

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 «Программное обеспечение средств вычислительной
техники и автоматизированных систем»

БАКАЛАВРСКАЯ РАБОТА

на тему

ПРИЛОЖЕНИЕ НА UNITY В ЖАНРЕ

ROGUELIKE

Студент Сурков А.А.
(подпись, дата)

Руководитель Зинкин С.А.
(подпись, дата)

Нормоконтролёр А.С. Бычков
(подпись, дата)

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

Заведующий кафедрой М.А. Митрохин

Работа защищена с отметкой хорошо (подпись)
(протокол заседания ГЭК от 09.06.25 № 8)

Секретарь ГЭК Гурин Е.И.
(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 Сурков Артём Анатольевич гр. 21BBB1 факультета вычислительной техники.
2. Тема работы: «Приложение на Unity в жанре Roguelike»
Тема утверждена приказом ПГУ № 601/о от "06" мая 2025 г.
3. Руководитель бакалаврской работы д.т.н., профессор кафедры ВТ Зишкин С.А.

4. Задание на работу (назначение разработки, исходные данные и т.п.)

Требуется разработать приложение на Unity в жанре Roguelike, которая должна соответствовать приведенным ниже функциональным и техническим требованиям. Основные игровые механики, такие как управление игровым персонажем, автоматическая атака персонажа по врагам, сложная система врагов, зависящая от погодных условий и времени суток, система усложнения игровой сессии путем добавления волн противников, которые становятся сложнее с каждой новой волной, также реализовать различные улучшения врагов, зависящие от погоды и времени суток, прогрессия внутри сессии, чтобы игроку не становилось скучно в течении игры.

5. Перечень подлежащих разработке вопросов

Требуется произвести анализ предметных областей: изучить аналоги существующих игр, изучить их основные игровые механики, изучить разработку приложений на Unity и его различные функционал.

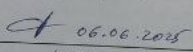
6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объём работы	Срок выполнения	Подпись руководителя
1.	Изучение предметной области	15	06.05.2025-08.05.2025	
2.	Анализ требований	5	09.05.2025-10.05.2025	
3.	Выбор технологий, применяемых при проектировании и разработке	10	11.05.2025-12.05.2025	
4.	Описание алгоритмов	20	13.05.2025-16.05.2025	
5.	Разработка приложения на Unity в жанре RogueLike	60	16.05.2025-28.05.2025	
6.	Оформление пояснительной записки	15	29.05.2025-30.05.2025	

Дата выдачи задания "06" мая 2025 г.

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

Задание к исполнению принял студент  Сурков А.А.
(подпись, дата)

Работу к защите допустить
Декан факультета  Фионова Л.Р.
(подпись, дата)

Реферат

Бакалаврская работа содержит 122 листа, 29 рисунков, 1 таблица, 12 источников и 3 приложения.

ИГРА, ROGUELIKE, SURVIVORS-LIKE, UNITY, C#, PIXEL ART, ОПТИМИЗАЦИЯ, КРОССПЛАТФОРМЕННОСТЬ

Цель работы – разработать игру в жанре RogueLike с PixelART графикой. Которая обеспечит высокую реиграбельность и кроссплатформенность.

Объектами разработки являются:

– игровое приложение с реализацией основных механик, характерных для данного жанра;

– система сохранения настроек и актуального прогресса;

– визуальное составляющая (пользовательский интерфейс).

В результате дипломного проектирования были назначены требования к разрабатываемому игровому приложению, спроектирована архитектура, и разработано кроссплатформенное приложение на Unity в жанре RogueLike.

Также были определены дальнейшие перспективы проекта.

ПГУ 1.090301.19.001 ПЗ

Изм.	Лист	№ докум.	Подпись	Дата
Разраб.		Сурков А.А.		28.06.21
Провер.		Зинкин С.А.		28.06.21
И. Контр.		Бычков А.С.		28.06.21
Утверд.		Ивантеев М.А.		28.06.21

Приложение на Unity в жанре
RogueLike

Пояснительная записка

Лист	Лист	Листов
	2	122
ФВТ, каф. ВТ		
гр. 21ВВВ1		

Содержание

Перечень сокращений и обозначений	4
Введение	5
1 «White Knight» - игра в жанре «RogueLike/Survivors-Like»	7
1.1 Анализ предметной области.....	7
1.1.1 Анализ существующих аналогов.....	8
1.1.2 Обоснование актуальности разработки программы	13
1.1.3 Постановка задачи.....	15
1.1.4 Выбор языка программирования	18
1.1.5 Выбор системы хранения данных	20
1.1.6 Графический интерфейс	21
1.1.7 Функциональные требования.....	22
1.2 Проектирование и разработка.....	23
1.2.1 Архитектура пользовательского интерфейса.....	33
1.2.2 Архитектура программы	41
1.2.3 Динамика работы приложения	43
1.2.4 Тестирование и отладка.....	45
2 Безопасность жизнедеятельности.....	46
2.1 Правила безопасности при использовании ПК.....	46
2.2 Правила безопасности при использовании смартфона.....	47
Заключение	21
Список использованных источников	50
Приложение А - UML – диаграммы программы.....	51
Приложение Б - Листинг программы.....	56
Приложение В - Презентация	122

Перечень сокращений и обозначений

FPS (Frames Per Second) – количество кадров в секунду, характеристика производительности игры.

UI (User Interface) – пользовательский интерфейс, взаимодействие игрока с игрой.

HUD (Heads-Up Display) – информационная панель, которая отображает важные данные игрового процесса.

JSON (JavaScript Object Notation) – текстовый формат обмена данными, который используется в качестве сохранения игровых настроек и прогресса.

MVC (Model-View-Controller) – архитектурный шаблон, который разделяет логику игры на модель, представление и контроллер.

Unity – игровой движок.

PixelART – жанр в графике, основанный на пиксельной обработке.

DLC (Downloadable Content) – дополнительный загружаемый контент для игры.

UML (Unified Modeling Language) – унифицированный язык моделирования.

URP (Universal Render Pipeline) – система рендеринга в Unity.

Object Pooling – метод оптимизации, суть которого повторное использование объектов.

Asset – цифровой объект, преимущественно состоящий из однотипных данных, неделимая сущность, которая представляет часть игрового контента и обладает некими свойствами.

					ПГУ 1.090301.19.001 ПЗ	Лист
						4
Изм.	Лист	№ докум.	Подпись	Дата		

Введение

В нашем мире компьютерные игры стали не только очень популярным развлечением, но и огромной частью современной культуры. С каждым годом игровая индустрия развивается все быстрее и быстрее, постоянно предлагая новые жанры игр, механики, развитие графики и все это привлекает миллионы игроков по всему миру. Одним из наиболее популярных направлений в последние 5 лет стали игры в жанре RogueLike/Survivors-Like, которые сочетают в себе элементы процедурной генерации, постоянного прогресса, динамичного геймплея и высокой реиграбельности.

Однако, несмотря на популярность этого жанра, многие проекты сталкиваются со следующими проблемами: недостаточное разнообразие игрового процесса, слабая оптимизация, отсутствие уникальных механик. В связи с этим моей задачей становится создание игры, которая соответствует современным стандартам жанра, но и предлагает новые решения для повышения интереса и удержания аудитории.

В данной выпускной квалификационной работе разрабатывается игра «White Knight» в жанре RogueLike/Survivors-Like, выполнена она в графическом стиле PixelART. Игра сочетает в себе классические элементы жанра, такие как автоматическая атака по врагам, процедурная генерация улучшений и возможность покупки постоянных улучшений, а также уникальные механики, включая динамическую смену дня и ночи, погодные эффекты. Игра создана с использованием современных технологий, включая Unity и URP, что обеспечивает высокую производительность и кроссплатформенность.

					ПГУ 1.090301.19.001 ПЗ	Лист
						5
Изм.	Лист	№ докум.	Подпись	Дата		

Цели выпускной квалификационной работы:

- изучить современные тенденции в жанре RogueLike/Survivors-Like и проанализировать существующие аналоги;
- определить ключевые проблемы игр;
- разработать архитектуру игры с использованием MVC для обеспечения высокой производительности и удобства дальнейшей работы;
- реализовать оптимизированные системы, такие как Object Pooling, для стабильного FPS даже при большом количестве врагов на экране;
- создать интуитивно понятный интерфейс и адаптацией под различные платформы (ПК и мобильные устройства);
- провести тестирование игры на разных устройствах, оценить баланс сложности и оптимизацию;
- определить перспективы дальнейшего развития проекта, включая добавление нового контента, интеграцию облачных сохранений и расширение платформенной поддержки.

					ПГУ 1.090301.19.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		6

1 «White Knight» - игра в жанре «RogueLike/Survivors-Like»

1.1 Анализ предметной области

Компьютерные игры с каждым годом набирают все большую популярность в обществе. Ими интересуются люди всех возрастов, мужчины и женщины. Это отличный способ расслабиться после тяжелого рабочего дня или отвлечься от окружающего ежедневного стресса. Прекрасный способ «убить время» по дороге на работу или в другой город. Замечательно времяпрепровождение, доставляющее удовольствие и поднимающее настроение.

В качестве дипломного проекта была выбрана игра в жанре «RogueLike», так как этот жанр, как никакой другой подходит абсолютно для любой группе людей. Каждый найдет в нем то, что приносит ему удовольствие. В последние несколько лет жанр игр «RogueLike» с элементами выживания начал набирать огромную популярность, как у разработчиков инди-игр, так и у игроков по всему миру.

Игры подобной направленности создали свой собственный поджанр, называемый «Survivors-Like». Свое название он получил от своего самого популярного представителя «Vampire Survivors». Причина успеха таких игр в их простоте и удержании игрока. Ключевые их особенности заключаются в минималистическом управлении (персонаж атакует автоматически, игроку достаточно управлять только передвижением по игровому полю), прогрессивная сложность (враги становятся сильнее каждое определенное время), рандомизированная прокачка (игрок выбирает улучшение своих характеристик или различные предметы, упрощающие игру, после повышения уровня), высокая реиграбельность (игровые сессии обычно не занимают больше 20 минут, но из-за того, что все улучшения выпадают в случайном порядке, ни одна из партий не будет похожа на предыдущую, что заставляет игрока продолжать и искать наиболее лучшие комбинации характеристик и предметов).

					ПГУ 1.090301.19.001 ПЗ	Лист
						7
Изм.	Лист	№ докум.	Подпись	Дата		

Сам по себе жанр «Survivors-Like» представляет собой сборник из нескольких игровых направлений, которые уже обрели популярность у игроков. Такие как: RogueLike элементы (процедурная генерация и смерть персонажа, которая обнуляет прогресс в сессии), инкрементальные игры (постепенное улучшение силы персонажа по ходу сессии), Bullet heaven (когда в игре присутствует много врагов, но они все слабее игрока).

1.1.1 Анализ существующих аналогов

Игр в жанре «RogueLike/Survivors-Like» огромное количество, но мы рассмотрим наиболее значимые аналоги.

Классикой данного жанра является «Vampire Survivors». Разработчиком игры является компания «Poncle». Игра была выпущена в 2022 году. Основные особенности заключаются в графике, которая сделана в стиле PixelART, автоматической атаке по врагам, прокачке через выбор предметов на каждом уровне, наличием реликвий (бонусы, которые покупаются в главном меню игры и дающие улучшения навсегда). Имеет ряд плюсов, перед другими представителями жанра. У нее понятные для любого человека система управления и дизайн, также присутствует хорошая оптимизация под любое устройство. Из минусов можно выделить постоянно повторяющиеся враги и отсутствие интересных и сложных механик, без которых со временем становится не интересно в нее играть.

Внешний вид игры представлен на рисунке 1.

					ПГУ 1.090301.19.001 ПЗ	Лист
						8
Изм.	Лист	№ докум.	Подпись	Дата		



Рисунок 1 – Внешний вид «Vampire Survivors»

Рассмотрим аналог «Brotato». Игру разработала компания «Blobfish» в 2022 году и реализовала в ней некоторые нововведения для жанра. В игре есть целых 6 мест в инвентаре под различные типы оружия, а некоторые персонажи обладают еще большим размером инвентаря. Разнообразные классы персонажей с уникальными особенностями и характеристиками. Большая система выбираемых предметов, которая была взята у игр в жанре МОБА. Эти нововведения дают ей некоторые преимущества среди конкурентов. Благодаря разнообразию классов и предметов, можно придумывать различные комбинации, что повышает вариативность игры и ее реиграбельность. Достаточно высокая динамика игры, которая не дает игроу расслабиться и заставляет всегда четко следить за происходящим на экране. Из минусов можно выделить маленькую игровую область, к концу сессии, при большом количестве врагов место становится очень мало. Также отсутствует разнообразие в типах врагов и боссов.

Внешний вид игры представлен на рисунке 2.

Изм.	Лист	№ докум.	Подпись	Дата



Рисунок 2 – Внешний вид «Brotato»

Следующий аналог – «Soulstone Survivors» разработанный в 2022 году. Разрабатывала «Soulstone Survivors» студия «Game Smithing Limited». Ключевой особенностью игры является наличие 3D-графики, что делает ее одной из единственных представителей жанра, имеющих 3D-графику. Также можно выделить систему прокачки при помощи рун, выпадающих с боссов. Ее особенность в виде 3D-графики является как плюсом, так и минусом, так как из-за хорошей графики игра не будет работать на любом устройстве в приемлемом FPS.

Внешний вид игры представлен на рисунке 3.



Рисунок 3 – Внешний вид «Soulstone Survivors»

«20 Minutes Till Dawn» также, как и другие аналоги была выпущена в 2022 году. Разработчики – компания «flanne». Игра имеет ручное управление стрельбы (но в настройках можно поставить авто-режим), темный готический сеттинг и уникальную систему заклинаний – это отличает ее от других аналогов. В ней можно собирать разнообразные комбинации скиллов, а также она имеет достаточно высокий уровень сложности, что могут оценить игроки, любящие трудности. Основным ее минусом является отсутствие контента, который можно изучить игроку в процессе игры.

Внешний вид игры представлен на рисунке 4

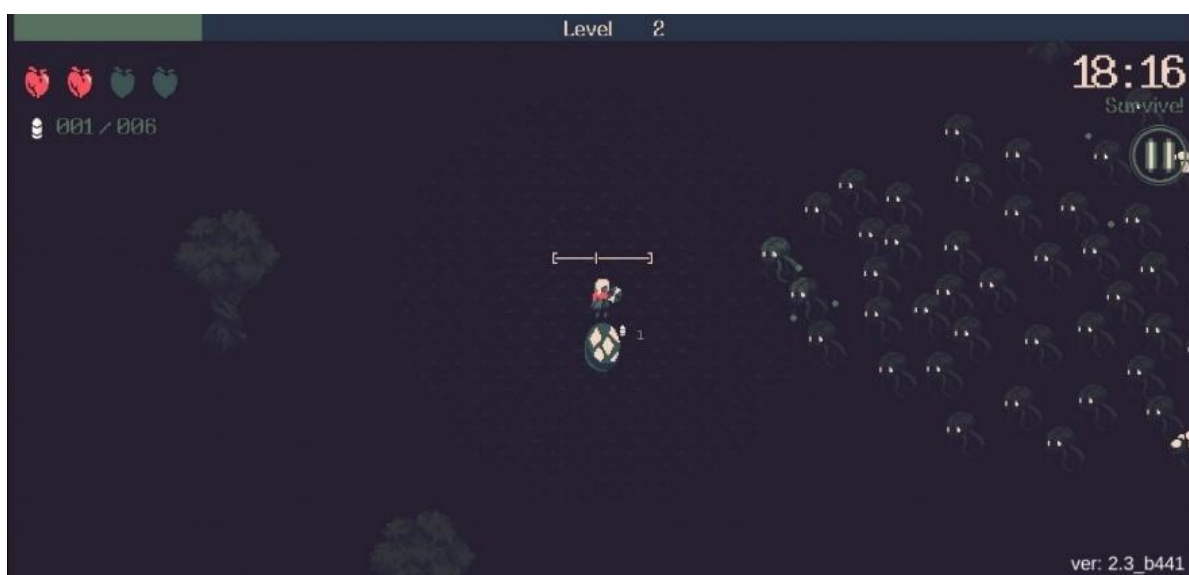


Рисунок 4 – Внешний вид «20 Minutes Till Dawn»

В таблице 1 представлен анализ аналогов.

Таблица 1 – Анализ аналогов

Название игры	Ключевые особенности геймплея	Удобство управления	Монетизация
Vampire Survivors	Автоматическая атака; Линейная прокачка; Реликвии для постоянных улучшений.	Клавиатура/джойстик.	Разовые покупки DLC, нет рекламы.
Brotato	6 оружий одновременно; Уникальные классы персонажей; Интересная система предметов.	Клавиатура/джойстик, хорошая адаптация под мобильные устройства.	Игра платная, также покупки DLC, как косметические, так и геймплейные.
Soulstone Survivors	Главная особенность-это 3D-графика, рунная система прокачки и высокая кастомизация.	Клавиатура. Новичку сложно разобраться с управлением из-за большого количества способностей.	Игра платная

Название игры	Ключевые особенности геймплея	Удобство управления	Монетизация
20 Minutes Till Dawn	Ручная стрельба, но можно поставить автоматическую Готический сеттинг; уникальные заклинания.	Клавиатура и мышь/ джойстик. Не очень хорошо адаптировано под мобильные устройства	Игра платная, в мобильной версии есть реклама.

1.1.2 Обоснование актуальности разработки программы

В данный момент жанр «RogueLike/Survivors-Like» очень успешен и актуален в игровой индустрии. Успешность «RogueLike/Survivors-Like» на рынке подтверждается следующими факторами:

- коммерческий успех первых игр в данном жанре («Vampire Survivors», «Brotato», «Soulstone Survivors», «20 Minutes Till Dawn»);
- высокий интерес игроков к примитивным, но увлекающим игровым механикам;
- развитие инди компаний, которым интересна быстрая разработка таких игр и дальнейшая их продажа.

Уже сделали много игр в данном жанре, но у всех существуют проблемы и недостатки. Можно выделить отсутствие вариативности, когда геймплей становится однообразным через 10 и более часов игры, плохой баланс игр, в некоторые бывает слишком легко играть, а другие настолько сложны, что не приносят никакого удовольствия и расслабления. К сожалению, многие проекты просто копируют оригинал жанра в виде «Vampire Survivors» и не хотят добавлять свои уникальные особенности.

Жанр «RogueLike/Survivors-Like» имеет потребность в новых решениях, поэтому игра «White Knight» предлагает динамическую смену окружающего мира, то есть в ней реализована система смены дня и ночи, а также погодных условий, из-за чего меняется тип появляющихся врагов. Она обладает хорошей оптимизацией и поддерживает стабильный FPS при огромном количестве врагов на экране.

Игра актуальна в своей технической части, потому что при ее разработке был использован Unity 2022+ и его современные инструменты для 2D. Она имеет поддержку мультиплатформенности, можно будет поиграть, как на ПК, так и на мобильных устройствах.

Также игру «White Knight» можно монетизировать, опубликовав ее в интернет-магазинах, например, в «Steam», «GooglePlay», «AppStore», «Rustore». В дальнейшем в нее можно добавить косметические предметы для персонажей, различные варианты внутриигровых покупок и рекламу.

1.1.3 Постановка задачи

Требуется разработать игру в жанре «Roguelike/Survivors-like» которая должна соответствовать приведенным ниже функциональным и техническим требованиям:

- перемещение с помощью клавиатуры и виртуального джойстика на телефоне;
- автоматическая атака персонажа по врагам;
- сложная система врагов, зависящая от погодных условий и времени суток;
- система усложнения игровой сессии путем добавления волн противников, которые становятся сложнее с каждой новой волной, также существуют различные улучшения врагов, зависящие от погоды и ночи;
- разнообразие врагов (у каждого типа врагов разные характеристики);
- за убийство врагов игрок получает кристаллы, которые нужны для повышения уровня, также с определенным шансом из врага выпадает «осколок души», которые нужны для покупки реликвий в главном меню игры;
- выбор из четырех случайных улучшений при достижении нового уровня;
- система реликвий (перманентные улучшения, которые покупаются в главном меню игры и позволяют улучшать своих персонажей от сессии к сессии);
- доступны на выбор несколько классов (рыцарь, маг, лучник) с уникальными типами оружия (меч, наносящий урон нескольким врагам в ближнем бою, огненный шар, который наносит урон по области, и лук со стрелами, который стреляет в ближайшего врага, на большом расстоянии).

Интерфейс должен состоять из главного меню, игрового HUD и панели завершения игры. В главном меню должен быть магазин реликвий (перманентных бонусов), кнопка «Играть», которая переносит игрока в меню выбора персонажа и кнопка настройки. Игровой HUD состоит из визуализации актуального уровня здоровья игрового персонажа, уровня, выбранных текущих улучшений и таймер нынешней сессии. На панель завершения игры, должна выводиться статистика после смерти (количество убитых врагов, продолжительность сессии, лучшее время сессии за все время). В настройках должна быть возможность настройки разрешения экрана, звука и языка.

Минимальные системные требования:

- ОС: Windows 7 (x64);
- процессор: Intel Core i3 / AMD Ryzen 3 (или аналоги);
- ОЗУ: 4 ГБ;
- видеокарта: Intel HD Graphics 4000/NVIDIA GTX 660/AMD Radeon 7850;
- место на диске: 500 МБ;
- разрешение экрана: 800×600.

