
```

```
          Viewing = not Viewing
          Input.set_mouse_mode(Input.MOUSE_MODE_VISIBLE)

     if event.is_action_pressed("ui_text_indent"):
          menu_open_action()

     if event.is_action_pressed("Interact") && nextRoomPos !=
Vector3.ZERO:
          if not inMenu:
               entering = true
               lastHeadPos = head.position
               lastDoor.get_parent().get_child(2).play("Open")
               playerAnimations.play("EnterRoom")
               global_position =
lastDoor.get_parent().get_child(4).global_position

     look_at(lastDoor.get_parent().get_child(5).global_position,
Vector3.UP)
               set_process_input(false)
               velocity = Vector3.ZERO
               direction = Vector3.ZERO

               door_open.play()
               if isCrouching:
                    toggle_crouch()
     if event.is_action_pressed("Interact") && inSaveZone:
          global_position = lastATM.get_child(1).global_position
          look_at(lastATM.get_parent().global_position, Vector3.UP)
          head.rotation = Vector3.ZERO
          velocity = Vector3.ZERO
          direction = Vector3.ZERO
          save_savefile()
          set_process_input(false)
          if isCrouching:
               toggle_crouch()
          playerAnimations.play("EnterATM")

     if event.is_action_pressed("Interact") && inShopZone:
          on_screen_ui.visible = !on_screen_ui.visible
          if not inMenu:
               entered_shop_zone.emit(tempMeta)
               menu_opened.emit()
          else:
               menu_closed.emit()
          inMenu = not inMenu
          Viewing = not Viewing
          Input.set_mouse_mode(Input.MOUSE_MODE_VISIBLE)
     if event.is_action_pressed("Crouch"):
          if not inMenu:
               toggle_crouch()

     if event.is_action_pressed("ActionSpecial"):
          if not inMenu and tempDialogue != null:
```

```
     DialogueManager.show_dialogue_balloon(tempDialogue,"start")

     if event is InputEventScreenTouch and event.pressed and not
is_picking_up and not inMenu:
          var from = camera.project_ray_origin(event.position)
          var to = from + camera.project_ray_normal(event.position)
* 1000.0
          var space_state = get_world_3d().direct_space_state
          var query = PhysicsRayQueryParameters3D.create(from, to)
          var result = space_state.intersect_ray(query)

          if result and result.collider.has_meta("inv") and
is_item_in_range(result.collider):
               start_pickup_animation(result.collider)

func toggle_crouch() -> void:
     isCrouching = not isCrouching
     var height
     var colliderHeight
     if isCrouching:
          height = crouchHeight
     else:
          height = standHeight
     collider.disabled =  not collider.disabled
     start_height_change(head.position.y, height)  # Изменяет
высоту персонажа

func start_height_change(start_height: float, target_height:
float) -> void:
     # Время изменения высоты
     var elapsed_time = 0.0

     while elapsed_time < 0.2:
          elapsed_time += get_process_delta_time()
          var t = elapsed_time / 0.2
          head.position.y = lerp(start_height, target_height, t)
          await get_tree().create_timer(0.01).timeout  # Ждем до
следующего кадра
     head.position.y = target_height  # Убедимся, что конечная
позиция установлена

# Основная функция для обработки строки
func process_items(input_string: String, inventory: Resource,
modeRemove: bool) -> void:
     # Разделяем строку по разделителю "!"
     var items = input_string.split("!")
     for item in items:
          item = item.strip_edges()  # Удаляем пробелы
          if item != "":
               # Используем регулярное выражение для извлечения
названия и количества
               var regex = RegEx.new()
```

```
                regex.compile(r"\[(.+?)\]\[(\d+)\]")  # Шаблон для
поиска
                var match = regex.search(item)
                if match:
                     var item_name =
match.get_string(1).strip_edges()  # Название предмета
                     var quantity = int(match.get_string(2))  #
Количество предметов
                     for _i in range(quantity):
                          if not modeRemove:

     inventory.add_item_by_ID(item_name,itemDatabase)  # Вызываем
функцию добавления предмета
                               else:

     inventory.remove_item_by_ID(item_name)  # Вызываем функцию
добавления предмета

                else:
                     print("Ошибка: некорректный формат для '" +
item + "'")

func process_collectables(input_string: String) -> void:
     # Разделяем строку по разделителю "!"
     var items = input_string.split("!")
     for item in items:
          item = item.strip_edges()  # Удаляем пробелы
          if item != "":
                # Используем регулярное выражение для извлечения
названия и координат
                var regex = RegEx.new()
                regex.compile(r"\[(.+?)\]\[([-
+]?\d*\.\d+|\d+)\]\[([-+]?\d*\.\d+|\d+)\]\[([-+]?\d*\.\d+|\d+)\]")
                var match = regex.search(item)
                if match:
                     var item_name =
match.get_string(1).strip_edges()  # Название предмета
                     var x = float(match.get_string(2))  #
Координата X
                     var y = float(match.get_string(3))  #
Координата Y
                     var z = float(match.get_string(4))  #
Координата Z

                     # Спавн модельки на указанных координатах
                     spawn_collectable(item_name, Vector3(x, y, z))
# Вызов функции спавна
                else:
                     print("Ошибка: некорректный формат для '" +
item + "'")

func spawn_collectable(itemID: String, pos: Vector3) -> void:
     var newCollectable
     match itemID:
```

```
          var id when "Tag" in itemID:
               newCollectable = load("res://models/Tag" + "/" +
itemID + "Collectable.tscn").instantiate()
          var id when "Coffee" in itemID:
               newCollectable = load("res://models/CoffeeCup" + "/"
+ itemID + "Collectable.tscn").instantiate()
          _:
               newCollectable = load("res://models/" + itemID + "/"
+ itemID + "Collectable.tscn").instantiate()
     map.add_child(newCollectable,true)
     newCollectable.global_position = pos

func load_savefile() -> bool:
     tempSAVE = FileAccess.open("user://save/tempSAVE.dat",
FileAccess.READ)
     if tempSAVE != null:
          # Загружаем переменные
          global_position = tempSAVE.get_var()
          global_rotation = tempSAVE.get_var()
          profileName = tempSAVE.get_var()
          elapsed_time = tempSAVE.get_var()
          playerItemsList = tempSAVE.get_var()
          player_menu.Bill = tempSAVE.get_var()
          collectableItemsList = tempSAVE.get_var()
          player_menu.hunger = tempSAVE.get_var()
          var npcInventories = tempSAVE.get_var()  # Инвентари NPC
          tempSAVE.close()

          # Обработка данных
          process_items(playerItemsList, playerInventory, false)
          process_collectables(collectableItemsList)

          # Загружаем инвентари NPC
          if npcInventories:
               for npc in get_tree().get_nodes_in_group("npcs"):
                    var npc_id = npc.get_meta("NPC_ID")  # У NPC
должен быть уникальный id
                    if npcInventories.has(npc_id):
                         process_items(npcInventories[npc_id],
npc.get_meta("ShopData").attachedInventory, false)
          return true
     else:
          return false

func save_savefile() -> bool:
     playerItemsList = ""
     collectableItemsList = ""

     # Формируем список предметов игрока
     for item in playerInventory.slots:
          playerItemsList += "[%s][%d]!" % [item.itemData.ID,
item.quantity]
```

```
     playerItemsList = playerItemsList.rstrip("!")  # Удаляем
последний "!"

     # Формируем список коллектируемых предметов
     for collectable in map.get_children():
          if "Collectable" in collectable.name:
               var newName =
collectable.name.replace("Collectable", "").rstrip("0123456789")
               collectableItemsList += "[%s][%.2f][%.2f][%.2f]!" %
[newName, collectable.global_position.x,
collectable.global_position.y, collectable.global_position.z]
     collectableItemsList = collectableItemsList.rstrip("!")  #
Удаляем последний "!"