В плане оптимизации игры требуется добиться стабильного FPS (не менее 60 кадров/с) при большом количестве врагов на экране. Для реализации этого требуется использовать систему Object Pool, в которой происходит переиспользование повторяющихся объектов (враги, снаряды). Также потребуется использовать URP (Universal Render Pipeline) для более эффективного рендеринга.

Будет реализовано локальное сохранение уровня текущего развития игрока (все купленные реликвии) и актуальных настроек (значения разрешения экрана, уровня музыки, игровых звуков и языка).

Требуется сделать понятный каждому UI (интерфейс игры) и реализовать для него поддержку разных разрешений экрана и платформ.

На рисунке 5 представлена диаграмма использования.

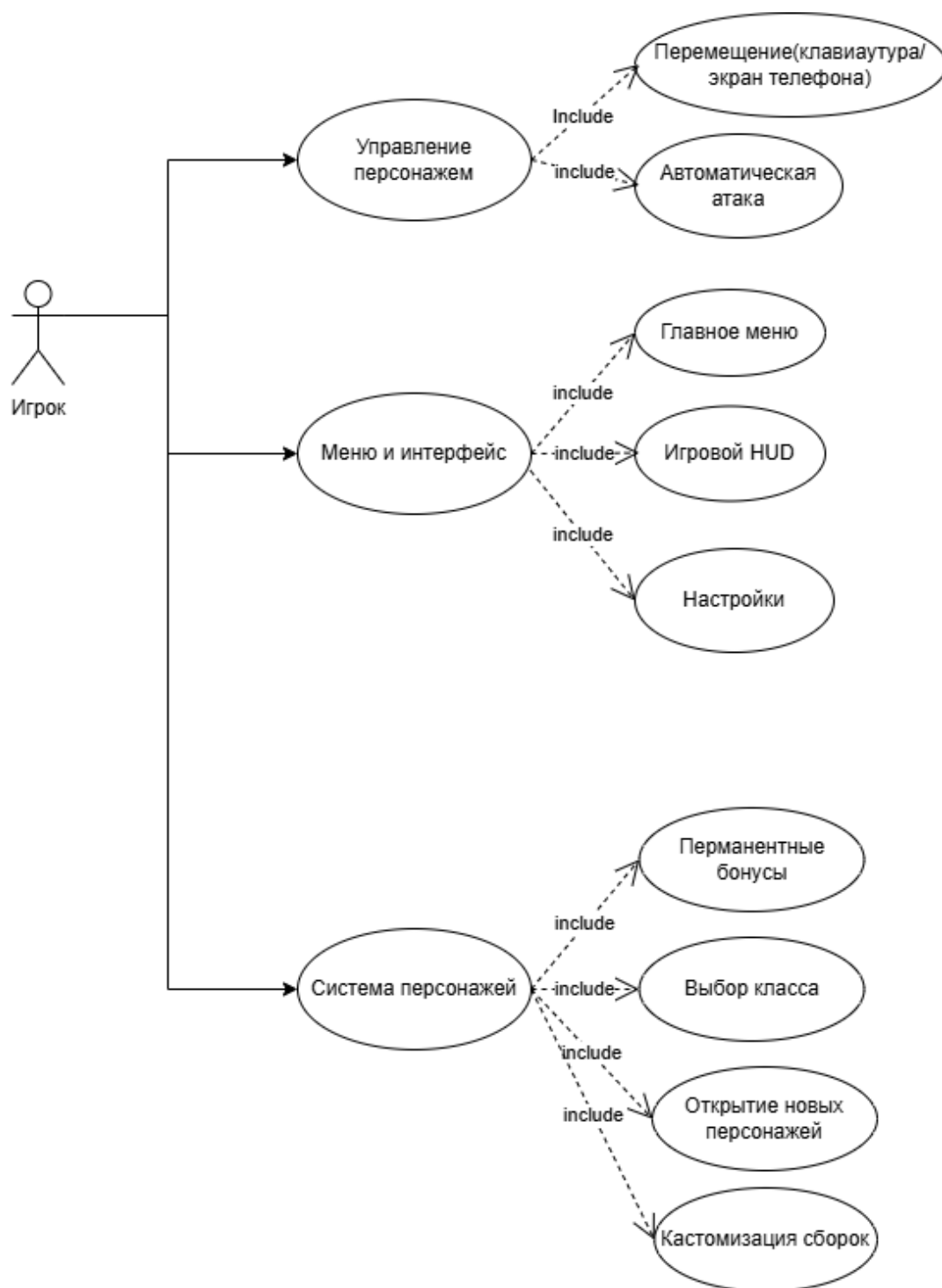


Рисунок 5 – UML-диаграмма вариантов использования

1.1.4 Выбор языка программирования

Для разработки игры «White Knight» важно выбрать наиболее подходящий язык программирования. Нужен язык, способный удовлетворить следующие требования:

- высокая производительность игры, не смотря на огромное количество активных объектов на игровой сцене (враги, кристаллы, снаряды);
- кроссплатформенность. Нужно чтобы игра работала одинаково на ПК и на мобильных устройствах, при этом практически без изменений существующего кода;
- наличие большого количества библиотек и современных инструментов для разработки игр.

Рассмотрим некоторые движки для игр и языки, на которых они работают. C++ с Unreal Engine имеет ряд преимуществ, таких как высокая производительность. C++ является одним из самых оптимизированных языков программирования. Благодаря оптимизации C++ и возможностям движка Unreal Engine, игры могут иметь большой и стабильный FPS даже если на сцене в этот момент будет огромное количество объектов. Имеет встроенный инструмент Blueprints. Визуальная система создания скриптов, которая позволяет быстро прототипировать игровые механики, при это не обладать глубокими знаниями программирования, что достаточно сильно ускоряет разработку игры. Также присутствует инструмент Niagara, нужный создания сложных визуальных эффектов, имеющий разнообразный функционал. С помощью него получается создавать действительно красивые и реалистичные погодные явления, эффекты взрывов, вспышек.

Из недостатков можно выделить сложность освоения. Unreal Engine вместе с C++ требует намного большего времени для изучения, тем более без опыта разработки игр на этом движке. Его избыточность для 2D, так как язык изначально создавался под объемные 3D-игры, функционал Unreal Engine заточен под 3D проекты, что делает его неподходящим для 2D-игры в PixelART.

Rust с Godot имеют такие преимущества, как производительность и безопасность. Rust может обеспечить высокое быстродействие кода, а также лучше работает с памятью, соответственно все ошибки, которые могут быть связаны с управлением памятью можно исключить, что очень важно для игр с большим количеством объектов на сцене. Является гибким движком, у которого открытый исходный код, с полной поддержкой, как 2D, так и 3D-разработки.

Недостатки движка заключаются в том, что он крайне слабо распространен в индустрии разработки игр, тем более для разработки игр на мобильные устройства, из-за этого найти готовые Assets или другие решения практически невозможно. Его инструменты весьма ограничены. Некоторые полезные функции, которые есть в Unity или Unreal Engine попросту отсутствуют в Godot. На нем очень сложно разрабатывать, так как документации с примерами кода в открытом доступе очень мало.

Для разработки «White Knight» идеально подходит C# вместе с Unity. Преимущества Unity перед другими языками заключаются в том, что он обладает достаточной производительностью. С помощью инструмента Burst Compiler, встроенного в Unity можно производить компиляцию кода сразу в машинный код, что позволяет достигнуть высокой скорости обработки большого количества врагов и снарядов на сцене. Также в нем есть кроссплатформенность. Благодаря Unity можно портировать игру на любую платформу, при этом изменить в коде, только управление персонажем. В Unity Store много открытых и бесплатных инструментов для 2D. В движке есть встроенные системы для создания механик, полная поддержка PixelART графики, возможность импортировать Assets со сторонних сайтов, а не только из официальных ресурсов и настраивать качество рендеринга для пиксельных объектов. Есть большое количество обучающих статей, как от самих разработчиков Unity, так и от других пользователей этого движка. Все это делает разработку удобнее и быстрее. Из недостатков можно выделить встроенный сборщик мусора, который при очистке памяти может вызывать задержки, поэтому приходится использовать Object Pool систему.

1.1.5 Выбор системы хранения данных.

Для хранения данных игре «White Knight» рассмотрим несколько систем.

SQLite обладающая реляционной структурой, что удобно для сложных запросов, таких как статистика игроков. Имеет лучшую производительность при большом количестве вводных данных, что хорошо подходит для больших проектов. Ее недостаток заключается в отсутствии встроенной синхронизации между устройствами.

PlayerPrefs встроенная система в Unity. Преимущество данной системы в ее простоте использования, а недостатки в том, что она подходит только для самых простых данных, таких как ключи прогресса и она очень неудобна при больших структурах и большим количеством параметров.

BinaryFormatter позволяет добиваться маленького размера файлов, что хорошо для мобильных игр, позволяет уменьшить конечный вес игры. Но у нее сложно читаемый формат, что сильно усложняет отладку и есть проблемы с поддержкой версий, при изменении класса могут сломаться старые сохранения.

На основе анализа приведенных аналогов лучше всего подходит JSON и PlayerPrefs в качестве системы хранения данных.

Преимущества:

- высокая гибкость, можно легко изменять и добавлять поля, что позволит развивать игру в дальнейшем. Добавлять новых персонажей, предметы в том числе для будущих DLC;
- есть поддержка вложенных структур, например конфиги врагов могут содержать подразделы с атаками, различными сопротивлениями урону;
- легко интегрируется с Unity благодаря встроенных функций и библиотек;
- работает на всех возможных платформах и мне не придется ее перенастраивать;
- при использовании не требуется наличие сервера;
- простая сериализация объектов на C#, можно сохранять прогресс игрока в один файл.

Недостатки:

- медленная скорость обработки больших данных. Если игра разрастется в сотни предметов или до огромного количества типов врагов, чтение и запись таких файлов будет занимать много времени;

- нет оптимизации запросов. Чтобы найти все объекты с определенным параметром придется искать вручную по всему файлу.

В JSON и PlayerPrefs будут храниться такие данные, как:

- настройки игры (разрешение экрана, громкость музыки и звуков, язык);
- актуальный прогресс игрока (купленные реликвии);
- конфиги персонажей (количество здоровья, урон, скорость передвижения).

1.1.6 Графический интерфейс

Для интерфейса использованы встроенные инструменты в Unity, такой как UGUI, который поддерживает Canvas, EventSystem, RectTransform, Button.

Canvas – это контейнер для всех UI-элементов, использован для главного меню и HUD на игровой сцене. EventSystem нужна для обработки ввода. RectTransform позволяет по-разному позиционировать элементы интерфейса. Нужно для адаптации под мобильные устройства и другие платформы. Button – интерактивные кнопки для главного меню, и других взаимодействий игрока с игрой.

Для внешнего вида текстов был выбран TextMeshPro. Делает шрифты более четкими и читабельными на разных разрешениях. Позволяет добавлять текстовым блокам тени, контуры, эффекты свечения. Можно добавить анимации.

Можно было бы использовать ImGui, но это уже устаревший интерфейс Unity, который отлично подходит, например, для отладки скриптов связанных с UI, но точно не подходит для финальной версии UI.

В качестве редактора для рисования персонажей и их анимирования в PixelART был выбран PIXELorama. Это бесплатный инструмент, без которого сложно представить создание моделек, картинок в PixelART. Он поддерживает наличие нескольких слоев.

Существует несколько аналогов:

- Aseprite имеет более разнообразный функционал, но главный его минус в том, что он платный;
- GraPhicsGal уже достаточно устаревший, мало функций и неудобный интерфейс;
- Photoshop самый базовый редактор, знакомый каждому, подходит под любые задачи, но его функционал заточен не для PixelART, что усложняет работу и делает его избыточным.

Редактор Figma был выбран в качестве инструмента для создания элементов интерфейса и фонов «Главное меню», «Окончание игры», «Пауза», «Настройки», «Магазин улучшений», «Магазин реликвий».

Из аналогов рассматривались:

- Adobe XD, который очень похож по своему функционалу на Figma, но не такой популярный;
- Inkscape, но он намного сложнее и неудобнее Figma, тем более для UI-дизайна.

1.1.7 Функциональные требования

Требования к игре «White Knight» на основе анализа аналогов:

- глубокий геймплей, включающий себя планирование своих сборок, на основе выбора улучшений;
- хорошая оптимизация, позволяющая играть в 60 FPS практически на любом устройстве и в 30 FPS на очень слабых;
- высокая реиграбельность, которая достигается процедурной генерацией;
- динамическая сложность, благодаря смене времени суток/погодных условий и зависимости появления типов врагов от этого.

1.2 Проектирование и разработка

Для создания игры «White Knight» в качестве шаблона проекта была выбрана пустая сцена, это позволило начинать разработку с нуля и настроить все необходимые элементы, как удобно.

Проектирование очень важный пункт в разработке игрового приложения. Оно помогает достигать высокой структуры кода, что значительно повышает его читабельность. Это упрощает дальнейшую поддержку код, добавление новых механик, персонажей, элементов UI. В рамках разработки был выбран принцип проектирования MVC (Model-View-Controller). Он делит логику игры на три важных составляющих.

Главный функционал модели (model) заключается в характеристиках персонажа и его оружия, характеристиках всех врагов, системе сохранения прогресса, системе прокачки персонажа внутри каждой сессии. Она совершенно не зависит от UI (визуального представления игры, с которым взаимодействует непосредственно сам игрок) и от логики управления персонажами. Внешний вид инспектора с моделькой игрового персонажа представлен на рисунке 6.

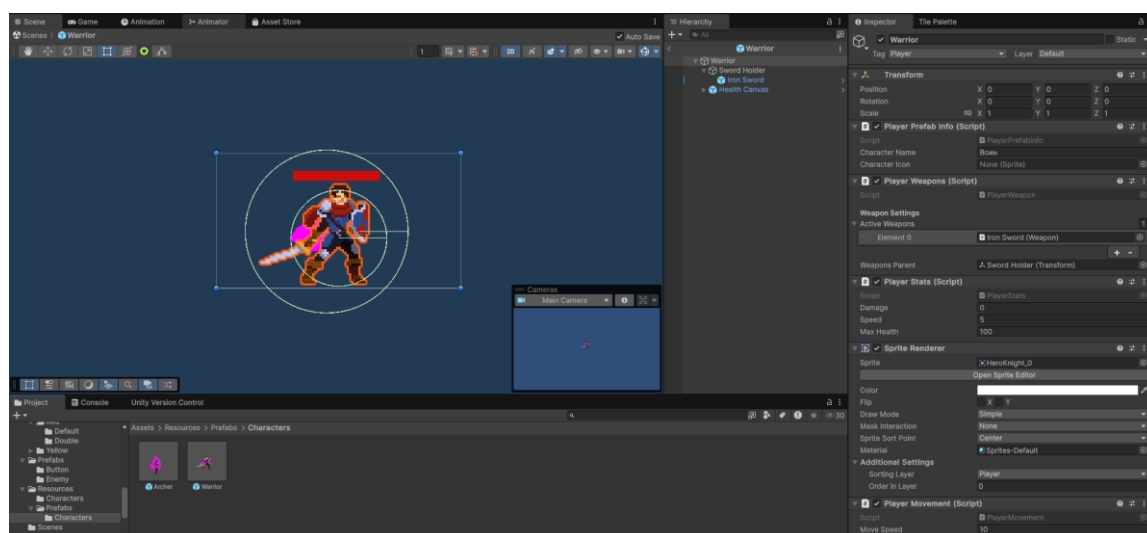


Рисунок 6 – Визуальное представление скриптов, относящихся к характеристикам на примере «Воина»

Пример кода скрипта для управление количеством здоровья персонажа представлен ниже.

```
using UnityEngine;
using UnityEngine.UI;
using System;
using UnityEngine.Events;
[System.Serializable]
public class PlayerHP : MonoBehaviour
{
    [SerializeField] private BloodEffect bloodEffect;
    [SerializeField] private Animator animator; // Добавлено
поле для аниматора
    public static PlayerHP instance;
    public float currentHealth, maxHealth;
    public Slider healthSlider;
    public event Action OnPlayerDeath;
    private void Awake()
    {
        instance = this;
        healthSlider.maxValue = maxHealth;
        healthSlider.value = currentHealth;

        // Автоматическое получение аниматора, если не назначен
в инспекторе
        if (animator == null)
        {
            animator = GetComponent<Animator>();
            if (animator == null)
            {
                Debug.LogWarning("Animator component not found on player!");
            }
        }
    }
}
```

```

    }

    void Start()
    {
        currentHealth = maxHealth;
    }

    public void TakeDamage(float damage)
    {
        currentHealth = Mathf.Max(currentHealth - damage, 0);
        healthSlider.value = currentHealth;
        if (bloodEffect != null)
        {
            bloodEffect.SpawnBlood(transform.position,
(int) damage);
        }
        if (currentHealth <= 0)
        {
            Die(); // Вызываем метод смерти
        }
    }

    private void Die()
    {
        // Активируем триггер смерти
        if (animator != null)
        {
            animator.SetTrigger("Die");
        }
        else
        {
            Debug.LogWarning("Cannot play death animation -
Animator not assigned!");
        }
        OnPlayerDeath?.Invoke();
    }

```

```

// Отключаем объект через 1 секунду (время на проигрывание
анимации)

    Invoke("DisablePlayer", 3f);
}

private void DisablePlayer()
{
    gameObject.SetActive(false);
}

private void OnDestroy()
{
    OnPlayerDeath = null;
}
}

```

В представление (view) входят все элементы UI, анимации ходьбы врагов, анимации ходьбы, бега, атаки, смерти персонажей, фоновая карта, всплывающие цифры нанесенного урона по врагу, визуализация погодных условий и смены времени суток, фоны в главном меню, меню паузы, меню окончания игры, меню улучшений, дизайн кнопок интерфейса, также дизайн моделей врагов, персонажей, иконки апгрейдов и реликвий. Внешний вид главного меню представлен на рисунке 7.

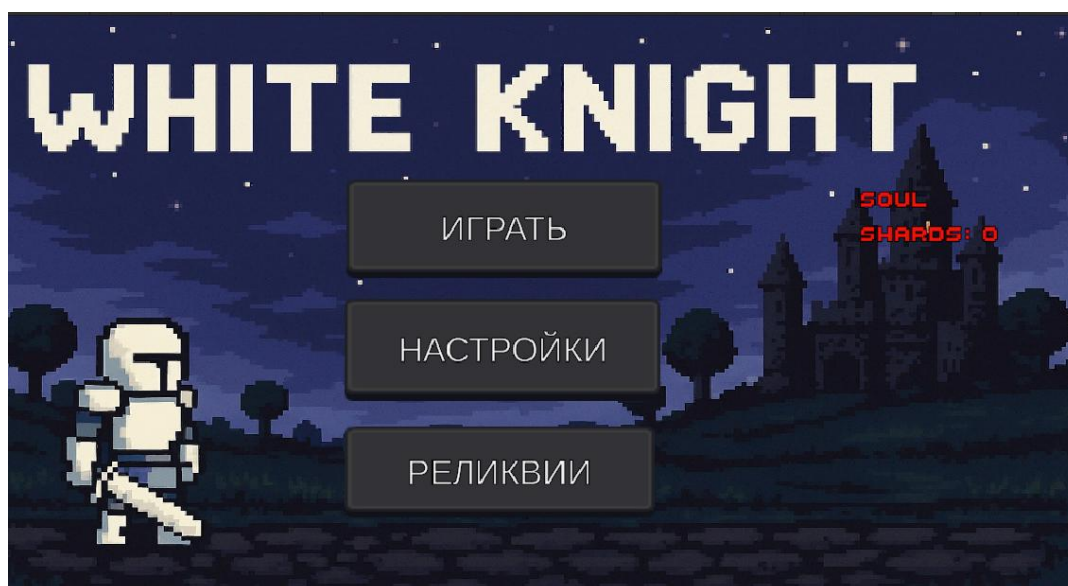


Рисунок 7 – Визуальное представление UI на примере «Главного меню»

Пример кода скрипта для управления UI главного меню представлен ниже.

```
using UnityEngine;

public class MenuManager : MonoBehaviour
{
    [Header("Canvases")]
    [SerializeField] private Canvas mainMenuCanvas;
    [SerializeField] private Canvas characterSelectionCanvas;
    [SerializeField] private Canvas settingsCanvas;

    private void Awake()
    {
        // Инициализация видимости при старте
        InitializeCanvases();
    }

    private void InitializeCanvases()
    {
        // Включаем только главное меню
        mainMenuCanvas.enabled = true;
        // Отключаем все остальные Canvas
        characterSelectionCanvas.enabled = false;
        settingsCanvas.enabled = false;
    }

    public void ShowSettings()
    {
        SetCanvasActive(mainMenuCanvas, false);
        SetCanvasActive(characterSelectionCanvas, false);
        SetCanvasActive(settingsCanvas, true);
    }

    public void ShowCharacterSelection()
    {
        SetCanvasActive(mainMenuCanvas, false);
        SetCanvasActive(characterSelectionCanvas, true);
        SetCanvasActive(settingsCanvas, false);}
}
```

```

public void BackToMainMenu()
{
    SetCanvasActive(settingsCanvas, false);
    SetCanvasActive(characterSelectionCanvas, false);
    SetCanvasActive(mainMenuCanvas, true);
}

private void SetCanvasActive(Canvas canvas, bool state)
{
    canvas.enabled = state;

    var canvasGroup = canvas.GetComponent<CanvasGroup>();
    if (canvasGroup != null)
    {
        canvasGroup.blocksRaycasts = state;
        canvasGroup.interactable = state;
    }
}
}

```

Контроллер (controller) включает в себя обработку всех действий игрока, например, управление движением персонажа, логика перемещения врагов, различные клики и нажатия по меню улучшений, взаимодействие с настройками. Внешний вид обработчика взаимодействия с игрой представлен на рисунке 8.

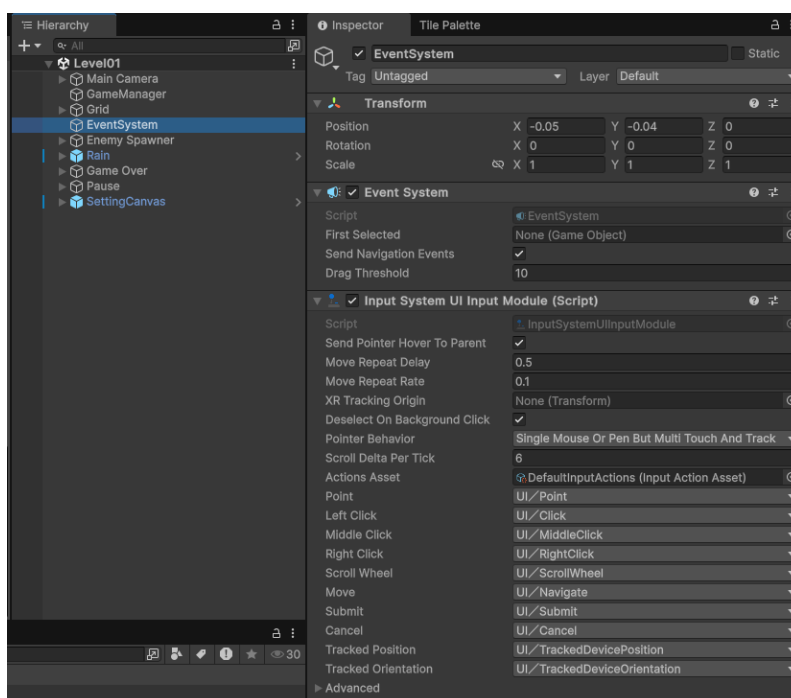


Рисунок 8 – «EventSystem» это обработчик взаимодействий игрока с игрой.

Пример кода скрипта, отвечающего за управление персонажем представлен ниже.

```
using UnityEngine;
public class PlayerMovement : MonoBehaviour
{
    public static PlayerMovement instance;

    public float moveSpeed = 5f;
    private Rigidbody2D rb;
    private Vector2 movement;
    private Animator animator;
    private SpriteRenderer spriteRenderer;

    private void Awake()
    {
        instance = this;
        rb = GetComponent<Rigidbody2D>();
        animator = GetComponent<Animator>();
        spriteRenderer = GetComponent<SpriteRenderer>();
    }
    private void Update()
    {
        // Получаем ввод
        movement.x = Input.GetAxisRaw("Horizontal");
        movement.y = Input.GetAxisRaw("Vertical");
        // Управление анимациями
        UpdateAnimations();
    }
    private void FixedUpdate()
    {
        rb.MovePosition(rb.position + movement * moveSpeed *
Time.fixedDeltaTime);
    }
}
```

Изм.	Лист	№ докум.	Подпись	Дата

```

private void UpdateAnimations()
{
    // Проверяем, движется ли персонаж
    bool isMoving = movement.magnitude > 0.1f;
    // Устанавливаем параметры аниматора
    animator.SetBool("IsRunning", isMoving);
    // Отражение спрайта по горизонтали
    if (movement.x > 0.1f)
    {
        spriteRenderer.flipX = false;
    }
    else if (movement.x < -0.1f)
    {
        spriteRenderer.flipX = true;
    }
}
}

```

Для того, чтобы сохранять характеристики персонажей и загружать их, используя JSON был реализован скрипт SaveSystem, который представлен ниже.

```

using UnityEngine;
using System.IO;
public static class SaveSystem
{
    private static string savePath =
Application.persistentDataPath + "/save.json";
    public static void SaveGame(PlayerData data)
    {
        string json = JsonUtility.ToJson(data);
        File.WriteAllText(savePath, json);
    }
}

```

```

public static PlayerData LoadGame()
{
    if (File.Exists(savePath))
    {
        string json = File.ReadAllText(savePath);
        return JsonUtility.FromJson<PlayerData>(json);
    }

    return new PlayerData(); // Новый файл
}
}

[System.Serializable]
public class PlayerData
{
    public int unlockedCharacters;
    public float permanentUpgrades;
    public int gold;
    public int highScore;
}

```

На рисунке 9 представлена UML- диаграмма взаимодействия.

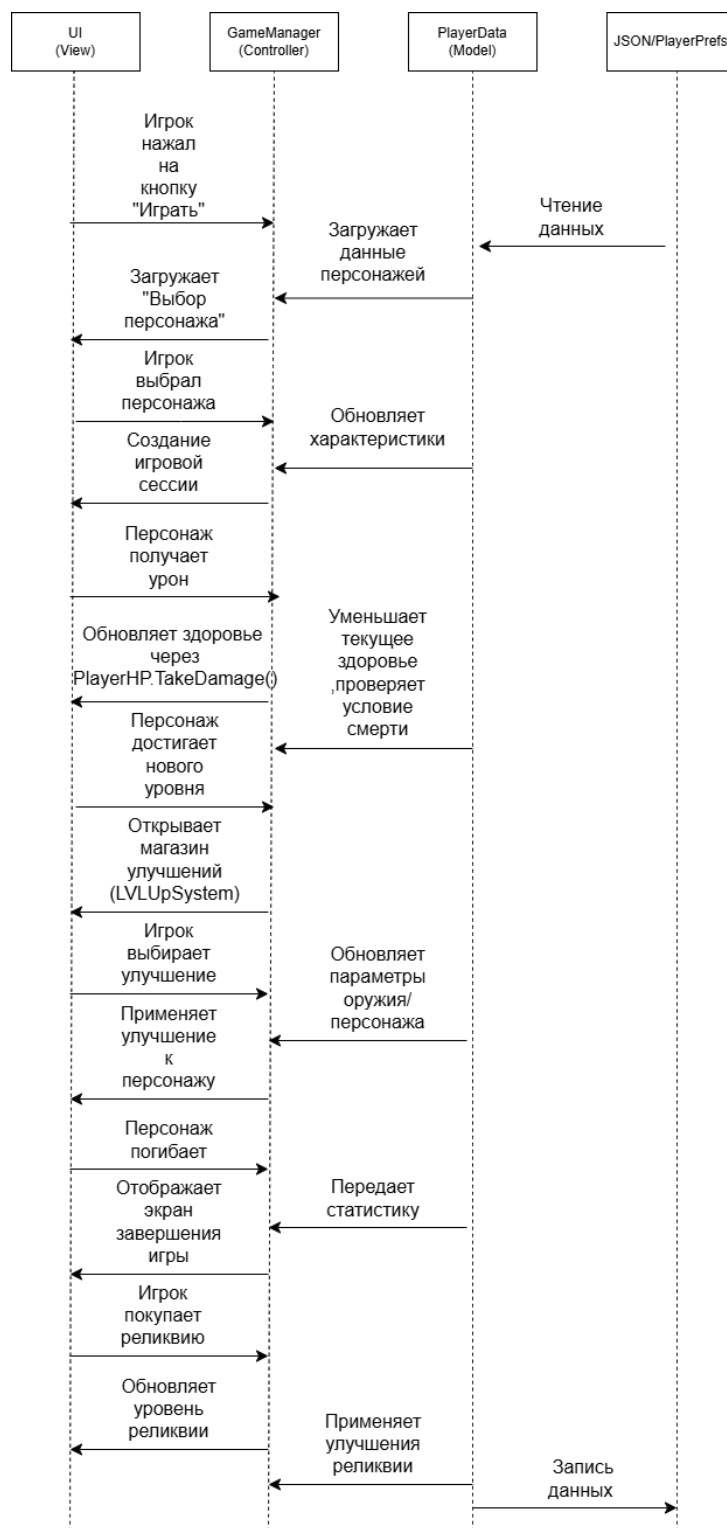


Рисунок 9 – UML - диаграмма взаимодействия

1.2.1 Архитектура пользовательского интерфейса

Интерфейс игры состоит из главного меню, панели настроек, магазина реликвий, игрового HUD, панели улучшений, экрана завершения игры.

Главное меню включает в себя кнопку для начала игры, внешний вид которой представлен на рисунке 10. После нажатия на которую происходит переход в меню выбора персонажа, визуализация на рисунке 11.

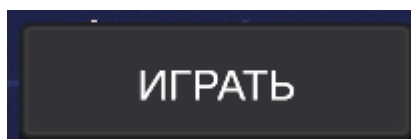


Рисунок 10 – Кнопка «Играть»



Рисунок 11 – Меню выбора персонажа

Изм.	Лист	№ докум.	Подпись	Дата

Также главное меню содержит кнопку «Настройки», внешний вид которой представлен на рисунке 12, которая переносит игрока в панель настроек, где он может настроить разрешение экрана, громкость музыки и звуков игры, язык. На рисунке 13 внешний вид настроек.

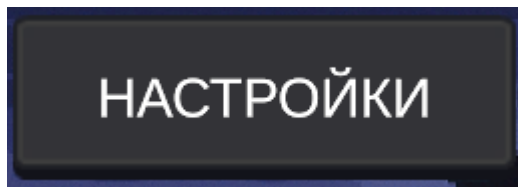


Рисунок 12 – Кнопка «Настройки»



Рисунок 13 – Меню настроек

Изм.	Лист	№ докум.	Подпись	Дата

Кнопка «Реликвии» на рисунке 14 переносит игрока в магазин реликвий, на рисунке 15, где он может покупать перманентные бонусы для открытых персонажей.



Рисунок 14 – Кнопка «Реликвии»

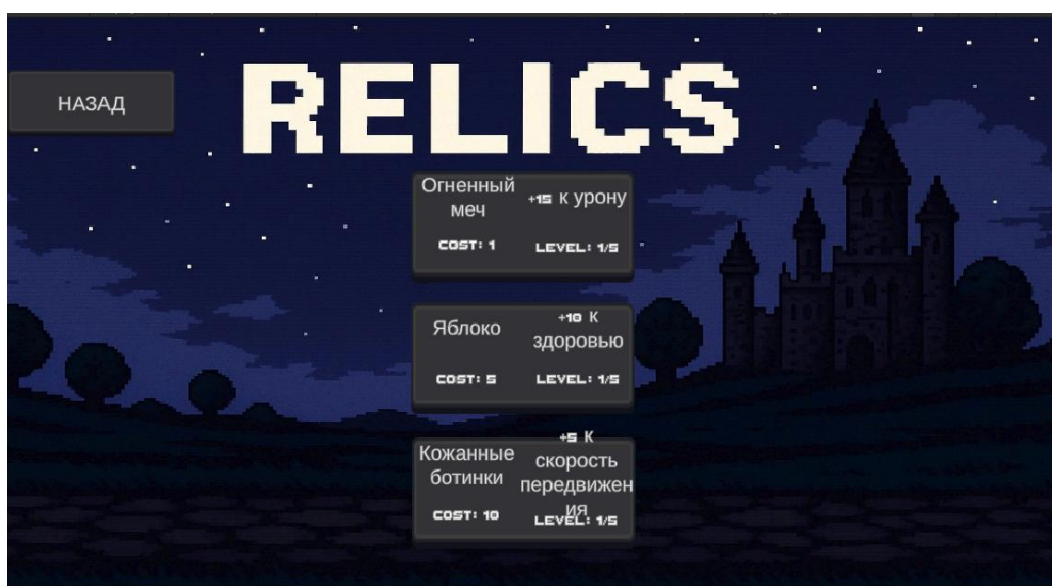


Рисунок 15 – Магазин «Реликвий»

Помимо этого, в «Главном меню» игры есть текстовый элемент, который показывает актуальное количество «осколков душ» у игрока. Это валюта, которая нужна для покупки реликвий, и она сохраняется от сессии к сессии. Внешний вид счетчика представлен на рисунке 16.



Рисунок 16 – Счетчик «Осколков душ»

Весь интерфейс игры был выполнен в стиле PixelART, с помощью бесплатного ПО PIXELorama и Figma.

Внутриигровой HUD состоит из нескольких важных элементов, позволяющих пользователю контролировать процесс игры. На рисунке 17 представлен внешний вид HUD;

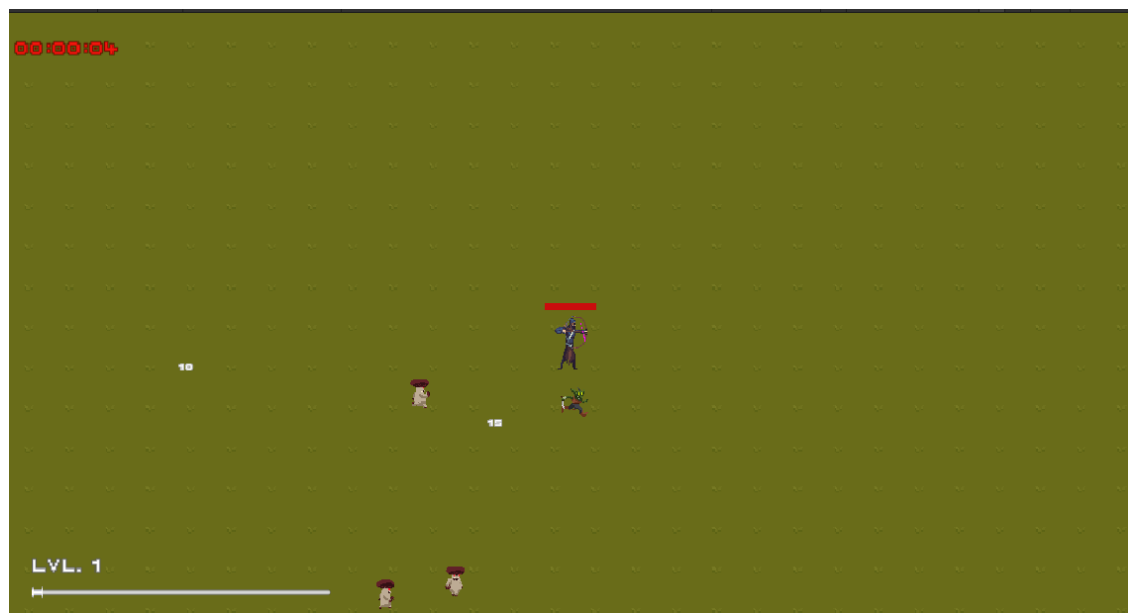


Рисунок 17 – Внешний вид внутриигрового HUD

На рисунке 18 визуализация HPbar – полоска над игровым персонажем, показывающая уровень здоровья на данный момент.



Рисунок 18 – Внешний вид HPbar персонажа

LVLprogress – заполняемая полоска, находящаяся в левом нижнем углу экрана. Над ней расположен текстовый элемент «LVL.», который показывает уровень персонажа на данный момент. Представление на рисунке 19.



Рисунок 19 – Внешний вид отображения текущего уровня и полоски прогресса

Панель выбора улучшений при повышении уровня. Она включает в себя 4 кнопки, которые отвечают за то или иное улучшение характеристик оружия или персонажа. Улучшения выпадают в случайном порядке и не повторяются на одном и том же уровне. На кнопках написаны название улучшений и их описания, чтобы игрок мог планировать свою дальнейшую сборку. Представление на рисунке 20.

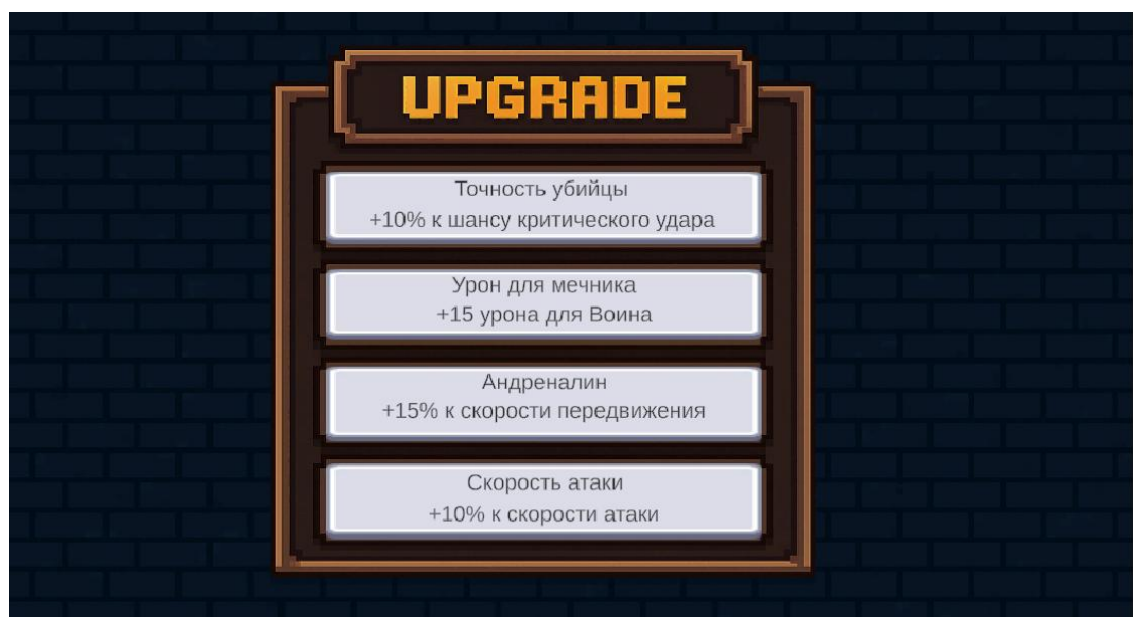


Рисунок 20 – Внешний вид «Магазина улучшений»

Актуальный набор купленных улучшений. После достижения нового уровня и выбора улучшения справа от времени игровой сессии появляется список названий выбранных улучшений. На рисунке 21 в верхнем левом углу внешний вид набора улучшений.



Рисунок 21 – Внешний вид отображения списка текущих улучшений

На рисунке 22 представлено время игровой сессии. Этот элемент HUD показывает, сколько по времени игрок уже смог выжить. Позволяет добавить элемент соревнования и цель ставить каждую сессию новый рекорд. Все элементы HUD обновляются во время игры с помощью привязки к скриптам.



Рисунок 22 – Время игровой сессии

Экран завершения игры показывает всю статистику за сессию (количество убитых врагов, продолжительность игровой сессии и лучшее время сессии за все время). Позволяет по нажатию кнопки «В главное меню» вернуться в главное меню и начать новую сессию. Внешний вид представлен на рисунке 23.

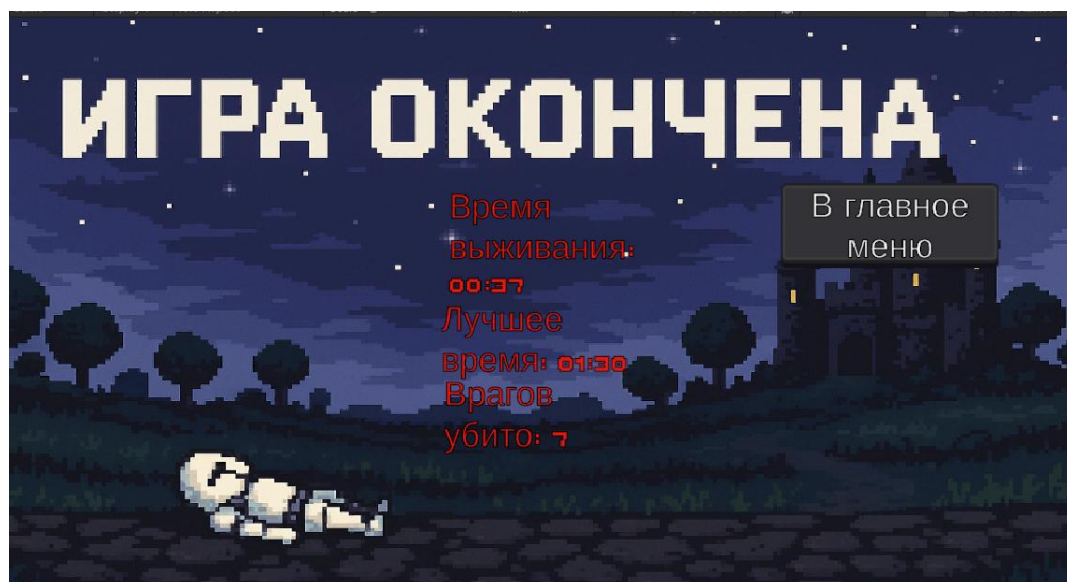


Рисунок 23 – Экран завершения игры

Для создания всех анимаций персонажей и врагов использована встроенная функция в Unity – Animation. Каждый спрайт был анимирован с помощью ключевых кадров. На рисунке 24 визуализация функции Animation.