     # Формируем инвентари NPC
     var npcInventories = {}
     for npc in get_tree().get_nodes_in_group("npcs"):
          var npcStr = ""
          for item in
npc.get_meta("ShopData").attachedInventory.slots:
               npcStr += "[%s][%d]!" % [item.itemData.ID,
item.quantity]
          npcInventories[npc.get_meta("NPC_ID")] =
npcStr.rstrip("!")  # Сохраняем без последнего "!"
          process_items(npcStr,
npc.get_meta("ShopData").attachedInventory, true)
     process_items(playerItemsList,playerInventory, 1)
     # Сохраняем данные
     tempSAVE = FileAccess.open("user://save/tempSAVE.dat",
FileAccess.WRITE)
     if tempSAVE != null:
          tempSAVE.store_var(global_position)
          tempSAVE.store_var(global_rotation)
          tempSAVE.store_var(profileName)
          tempSAVE.store_var(elapsed_time)
          tempSAVE.store_var(playerItemsList)
          tempSAVE.store_var(player_menu.Bill)
          tempSAVE.store_var(collectableItemsList)
          tempSAVE.store_var(player_menu.hunger)
          tempSAVE.store_var(npcInventories)
          tempSAVE.close()
          return true
     else:
          return false

func get_time_period(hours: int) -> String:
     if hours >= 6 and hours < 12:
          return "morning"
     elif hours >= 12 and hours < 18:
          return "day"
     elif hours >= 18 and hours < 24:
          return "evening"
     else:
          return "night"
```

```
func update_environment_and_music(period: String):
     var ambient_color: Color
     var sky_color: Color
     var music_stream: AudioStream
     var fog_color: Color
     var fog_density: float

     match period:
          "morning":
               # Туманное холодное утро с синевой
               ambient_color = Color(0.15, 0.15, 0.2)
               sky_color = Color(0.2, 0.25, 0.3)
               fog_color = Color(0.3, 0.35, 0.4)
               fog_density = 0.02
               music_stream =
preload("res://audio/tracks/Podval.mp3")

          "day":
               # Мрачный "дневной" свет с зеленоватым оттенком
               ambient_color = Color(0.25, 0.3, 0.25)
               sky_color = Color(0.15, 0.2, 0.15)
               fog_color = Color(0.4, 0.45, 0.4)
               fog_density = 0.015
               music_stream =
preload("res://audio/tracks/CorridorStruggle.mp3")

          "evening":
               # Кроваво-красный закат с глубокими тенями
               ambient_color = Color(0.25, 0.1, 0.1)
               sky_color = Color(0.35, 0.15, 0.15)
               fog_color = Color(0.5, 0.2, 0.2)
               fog_density = 0.025
               music_stream =
preload("res://audio/tracks/ChaosDarkening.mp3")

          "night":
               # Почти полная темнота с синим оттенком
               ambient_color = Color(0.05, 0.05, 0.1)
               sky_color = Color(0.02, 0.02, 0.05)
               fog_color = Color(0.1, 0.1, 0.15)
               fog_density = 0.03
               music_stream =
preload("res://audio/tracks/Dialog.mp3")

     # Настройка окружения
     var env = world_environment.environment
     var tween = create_tween().set_parallel(true)

     # Основные параметры
     tween.tween_property(env, "ambient_light_color",
ambient_color, 5.0)
     tween.tween_property(env, "background_color", sky_color, 5.0)
```

```
     # Настройки тумана
     tween.tween_property(env, "fog_light_color", fog_color, 5.0)
     tween.tween_property(env, "fog_density", fog_density, 5.0)

     # Дополнительные эффекты для хоррора
     tween.tween_property(env, "tonemap_exposure", 0.8 if period
== "night" else 1.0, 5.0)
     tween.tween_property(env, "ssao_enabled", period == "night",
5.0)

     # Смена музыки
     if track.stream != music_stream:
          change_music(music_stream)

func change_music(new_stream: AudioStream):
     var fade_time = 1.0
     # Фейд-аут текущего трека
     var tween_out = create_tween()
     tween_out.tween_property(track, "volume_db", -20.0,
fade_time)
     await tween_out.finished
     track.stop()

     # Фейд-ин нового трека
     track.stream = new_stream
     track.volume_db = -20.0
     track.play()
     var tween_in = create_tween()
     tween_in.tween_property(track, "volume_db", 8.0, fade_time)

func onItemThrown(item_instance: RigidBody3D) -> void:
     item_instance.position = global_position
     map.add_child(item_instance,true)  # Добавляем экземпляр в
текущую сцену

func _on_area_3d_area_entered(area: Area3D) -> void:
     if area.name == "ShopZone":
          inShopZone = true
          tempMeta = area.get_meta("ShopData")
     if area.has_meta("dialogueData"):
          tempDialogue = area.get_meta("dialogueData")
     if area.has_meta("nextRoom"):
          nextRoomPos = area.get_meta("nextRoom")
          lastDoor = area
     if area.name == "SaveZone":
          lastATM = area
          inSaveZone = true

func _on_area_3d_area_exited(area: Area3D) -> void:
     if area.name == "ShopZone":
          inShopZone = false
     if area.has_meta("dialogueData"):
          tempDialogue = null
     if area.has_meta("nextRoom") && not entering:
```

```
            nextRoomPos = Vector3.ZERO
            print("reset")
        if area.name == "SaveZone":
            inSaveZone = false

func _on_player_animations_animation_finished(anim_name:
StringName) -> void:
        if anim_name == "EnterATM":
            get_tree().change_scene_to_file("res://save_menu.tscn")
        if anim_name == "EnterRoom":
            playerAnimations.play("Default")
            lastDoor.get_parent().get_child(2).stop()
            set_process_input(true)
            global_position = nextRoomPos
            entering = false
            nextRoomPos = Vector3.ZERO
            head.position = lastHeadPos
            match lastDoor.get_meta("nextTrack"):
                "Office":
                    track.stream =
load("res://audio/tracks/Office.mp3")
                    Global.update_music_object(get_tree().root)
                    current_location = "Office"

                "Hub":
                    track.stream =
load("res://audio/tracks/Dialog.mp3")
                    Global.update_music_object(get_tree().root)
                    current_location = "Hub"
                _:
                    track.stop()
            door_close.play()

func menu_open_action() -> void:
            on_screen_ui.visible = !on_screen_ui.visible
            if not inMenu:
                exited_shop_zone.emit()
                menu_opened.emit()
            else:
                menu_closed.emit()
            inMenu = not inMenu
            Viewing = not Viewing
            Input.set_mouse_mode(Input.MOUSE_MODE_VISIBLE)

func start_pickup_animation(item: Node3D):
    is_picking_up = true
    picked_item = item

    # Отключаем взаимодействие
    picked_item.set_collision_layer(0)
    picked_item.set_collision_mask(0)

    # Настройка анимации
```

```
     var target_position = global_transform.origin +
global_transform.basis.z

     tween = create_tween().set_parallel(true)
     tween.tween_property(picked_item, "global_position",
target_position, 0.8)\
          .set_trans(Tween.TRANS_BACK).set_ease(Tween.EASE_OUT)

     tween.finished.connect(func():

     playerInventory.add_item(picked_item.get_meta("inv").slots[0]
)
          player_menu.recalculate_bill()
          picked_item.queue_free()
          is_picking_up = false
          picked_item = null
     )

func is_item_in_range(item: Node3D) -> bool:
     var camera_pos = camera.global_transform.origin
     return item.global_position.distance_to(camera_pos) <= 2.0
```

Файл «PlayerMenu.gd»

```
extends Panel

enum MenuMode {INVENTORY, SHOP, EXCHANGE, OPTIONS, TUTORIAL}

@onready var control: Control = $".."
@onready var labelitemname: Label =
$PlayerMenuGeneralContainer/VBoxContainer2/ITEMUICONTAINER/LABELIT
EMNAME
@onready var descriptiontext: RichTextLabel =
$PlayerMenuGeneralContainer/VBoxContainer2/DESCRIPTIONUICONTAINER/
MarginContainer/HBoxContainer/DESCRIPTIONTEXT
@onready var buttonlabel: Label =
$PlayerMenuGeneralContainer/VBoxContainer2/TextureButton/BUTTONLAB
EL
@onready var place_holder: Node3D =
$PlayerMenuGeneralContainer/VBoxContainer2/ITEMUICONTAINER/HBoxCon
tainer/SubViewportContainer/SubViewport/PlaceHolder
@onready var itemuicontainer: VBoxContainer =
$PlayerMenuGeneralContainer/VBoxContainer2/ITEMUICONTAINER
@onready var descriptionuicontainer: HBoxContainer =
$PlayerMenuGeneralContainer/VBoxContainer2/DESCRIPTIONUICONTAINER
@onready var texture_button: TextureButton =
$PlayerMenuGeneralContainer/VBoxContainer2/TextureButton
@onready var texture_rect: TextureRect =
$PlayerMenuGeneralContainer/VBoxContainer2/TextureRect
@onready var label: Label =
$PlayerMenuGeneralContainer/VBoxContainer2/STATUSUICONTAINER/HBoxC
ontainer/VBoxContainer/Label
@onready var label_2: RichTextLabel =
$PlayerMenuGeneralContainer/VBoxContainer2/STATUSUICONTAINER/HBoxC
ontainer/VBoxContainer/Label2
```

```
@onready var labelitems: Label =
$PlayerMenuGeneralContainer/VBoxContainer2/ITEMUICONTAINER/LABELIT
EMS
@onready var texture_button_2: TextureButton =
$PlayerMenuGeneralContainer/VBoxContainer2/ITEMUICONTAINER/Texture
Button2
@onready var track: AudioStreamPlayer2D =
get_parent().get_parent().get_child(4)
@onready var player_menu_general_container: MarginContainer =
$PlayerMenuGeneralContainer
@onready var settings_button: TouchScreenButton = $SettingsButton
@onready var options: Control = $"../Options"
@onready var tutorial_button: TouchScreenButton = $TutorialButton
@onready var tutorial_prev: Control = $"../TutorialPrev"
@onready var preview_tutorial: VideoStreamPlayer =
$"../TutorialPrev/PreviewTutorial"
@onready var playerRef: CharacterBody3D =
get_node("../../SubViewportContainer/SubViewport/Player")
@onready var confirm_exit: ColorRect = $"../Options/ConfirmExit"

var Bill: int = 1000
var hunger: int = 1000
var hungerTimer: Timer
var tempSAVE
var profileName
var shopData: Resource
var playerInventory: Resource =
preload("res://data/inventory/PlayerInventory.tres")
var tutorialInventory: Resource =
preload("res://data/inventory/TutorialInventory.tres")

var itemDatabase: Resource =
preload("res://data/inventory/Database.tres")
var currentInventory: Resource
var tempModel : Node
var currentSlot: int = 0
var inMenu: bool = false
var current_mode_handler: MenuModeHandler
signal itemThrown(item_instance: RigidBody3D)
signal yellow_card_obtained

class MenuModeHandler:
     var panel: Control
     var menu: Control

     func _init(menu_panel: Control, menu_instance: Control):
          panel = menu_panel
          menu = menu_instance

     func enter() -> void: pass
     func exit() -> void: pass
     func handle_input(event: InputEvent) -> void: pass
     func update_display() -> void: pass
     func handle_action() -> void: pass
```

```
class InventoryHandler extends MenuModeHandler:
     func enter() -> void:
          menu.currentInventory = menu.playerInventory
          menu.label.text = str(menu.profileName)
          menu.label_2.text = "Голод: " + str(menu.hunger / 10)
          var slot_count = menu.currentInventory.slots.size()
          menu.currentSlot = clamp(menu.currentSlot, 0, max(0,
slot_count - 1))
          menu.texture_button_2.visible = true
          menu.settings_button.visible = true
          menu.tutorial_button.visible = true
     func handle_input(event: InputEvent) -> void:
          if event.is_action_pressed("ui_right"):
               menu.currentSlot = wrapi(menu.currentSlot + 1, 0,
max(1, menu.currentInventory.slots.size()))
          elif event.is_action_pressed("ui_left"):
               menu.currentSlot = wrapi(menu.currentSlot - 1, 0,
max(1, menu.currentInventory.slots.size()))

     func update_display() -> void:
          menu.texture_button.disabled = false
          var slot_count = menu.currentInventory.slots.size()
          menu.labelitems.text = "ВЕЩИ (%d/%d)" % [menu.currentSlot
+ 1 if slot_count > 0 else 0, slot_count]
          menu._update_common_display()
          menu.texture_button.self_modulate =
Color(1.991,0.68,0.24,1)

     menu.buttonlabel.add_theme_font_size_override("font_size",
51)
          if menu.currentInventory.slots.size() == 0:
               return
          var currentType =
menu.currentInventory.slots[menu.currentSlot].itemData.Type
          match currentType:
               0: menu.buttonlabel.text = "Съесть"
               1: menu.buttonlabel.text = "Использовать"
               _:
                    menu.texture_button.disabled = true
                    menu.buttonlabel.text = ""

     func handle_action() -> void:
          if menu.currentInventory.slots.size() <=
menu.currentSlot:
               return
          if
menu.currentInventory.slots[menu.currentSlot].itemData.Type == 0:
               menu.hunger +=
menu.currentInventory.slots[menu.currentSlot].itemData.Stat
               menu.hunger = clamp(menu.hunger,0,1000)
          menu.currentInventory.use_item(menu.currentSlot)

class ShopHandler extends MenuModeHandler:
```

```
	func enter() -> void:
		menu.currentInventory = menu.shopData.attachedInventory
		menu.label.text = menu.shopData.Name
		menu.label_2.text = menu.shopData.Replica
		menu.labelitems.text = "АССОРТИМЕНТ"
		menu.texture_button_2.visible = false
		menu.settings_button.visible = false
		menu.tutorial_button.visible = false

	func update_display() -> void:
		if menu.currentInventory.slots.is_empty():
			menu.texture_button.disabled = true
			menu.buttonlabel.text = ""
			return
		menu._update_common_display()

	menu.buttonlabel.add_theme_font_size_override("font_size",
51)
		if
menu.currentInventory.check_if_affordable(menu.currentInventory.sl
ots[menu.currentSlot].itemData, menu.playerInventory):
			menu.texture_button.disabled = false
			menu.buttonlabel.text = "Купить %d₽" %
menu.currentInventory.slots[menu.currentSlot].itemData.Price
			menu.texture_button.self_modulate =
Color(0.61,0.68,0.72,1)


		else:
			menu.texture_button.disabled = true
			menu.buttonlabel.text = "Нужно %d₽" %
menu.currentInventory.slots[menu.currentSlot].itemData.Price


	func handle_action() -> void:
		if menu.currentInventory.slots.is_empty():
			return

		var original_size = menu.currentInventory.slots.size()

		menu.Bill -=
menu.currentInventory.slots[menu.currentSlot].itemData.Price

	menu.currentInventory.buy_item(menu.currentInventory.slots[me
nu.currentSlot].itemData, menu.playerInventory)

	menu.playerInventory.add_item(menu.currentInventory.slots[men
u.currentSlot])
		menu.currentInventory.remove_item(menu.currentSlot)

		# Корректировка слота
		if original_size > 1:
			menu.currentSlot = clamp(menu.currentSlot, 0,
menu.currentInventory.slots.size() - 1)
```