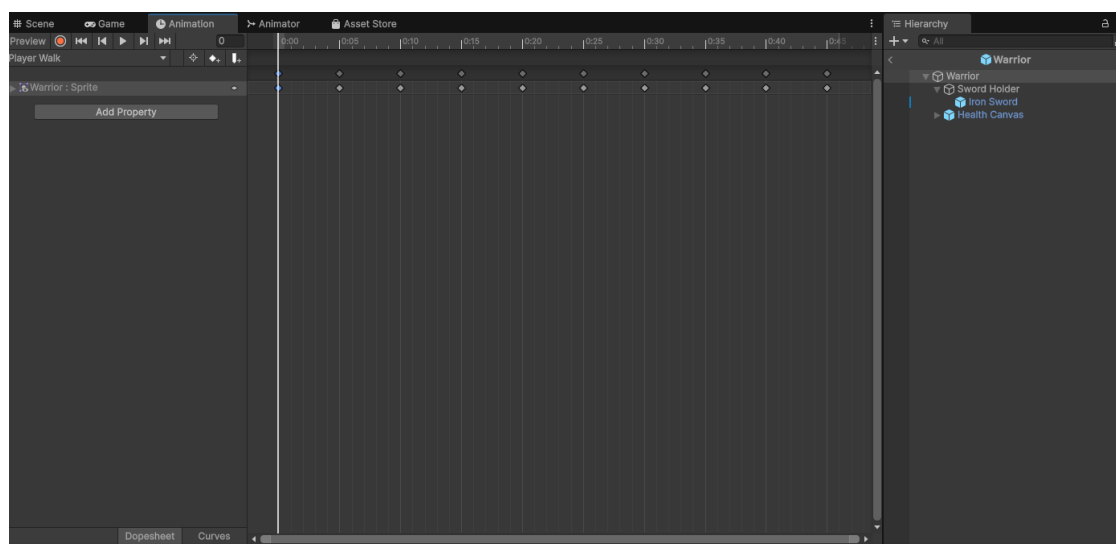


Рисунок 24 – Animation

Берется каждый кадр в отдельности и добавляется на панель Animation в выбранные промежутки времени, после чего Unity сам создает анимацию. Внешний вид Sprite можно посмотреть на рисунке 25.

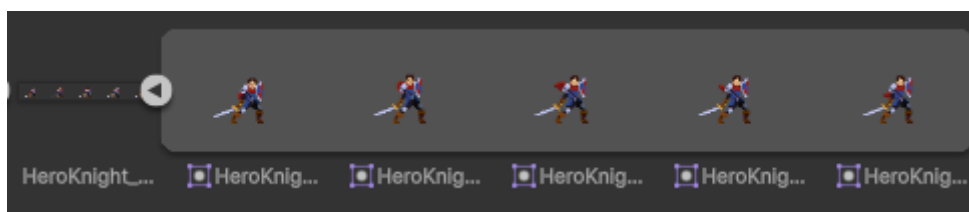


Рисунок 25 – внешний вид Sprite

Для синхронизации с логикой игры был использован функционал Animator, позволяющий соединять и настраивать анимационные клипы, созданные в Animation. Он нужен, чтобы построить связи и последовательность анимаций. Синхронизация достигается путем написания скрипов, контролирующих логику включения анимаций и настройки задержек. Внешний вид Animator для «Воина» представлен на рисунке 26.

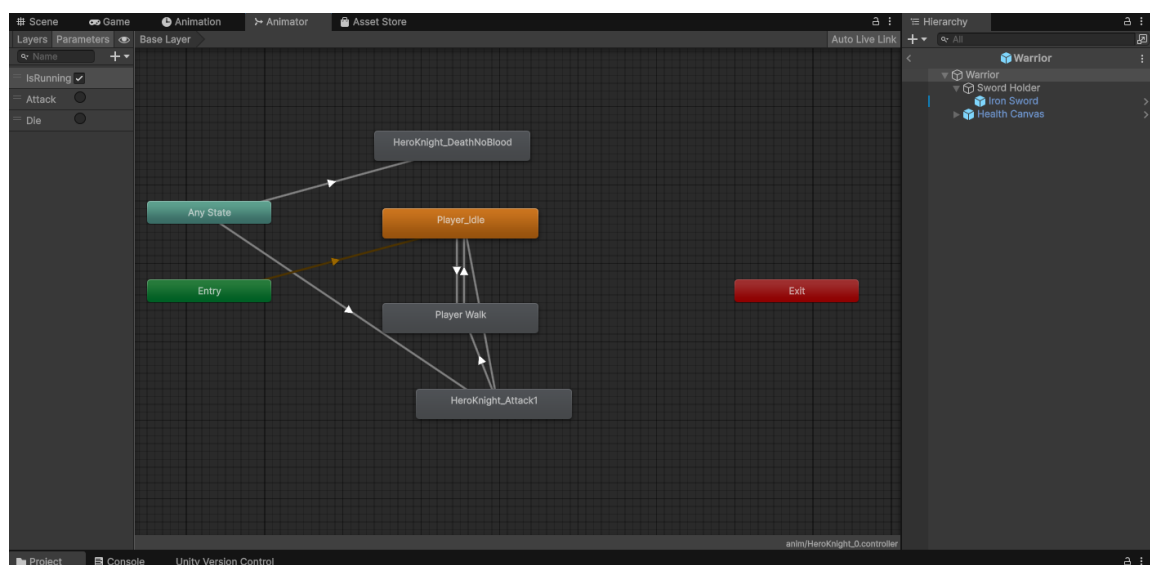


Рисунок 26 – Animator для персонажа «Воин»

1.2.2 Архитектура программы

Архитектура игрового приложения «White Knight» состоит из нескольких основных модулей, полная структура которых представлена на рисунке 27.

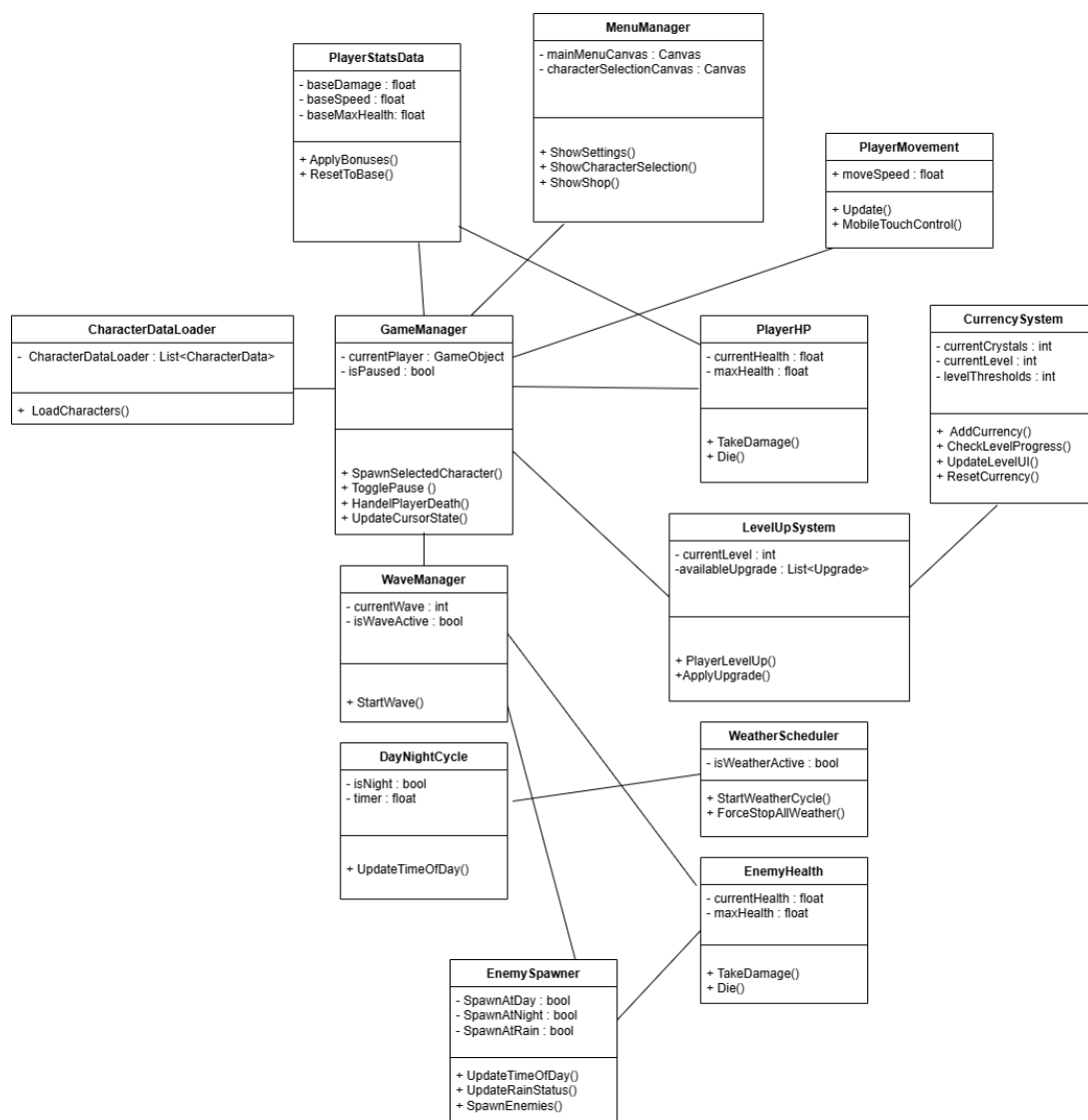


Рисунок 27 – UML-диаграмма классов, архитектура программы

Геймплей: включает в себя системы управления персонажами, появление врагов, их передвижение, обработку нанесения урона врагу и персонажу, систему улучшений, изменение погоды и времени суток

За геймплей отвечают следующие скрипты и классы:

- PlayerMovement(). Управление персонажами;
- PlayerHP(). Состояние здоровья персонажа и нанесения ему урона;

- EnemyMovement(). Передвижение врагов;
- LvlUpSystem(). Система улучшений;
- PlayerWeapon(). Логика оружия персонажей;
- EnemySpawner(). Появление врагов на сцене;
- WaveManager(). Управление настройками волн врагов;
- PlayerStats(). Характеристики персонажа;
- GameManager(). Менеджер управления основных скриптов;
- Weapon(). Характеристики оружия персонажей и логика нанесения урона врагам;
- CurrencySystem(). Система повышения уровня;
- EnemyHealth(). Здоровье врагов, получение урона от персонажей, логика выпадения кристаллов и осколков;
- DayNightCicle(). Логика смены времени суток;
- WeatherSystem(). Логика смены погодных условий;
- CharacterSelectionSystem(). Выбор игрового персонажа.

Модуль сохранений записывает и загружает прогресс игрока на данный момент, актуальные настройки, использует для этого JSON систему для хранения данных.

За модуль сохранения отвечают следующие скрипты и классы:

- CharacterData(). Класс для определения характеристик, которые нужно сохранять и загружать;
- CharacterDataLoader(). Загрузка из JSON файлов.

Интерфейс управляет отображением всех визуальных панелей (главного меню, HUD, экрана завершения игры, магазина улучшений, панель паузы).

За интерфейс отвечают следующие скрипты и классы:

- DamagePupUp(). Появление нанесенного урона врагам;
- DamagePupUpPool(). Pool для оптимизации урона;
- MenuManager(). Управление главным меню игры;
- GameManager(). Менеджер управления основных скриптов;

- TogglePause(). Появление панели паузы;
- OpenSettings(). Открытие настроек;
- CloseSettings(). Закрытие настроек;
- ResumeGame(). Возобновление игры после паузы;
- ShowGameOverPanel(). Появление панели окончания игры;
- SoundManager(). Управление уровнем музыки и звуков;
- ResolutionSetting(). Настройка разрешения экрана;
- Language(). Смена языка игры в настройках.

Модуль звука отвечает за воспроизведение музыки игры и игровых эффектов:

- SoundManager(). Управляет громкостью звуков через настройки.

1.2.3 Динамика работы приложения

- главное меню, где игрок начинает новую игру, а далее выбирает игрового персонажа, корректирует настройки и покупает реликвии;
- игровой процесс – управление передвижением персонажа, нанесение урона врагам, сбор кристаллов для повышения уровня и выбора улучшений;
- панель выбора улучшений, который появляется по достижению нового уровня;
- панель завершения игры появляется после смерти игрового персонажа, на ней показывается статистика за сессию и предлагается вернуться в главное меню, для запуска новой сессии.

На рисунке 28 представлена диаграмма состояний для динамики работы.

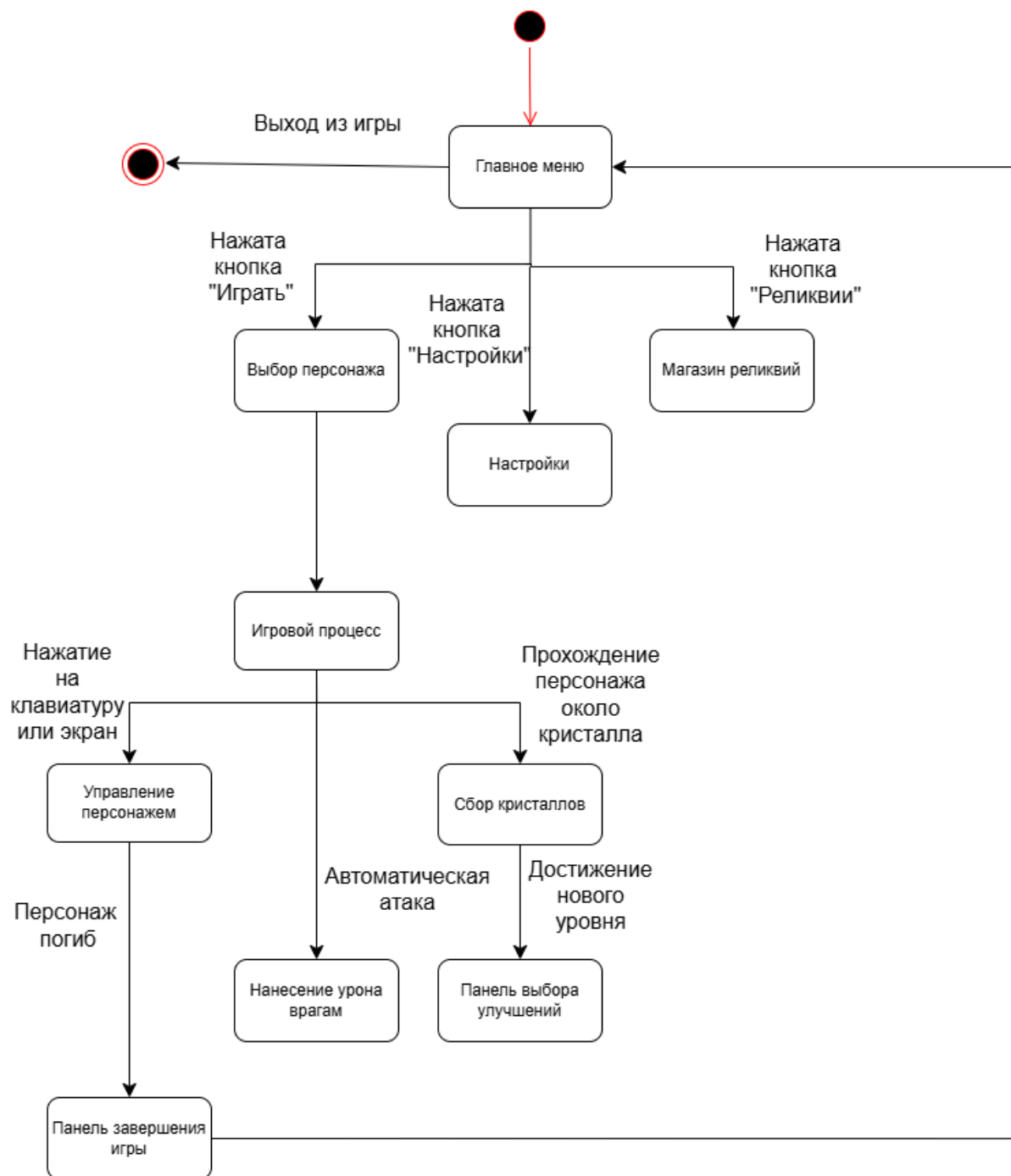


Рисунок 28 - UML-диаграмма состояний

1.2.4 Тестирование и отладка

Для тестирования игры были использованы различные платформы (ПК на Win10 и Android), чтобы выявить различные возникающие ошибки, баги, проверить стабильность игры и производительность.

При отладки были найдены следующие проблемы:

- оптимизация игры, когда на сцене находилось огромное количество врагов, была достигнута путем использования ObjectPool, для всех основных игровых объектов;
- баги при сохранении прогресса, когда игра была завершена некорректно, баги при загрузке характеристик из JSON файлов на Android;
- плохое построение баланса сложности игры на разных стадиях прогресса в сессии. Когда-то было слишком легко, когда-то чрезвычайно сложно.

Внешний вид игры на Android представлен на рисунке 29.



Рисунок 29 – внешний вид игры на Android

2 Безопасность жизнедеятельности

2.1 Правила безопасности при использовании ПК

При разработке игрового приложения на Unity чрезвычайно важно учитывать правила безопасности в работе с персональным компьютером. Ниже привожу основные риски и рекомендации по их устранению, которые основаны на требованиях ГОСТ 12.0.003-2015.

Риск ухудшения зрения и утомления глаз. Длительная работа перед монитором без перерыва очень быстро может привести к усталости глаз, а со временем к ухудшению зрения. Неправильное освещение рабочего места и различные блики на экране, также увеличивают напряжение на глаза.

Рекомендации:

– нужно делать перерывы каждые 40-60 минут и делать разминку для глаз (каждые 20 минут смотреть на другой объект на расстоянии 6 метров на протяжении 20 секунд);

– требуется настроить яркость и контраст монитора, чтобы при текущем уровне освещения в помещении на экране монитора не появлялись блики и отсветы;

– монитор должен находиться на расстоянии не менее 50-70 см от глаз, а верхний край монитора необходимо расположить на уровне или немного ниже уровня глаз.

Риск травм и заболеваний опорно-двигательной системы. При работе за компьютером в неправильной позе могут развиваться боли в спине, шее и запястьях. Если ваше рабочее кресло не эргономичное, это создает дополнительную нагрузку на позвоночник и шею.

Рекомендации:

– необходимо использовать кресло, у которого есть функция поддержки поясницы и регулирования уровня высоты, при этом ваши стопы должны полностью стоять на полу, а колени должны быть согнуты под углом в 90 градусов;

– при работе с клавиатурой и мышкой рекомендуется использовать подставки для запястий.

Риск пожара и электрического удара. Чрезмерная нагрузка на розетки, использование кабелей, у которых нарушена изоляционная оплетка или любого другого неисправного оборудования может привести к возгоранию рабочего помещения или удару электрическим током.

Рекомендация – перед подключением оборудования к электрической сети, необходимо убедиться в исправности розетки, проверить кабель на повреждение оплетки и только после этого приступать к подключению.

2.2 Правила безопасности при использовании смартфона

При тестировании и использовании игрового приложения на мобильных устройствах необходимо знать о различных рисках для здоровья и рекомендации по их устранению.

Риск зависимости и негативного воздействия на здоровье. Постоянное использование смартфона может привести к психологической зависимости от использования устройства.

Рекомендации:

– необходимо контролировать себя и ограничивать время использования смартфона в течение дня, тем более перед сном;

– нужно делать перерывы каждые 30–40 минут для того, чтобы размять глаза и дать им время отдохнуть.

Негативное влияние на сон. Синий свет экрана смартфона уменьшает выработку мелатонина, что может привести к нарушению сна.

Рекомендации:

- перед сном включайте режим "Сна" или используйте приложения, фильтрующие синий свет;
- откажитесь от использования смартфона за 1–2 часа до сна.

Риск отвлечения внимания водителя. Использование смартфона во время вождения отвлекает внимание водителя, тем самым значительно повышает риск аварий. Для предотвращения аварийных ситуаций никогда не используйте смартфон за рулем.

					ПГУ 1.090301.19.001 ПЗ	Лист
						48
Изм.	Лист	№ докум.	Подпись	Дата		

Заключение

В рамках ВКР было разработано игровое приложение в жанре «RogueLike» при использовании игрового движка Unity и языка программирования C#.

Разработанная игра «White Knight» реализована в соответствии с поставленными в начале дипломной работы техническими и функциональными требованиями. Получилось добиться высокого уровня производительности, разнообразить геймплей, чтобы игроку не становилось скучно и портировать игру не только на ПК, но и на телефон. Использовались Unity DOTS и Object Pool и это позволило достичь стабильного FPS, при нахождении большого количества врагов на сцене. Использование Burst Compiler и Hybrid Renderer обеспечило отличную оптимизацию ресурсов и поддержку устройств с разными техническими характеристиками, что позволит поиграть в мою игру людям с не самыми производительными устройствами.

В дальнейшем можно выложить игру на платформы и продолжить ее поддержку.

Постоянно добавлять новый контент. Новые персонажи с уникальными способностями и типами оружия, добавить новые карты с уникальными погодными условиями и соответствующими врагами, интегрировать систему облачных сохранений, адаптация игры для других платформ, например IOS, добавить косметические предметы и различные игровые ивенты для удержания игрока, за которые он будет получать уникальные награды.

					ПГУ 1.090301.19.001 ПЗ	Лист
						49
Изм.	Лист	№ докум.	Подпись	Дата		

Список использованных источников

1. Голдрат, Дж. Unity в действии. Разработка многоплатформенных игр на C# / Дж. Голдрат. – СПб.: Питер, 2021. – 480с.
2. Бонд, Д. Игровой движок Unity. Руководство по разработке игр / Д. Бонд. – М.: ДМК Пресс, 2022. – 352с.
3. Фаулер, М. Архитектура корпоративных программных приложений / М. Фаулер. – М.: Вильямс, 2020. – 544с.
4. Эриксон, К. Искусство оптимизации приложений на Unity / К. Эриксон. – СПб.: БХВ-Петербург, 2023. – 320с.
5. Иванов, А.А. Оптимизация игровых процессов в жанре RogueLike // Журнал "Программная инженерия", 2023.
6. Smith, J. Procedural Generation in Modern Roguelike Games // Game Development Journal, 2022.
7. Unity Technologies. Unity Official Documentation: Universal Render Pipeline (URP) [Электронный ресурс] – URL: <https://docs.unity3d.com/Manual/URP.html> (дата обращения 08.05.2025).
8. Официальный сайт Unity [Электронный ресурс] – URL: <https://unity.com> (дата обращения 08.05.2025).
9. Unity Learn [Электронный ресурс] – URL: <https://learn.unity.com> (дата обращения 10.05.2025).
10. Документация по C# [Электронный ресурс] – URL: <https://learn.microsoft.com/ru-ru/dotnet/csharp/>
11. Руководство по JSON в Unity [Электронный ресурс] – URL: <https://docs.unity3d.com/Manual/JSONSerialization.html> (дата обращения 10.05.2025).
12. Описание шаблона проектирования MVC [Электронный ресурс] – URL: https://pub.dev/documentation/mvc_pattern/latest/ (дата обращения 20.05.2025).

Приложение А

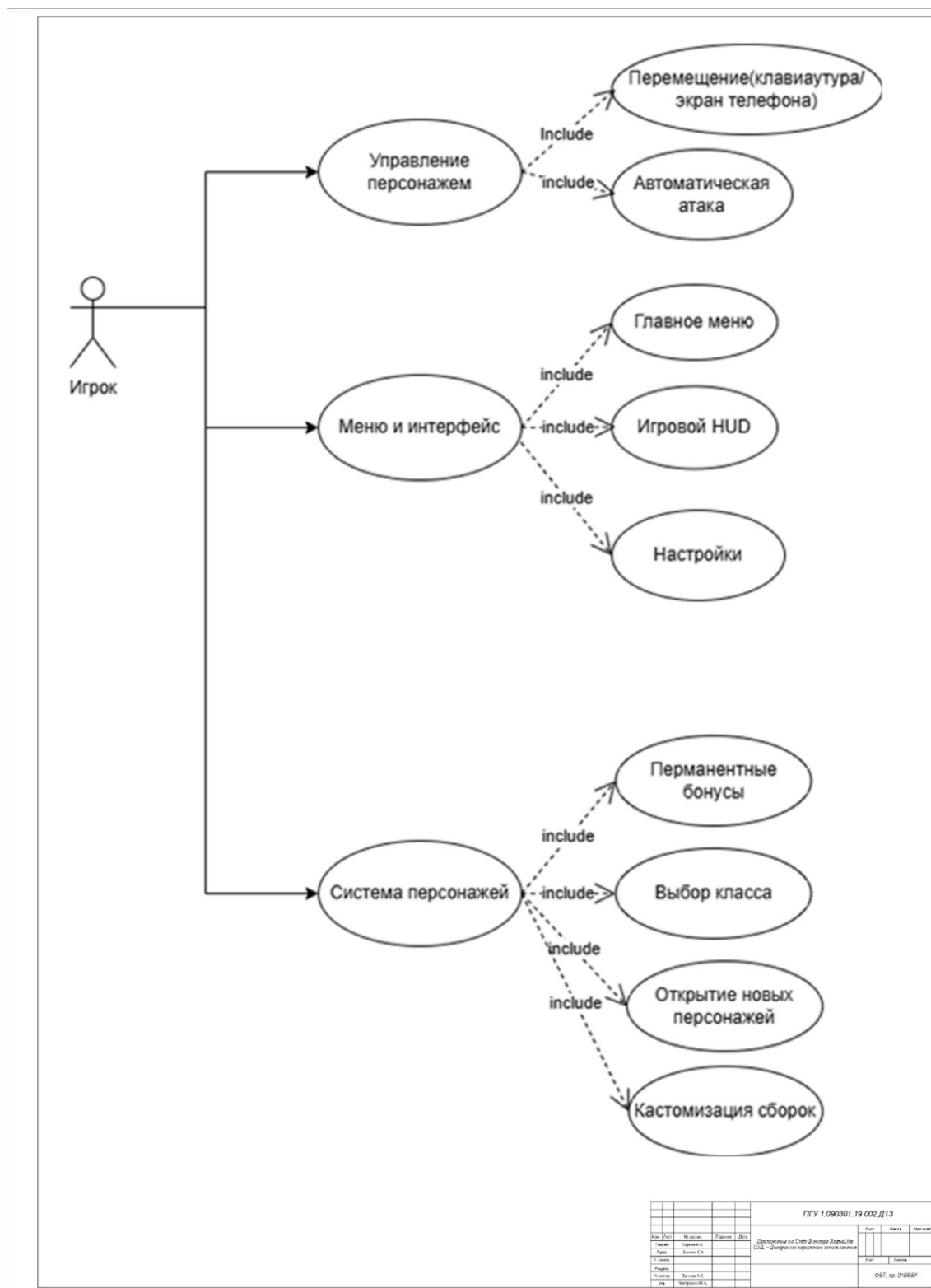
(Обязательное)

Приложение на Unity в жанре RogueLike

UML-диаграммы

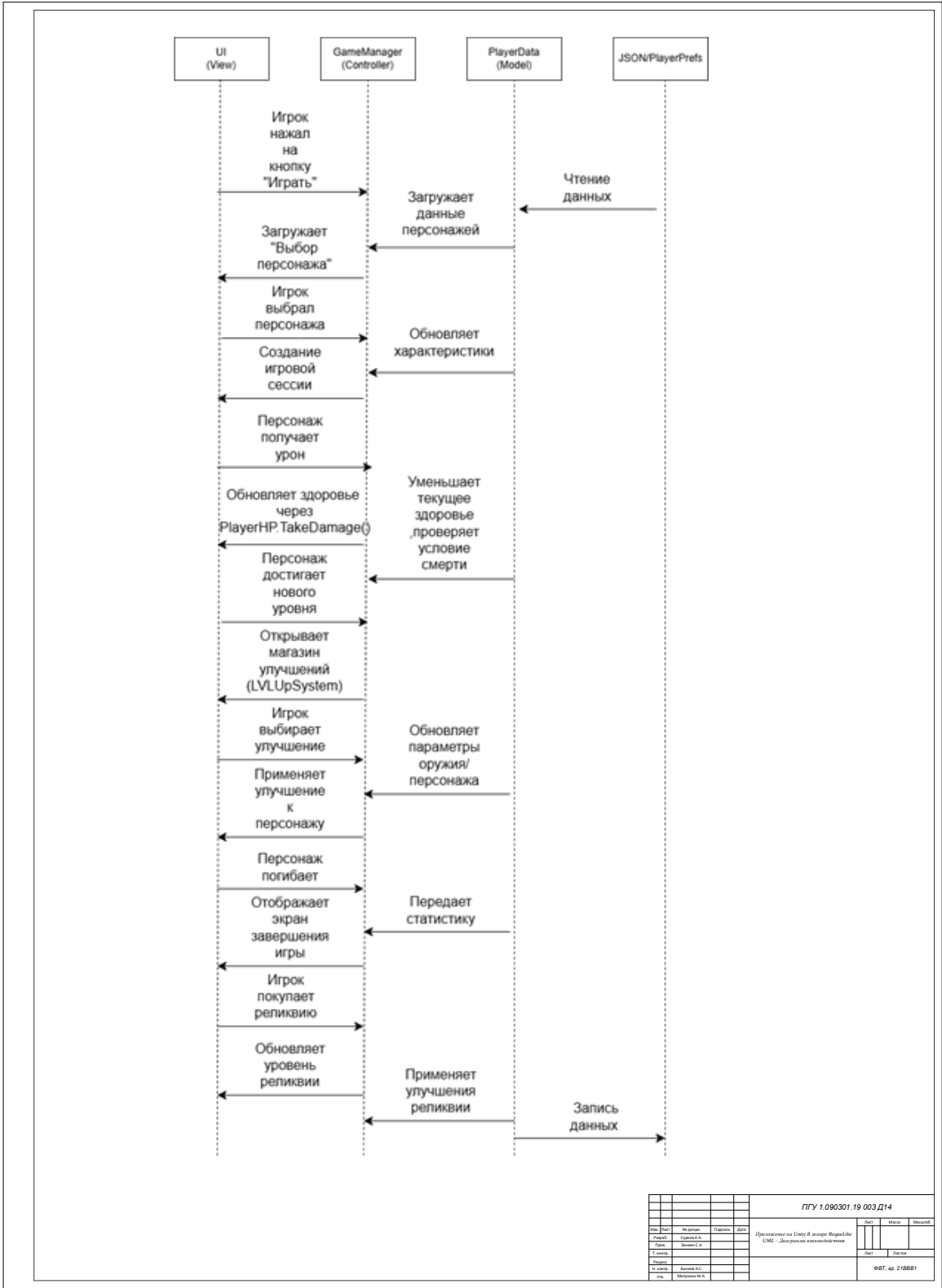
					ПГУ 1.090301.19.001 ПЗ	Лист
						51
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение А.1 – UML – диаграмма вариантов использования

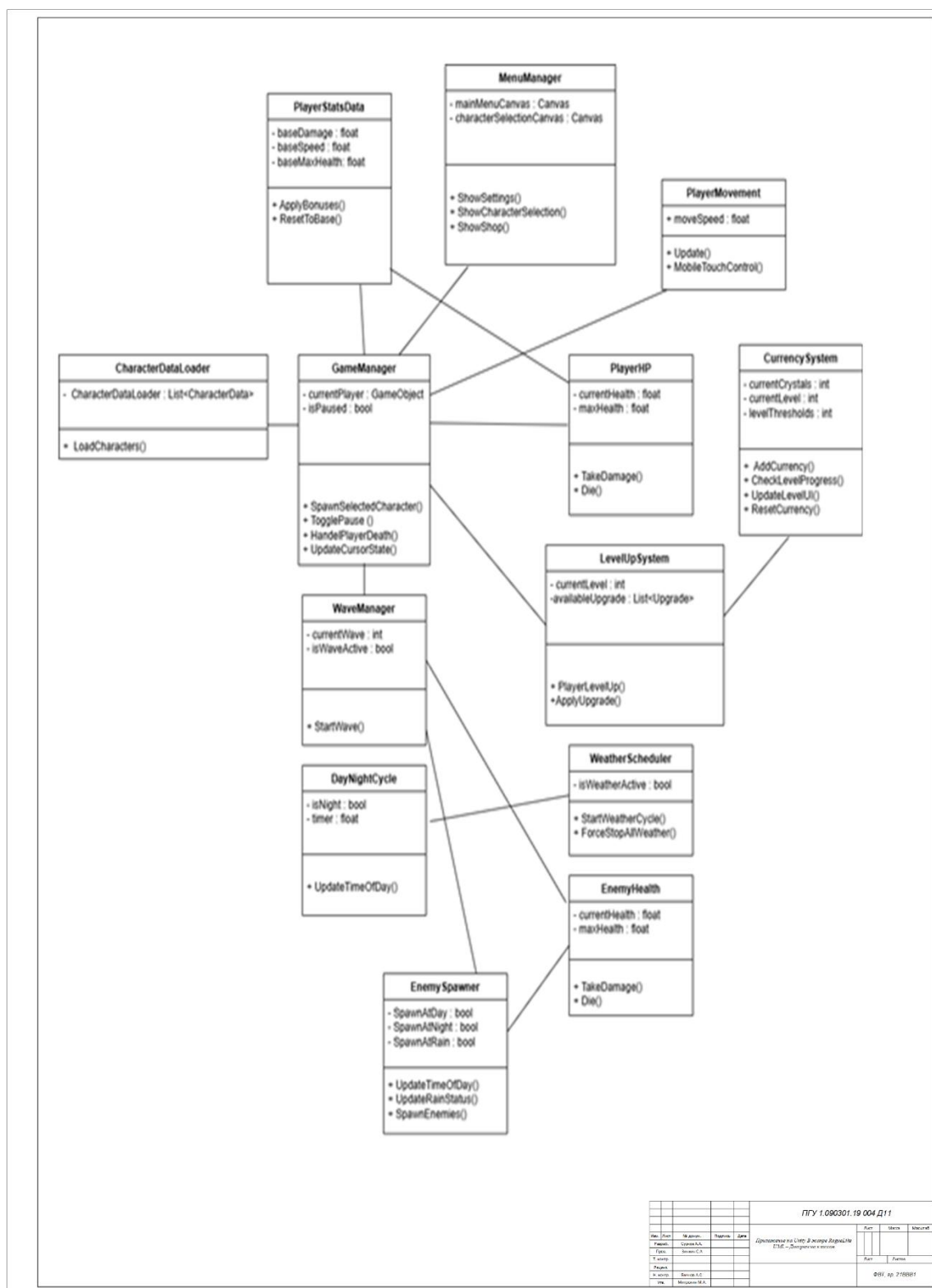


Изм.	Лист	№ докум.	Подпись	Дата

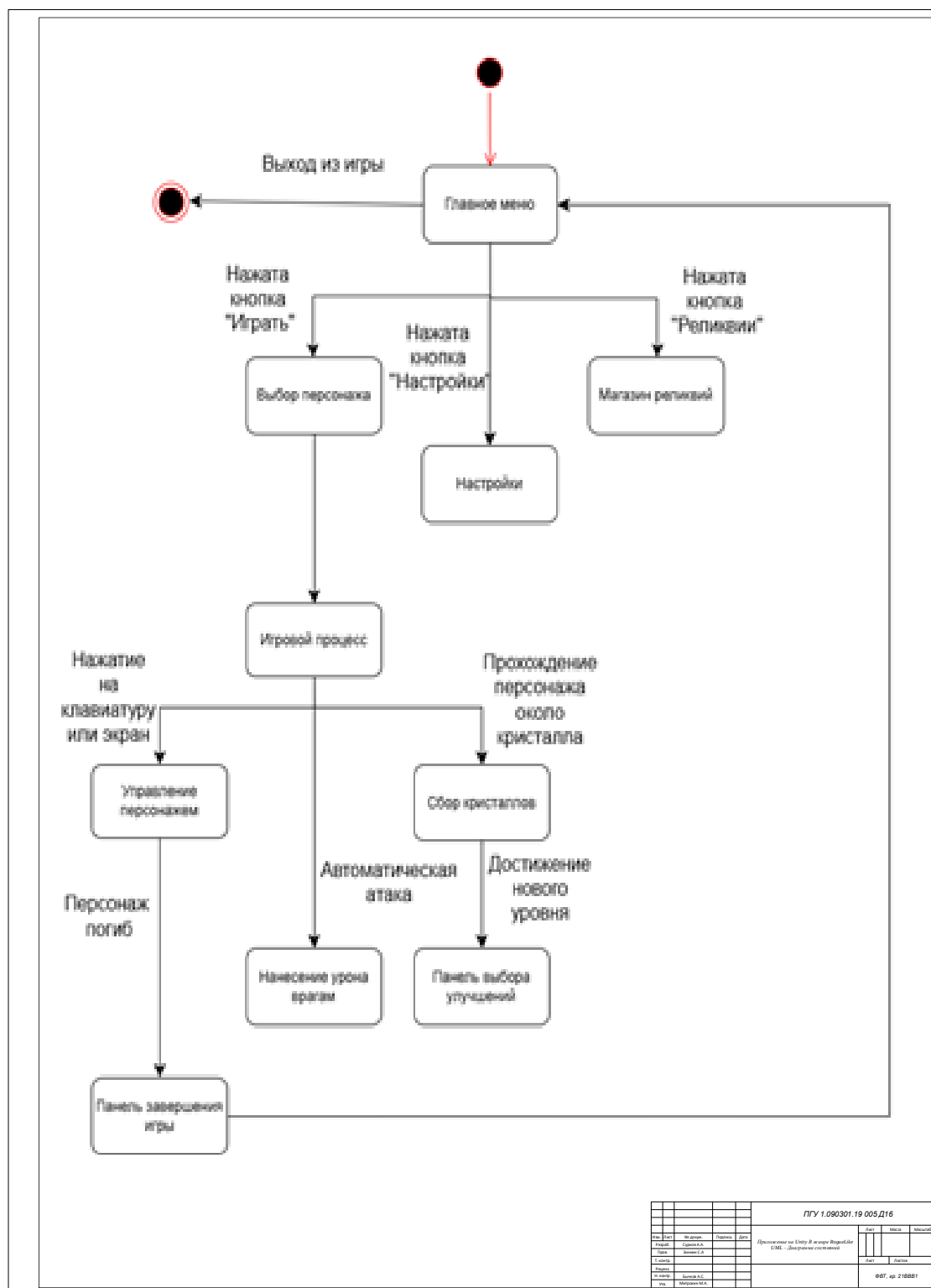
Приложение А.2 – UML – диаграмма взаимодействия



Приложение А.3 – UML – диаграмма классов



Приложение А.4 – UML – диаграмма состояний



Приложение Б

(Обязательное)

Игровое приложение на Unity в жанре RogueLike

Листинг программы

					ПГУ 1.090301.19.001 ПЗ	Лист
						56
Изм.	Лист	№ докум.	Подпись	Дата		

Файл «Arrow.cs»

```
using UnityEngine;

public class Arrow : MonoBehaviour
{
    public float damage = 10f;
    public float speed = 15f;
    public GameObject hitEffect;

    private Transform target;
    private Rigidbody2D rb;
    private bool isActive;

    private void Awake()
    {
        rb = GetComponent<Rigidbody2D>();
    }

    public void Initialize(Transform target, float damage, bool
isCritical)
    {
        this.target = target;
        this.damage = damage;
        isActive = true;
    }

    private void Update()
    {
        if (!isActive) return;

        if (target != null)
        {
            Vector2 direction = (target.position -
transform.position).normalized;
            rb.linearVelocity = direction * speed; // Изменено
здесь
```

					ПГУ 1.090301.19.001 ПЗ	Лист
						57
Изм.	Лист	№ докум.	Подпись	Дата		

```

        float angle = Mathf.Atan2(direction.y, direction.x) *
Mathf.Rad2Deg;

        transform.rotation = Quaternion.AngleAxis(angle,
Vector3.forward);
    }
    else
    {
        ReturnToPool();
    }
}

private void OnTriggerEnter2D(Collider2D collision)
{
    if (!isActive) return;

    if (collision.TryGetComponent(out EnemyHealth enemy))
    {
        enemy.TakeDamage(damage);
        SpawnHitEffect();
        ReturnToPool();
    }
    else if (collision.gameObject.layer ==
LayerMask.NameToLayer("Obstacles"))
    {
        SpawnHitEffect();
        ReturnToPool();
    }
}

private void SpawnHitEffect()
{
    if (hitEffect != null)
    {
        Instantiate(hitEffect, transform.position,
Quaternion.identity);
    }
}
}

```

```

private void ReturnToPool()
{
    isActive = false;
    rb.linearVelocity = Vector2.zero; // И здесь
    ArrowPool.Instance.ReturnArrow(gameObject);
}
}

```

Файл «ArrowPool.cs»

```

using UnityEngine;
using System.Collections.Generic;

public class ArrowPool : MonoBehaviour
{
    public static ArrowPool Instance;
    public GameObject arrowPrefab;
    public int poolSize = 10;

    private Queue<GameObject> arrowPool = new
Queue<GameObject>();

    private void Awake()
    {
        Instance = this;
        InitializePool();
    }

    private void InitializePool()
    {
        for (int i = 0; i < poolSize; i++)
        {
            GameObject arrow = Instantiate(arrowPrefab);
            arrow.SetActive(false);
            arrowPool.Enqueue(arrow);
        }
    }
}

```

```

public GameObject GetArrow()
{
    if (arrowPool.Count == 0)
    {
        ExpandPool();
    }
    GameObject arrow = arrowPool.Dequeue();
    arrow.SetActive(true);
    return arrow;
}

public void ReturnArrow(GameObject arrow)
{
    arrow.SetActive(false);
    arrowPool.Enqueue(arrow);
}

private void ExpandPool()
{
    for (int i = 0; i < poolSize / 2; i++)
    {
        GameObject arrow = Instantiate(arrowPrefab);
        arrow.SetActive(false);
        arrowPool.Enqueue(arrow);
    }
}
}