```
            else:
                menu.currentSlot = 0

            menu.recalculate_bill()
            menu.update_menu()
            menu._update_common_display()

class ExchangeHandler extends MenuModeHandler:
    func enter() -> void:
            menu.currentInventory = menu.shopData.attachedInventory
            menu.label.text = menu.shopData.Name
            menu.label_2.text = menu.shopData.Replica
            menu.labelitems.text = "Обменять"
            menu.tutorial_button.visible = false
            menu.texture_button_2.visible = false
            menu.settings_button.visible = false

    func update_display() -> void:
            if menu.currentInventory.slots.is_empty():
                menu.labelitems.text = "Обменять"
                menu.texture_button.disabled = true
                return
            menu._update_common_display()
            if
menu.currentInventory.check_if_affordable(menu.currentInventory.sl
ots[menu.currentSlot].itemData, menu.playerInventory):
                menu.labelitems.text = "Обменять %s" %
menu.currentInventory.slots[menu.currentSlot].itemData.Payment.Nam
e
                menu.labelitemname.text = "на " +
menu.labelitemname.text
                menu.buttonlabel.text = "Обменять"
                menu.texture_button.self_modulate =
Color(0.67,0.2,0.2,1)
                menu.texture_button.disabled = false

    menu.buttonlabel.add_theme_font_size_override("font_size",
51)

            else:
                menu.texture_button.disabled = true
                menu.buttonlabel.text = "Нужен %s" %
menu.currentInventory.slots[menu.currentSlot].itemData.Payment.Nam
e

    menu.buttonlabel.add_theme_font_size_override("font_size",
14)

    func handle_action() -> void:
            for i in menu.playerInventory.slots:
                if i.itemData.Name ==
menu.currentInventory.slots[menu.currentSlot].itemData.Payment.Nam
e:
```

```
     menu.playerInventory.remove_item(menu.playerInventory.slots.f
ind(i))
          if
menu.currentInventory.slots[menu.currentSlot].itemData.ID ==
"Money":
               menu.Bill +=
menu.currentInventory.slots[menu.currentSlot].itemData.Stat
               menu.recalculate_bill()
          else:

     menu.playerInventory.add_item(menu.currentInventory.slots[men
u.currentSlot])
          menu.currentInventory.remove_item(menu.currentSlot)
          menu._update_common_display()
          menu.check_quest_items()

class OptionsHandler extends MenuModeHandler:
     func enter() -> void:
          menu.player_menu_general_container.visible = false
          menu.options.visible = true
          menu.settings_button.visible = false
          menu.tutorial_button.visible = false

     func exit() -> void:
          menu.player_menu_general_container.visible = true
          menu.options.visible = false
          menu.settings_button.visible = true
          menu.tutorial_button.visible = true

class TutorialHandler extends MenuModeHandler:
     func enter() -> void:
          menu.currentInventory = menu.tutorialInventory
          menu.label.text = "Обучение"
          menu.label_2.text = "Нажимайте на стрелки по обе стороны,
чтобы переключать подсказки."
          var slot_count = menu.currentInventory.slots.size()
          menu.currentSlot = clamp(menu.currentSlot, 0, max(0,
slot_count - 1))
          menu.texture_button_2.visible = false
          menu.settings_button.visible = false
          menu.tutorial_button.visible = false
          menu.tutorial_prev.visible = false
          menu.player_menu_general_container.visible = true
     func handle_input(event: InputEvent) -> void:
          if event.is_action_pressed("ui_right"):
               menu.currentSlot = wrapi(menu.currentSlot + 1, 0,
max(1, menu.currentInventory.slots.size()))
          elif event.is_action_pressed("ui_left"):
               menu.currentSlot = wrapi(menu.currentSlot - 1, 0,
max(1, menu.currentInventory.slots.size()))

     func update_display() -> void:
          menu.texture_button.disabled = false
```

```
          var slot_count = menu.currentInventory.slots.size()
          menu.labelitems.text = "ПОДСКАЗКИ (%d/%d)" %
[menu.currentSlot + 1 if slot_count > 0 else 0, slot_count]
          menu._update_common_display()
          menu.texture_button.self_modulate = Color("ac9600")

     menu.buttonlabel.add_theme_font_size_override("font_size",
51)
          if menu.currentInventory.slots.size() == 0:
               return
          var currentType =
menu.currentInventory.slots[menu.currentSlot].itemData.Type
          match currentType:
               1: menu.buttonlabel.text = "Посмотреть"
               _:
                    menu.texture_button.disabled = true
                    menu.buttonlabel.text = ""

     func handle_action() -> void:
          if menu.currentInventory.slots.size() <=
menu.currentSlot:
               return
          if
menu.currentInventory.slots[menu.currentSlot].itemData.Type == 1:

     menu.preview_tutorial.change_video("res://tutorials/" +
menu.currentInventory.slots[menu.currentSlot].itemData.ID +
".ogv")
               menu.player_menu_general_container.visible = false
               menu.tutorial_prev.visible = true
               menu.settings_button.visible = false
          menu.tutorial_button.visible = false

func change_mode(new_mode: MenuMode) -> void:
     if current_mode_handler:
          current_mode_handler.exit()

     match new_mode:
          MenuMode.INVENTORY: current_mode_handler =
InventoryHandler.new(self, self)
          MenuMode.SHOP: current_mode_handler =
ShopHandler.new(self, self)
          MenuMode.EXCHANGE: current_mode_handler =
ExchangeHandler.new(self, self)
          MenuMode.OPTIONS: current_mode_handler =
OptionsHandler.new(self, self)
          MenuMode.TUTORIAL: current_mode_handler =
TutorialHandler.new(self, self)


          _: push_error("Unhandled menu mode")

     current_mode_handler.enter()
     currentSlot = 0
```

```
     update_menu()

func _ready() -> void:
     hungerTimer = Timer.new()
     add_child(hungerTimer)
     hungerTimer.wait_time = 5
     hungerTimer.timeout.connect(update_hunger_status.bind(-2))
     hungerTimer.start()
     change_mode(MenuMode.INVENTORY)
     tempSAVE = FileAccess.open("user://save/tempSAVE.dat",
FileAccess.READ)
     for i in range(8):
          var parseVar = tempSAVE.get_var()
          if i == 2:
               profileName = parseVar
          if i == 5:
               Bill = parseVar
          if i == 7:
               hunger = parseVar
     tempSAVE.close()
     recalculate_bill()
     itemThrown.connect(playerRef.onItemThrown)

func _input(event) -> void:
     if inMenu:
          control.visible = true
          if event.is_action_pressed("ui_right"):
               currentSlot = wrapi(currentSlot + 1, 0, max(1,
currentInventory.slots.size()))
               update_menu()
          elif event.is_action_pressed("ui_left"):
               currentSlot = wrapi(currentSlot - 1, 0, max(1,
currentInventory.slots.size()))
               update_menu()
     else:
          control.visible = false

func update_menu() -> void:
     current_mode_handler.update_display()

func _update_item_model(slot: SlotData) -> void:
     for n in place_holder.get_children():
          place_holder.remove_child(n)
          n.queue_free()

     if slot == null:
          return

     tempModel = slot.itemData.Model.instantiate()
     for n in tempModel.get_children():
          if n.name == "Tag":
               n.text = str(slot.itemData.Stat)
     place_holder.add_child(tempModel)
     tempModel.global_position = place_holder.global_position
```

```
func _update_common_display() -> void:
     texture_rect.texture = currentInventory.Avatar
     currentSlot = clamp(currentSlot, 0, max(0,
currentInventory.slots.size() - 1))

     if currentSlot >= 0 and currentInventory.slots.size() > 0:
          itemuicontainer.visible = true
          descriptionuicontainer.visible = true
          var slot = currentInventory.slots[currentSlot]
          descriptiontext.text = slot.itemData.Description
          labelitemname.text = slot.itemData.Name

          if slot.quantity > 1:
               if slot.itemData.Type == 3:
                    labelitemname.text += " %dруб." % slot.quantity
               else:
                    labelitemname.text += " x%d" % slot.quantity

          _update_item_model(slot)
     else:
          itemuicontainer.visible = false
          texture_button.disabled = true
          buttonlabel.text = ""
          descriptionuicontainer.visible = false

func update_hunger_status(num:int) -> void:
     hunger += num
     hunger = clamp(hunger,0,1000)
     var newPitch: float = (hunger * 0.001)
     track.pitch_scale = newPitch
     update_menu()

func recalculate_bill() -> void:
     for items in playerInventory.slots:
          if items.itemData.ID == "Card" and items.quantity < Bill:
               for coin in range(Bill - items.quantity):
                    playerInventory.add_item(items)

func _on_texture_button_button_down(arg1) -> void:
     get_node(arg1).get_child(0).position.y += 2

func _on_texture_button_button_up(arg1) -> void:
     get_node(arg1).get_child(0).position.y -= 2

func _on_player_menu_closed() -> void:
     inMenu = not inMenu

func _on_player_menu_opened() -> void:
     inMenu = not inMenu

func _on_texture_button_pressed() -> void:
     current_mode_handler.handle_action()
     update_menu()
```

```
func _on_player_entered_shop_zone(meta: Resource) -> void:
     shopData = meta
     change_mode(MenuMode.SHOP if shopData.Type == 0 else
MenuMode.EXCHANGE)
     _update_common_display()

func _on_player_exited_shop_zone() -> void:
     change_mode(MenuMode.INVENTORY)
     _update_common_display()

func check_quest_items() -> void:
     for items in playerInventory.slots:
          if items.itemData.ID == "YellowCard":
               emit_signal("yellow_card_obtained")

func _on_texture_button_2_pressed() -> void:
     if currentInventory.slots.is_empty():
          return
     var throwItemModelPath =
currentInventory.slots[currentSlot].itemData.ID
     var item_instance
     if throwItemModelPath.contains("Tag"):
          var number_part
          var tag_part
          for i in range(throwItemModelPath.length()):
               if throwItemModelPath[i].is_valid_int():
                    number_part = throwItemModelPath.substr(i,
throwItemModelPath.length() - i)
                    tag_part = throwItemModelPath.substr(0, i)
                    item_instance = load("res://models/" + tag_part
+ "/" + tag_part + number_part +"Collectable.tscn").instantiate()
                    break
     else:
          match throwItemModelPath:
               var path when "Coffee" in throwItemModelPath:
                    item_instance = load("res://models/CoffeeCup/"
+ throwItemModelPath + "Collectable.tscn").instantiate()
               _:
                    item_instance = load("res://models/" +
throwItemModelPath + "/" + throwItemModelPath +
"Collectable.tscn").instantiate()

     itemThrown.emit(item_instance)
     if currentInventory.slots[currentSlot].itemData.ID == "Card":
          for i in range(Bill):
               currentInventory.remove_item(currentSlot)
     else:
          currentInventory.remove_item(currentSlot)
     update_menu()

func _on_texture_button_exit_pressed() -> void:
     playerRef.menu_open_action()
```

```
func _on_back_button_pressed() -> void:
	change_mode(MenuMode.INVENTORY)
	_update_common_display()

func _on_music_mode_button_toggled(toggled_on: bool) -> void:
	Global.musicEnabled = toggled_on
	Global.update_music_object(get_tree().root)

func _on_low_end_mode_button_toggled(toggled_on: bool) -> void:
	Global.shaderDisabled = toggled_on
	Global.update_viewport_object(get_tree().root)

func _on_to_main_menu_button_pressed() -> void:
	playerInventory.slots.clear()
	Global.sceneHistory.append("res://MenuScene.tscn")
	get_tree().change_scene_to_file("res://MenuScene.tscn")

func _on_settings_button_released() -> void:
	change_mode(MenuMode.OPTIONS)

func _on_tutorial_button_released() -> void:
	change_mode(MenuMode.TUTORIAL)


func _on_back_tutor_button_pressed() -> void:
	preview_tutorial.stop()
	change_mode(MenuMode.TUTORIAL)


func _on_back_to_settings_button_pressed() -> void:
	confirm_exit.visible = false


func _on_to_confirm_exit_button_pressed() -> void:
	confirm_exit.visible = true
```

Файл «save_menu.gd»

```
extends Control

@export var SaveSlot: PackedScene
@export var SaveSlotRemoveButton: PackedScene
@export var SaveSlotUnit: PackedScene
@onready var v_box_container: VBoxContainer =
$MarginContainer/HBoxContainer/ScrollContainer/VBoxContainer
@onready var back_button: Button =
$MarginContainer/HBoxContainer/HBoxContainer/BackButton
@onready var load_button: Button =
$MarginContainer/HBoxContainer/LoadButton
@onready var animation_player: AnimationPlayer = $AnimationPlayer
@onready var save_button: Button =
$MarginContainer/HBoxContainer/SaveButton
@onready var sub_viewport_container: SubViewportContainer =
$SubViewportContainer

var pathPrefix = "user://save/"
```

```
var currentSlotButton: Button
var saveMode: String = "NewSlot"
var transCall: Callable
func _ready() -> void:
     load_button.disabled = true
     set_process_input(true)
     Input.set_mouse_mode(Input.MOUSE_MODE_VISIBLE)
     if Global.sceneHistory[-1] == "res://MenuScene.tscn":
          back_button.visible = false
          save_button.visible = false
     load_save_slots()  # Загружаем слоты сохранений при открытии
меню
     Global.update_settings()


func _process(delta: float) -> void:
          if currentSlotButton and  not
currentSlotButton.button_pressed:
               currentSlotButton = null
               load_button.disabled = true

func load_save_slots() -> void:
     var dir = DirAccess.open(pathPrefix)  # Открываем директорию
сохранений
     if dir:  # Проверяем, успешно ли открыта директория
          dir.list_dir_begin()
          var file_name = dir.get_next()  # Получаем первое имя
файла
          while file_name != "":
               if file_name != "tempSAVE.dat" and
file_name.ends_with(".dat"):  # Проверяем, является ли файл файлом
сохранения
                    create_save_slot_button(pathPrefix + file_name,
"Load")
               file_name = dir.get_next()  # Получаем следующее имя
файла


func get_slotButton(slotButton: Button) -> void:
     load_button.disabled = false
     currentSlotButton = slotButton
     var container = currentSlotButton.get_parent().get_parent()
     if currentSlotButton.button_pressed:
          saveMode = "Overwrite"

     container.get_parent().get_parent().get_child(1).get_child(0)
.text = "ПЕРЕЗАПИСАТЬ"
     else:
          saveMode = "NewSlot"

     container.get_parent().get_parent().get_child(1).get_child(0)
.text ="НОВОЕ
          СОХРАНЕНИЕ"
```

```
     for element in container.get_children():
          if element.get_child(0) != currentSlotButton:
               element.get_child(0).button_pressed = false

func copy_tempSave_to(file_name: String) -> void:
          var tempSAVE =
FileAccess.open("user://save/tempSAVE.dat", FileAccess.READ)
          var newSAVE = FileAccess.open(file_name,
FileAccess.WRITE_READ)
          var line =
FileAccess.get_file_as_bytes("user://save/tempSAVE.dat")
          newSAVE.store_buffer(line)
          tempSAVE.close()
          newSAVE.close()

# Функция для поиска максимального индекса существующих файлов
func get_max_save_index(base_name: String) -> int:
     var max_counter = 0
     var dir = DirAccess.open(pathPrefix)
     if dir:
          dir.list_dir_begin()
          while true:
               var file = dir.get_next()
               if file == "":
                    break
               if file.begins_with(base_name + " - ") and
file.ends_with(".dat"):
                    var number = file.get_slice(" - ", 1).to_int()
                    if number > max_counter:
                         max_counter = number
     max_counter += 1
     return max_counter

func create_save_slot_button(file_name: String, save_mode:
String):
     if saveMode == "NewSlot" or saveMode == "Load":
          var newSaveSlotUnit = SaveSlotUnit.instantiate()
          if save_mode == "NewSlot":
               copy_tempSave_to(file_name)
          newSaveSlotUnit.get_child(0).set_meta("newSavePath",
file_name)
          var newName = file_name.get_slice("/", 3)
          newName = newName.get_basename()

     newSaveSlotUnit.get_child(0).get_child(0).get_child(0).get_ch
ild(1).text = newName

     newSaveSlotUnit.get_child(0).get_child(0).get_child(1).text =
get_profile_time(file_name)
          v_box_container.add_child(newSaveSlotUnit)
          v_box_container.move_child(newSaveSlotUnit,0)
     else:
          # Получаем имя файла из метаданных кнопки
          file_name = currentSlotButton.get_meta("newSavePath")
```