```

Файл «Camera.cs»

```

using UnityEngine;

public class CameraController : MonoBehaviour
{
    private GameObject player;

    private void Start()
    {

```

Изм.	Лист	№ докум.	Подпись	Дата

```

        GameManager.OnPlayerSpawned += SetPlayer;
    }
    private void SetPlayer(GameObject spawnedPlayer)
    {
        player = spawnedPlayer;
        Debug.Log("Камера получила персонажа: " + player.name);
    }
    void Update()
    {
        if (player != null)
            transform.position = new
Vector3(player.transform.position.x, player.transform.position.y, -10f);
    }

    private void OnDestroy()
    {
        GameManager.OnPlayerSpawned -= SetPlayer;
    }
}

```

Файл «CharacterData.cs»

```

using UnityEngine;
using UnityEngine.UI; // Добавляем для Button
[System.Serializable]
public class CharacterData
{
    public Button selectionButton; // Кнопка выбора (назначается
вручную)
    public GameObject characterPrefab; // Префаб персонажа
    [HideInInspector] public string characterName; // Будет браться
с префаба
    [HideInInspector] public Sprite characterIcon; // Будет браться
с кнопки
}

```

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.19.001 ПЗ

Лист

61

Файл «CharacterDataLoader.cs»

```
// CharacterDataLoader.cs
using System.Collections.Generic;
using UnityEngine;
public static class CharacterDataLoader
{
    private static List<CharacterData> _characters;
    public static void LoadCharacters()
    {
        TextAsset jsonFile =
Resources.Load<TextAsset>("Characters/characters");
        _characters =
JsonUtility.FromJson<CharacterList>(jsonFile.text).characters;
    }
    public static CharacterData GetCharacter(int id)
    {
        return _characters.Find(c => c.id == id);
    }
    public static List<CharacterData> GetAllCharacters()
    {
        return _characters;
    }
}
```

Файл «CharacterSelectionSystem.cs»

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

public class CharacterSelectionManager : MonoBehaviour
{
    [Header("Настройки персонажей")]
```

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.19.001 ПЗ

Лист

62

```

        public List<CharacterData> characters = new
List<CharacterData>();
        [Header("Настройки игры")]
        [SerializeField] private string gameSceneName = "Level01";
        private GameObject selectedCharacterPrefab;
        private void Start()
        {
            // Назначаем обработчики для всех кнопок
            foreach (CharacterData character in characters)
            {
                // Получаем данные с префаба
                PlayerPrefabInfo info =
character.characterPrefab.GetComponent<PlayerPrefabInfo>();
                if (info != null)
                {
                    character.characterName = info.characterName;
                    // Настраиваем кнопку
                    Image icon =
character.selectionButton.GetComponentInChildren<Image>();
                    if (icon != null) character.characterIcon =
icon.sprite;

                    Text buttonText =
character.selectionButton.GetComponentInChildren<Text>();
                    if (buttonText != null) buttonText.text =
info.characterName;
                }
                // Назначаем обработчик
                character.selectionButton.onClick.AddListener(() =>
SelectCharacter(character));
            }
        }
    }

```

Изм.	Лист	№ докум.	Подпись	Дата

```

public void SelectCharacter(CharacterData character)
{
    selectedCharacterPrefab = character.characterPrefab;
    Debug.Log("Выбран: " + character.characterName);
    StartGame();
}

public void StartGame()
{
    if (selectedCharacterPrefab != null)
    {
        Debug.Log("Загрузка сцены: " + gameSceneName);
        Debug.Log("Выбранный префаб: " +
selectedCharacterPrefab.name);
        PlayerPrefs.SetString("SelectedCharacterPrefab",
            selectedCharacterPrefab.name);
        SceneManager.LoadScene(gameSceneName);
    }
    else
    {
        Debug.LogWarning("Персонаж не выбран!");
    }
}
}

```

Файл «Crystal.cs»

```

using UnityEngine;

public class Crystal : MonoBehaviour
{
    [Header("Настройки")]
    public int value = 1;
    public float magnetRadius = 3f;
    public float attractSpeed = 8f;
    public GameObject collectEffect;
}

```

```

private Transform player;
private bool isCollected;
private CrystalPool pool;

public void Initialize(CrystalPool crystalPool)
{
    pool = crystalPool;
    player =
GameObject.FindGameObjectWithTag("Player").transform;
    isCollected = false;
    Invoke(nameof(EnableMagnet), 0.5f);
}

private void EnableMagnet()
{
    float distance = Vector3.Distance(transform.position,
player.position);
    if (distance <= magnetRadius)
    {
        isCollected = true;
    }
}

private void Update()
{
    if (!isCollected) return;
    transform.position = Vector3.MoveTowards(
        transform.position,
        player.position,
        attractSpeed * Time.deltaTime
    );
}

private void OnTriggerEnter2D(Collider2D other)
{
    if (other.CompareTag("Player") && !isCollected)
    {

```

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.19.001 ПЗ

Лист

65

```

        CollectCrystal();
    }
}

private void CollectCrystal()
{
    isCollected = true;
    CurrencySystem.instance.AddCurrency(value,
transform.position);

    if (collectEffect != null)
    {
        Instantiate(collectEffect, transform.position,
Quaternion.identity);
    }

    // Возвращаем в пул вместо уничтожения
    pool.ReturnToPool(this);
}

public void ResetCrystal()
{
    isCollected = false;
    CancelInvoke(nameof(EnableMagnet));
}
}

```

Файл «CrystalPool.cs»

```

using UnityEngine;
using System.Collections.Generic;
public class CrystalPool : MonoBehaviour
{
    public GameObject crystalPrefab;
    public int poolSize = 20;
    private List<Crystal> crystalPool = new List<Crystal>();
}

```

```

private void Start()
{
    InitializePool();
}

private void InitializePool()
{
    for (int i = 0; i < poolSize; i++)
    {
        CreateCrystal();
    }
}

private Crystal CreateCrystal()
{
    GameObject crystalObj = Instantiate(crystalPrefab);
    crystalObj.SetActive(false);
    Crystal crystal = crystalObj.GetComponent<Crystal>();
    crystal.Initialize(this);
    crystalPool.Add(crystal);
    return crystal;
}

public Crystal GetCrystal(Vector3 position)
{
    foreach (Crystal crystal in crystalPool)
    {
        if (!crystal.gameObject.activeInHierarchy)
        {
            crystal.transform.position = position;
            crystal.gameObject.SetActive(true);
            crystal.ResetCrystal();
            return crystal;
        }
    }

    // Если все кристаллы заняты, создаем новый
    Crystal newCrystal = CreateCrystal();

```

```

        newCrystal.transform.position = position;
        newCrystal.gameObject.SetActive(true);
        return newCrystal;
    }

    public void ReturnToPool(Crystal crystal)
    {
        crystal.gameObject.SetActive(false);
    }
}

```

Файл «CurrencySystem.cs»

```

using System;
using UnityEngine;
using UnityEngine.UI;
using TMPro;
public class CurrencySystem : MonoBehaviour
{
    public static CurrencySystem instance;

    [Header("Настройки валюты")]
    public int currentCrystals;
    public event Action<int> OnCrystalsUpdated;

    [Header("Настройки уровня")]
    public int currentLevel = 1;
    public int[] levelThresholds = { 5, 10, 15, 20, 25, 30, 35, 40,
45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95, 100}; // Кристаллы для
каждого уровня
    public Slider levelProgressSlider; // Ссылка на UI Slider
    public TMP_Text levelText; // Текст уровня (например, "Ур. 3")

    [Header("Звуковые эффекты")]
    public AudioClip collectSound;
    public AudioClip levelUpSound;
    public float soundVolume = 0.5f;
}

```

```

private void Awake()
{
    if (instance == null)
    {
        instance = this;
        DontDestroyOnLoad(gameObject); // Сохраняем между
сценами
    }
    else
    {
        Destroy(gameObject);
    }
}

private void Start()
{
    UpdateLevelUI();
}

public void AddCurrency(int amount, Vector3 position)
{
    currentCrystals += amount;
    CheckLevelProgress();
    OnCrystalsUpdated?.Invoke(currentCrystals);

    if (collectSound != null)
    {
        AudioSource.PlayClipAtPoint(collectSound, position,
soundVolume);
    }
}

private void CheckLevelProgress()
{
    bool leveledUp = false;

    // Проверяем, достигли ли мы нового уровня
    while (currentLevel <= levelThresholds.Length &&

```

```

currentCrystals >= levelThresholds[currentLevel - 1])
    {
        currentLevel++;
        leveledUp = true;
    }
    if (leveledUp)
    {
        if (levelUpSound != null)
        {
            AudioSource.PlayClipAtPoint(levelUpSound,
Camera.main.transform.position, soundVolume);
        }
        // Вызываем панель апгрейдов
        if (LevelUpSystem.instance != null)
        {
            LevelUpSystem.instance.PlayerLevelUp();
        }
    }
    UpdateLevelUI();
}
private void UpdateLevelUI()
{
    if (levelProgressSlider != null)
    {
        if (currentLevel <= levelThresholds.Length)
        {
            int prevThreshold = currentLevel > 1 ?
levelThresholds[currentLevel - 2] : 0;
            int nextThreshold = levelThresholds[currentLevel -
1];

            levelProgressSlider.value = (float)(currentCrystals
- prevThreshold) / (nextThreshold - prevThreshold);

```

```

        }
        else
        {
            levelProgressSlider.value = 1f; // Максимальный
уровень
        }
    }

    if (levelText != null)
    {
        levelText.text = $"LVL. {currentLevel}";
    }
}

public bool SpendCrystals(int amount)
{
    if (currentCrystals >= amount)
    {
        currentCrystals -= amount;
        OnCrystalsUpdated?.Invoke(currentCrystals);
        UpdateLevelUI();
        return true;
    }
    return false;
}

public void ResetCurrency()
{
    currentCrystals = 0;
    currentLevel = 1;
    OnCrystalsUpdated?.Invoke(currentCrystals);
    UpdateLevelUI();
}
}

```

Файл «DamagePupUp.cs»

```

using TMPro;
using UnityEngine;

public class DamagePupUp : MonoBehaviour
{
    [Header("Настройки")]
    public float moveSpeed = 0.5f;
    public float lifeTime = 1f;
    public Vector3 randomOffset = new Vector3(0.3f, 0.3f, 0);

    [SerializeField] private TextMeshProUGUI textMesh; // Ссылка
    // через сериализуемое поле
    private float timer;
    private Vector3 startPosition;

    void Awake()
    {
        // Дублирующая проверка, если забыли назначить в инспекторе
        if (textMesh == null)
        {
            textMesh = GetComponentInChildren<TextMeshProUGUI>();
            if (textMesh == null) Debug.LogError("TextMeshProUGUI
не найден!", this);
        }
    }

    public void Setup(int damage, Vector3 position)
    {
        if (textMesh == null)
        {
            Debug.LogError("TextMeshProUGUI не назначен!", this);
            return;
        }
    }
}

```

```

startPosition = position + new Vector3(
    Random.Range(-randomOffset.x, randomOffset.x),
    Random.Range(-randomOffset.y, randomOffset.y),
    0
);
transform.position = startPosition;
textMesh.text = damage.ToString();
textMesh.color = Color.white;
timer = 0;
gameObject.SetActive(true);
// Безопасная проверка камеры
if (Camera.main != null)
{
    transform.rotation = Camera.main.transform.rotation;
}
}
void Update()
{
    timer += Time.deltaTime;
    transform.position = startPosition + new Vector3(0,
moveSpeed * timer, 0);
    if (timer > lifeTime * 0.7f)
    {
        float alpha = 1 - (timer - lifeTime * 0.7f) / (lifeTime
* 0.3f);
        textMesh.color = new Color(1, 1, 1, alpha);
    }
    if (timer >= lifeTime)
    {
        if (DamagePopupPool.InstanceExists())
        {
            DamagePopupPool.instance.ReturnToPool(this);
        }
    }
}

```

```

        else
        {
            gameObject.SetActive(false);
        }
    }
}

```

Файл «DamagePupUpPool.cs»

```

using UnityEngine;
using System.Collections.Generic;

public class DamagePopupPool : MonoBehaviour
{
    private static DamagePopupPool _instance;
    public static DamagePopupPool instance
    {
        get
        {
            if (_instance == null)
            {
                _instance =
Object.FindAnyObjectByType<DamagePopupPool>();
                if (_instance == null)
                {
                    Debug.LogError("DamagePopupPool не найден на сцене!");
                }
            }
            return _instance;
        }
    }

    public static bool InstanceExists() => _instance != null;
    [SerializeField] private GameObject popupPrefab;
    [SerializeField] private int poolSize = 20;

```

					ПГУ 1.090301.19.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		74

```

private List<DamagePopup> popupPool = new List<DamagePopup>();
private void Awake()
{
    if (_instance != null && _instance != this)
    {
        Destroy(gameObject);
        return;
    }
    _instance = this;
    if (popupPrefab == null)
    {
        Debug.LogError("Popup prefab не назначен!", this);
        return;
    }
    if (popupPrefab.GetComponent<DamagePopup>() == null)
    {
        Debug.LogError("Префаб должен содержать компонент
DamagePopup!", this);
        return;
    }
    InitializePool();
}

private void InitializePool()
{
    for (int i = 0; i < poolSize; i++)
    {
        CreatePopup();
    }
}

private DamagePopup CreatePopup()
{

```

```

        GameObject popupObj = Instantiate(popupPrefab, transform);
        popupObj.SetActive(false);
        DamagePopup popup = popupObj.GetComponent<DamagePopup>();

        if (popup == null)
        {
            Debug.LogError("Не удалось создать DamagePopup!",
this);

            Destroy(popupObj);
            return null;
        }

        popupPool.Add(popup);
        return popup;
    }

    public DamagePopup GetPopup()
    {
        if (popupPool.Count == 0)
        {
            Debug.LogWarning("Пул пуст, создаю новый поппап");
            return CreatePopup();
        }

        foreach (DamagePopup popup in popupPool)
        {
            if (popup != null &&
!popup.gameObject.activeInHierarchy)
            {
                return popup;
            }
        }

        // Если все заняты или null - создаем новый
        Debug.Log("Все поппапы заняты, создаю дополнительный");
        return CreatePopup();
    }

```

```

public void ReturnToPool (DamagePopup popup)
{
    if (popup == null) return;

    popup.gameObject.SetActive (false);
    popup.transform.SetParent (transform);
}
}

```

Файл «DayNightCicle.cs»

```

using UnityEngine;
using UnityEngine.Tilemaps;

public class DayNightCycle : MonoBehaviour
{
    public static DayNightCycle instance;

    [Header ("Настройки")]
    public float dayDuration = 60f;
    public float nightDuration = 45f;
    public Color dayColor = Color.white;
    public Color nightColor = new Color (0.2f, 0.2f, 0.4f);

    [Header ("Ссылки")]
    public Tilemap backgroundTilemap;
    public EnemySpawner enemySpawner;

    private float timer;
    private bool isNight = false;
}

```

```

        public bool IsNight => isNight;
        public float CurrentPhaseDuration => isNight ? nightDuration :
dayDuration;
        public float CurrentPhaseProgress => timer /
CurrentPhaseDuration;
        private void Awake()
        {
            if (instance == null)
            {
                instance = this;
            }
            else
            {
                Destroy(gameObject);
            }
        }
        private void Update()
        {
            timer += Time.deltaTime;

            if (timer >= CurrentPhaseDuration)
            {
                timer = 0;
                isNight = !isNight;
                enemySpawner?.UpdateTimeOfDay(isNight);
            }

            // Плавное изменение цвета тайлов
            backgroundTilemap.color = isNight
                ? Color.Lerp(nightColor, dayColor,
CurrentPhaseProgress)
                : Color.Lerp(dayColor, nightColor,
CurrentPhaseProgress);
        }
        public void InitializeComponents()

```

					ПГУ 1.090301.19.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		78

```

{
    if (backgroundTilemap == null)
    {
        backgroundTilemap =
Object.FindAnyObjectByType<Tilemap>();
        if (backgroundTilemap == null)
        {
            Debug.LogError("Background Tilemap not assigned and
not found in scene");
        }
    }

    if (enemySpawner == null)
    {
        enemySpawner =
Object.FindAnyObjectByType<EnemySpawner>();
    }
}

```

Файл «EnemySpawner.cs»

```

using UnityEngine;
using System.Collections.Generic;
using System.Collections;
public class EnemySpawner : MonoBehaviour
{
    public static EnemySpawner instance;

    [System.Serializable]
    public class EnemyGroup
    {
        public GameObject prefab;
        public bool spawnAtDay = true;
        public bool spawnAtNight = true;
        public bool spawnAtRain = false; // Новое поле для спавна
во время дождя
        [Range(0.1f, 10f)] public float spawnWeight = 1f;
    }
}

```

```

    }

    [Header("Spawn Settings")]
    public Transform minSpawn;
    public Transform maxSpawn;
    public int initialPoolSize = 10;

    [Header("Enemy Groups")]
    public List<EnemyGroup> enemyGroups;

    [Header("Boss Settings")]
    public GameObject bossPrefab;

    [Header("Time of Day Modifiers")]
    public float nightEnemyHealthMultiplier = 1.5f;
    public float nightEnemySpeedMultiplier = 1.2f;
    [Header("Rain Modifiers")]
    public float rainEnemyHealthMultiplier = 1.3f;
    public float rainEnemySpeedMultiplier = 0.9f;
    private Dictionary<GameObject, Queue<GameObject>> enemyPools =
new Dictionary<GameObject, Queue<GameObject>>();
    private List<GameObject> activeEnemies = new
List<GameObject>();
    private bool isNightTime;
    private bool isRaining;
    private void Awake()
    {
        if (instance != null && instance != this)
        {
            Destroy(gameObject);
            return;
        }
        instance = this;
        InitializeAllPools();
    }

```

```

private void InitializeAllPools()
{
    foreach (var group in enemyGroups)
    {
        if (!enemyPools.ContainsKey(group.prefab))
        {
            enemyPools[group.prefab] = new Queue<GameObject>();

            for (int i = 0; i < initialPoolSize; i++)
            {
                CreateEnemy(group.prefab);
            }
        }
    }
}

private GameObject CreateEnemy(GameObject prefab)
{
    var enemy = Instantiate(prefab, transform);
    enemy.SetActive(false);
    enemyPools[prefab].Enqueue(enemy);
    return enemy;
}

public void UpdateTimeOfDay(bool isNight)
{
    if (isNightTime != isNight)
    {
        isNightTime = isNight;
        Debug.Log($"Time changed to: {(isNight ? "Night" :
"Day")}");

        UpdateActiveEnemiesModifiers();
    }
}

public void UpdateRainStatus(bool raining)

```

```

{
    if (isRaining != raining)
    {
        isRaining = raining;
        Debug.Log($"Rain status changed to: {(raining ?
"Raining" : "Not Raining")}");
        UpdateActiveEnemiesModifiers();
    }
}

private void UpdateActiveEnemiesModifiers()
{
    foreach (var enemy in activeEnemies)
    {
        ApplyTimeOfDayModifiers(enemy);
    }
}

public void SpawnEnemies(int count)
{
    StartCoroutine(SpawnEnemiesWithDelay(count));
}

private IEnumerator SpawnEnemiesWithDelay(int count)
{
    for (int i = 0; i < count; i++)
    {
        SpawnSingleEnemy();
        yield return new WaitForSeconds(GetSpawnDelay());
    }
}

private void SpawnSingleEnemy()
{
    var availableGroups = GetAvailableEnemyGroups();
    if (availableGroups.Count == 0) return;
    var selectedGroup = GetRandomEnemyGroup(availableGroups);

```

```

        var prefab = selectedGroup.prefab;

        if (!enemyPools.ContainsKey(prefab)) return;

        GameObject enemy = enemyPools[prefab].Count > 0
            ? enemyPools[prefab].Dequeue()
            : CreateEnemy(prefab);

        SetupEnemy(enemy, prefab);
    }

    private void SetupEnemy(GameObject enemy, GameObject prefab)
    {
        enemy.transform.position = GetRandomSpawnPoint();
        enemy.SetActive(true);
        activeEnemies.Add(enemy);
        ApplyTimeOfDayModifiers(enemy);
    }

    public void SpawnBoss()
    {
        if (bossPrefab == null) return;
        var boss = Instantiate(bossPrefab, GetRandomSpawnPoint(),
Quaternion.identity);
        activeEnemies.Add(boss);
        ApplyTimeOfDayModifiers(boss);
    }

    private List<EnemyGroup> GetAvailableEnemyGroups()
    {
        return enemyGroups.FindAll(g =>
            (isNightTime && g.spawnAtNight) ||
            (isRaining && g.spawnAtRain) ||
            (!isNightTime && !isRaining && g.spawnAtDay));
    }

```