```
          # Проверяем, существует ли файл
          if FileAccess.file_exists(file_name):
               # Удаляем файл, если он существует
               DirAccess.remove_absolute(file_name)
          # Генерируем новое имя файла
          file_name = pathPrefix + get_profile_name() + ".dat"

          var base_name = get_profile_name()
          var new_file_name = file_name
          # Генерируем уникальное имя файла
          new_file_name = pathPrefix + base_name + " - " +
str(get_max_save_index(base_name)) + ".dat"
          # Копируем временный файл в новое имя
          copy_tempSave_to(new_file_name)
          # Обновляем метаданные кнопки
          currentSlotButton.set_meta("newSavePath", new_file_name)

     currentSlotButton.get_child(0).get_child(0).get_child(1).text
= new_file_name.get_slice("/", 3).get_basename()
          currentSlotButton.get_child(0).get_child(1).text =
get_profile_time(new_file_name)


func _input(event) -> void:
     if event.is_action_pressed("ui_cancel"):
          Global.sceneHistory.append("res://world.tscn")
          get_tree().change_scene_to_file("res://world.tscn")

func _on_save_button_pressed() -> void:
     create_save_slot_button("user://save/"+ get_profile_name() +
" - " +str(get_max_save_index(get_profile_name())) + ".dat",
saveMode)


func get_profile_name() -> String:
     var tempSAVE = FileAccess.open("user://save/tempSAVE.dat",
FileAccess.READ)
     var profileName
     for i in range(3):
          profileName = tempSAVE.get_var()

     return profileName

func get_profile_time(path:String) -> String:
     var tempSAVE = FileAccess.open(path, FileAccess.READ)
     var profileTime
     for i in range(4):
          profileTime = tempSAVE.get_var()
           # Вычисляем минуты и секунды
     var minutes: int = int(profileTime) / 60
     var seconds: int = int(profileTime) % 60

     # Форматируем строку для отображения
```

```
	var time_string = "Время в игре: " + str(minutes) + "m " +
str(seconds) + "s"
	return time_string

func get_removeSlotButton(removeSlotButton: Button) -> void:
	var fileToRemove: String =
removeSlotButton.get_parent().get_child(0).get_meta("newSavePath")
	DirAccess.remove_absolute(fileToRemove)
	removeSlotButton.get_parent().queue_free()
func _on_load_button_pressed() -> void:
	var saveFromSlot = currentSlotButton.get_meta("newSavePath")
	var tempSAVE = FileAccess.open("user://save/tempSAVE.dat",
FileAccess.WRITE_READ)
	var newSAVE = FileAccess.open(saveFromSlot, FileAccess.READ)
	var line = newSAVE.get_file_as_bytes(saveFromSlot)
	tempSAVE.store_buffer(line)
	tempSAVE.close()
	newSAVE.close()
	animation_player.play("SaveOut")
	transCall = func ():
		Global.sceneHistory.append("res://world.tscn")
		get_tree().change_scene_to_file("res://world.tscn")


func _on_to_menu_button_pressed() -> void:
	animation_player.play("SaveOut")
	transCall = func ():
		Global.sceneHistory.append("res://MenuScene.tscn")
		get_tree().change_scene_to_file("res://MenuScene.tscn")

func _on_back_button_pressed() -> void:
	animation_player.play("SaveOut")
	transCall = func ():
		Global.sceneHistory.append("res://world.tscn")
		get_tree().change_scene_to_file("res://world.tscn")

func _on_animation_player_animation_finished(anim_name:
StringName) -> void:
	if anim_name == "SaveOut":
		transCall.call()
```

Файл «save_slot_button.gd»

```
extends Button
var tempSAVE
# Время, в течение которого должно произойти второе нажатие
const DOUBLE_CLICK_TIME = 200

# Переменные для отслеживания времени и состояния
var last_click_time = 0.0
var click_count = 0
signal instaLoad
signal saveMode(buttonStatus:bool)
signal sendSelf(slotButton: Button)
func _ready() -> void:
```

```
     instaLoad.connect(Callable(get_parent().get_parent().get_pare
nt().get_parent().get_parent().get_parent(),
"_on_load_button_pressed"))
     saveMode.connect(Callable(get_parent().get_parent().get_paren
t().get_parent().get_parent().get_parent(),
"_on_save_button_pressed"))
     sendSelf.connect(Callable(get_parent().get_parent().get_paren
t().get_parent().get_parent().get_parent(), "get_slotButton"))
func _on_pressed() -> void:
     sendSelf.emit(self)
     var current_time = Time.get_ticks_msec()  # Получаем текущее
время в миллисекундах
     # Проверяем, произошло ли двойное нажатие
     if current_time - last_click_time <= DOUBLE_CLICK_TIME:
          click_count += 1
          if click_count == 2:
               _on_double_click()  # Вызываем метод для обработки
двойного нажатия
               click_count = 0  # Сбрасываем счетчик
     else:
          click_count = 1  # Сбрасываем счетчик, если время
превышено

     last_click_time = current_time  # Обновляем время последнего
нажатия
     print(current_time)
func _on_double_click() -> void:
     instaLoad.emit()
     print(instaLoad.get_connections())
```

Файл «start_cutscene_1.gd»

```
extends Node3D

var tempSAVE
@onready var click: AudioStreamPlayer2D = $Click
@onready var animation_player: AnimationPlayer =
$SubViewportContainer/SubViewport/AnimationPlayer
@onready var sub_viewport_container: SubViewportContainer =
$SubViewportContainer
const INTRO = preload("res://data/dialogue/Intro.dialogue")

var tempR: String
var step_one: bool = false
var start_position = Vector3(-123.456, 0.765, 1.069)
var start_rotation =  Vector3(0,0,0)
var charPerSec: float = 0.1
func _ready() -> void:
     var Dirs = DirAccess
     Dirs.open("user://")
     Dirs.make_dir_recursive_absolute("user://save")
     Global.update_settings()

func _input(event):
     if Input.is_anything_pressed() && not step_one:
          animation_player.stop()
```

```
            tempR = "Wiggle" + str(randi_range(1, 3))
            animation_player.play(tempR)
            click.play()



func _on_control_name_entered(thatName: String) -> void:
     step_one = true
     animation_player.play("UiFade")
     tempSAVE = FileAccess.open("user://save/tempSAVE.dat",
FileAccess.WRITE)
     tempSAVE.store_var(start_position)
     tempSAVE.store_var(start_rotation)
     tempSAVE.store_var(thatName)
     tempSAVE.store_var(0.0)
     tempSAVE.store_var("[Card][1]![Tag90][1]!")
     tempSAVE.store_var(7000)
     tempSAVE.store_var("")
     tempSAVE.store_var(1000)
     tempSAVE.close()

func _on_animation_player_animation_finished(anim_name:
StringName) -> void:
     if anim_name == "UiFade":
          animation_player.play("EnterTurnstile")
     if anim_name == "EnterTurnstile":
          animation_player.play("Tale")
          DialogueManager.show_dialogue_balloon(INTRO,"start")



     if anim_name == "Tale" or anim_name == "voice":
          Global.sceneHistory.append("res://start_cutscene_1.tscn")
          get_tree().change_scene_to_file("res://world.tscn")

func check_music() -> void:
     Global.update_music_object(get_tree().root)
```

Файл «menu_scene.gd»

```
extends Node3D
var transitionFlag: bool = false
@onready var load_game_button: Button =
$VBoxContainer/HBoxContainer/LoadGameButton
@onready var sub_viewport_container: SubViewportContainer =
$SubViewportContainer
@onready var animation_player: AnimationPlayer = $AnimationPlayer
var settingUIList = []
var menuUIList = []
var transScene: String

func _ready() -> void:
     var dir = DirAccess
     if dir.open("user://save/") == null: load_game_button.visible
= false
```

```
	collect_ui_elements(get_tree().root)
	Global.update_settings()

# Рекурсивная функция для обхода всех узлов
func collect_ui_elements(node):
	if node.has_meta("SettingUI"):
		settingUIList.append(node)
	if node.has_meta("MenuUI"):
		menuUIList.append(node)
	# Проходим по всем детям текущего узла
	for child in node.get_children():
		collect_ui_elements(child)

func _on_new_game_button_pressed() -> void:
	transScene = "start_cutscene_1.tscn"
	animation_player.play("StartGame")

func _on_load_game_button_pressed() -> void:
	transScene = "save_menu.tscn"
	animation_player.play("StartGame")

func _on_animation_player_animation_finished(anim_name:
StringName) -> void:
	if anim_name == "StartGame":
		Global.sceneHistory.append("res://MenuScene.tscn")
		get_tree().change_scene_to_file(transScene)

func _on_low_end_mode_button_toggled(toggled_on: bool) -> void:
	Global.shaderDisabled = toggled_on
	Global.update_viewport_object(get_tree().root)

func _on_options_button_pressed() -> void:
	for element in menuUIList:
		element.visible = false
	for element in settingUIList:
		element.visible = true

func _on_back_button_pressed() -> void:
	for element in menuUIList:
		element.visible = true
	for element in settingUIList:
		element.visible = false
```

Файл «remove_button.gd»

```
extends Button
signal sendSelf(slotRemoveButton: Button)

func _ready() -> void:
	sendSelf.connect(Callable(get_parent().get_parent().get_paren
t().get_parent().get_parent().get_parent(),
"get_removeSlotButton"))


func _on_pressed() -> void:
	sendSelf.emit(self)
```

```
     print(sendSelf.get_connections())
```

Файл «MapGeneraor.gd»

```
extends Node

@export var grid_map: GridMap
@export var cell_scale: int = 10  # Масштаб 1 ячейки в пикселях

func generate_map_image():
     # Получаем все занятые ячейки
     var cells = grid_map.get_used_cells()
     if cells.is_empty():
          push_warning("GridMap is empty!")
          return

     # Получаем MeshLibrary
     var mesh_lib = grid_map.mesh_library
     if not mesh_lib:
          push_error("GridMap has no MeshLibrary assigned!")
          return

     # Рассчитываем границы вручную
     var min_x = INF
     var max_x = -INF
     var min_z = INF
     var max_z = -INF

     for cell in cells:
          min_x = min(min_x, cell.x)
          max_x = max(max_x, cell.x)
          min_z = min(min_z, cell.z)
          max_z = max(max_z, cell.z)

     var width = (max_x - min_x + 1) * cell_scale
     var height = (max_z - min_z + 1) * cell_scale

     # Создаем изображение
     var img = Image.create(width, height, false,
Image.FORMAT_RGBA8)
     img.fill(Color.TRANSPARENT)

     # Настраиваем цвет стен
     var wall_color = Color.BLACK

     # Перебираем все ячейки
     for cell in cells:
          var item = grid_map.get_cell_item(cell)
          var item_name = mesh_lib.get_item_name(item)

          if "WALL" in item_name.to_upper():
               var grid_x = cell.x - min_x
               var grid_z = cell.z - min_z
               draw_cell_fill(img, grid_x, grid_z, wall_color)
```

```
     # Создаем текстуру и отображаем
     var textureMap = ImageTexture.create_from_image(img)
     self.texture = textureMap

     # Сохраняем в файл (опционально)
     img.save_png("user://map_overview.png")

func draw_cell_fill(img: Image, x: int, z: int, color: Color):
     var rect = Rect2i(
          x * cell_scale,
          z * cell_scale,
          cell_scale,
          cell_scale
     )
     img.fill_rect(rect, color)

func _ready():
     generate_map_image()
```

Файл «name_prompt_ui.gd»

```
extends Control

@onready var namePr: LineEdit =
$Panel/MarginContainer/VBoxContainer/HBoxContainer/NamePr
@onready var name_pr_2: Label =
$Panel/MarginContainer/VBoxContainer/HBoxContainer/NamePr2
signal name_entered(thatName: String)

# Called when the node enters the scene tree for the first time.
func _ready() -> void:
     pass # Replace with function body.


# Called every frame. 'delta' is the elapsed time since the
previous frame.
func _process(delta: float) -> void:
     pass