```

private EnemyGroup GetRandomEnemyGroup(List<EnemyGroup> groups)
{
    float totalWeight = 0;
    foreach (var group in groups)
    {
        totalWeight += group.spawnWeight;
    }
    float randomPoint = Random.Range(0, totalWeight);
    float currentWeight = 0;

    foreach (var group in groups)
    {
        currentWeight += group.spawnWeight;
        if (randomPoint <= currentWeight)
            return group;
    }
    return groups[0];
}

private void ApplyTimeOfDayModifiers(GameObject enemy)
{
    var health = enemy.GetComponent<EnemyHealth>();
    if (health != null)
    {
        health.ResetHealth();
        if (isNightTime)
        {
            health.maxHealth *= nightEnemyHealthMultiplier;
        }
        else if (isRaining)
        {
            health.maxHealth *= rainEnemyHealthMultiplier;
        }
        health.currentHealth = health.maxHealth;
    }
}

```

```

var movement = enemy.GetComponent<EnemyMovement>();
if (movement != null)
{
    if (isNightTime)
    {
        movement.speed *= nightEnemySpeedMultiplier;
    }
    else if (isRaining)
    {
        movement.speed *= rainEnemySpeedMultiplier;
    }
}
}

public void ReturnToPool(GameObject enemy)
{
    if (enemy == null) return;
    enemy.SetActive(false);
    activeEnemies.Remove(enemy);

    foreach (var pair in enemyPools)
    {
        if (enemy.name.StartsWith(pair.Key.name))
        {
            ResetEnemyModifiers(enemy);
            pair.Value.Enqueue(enemy);
            return;
        }
    }
    Destroy(enemy);
}

private void ResetEnemyModifiers(GameObject enemy)
{

```

Изм.	Лист	№ докум.	Подпись	Дата

```

        var health = enemy.GetComponent<EnemyHealth>();
        if (health != null)
        {
            health.ResetHealth();
        }
        var movement = enemy.GetComponent<EnemyMovement>();
        if (movement != null)
        {
            // movement.ResetSpeed();
        }
    }

    private Vector3 GetRandomSpawnPoint()
    {
        float x = Random.Range(minSpawn.position.x,
maxSpawn.position.x);
        float y = Random.Range(minSpawn.position.y,
maxSpawn.position.y);
        return new Vector3(x, y, 0);
    }

    public bool HasActiveEnemies()
    {
        return activeEnemies.Count > 0;
    }

    public int GetActiveEnemiesCount()
    {
        return activeEnemies.Count;
    }

    private float GetSpawnDelay()
    {
        return Random.Range(WaveManager.instance.minSpawnDelay,
WaveManager.instance.maxSpawnDelay);
    }
}

```

Файл «EnemyHealth.cs»

```
using UnityEngine;

public class EnemyHealth : MonoBehaviour
{
    public float maxHealth = 30f;
    [SerializeField] public float currentHealth;
    public GameObject deathEffect;
    public GameObject crystalPrefab; // Префаб кристалла
    public int minCrystals = 1; // Минимальное количество кристаллов
    public int maxCrystals = 3; // Максимальное количество
кристаллов
    public float crystalDropForce = 2f; // Сила разлета кристаллов
    private void Start()
    {
        currentHealth = maxHealth;
    }
    public void ResetHealth()
    {
        currentHealth = maxHealth;
    }
    public void TakeDamage(float damage)
    {
        currentHealth -= damage;
        // Показываем попап урона
        DamagePopup popup = DamagePopupPool.instance.GetPopup();
        popup.Setup((int)damage, transform.position);

        if (currentHealth <= 0)
        {
            Die();
        }
    }
    private void Die()
```

```

{
    // Уведомляем статистику о убийстве врага
    EnemyStats.Instance?.EnemyKilled();
    // Получаем CrystalPool
    CrystalPool crystalPool =
Object.FindAnyObjectByType<CrystalPool>();
    // Спавним кристаллы через пул
    int crystalsToDrop = Random.Range(1, 4);
    for (int i = 0; i < crystalsToDrop; i++)
    {
        crystalPool.GetCrystal(transform.position +
Random.insideUnitSphere * 0.5f);
    }
    // 30% шанс выпадения души (1-3 шарда)
    if (Random.value < 0.3f)
    {
        int shardsToDrop = Random.Range(1, 4);
        SoulShard shard = SoulShardPool.Instance.GetShard(
            transform.position + Random.insideUnitSphere *
0.5f,
            shardsToDrop
        );
    }
    EnemySpawner.instance.ReturnToPool(gameObject);
}
private void DropCrystals()
{
    if (crystalPrefab == null) return;

    int crystalsToDrop = Random.Range(minCrystals, maxCrystals
+ 1);

    for (int i = 0; i < crystalsToDrop; i++)

```

Изм.	Лист	№ докум.	Подпись	Дата

```

        {
            GameObject crystal = Instantiate(crystalPrefab,
transform.position, Quaternion.identity);

            // Добавляем случайный разлет
            Vector2 randomDirection = new Vector2(
                Random.Range(-1f, 1f),
                Random.Range(0.5f, 1f)
            ).normalized;

            Rigidbody2D rb = crystal.GetComponent<Rigidbody2D>();
            if (rb != null)
            {
                rb.AddForce(randomDirection * crystalDropForce,
ForceMode2D.Impulse);
            }
        }
    }
}

```

Файл «EnemyMovement.cs»

```

using UnityEngine;

public class EnemyMovement : MonoBehaviour
{
    private Transform player;
    private Rigidbody2D rb;
    private Vector2 movement;
    public float speed, damage;
    private float hitCounter = 0f;
    public float hitWaitTime = 1f; // Интервал между ударами (в
секундах)
    private PlayerHP playerHP;
}

```

Изм.	Лист	№ докум.	Подпись	Дата

```

void Start()
{
    rb = GetComponent<Rigidbody2D>();
    speed = Random.Range(speed * 0.8f, speed * 1.2f);
    player =
Object.FindAnyObjectByType<PlayerMovement>().transform;
}
void Update()
{
    if (player != null)
    {
        Vector3 direction = player.position -
transform.position;
        direction.Normalize();
        movement = direction;
    }
}
private void FixedUpdate()
{
    if (player != null)
    {
        MoveChar(movement);

        // Наносим урон каждые hitWaitTime секунд, пока враг
касается игрока
        if (hitCounter <= 0f && playerHP != null)
        {
            playerHP.TakeDamage(damage);
            hitCounter = hitWaitTime;
        }
        else
        {
            hitCounter -= Time.deltaTime;
        }
    }
}

```

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.19.001 ПЗ

Лист

90

```

        private void MoveChar(Vector2 direction)
        {
            rb.MovePosition((Vector2)transform.position + (direction *
speed * Time.deltaTime));
        }

        private void OnCollisionEnter2D(Collision2D collision)
        {
            if (collision.gameObject.TryGetComponent<PlayerHP>(out
playerHP))
            {
                // Наносим урон сразу при первом столкновении
                if (hitCounter <= 0f)
                {
                    playerHP.TakeDamage(damage);
                    hitCounter = hitWaitTime;
                }
            }
        }

        private void OnCollisionExit2D(Collision2D collision)
        {
            if (collision.gameObject.GetComponent<PlayerHP>() != null)
            {
                playerHP = null;
            }
        }

```

Файл «EnemyStats.cs»

```

using UnityEngine;

public class EnemyStats : MonoBehaviour
{
    public static EnemyStats Instance { get; private set; }
}

```

```

private int enemiesKilled = 0;

private void Awake()
{
    if (Instance == null)
    {
        Instance = this;
        DontDestroyOnLoad(gameObject);
    }
    else
    {
        Destroy(gameObject);
    }
}

public void EnemyKilled()
{
    enemiesKilled++;
}

public int GetEnemiesKilled()
{
    return enemiesKilled;
}

public void ResetStats()
{
    enemiesKilled = 0;
}
}

```

Файл «GameManager.cs»

```

using UnityEngine;
using System.Collections.Generic;
using TMPro;
using UnityEngine.SceneManagement;
using System;

```

```

public class GameManager : MonoBehaviour
{
    [Header("Player Settings")]
    [SerializeField] private Vector3 spawnPosition = Vector3.zero;
    [SerializeField] private List<GameObject> characterPrefabs =
new List<GameObject>();

    [Header("UI Settings")]
    [SerializeField] private Canvas gameOverCanvas;
    [SerializeField] private Canvas pauseCanvas;
    [SerializeField] private Canvas settingsCanvas;
    [SerializeField] private TMP_Text survivalTimeText;
    [SerializeField] private TMP_Text bestTimeText;
    [SerializeField] private TMP_Text pauseTimeText;
    [SerializeField] private TMP_Text enemiesKilledText;

    public static GameManager instance;
    public static event Action<GameObject> OnPlayerSpawned;
    public static event Action OnPlayerDeath;
    private GameObject currentPlayer;
    private bool isPaused = false;
    private void Awake()
    {
        if (instance == null)
        {
            instance = this;
        }
        else
        {
            Destroy(gameObject);
            return;
        }
        InitializeUI();
        Time.timeScale = 1f;
    }
}

```

```

    }

    private void InitializeUI()
    {
        if (gameOverCanvas != null) gameOverCanvas.enabled = false;
        if (pauseCanvas != null)
pauseCanvas.gameObject.SetActive(false);
        if (settingsCanvas != null)
settingsCanvas.gameObject.SetActive(false);
    }

    private void Start()
    {
        StartNewSession();
        SpawnSelectedCharacter();

        if (PlayerHP.instance != null)
        {
            PlayerHP.instance.OnPlayerDeath += HandlePlayerDeath;
        }
        UpdateCursorState();
    }

    private void Update()
    {
        if (Input.GetKeyDown(KeyCode.Escape))
        {
            if (settingsCanvas != null &&
settingsCanvas.gameObject.activeSelf)
            {
                CloseSettings();
            }
            else if (!IsAnyUIOpen())
            {
                TogglePause();
            }
        }
    }

```

```

    }

    }

    public bool IsAnyUIOpen()
    {
        return (gameOverCanvas != null && gameOverCanvas.enabled)
||
        (LevelUpSystem.instance != null &&
LevelUpSystem.instance.IsLevelUpInProgress());
    }

    public void TogglePause()
    {
        if (IsAnyUIOpen()) return;

        isPaused = !isPaused;

        if (pauseCanvas != null)
        {
            pauseCanvas.gameObject.SetActive(isPaused);
            if (isPaused && SessionTimer.Instance != null &&
pauseTimeText != null)
            {
                pauseTimeText.text = "Игровое время: " +
FormatTime(SessionTimer.Instance.GetCurrentTime());
            }
        }

        Time.timeScale = isPaused ? 0f : 1f;
        UpdateCursorState();
    }

    public void OpenSettings()
    {
        if (pauseCanvas != null)
pauseCanvas.gameObject.SetActive(false);
        if (settingsCanvas != null)
settingsCanvas.gameObject.SetActive(true);
        UpdateCursorState();
    }
}

```

Изм.	Лист	№ докум.	Подпись	Дата

```

public void UpdateCursorState()
{
    bool shouldShowCursor = isPaused ||
        (gameOverCanvas != null &&
gameOverCanvas.enabled) ||
        (LevelUpSystem.instance != null &&
LevelUpSystem.instance.IsLevelUpInProgress()) ||
        (settingsCanvas != null &&
settingsCanvas.gameObject.activeSelf);

    Cursor.visible = shouldShowCursor;
    Cursor.lockState = shouldShowCursor ? CursorLockMode.None :
CursorLockMode.Locked;
}

public void CloseSettings()
{
    if (settingsCanvas != null)
settingsCanvas.gameObject.SetActive(false);
    if (pauseCanvas != null)
pauseCanvas.gameObject.SetActive(true);
    UpdateCursorState();
}

public void ResumeGame()
{
    isPaused = false;
    if (pauseCanvas != null)
pauseCanvas.gameObject.SetActive(false);
    Time.timeScale = 1f;
    UpdateCursorState();
}

private void HandlePlayerDeath()
{
    EndSession();
    ShowGameOverPanel();
}

```

```

private void ShowGameOverPanel()
{
    if (gameOverCanvas != null)
    {
        float currentTime =
SessionTimer.Instance.GetCurrentTime();
        float bestTime =
PlayerPrefs.GetFloat("BestSessionTime", 0f);
        int kills = EnemyStats.Instance?.GetEnemiesKilled() ??
0;

        survivalTimeText.text = $"Время выживания:
{FormatTime(currentTime)}";
        bestTimeText.text = $"Лучшее время:
{FormatTime(bestTime)}";
        enemiesKilledText.text = $"Врагов убито: {kills}";

        // Сохраняем рекорд убийств
        int bestKills = PlayerPrefs.GetInt("BestEnemiesKilled",
0);

        if (kills > bestKills)
        {
            PlayerPrefs.SetInt("BestEnemiesKilled", kills);
        }

        gameOverCanvas.enabled = true;
        Time.timeScale = 0f;
        UpdateCursorState();
    }
}

private void StartNewSession()
{
    if (SessionTimer.Instance != null)
    {
        SessionTimer.Instance.ResetTimer();
        SessionTimer.Instance.StartTimer();
    }
}

```

```

        // Сбрасываем статистику убийств при начале новой сессии
        EnemyStats.Instance?.ResetStats();
    }

    public void ReturnToMainMenu()
    {
        if (currentPlayer != null) Destroy(currentPlayer);
        Time.timeScale = 1f;
        SceneManager.LoadScene("MainMenu");
    }

    private string FormatTime(float timeInSeconds)
    {
        int minutes = (int)(timeInSeconds / 60);
        int seconds = (int)(timeInSeconds % 60);
        return $"{minutes:00}:{seconds:00}";
    }

    private void EndSession()
    {
        if (SessionTimer.Instance != null)
        {
            SessionTimer.Instance.StopTimer();

            SaveSessionStats(SessionTimer.Instance.GetCurrentTime());
        }
    }

    private void SaveSessionStats(float sessionTime)
    {
        PlayerPrefs.SetFloat("LastSessionTime", sessionTime);
        float bestTime = PlayerPrefs.GetFloat("BestSessionTime",
0f);

```

```

        if (sessionTime > bestTime)
PlayerPrefs.SetFloat("BestSessionTime", sessionTime);

        PlayerPrefs.Save();
    }

    private void SpawnSelectedCharacter()
    {
        if (currentPlayer != null) Destroy(currentPlayer);

        string prefabName =
PlayerPrefs.GetString("SelectedCharacterPrefab");

        GameObject characterPrefab = characterPrefabs.Find(p =>
p.name == prefabName) ??

                                (characterPrefabs.Count > 0 ?
characterPrefabs[0] : null);

        if (characterPrefab != null)
        {
            currentPlayer = Instantiate(characterPrefab,
spawnPosition, Quaternion.identity);

            currentPlayer.name = characterPrefab.name;
            currentPlayer.tag = "Player";
            OnPlayerSpawned?.Invoke(currentPlayer);
        }
    }
}

```

Файл «Language.cs»

```

// LocalizationManager.cs
using TMPro;
using UnityEngine;
using UnityEngine.UI;

public class LocalizationManager : MonoBehaviour
{

    [SerializeField] private TMP_Dropdown
        languageDropdown

```

```

        public void SetLanguage(int languageIndex)
        {
            string language =
languageDropdown.options[languageIndex].text;
            PlayerPrefs.SetString("Language", language);
        }
    }
}

```

Файл «LvlUpSystem.cs»

```

using UnityEngine;
using System.Collections.Generic;
using TMPro;
using UnityEngine.UI;
using static PlayerWeapons;

public class LevelUpSystem : MonoBehaviour
{
    [Header("UI Settings")]
    public GameObject levelUpPanel;
    public TMP_Text levelText;

    [Header("Upgrade Settings")]
    public List<Upgrade> availableUpgrades;
    public List<UpgradeButton> upgradeButtons;

    [Header("Selected Upgrades Display")]
    public Transform upgradesDisplayParent;
    public GameObject upgradeDisplayPrefab;
    public Vector2 firstUpgradePosition = new Vector2(20, -20);
    public float verticalSpacing = 30f;
    public int maxDisplayedUpgrades = 10;
    public static LevelUpSystem instance;
    private int currentLevel = 1;
    private bool isLevelUpInProgress = false;
    private List<GameObject> activeUpgradeDisplays = new
List<GameObject>();

```

Изм.	Лист	№ докум.	Подпись	Дата

```

private void Awake()
{
    if (instance == null)
    {
        instance = this;
        if (levelUpPanel != null)
levelUpPanel.SetActive(false);
    }
    else
    {
        Destroy(gameObject);
    }
}

public bool IsLevelUpInProgress() => isLevelUpInProgress;
public void PlayerLevelUp()
{
    if (isLevelUpInProgress) return;

    currentLevel++;
    levelText.text = $"Level: {currentLevel}";
    ShowUpgradeOptions();
}

private void ShowUpgradeOptions()
{
    isLevelUpInProgress = true;
    Time.timeScale = 0f;
    levelUpPanel.SetActive(true);
    SetupUpgradeButtons();
    GameManager.instance?.UpdateCursorState();
}

private void SetupUpgradeButtons()
{
    foreach (var button in upgradeButtons)
    {

```

```

        button.gameObject.SetActive(false);
        button.ClearButton();
    }
    List<Upgrade> shuffledUpgrades = new
List<Upgrade>(availableUpgrades);
    ShuffleUpgrades(shuffledUpgrades);

    HashSet<string> usedNames = new HashSet<string>();
    int buttonsSetup = 0;

    foreach (var upgrade in shuffledUpgrades)
    {
        if (buttonsSetup >= upgradeButtons.Count) break;

        if (!usedNames.Contains(upgrade.name))
        {
            upgradeButtons[buttonsSetup].Setup(
                upgrade.name,
                upgrade.description,
                upgrade.icon
            );
            upgradeButtons[buttonsSetup].SetUpgradeData(upgrade);

upgradeButtons[buttonsSetup].gameObject.SetActive(true);
            usedNames.Add(upgrade.name);
            buttonsSetup++;
        }
    }
}

private void ShuffleUpgrades(List<Upgrade> upgrades)
{
    for (int i = 0; i < upgrades.Count; i++)
    {
        Upgrade temp = upgrades[i];
        int randomIndex = Random.Range(i, upgrades.Count);

```

```

        upgrades[i] = upgrades[randomIndex];
        upgrades[randomIndex] = temp;
    }
}

public void ApplyUpgrade(Upgrade upgrade)
{
    if (upgrade == null || !isLevelUpInProgress) return;

    ApplyUpgradeEffect(upgrade);
    AddUpgradeToDisplay(upgrade);
    if (SynergySystem.instance != null)
SynergySystem.instance.CheckForSynergies(availableUpgrades);
    CloseLevelUpPanel();
}

private void AddUpgradeToDisplay(Upgrade upgrade)
{
    if (upgradesDisplayParent == null || upgradeDisplayPrefab
== null) return;

    // Если достигнут лимит, удаляем самое старое улучшение
    if (activeUpgradeDisplays.Count >= maxDisplayedUpgrades &&
activeUpgradeDisplays.Count > 0)
    {
        GameObject oldestUpgrade = activeUpgradeDisplays[0];
        activeUpgradeDisplays.RemoveAt(0);
        Destroy(oldestUpgrade);
    }

    // Создаем новый элемент отображения
    GameObject newUpgradeDisplay =
Instantiate(upgradeDisplayPrefab, upgradesDisplayParent);

    // Настраиваем текст
    TMP_Text upgradeText =
newUpgradeDisplay.GetComponentInChildren<TMP_Text>();
    if (upgradeText != null)

```

```

        {
            upgradeText.text = upgrade.name;
        }

        // Настраиваем иконку, если есть
        Image upgradeIcon =
newUpgradeDisplay.GetComponentInChildren<Image>();
        if (upgradeIcon != null && upgrade.icon != null)
        {
            upgradeIcon.sprite = upgrade.icon;
            upgradeIcon.preserveAspect = true;
        }

        // Позиционируем
        RectTransform rectTransform =
newUpgradeDisplay.GetComponent<RectTransform>();
        if (rectTransform != null)
        {
            rectTransform.anchoredPosition = new Vector2(
                firstUpgradePosition.x,
                firstUpgradePosition.y -
(activeUpgradeDisplays.Count * verticalSpacing)
            );
        }
        activeUpgradeDisplays.Add(newUpgradeDisplay);
    }

    private void ApplyUpgradeEffect(Upgrade upgrade)
    {
        switch (upgrade.type)
        {
            case UpgradeType.Weapon:

PlayerWeapons.Instance?.AddWeapon(upgrade.weaponPrefab);
                break;
            case UpgradeType.Passive:

```

```

        PlayerStats.instance?.ApplyPassive(upgrade.statBonus);

        break;

        case UpgradeType.WeaponUpgrade:

PlayerWeapons.Instance?.UpgradeWeapon(upgrade.weaponType,
upgrade.statBonus);

        break;

    }

}

private void CloseLevelUpPanel()
{
    isLevelUpInProgress = false;
    Time.timeScale = 1f;
    levelUpPanel.SetActive(false);
    GameManager.instance?.UpdateCursorState();
}

public void ResetLevelSystem()
{
    currentLevel = 1;
    levelText.text = $"Level: {currentLevel}";
    if (levelUpPanel != null) levelUpPanel.SetActive(false);
    isLevelUpInProgress = false;
    Time.timeScale = 1f;
    // Очищаем отображение улучшений
    foreach (var display in activeUpgradeDisplays)
    {
        if (display != null) Destroy(display);
    }
    activeUpgradeDisplays.Clear();
    GameManager.instance?.UpdateCursorState();
}