func _input(event: InputEvent) -> void:
     if event.is_action_pressed("ui_accept"):
          namePr.editable = false
          namePr.selecting_enabled = false
          name_pr_2.text = namePr.text
          namePr.visible = false
          name_pr_2.visible = true
          name_entered.emit(name_pr_2.text)
```

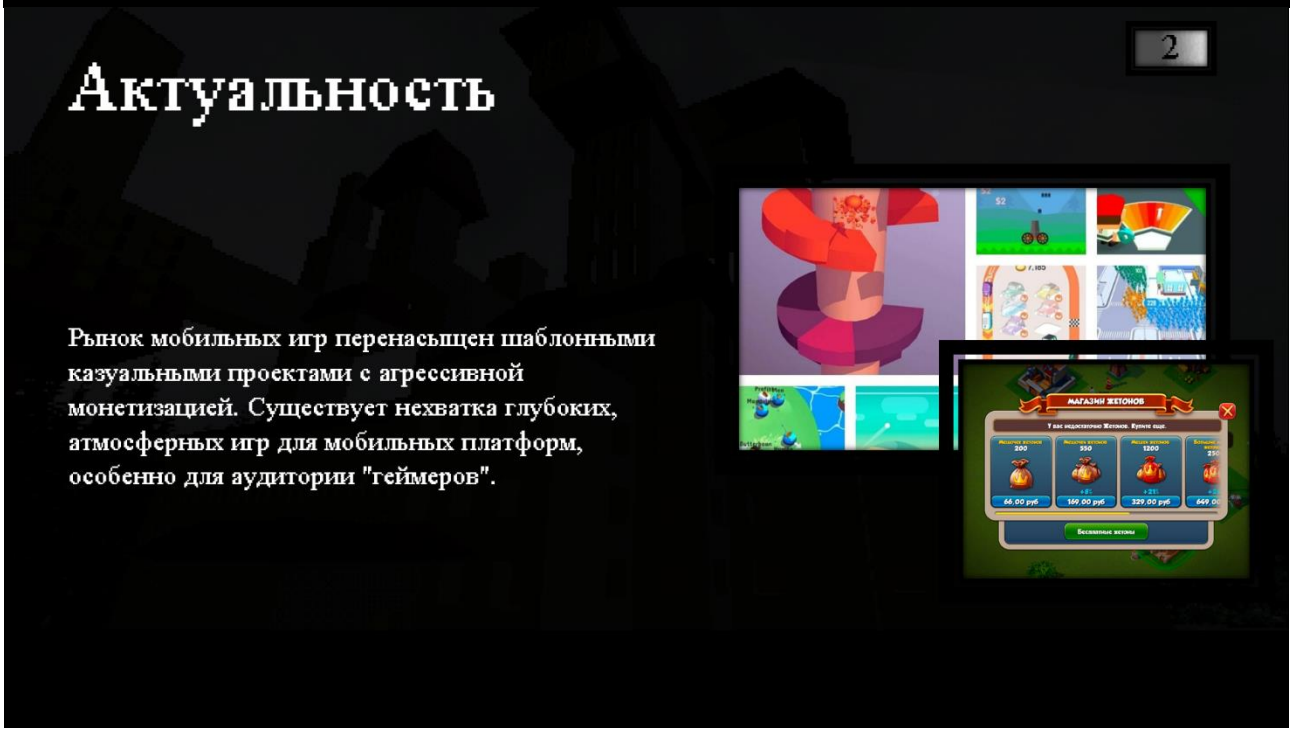
Приложение С

(Обязательное)

Игровое приложение «Escape Building-9»

Презентация

1
Мобильное приложение
‘Escape Building-9’
Автор работы:
Кирмасов И. Д. (уч. гр. 21ВВП2)
Научный руководитель:
д.т.н., профессор кафедры ВТ ПГУ
Зинкин С. А.
Пензенский государственный университет
2025 г.

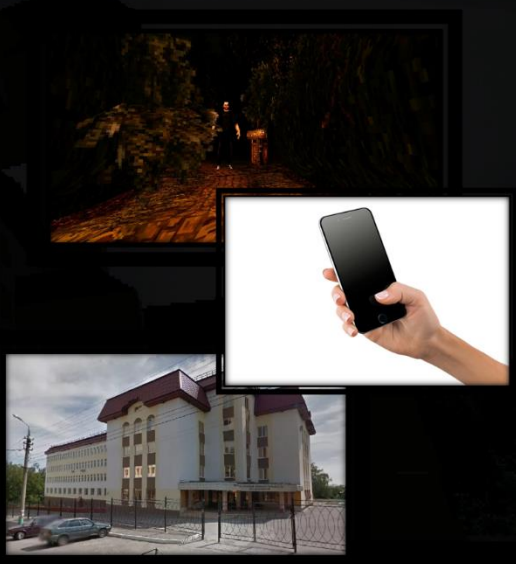

2
Актуальность
Рынок мобильных игр перенасыщен шаблонными казуальными проектами с агрессивной монетизацией. Существует нехватка глубоких, атмосферных игр для мобильных платформ, особенно для аудитории "геймеров".
МАГАЗИН ЖЕТОНОВ

Цель

3

Разработать качественное мобильное приложение в жанре survival-horror с элементами RPG, обеспечивающее погружение за счет:

- Уникальной ретро-стилистики PS1 (низкополигональная 3D-графика).
- Адаптированных механик под сенсорное управление.
- Знакомых локаций на основе архитектуры девятого корпуса ПГУ.



Технологический стек

4

Движок: Godot Engine 4.4, язык программирования GDScript

Графика: Blender (модели), Adobe Photoshop (текстуры, UI).

5

Ключевые разработки

Что разработано:

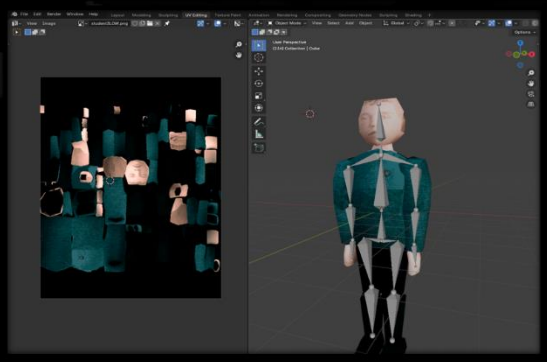
Игровое приложение для Android: Системы управления (персонаж, инвентарь, диалоги), сохранений, обучающая система, система квестов.

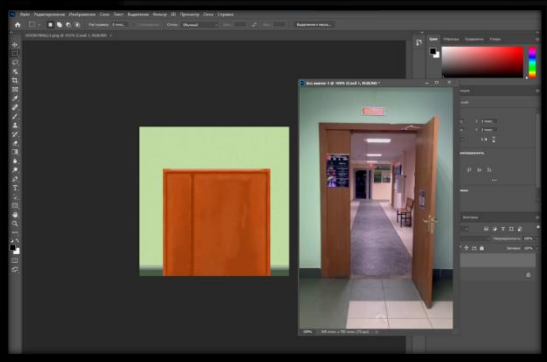
6

Ключевые разработки

Что разработано:

Контент: Стилизованные 3D-модели и текстуры (Blender, Photoshop), локации на основе 9 корпуса ПГУ, адаптивный UI.





7

Ключевые разработки

Что разработано:

Геймдизайн: Смесь механик survival-horror (напряжение, ресурсы) и RPG (инвентарь, диалоги, прогресс).

Мониторинг состояния персонажа в серии Resident Evil, как пример survival-horror механики

Прогрессия персонажа за счёт открытого перемещения по миру и взаимодействия с NPC в серии The Elder Scrolls, как пример RPG механики

8

Результаты

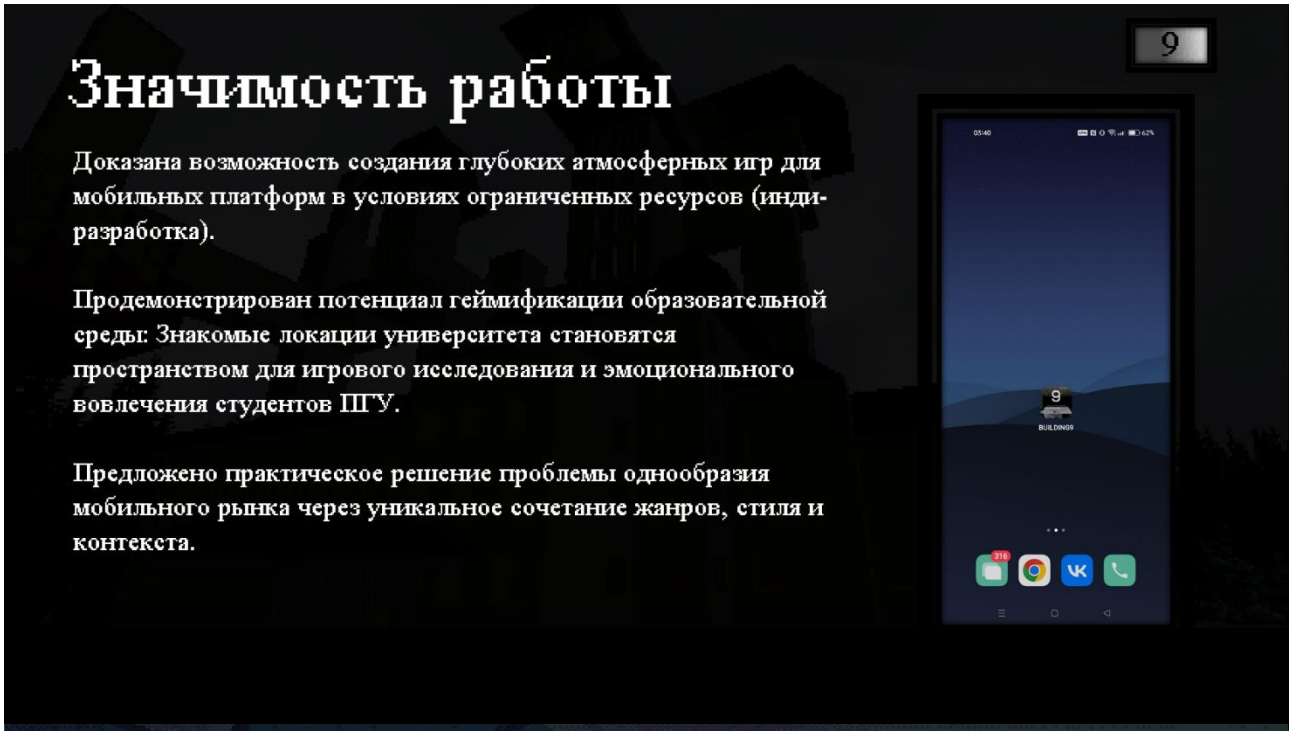
Создано полностью функциональное мобильное приложение, готовое к публикации. Его ключевые характеристики:

-Стабильные 30 FPS достигнуты даже на бюджетных Android-устройствах.

-Интуитивное сенсорное управление: Крупные кнопки, контекстные действия, удобные меню.

-Динамические системы: инвентарь персонажа и всех NPC, сохранения, диалоги, квесты, смена времени суток.

-Достигнута уникальная атмосфера за счет ретро-графики, динамического освещения, эмбиентного звука и узнаваемых элементов локации девятого корпуса ПГУ.


9
Значимость работы
Доказана возможность создания глубоких атмосферных игр для мобильных платформ в условиях ограниченных ресурсов (инди-разработка).
Продемонстрирован потенциал геймификации образовательной среды: Знакомые локации университета становятся пространством для игрового исследования и эмоционального вовлечения студентов ПГУ.
Предложено практическое решение проблемы однообразия мобильного рынка через уникальное сочетание жанров, стиля и контекста.

10
Спасибо за внимание!
КОНТАКТНЫЕ ДАННЫЕ
@INSOMNIAPRISONER