}

[System.Serializable]

```

```

public class Upgrade
{
    public string name;
    public string description;
    public UpgradeType type;
    public WeaponType weaponType;
    public GameObject weaponPrefab;
    public StatBonus statBonus;
    public Sprite icon;
    public SynergyType[] possibleSynergies;
}

public enum UpgradeType { Weapon, Passive, WeaponUpgrade }

[System.Serializable]
public class StatBonus
{
    public float damageBonus;
    public float speedBonus;
    public float healthBonus;
    public float attackSpeedBonus;
    public float critChanceBonus;
    public float critMultiplierBonus;
}

```

Файл «MagicProjectilePool.cs»

```

using UnityEngine;
using System.Collections.Generic;

public class MagicProjectilePool : MonoBehaviour
{
    public static MagicProjectilePool Instance;
    public GameObject magicProjectilePrefab;
    public int poolSize = 10;
}

```

```

        private Queue<GameObject> magicProjectilePool = new
Queue<GameObject>();

        private void Awake()
        {
            Instance = this;
            InitializePool();
        }

        private void InitializePool()
        {
            for (int i = 0; i < poolSize; i++)
            {
                GameObject projectile =
Instantiate(magicProjectilePrefab);
                projectile.SetActive(false);
                magicProjectilePool.Enqueue(projectile);
            }
        }

        public GameObject GetMagicProjectile()
        {
            if (magicProjectilePool.Count == 0)
            {
                ExpandPool();
            }

            GameObject projectile = magicProjectilePool.Dequeue();
            projectile.SetActive(true);
            return projectile;
        }

        public void ReturnMagicProjectile(GameObject projectile)
        {
            projectile.SetActive(false);
            magicProjectilePool.Enqueue(projectile);
        }

```

```

private void ExpandPool()
{
    for (int i = 0; i < poolSize / 2; i++)
    {
        GameObject projectile =
Instantiate(magicProjectilePrefab);
        projectile.SetActive(false);
        magicProjectilePool.Enqueue(projectile);
    }
}
}

```

Файл «MagicProjectile.cs»

```

using UnityEngine;
public class MagicProjectile : MonoBehaviour
{
    public float damage = 15f;
    public float speed = 8f;
    public GameObject hitEffect;
    public float splashRadius = 1.5f;
    private Transform target;
    private Rigidbody2D rb;
    private bool isCritical;
    private bool isActive;

    private void Awake()
    {
        rb = GetComponent<Rigidbody2D>();
    }

    public void Initialize(Transform target, float damage, bool
isCritical, float splashRadius)
    {
        this.target = target;
        this.damage = damage;
        this.isCritical = isCritical;
    }
}

```

```

        this.splashRadius = splashRadius;
        isActive = true;
    }

    private void Update()
    {
        if (!isActive) return;

        if (target != null)
        {
            Vector2 direction = (target.position -
transform.position).normalized;
            rb.linearVelocity = direction * speed;

            float angle = Mathf.Atan2(direction.y, direction.x) *
Mathf.Rad2Deg;
            transform.rotation = Quaternion.AngleAxis(angle,
Vector3.forward);
        }
        else
        {
            ReturnToPool();
        }
    }

    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (!isActive) return;

        if (collision.TryGetComponent(out EnemyHealth enemy))
        {
            // Наносим урон основному target
            enemy.TakeDamage(damage);

            // Наносим урон всем врагам в радиусе
            Collider2D[] splashTargets =
Physics2D.OverlapCircleAll(transform.position, splashRadius);
            foreach (Collider2D splashTarget in splashTargets)

```

```

        {
            if (splashTarget.TryGetComponent(out EnemyHealth
splashEnemy) && splashEnemy != enemy)
            {
                splashEnemy.TakeDamage(damage * 0.5f); //
Половина урона по splash
            }
        }
        SpawnHitEffect();
        ReturnToPool();
    }
    else if (collision.gameObject.layer ==
LayerMask.NameToLayer("Obstacles"))
    {
        SpawnHitEffect();
        ReturnToPool();
    }
}

private void SpawnHitEffect()
{
    if (hitEffect != null)
    {
        GameObject effect = Instantiate(hitEffect,
transform.position, Quaternion.identity);
        if (isCritical &&
effect.TryGetComponent<ParticleSystem>(out var ps))
        {
            var main = ps.main;
            main.startColor = Color.cyan; // Изменяем цвет для
критического попадания
        }
    }
}

private void ReturnToPool()
{

```

```

        isActive = false;
        rb.linearVelocity = Vector2.zero;

MagicProjectilePool.Instance.ReturnMagicProjectile(gameObject);
    }
}

```

Файл «MenuManager.cs»

```

using UnityEngine;

public class MenuManager : MonoBehaviour
{
    [Header("Canvases")]
    [SerializeField] private Canvas mainMenuCanvas;
    [SerializeField] private Canvas characterSelectionCanvas;
    [SerializeField] private Canvas settingsCanvas;
    [SerializeField] private Canvas relicShopCanvas; // Добавлен
новый Canvas для магазина реликвий

    private void Awake()
    {
        // Инициализация всех канвасов
        InitializeCanvases();
    }

    private void InitializeCanvases()
    {
        // Активируем только главное меню
        mainMenuCanvas.enabled = true;

        // Деактивируем все остальные Canvas
        characterSelectionCanvas.enabled = false;
        settingsCanvas.enabled = false;
        relicShopCanvas.enabled = false; // Инициализация нового
Canvas
    }
}

```

```

public void ShowSettings()
{
    SetCanvasActive(mainMenuCanvas, false);
    SetCanvasActive(characterSelectionCanvas, false);
    SetCanvasActive(settingsCanvas, true);
    SetCanvasActive(relicShopCanvas, false);
}

public void ShowCharacterSelection()
{
    SetCanvasActive(mainMenuCanvas, false);
    SetCanvasActive(characterSelectionCanvas, true);
    SetCanvasActive(settingsCanvas, false);
    SetCanvasActive(relicShopCanvas, false);
}

// Новый метод для показа магазина реликвий
public void ShowRelicShop()
{
    SetCanvasActive(mainMenuCanvas, false);
    SetCanvasActive(characterSelectionCanvas, false);
    SetCanvasActive(settingsCanvas, false);
    SetCanvasActive(relicShopCanvas, true);
}

public void BackToMainMenu()
{
    SetCanvasActive(settingsCanvas, false);
    SetCanvasActive(characterSelectionCanvas, false);
    SetCanvasActive(relicShopCanvas, false);
    SetCanvasActive(mainMenuCanvas, true);
}

private void SetCanvasActive(Canvas canvas, bool state)
{
    canvas.enabled = state;
    var canvasGroup = canvas.GetComponent<CanvasGroup>();

```

```

        if (canvasGroup != null)
        {
            canvasGroup.blocksRaycasts = state;
            canvasGroup.interactable = state;
        }
    }
}

```

Файл «PlayerHP.cs»

```

using UnityEngine;
using UnityEngine.UI;
using System;
using UnityEngine.Events;
[System.Serializable]
public class PlayerHP : MonoBehaviour
{
    [SerializeField] private BloodEffect bloodEffect;
    [SerializeField] private Animator animator;
    public static PlayerHP instance;
    public float currentHealth, maxHealth;
    public Slider healthSlider;
    public event Action OnPlayerDeath;
    private void Awake()
    {
        instance = this;
        healthSlider.maxValue = maxHealth;
        healthSlider.value = currentHealth;

        // Автоматическое получение аниматора, если не назначен в
инспекторе
        if (animator == null)
        {
            animator = GetComponent<Animator>();
            if (animator == null)

```

```

        {
            Debug.LogWarning("Animator component not found on
player!");
        }
    }

    void Start()
    {
        currentHealth = maxHealth;
    }

    public void TakeDamage(float damage)
    {
        currentHealth = Mathf.Max(currentHealth - damage, 0);
        healthSlider.value = currentHealth;

        if (bloodEffect != null)
        {
            bloodEffect.SpawnBlood(transform.position,
(int) damage);
        }
        if (currentHealth <= 0)
        {
            Die(); // Вызываем метод смерти
        }
    }

    private void Die()
    {
        // Активируем триггер смерти
        if (animator != null)
        {
            animator.SetTrigger("Die");
        }
        else
        {

```

					ПГУ 1.090301.19.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		114

```

        Debug.LogWarning("Cannot play death animation -
Animator not assigned!");
    }
    OnPlayerDeath?.Invoke();
    // Отключаем объект через 1 секунду (время на проигрывание
анимации)
    Invoke("DisablePlayer", 3f);

}

private void DisablePlayer()
{
    gameObject.SetActive(false);
}

private void OnDestroy()
{
    OnPlayerDeath = null;
}
}

```

Файл «PlayerMovement.cs»

```

using UnityEngine;
using UnityEngine.EventSystems;

[RequireComponent(typeof(Rigidbody2D), typeof(Animator),
typeof(SpriteRenderer))]
public class PlayerMovement : MonoBehaviour
{
    public static PlayerMovement instance;
    public float moveSpeed = 5f;
    private Rigidbody2D rb;
    private Vector2 movement;
    private Animator animator;
    private SpriteRenderer spriteRenderer;
    // Мобильное управление
    [Header("Mobile Controls")]
}

```

Изм.	Лист	№ докум.	Подпись	Дата

```

public bool useMobileControls = false;

[SerializeField] private GameObject joystickGameObject;
[SerializeField] private DynamicJoystick dynamicJoystick;

// Для касаний
private bool isDragging = false;
private Vector2 touchStartPos;

private void Awake()
{
    instance = this;
    rb = GetComponent<Rigidbody2D>();
    animator = GetComponent<Animator>();
    spriteRenderer = GetComponent<SpriteRenderer>();

#if UNITY_ANDROID || UNITY_IOS
    useMobileControls = true;
#endif

    // Автоматический поиск джойстика
    FindJoystick();

    if (joystickGameObject != null)
    {
        joystickGameObject.SetActive(useMobileControls);
    }
}

private void FindJoystick()
{
    joystickGameObject =
GameObject.FindGameObjectWithTag("Joystick");

    if (joystickGameObject == null)
    {

```

```

        dynamicJoystick = FindObjectOfType<DynamicJoystick>();
        if (dynamicJoystick != null)
        {
            joystickGameObject = dynamicJoystick.gameObject;
        }
    }
    else
    {
        dynamicJoystick =
joystickGameObject.GetComponent<DynamicJoystick>();
    }

    if (dynamicJoystick == null && useMobileControls)
    {
        Debug.LogWarning("DynamicJoystick not found on the
scene! Mobile controls will not work properly.");
    }
}

private void Update()
{
    if (!useMobileControls)
    {
        // Клавиатурное управление
        movement.x = Input.GetAxisRaw("Horizontal");
        movement.y = Input.GetAxisRaw("Vertical");
    }
    else
    {
        // Мобильное управление
        if (dynamicJoystick != null)
        {
            movement.x = dynamicJoystick.Horizontal;
            movement.y = dynamicJoystick.Vertical;
        }
        else

```

```

        {
            MobileTouchControl();
        }
    }

    UpdateAnimations();
}

private void MobileTouchControl()
{
    if (Input.touchCount > 0)
    {
        Touch touch = Input.GetTouch(0);

        // Проверяем, что касание не на UI элементе
        if
(EventSystem.current.IsPointerOverGameObject(touch.fingerId))
            return;
        switch (touch.phase)
        {
            case TouchPhase.Began:
                touchStartPos = touch.position;
                isDragging = true;
                break;
            case TouchPhase.Moved:
            case TouchPhase.Stationary:
                Vector2 touchDelta = touch.position -
touchStartPos;

                movement = touchDelta.normalized;
                break;
            case TouchPhase.Ended:
            case TouchPhase.Canceled:
                movement = Vector2.zero;
                isDragging = false;
                break;
        }
    }
}

```

```

    }

    }

    else
    {
        movement = Vector2.zero;
    }
}

private void FixedUpdate()
{
    rb.MovePosition(rb.position + movement * moveSpeed *
Time.fixedDeltaTime);
}

private void UpdateAnimations()
{
    bool isMoving = movement.magnitude > 0.1f;
    animator.SetBool("IsRunning", isMoving);

    if (movement.x > 0.1f) spriteRenderer.flipX = false;
    else if (movement.x < -0.1f) spriteRenderer.flipX = true;
}
}

```

Файл «PlayerPrefabInfo.cs»

```

using UnityEngine;

public class PlayerPrefabInfo : MonoBehaviour
{
    public string characterName = "Персонаж";
    public Sprite characterIcon; // Можно использовать для UI
}

```

Файл «PlayerStats.cs»

```

// PlayerStats.cs
using UnityEngine;
using System.Collections.Generic;

```

```

public class PlayerStats : MonoBehaviour
{
    public static PlayerStats instance;
    // Базовые характеристики
    public float baseDamage = 10f;
    public float baseSpeed = 5f;
    public float baseMaxHealth = 100f;
    // Текущие характеристики (база + бонусы)
    public float currentDamage { get; private set; }
    public float currentSpeed { get; private set; }
    public float currentMaxHealth { get; private set; }

    // Список активных бонусов от реликвий
    private List<RelicBonus> activeRelicBonuses = new
List<RelicBonus>();

    private void Awake()
    {
        instance = this;
        ResetToBaseStats();
    }

    public void ResetToBaseStats()
    {
        currentDamage = baseDamage;
        currentSpeed = baseSpeed;
        currentMaxHealth = baseMaxHealth;
        UpdateHealthUI();
    }

    public void ApplyPassive(StatBonus bonus)
    {
        currentDamage += bonus.damageBonus;
        currentSpeed += bonus.speedBonus;
        currentMaxHealth += bonus.healthBonus;
        UpdateHealthUI();
    }
}

```

Изм.	Лист	№ докум.	Подпись	Дата

```

    }

    public void ApplyRelicBonuses(List<Relic> relics)
    {
        ResetToBaseStats();
        activeRelicBonuses.Clear();
        // Собираем все бонусы от реликвий
        foreach (var relic in relics)
        {
            if (relic.currentLevel > 0)
            {
                var bonus = new RelicBonus(
                    relic.damagePerLevel * relic.currentLevel,
                    relic.speedPerLevel * relic.currentLevel,
                    relic.healthPerLevel * relic.currentLevel
                );
                activeRelicBonuses.Add(bonus);
            }
        }
        // Применяем все бонусы
        foreach (var bonus in activeRelicBonuses)
        {
            currentDamage += bonus.damageBonus;
            currentSpeed += bonus.speedBonus;
            currentMaxHealth += bonus.healthBonus;
        }
        UpdateHealthUI();
    }

    private void UpdateHealthUI()
    {
        if (PlayerHP.instance != null)
        {
            PlayerHP.instance.maxHealth = currentMaxHealth;
            PlayerHP.instance.currentHealth = currentMaxHealth;
        }
    }
}

```

Приложение В

(Обязательное)

Игровое приложение на Unity в жанре RogueLike

Презентация

					ПГУ 1.090301.19.001 ПЗ	Лист
						122
Изм.	Лист	№ докум.	Подпись	Дата		

ПРИЛОЖЕНИЕ НА UNITY В ЖАНРЕ ROGUELIKE

Автор работы: студент группы 21BBB1

Сурков А.А.

Научный руководитель: д.т.н., профессор кафедры ВТ
Зинкин С.А.

ЦЕЛЬ РАБОТЫ: РАЗРАБОТАТЬ ИГРУ В ЖАНРЕ ROGUELIKE С PIXELART ГРАФИКОЙ. КОТОРАЯ ОБЕСПЕЧИТ ВЫСОКУЮ РЕИГРАБЕЛЬНОСТЬ И КРОССПЛАТФОРМЕННОСТЬ.

Объектами разработки являются:

- 1) игровое приложение с реализацией основных механик, характерных для данного жанра;
- 2) система сохранения настроек и актуального прогресса;
- 3) визуальная составляющая (пользовательский интерфейс).

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.19.001 ПЗ

Лист

123

В ПРОЦЕССЕ РАЗРАБОТКИ ДОЛЖНЫ БЫТЬ РЕШЕНЫ СЛЕДУЮЩИЕ ЗАДАЧИ:

- 1) изучить современные тенденции в жанре RogueLike/Survivors-Like и проанализировать существующие аналоги;
- 2) определить ключевые проблемы игр;
- 3) реализовать оптимизированные системы, такие как Object Pooling, для стабильного FPS даже при большом количестве врагов на экране;
- 4) создать интуитивно понятный интерфейс с адаптацией под различные платформы (ПК и мобильные устройства);
- 5) провести тестирование игры на разных устройствах, оценить баланс сложности и оптимизацию;
- 6) определить перспективы дальнейшего развития проекта, включая добавление нового контента, интеграцию облачных сохранений и расширение платформенной поддержки.

АНАЛОГИ:

- «Vampire Survivors»
 - «Brotato»
- «Soulstone Survivors»
- «20 Minutes Till Dawn»

Изм.	Лист	№ докум.	Подпись	Дата

«VAMPIRE SURVIVORS»



«BROTATO»



Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.19.001 ПЗ

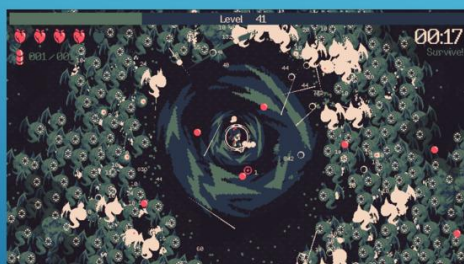
Лист

125

«SOULSTONE SURVIVORS»



«20 MINUTES TILL DAWN»



Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.19.001 ПЗ

Лист

126

ВЫБОР ДВИЖКА И СИСТЕМЫ ХРАНЕНИЯ ДАННЫХ



Язык
программирования



Движок игры



Система хранения
данных

ПРИ РАЗРАБОТКЕ ГРАФИЧЕСКОГО ИНТЕРФЕЙСА
БЫЛИ ИСПОЛЬЗОВАНЫ СЛЕДУЮЩИЕ
ИНСТРУМЕНТЫ:

- UGUI
- Для внешнего вида текстов был выбран TextMeshPro
- PIXELorama
- Figma

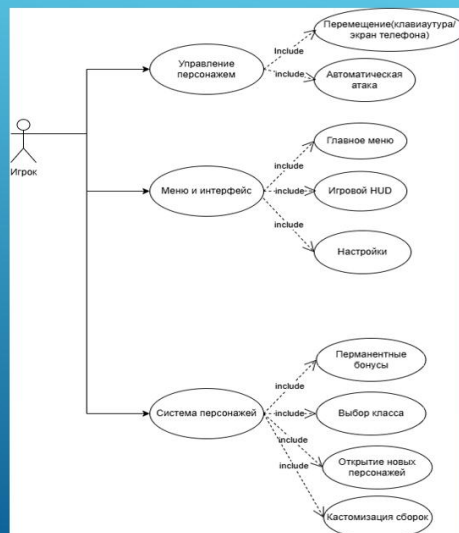
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.19.001 ПЗ

Лист

127

ДИАГРАММА ВАРИАНТОВ ИСПОЛЬЗОВАНИЯ



ОСНОВНЫЕ МОДУЛИ:

- Модель (model);
- Представление (view);
- Контроллер (controller);

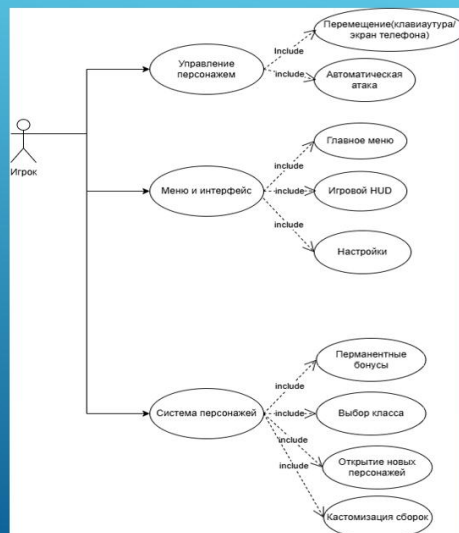
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.19.001 ПЗ

Лист

128

ДИАГРАММА ВАРИАНТОВ ИСПОЛЬЗОВАНИЯ



ОСНОВНЫЕ МОДУЛИ:

- Модель (model);
- Представление (view);
- Контроллер (controller);

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.19.001 ПЗ

Лист

129

ЗАКЛЮЧЕНИЕ

Разработанная игра «White Knight» реализована в соответствии с поставленными в начале дипломной работы техническими и функциональными требованиями. Получилось добиться высокого уровня производительности, разнообразить геймплей, чтобы игроку не становилось скучно и портировать игру не только на ПК, но и на телефон.

В дальнейшем можно выложить игру на платформы и продолжить ее поддержку.

- Постоянно добавлять новый контент, например:
- новые персонажи с уникальными способностями и типами оружия;
- добавить новые карты с уникальными погодными условиями и соответствующими врагами;
- интегрировать систему облачных сохранений;
- адаптация игры для других платформ, например IOS;
- добавление косметических предметов;
- различные игровые ивенты для удержания игрока, за которые он будет получать уникальные награды.

СПАСИБО ЗА ВНИМАНИЕ!!!

					ПГУ 1.090301.19.001 ПЗ	Лист
						130
Изм.	Лист	№ докум.	Подпись	Дата		

