

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

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

Факультет
вычислительной техники

Кафедра
МОиПЭВМ

Направление подготовки 09.03.04 «Программная инженерия»

БАКАЛАВРСКАЯ РАБОТА

на тему

Программная реализация компьютерной игры
«Битва с боссом»

Студент _____ Абдраманов Анатолий Талгатович
(подпись, дата) (ФИО полностью)

Руководитель _____ Кольчугина Е.А.
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр _____ Такташкин Д.В.
(подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от _____ № _____)

Заведующий кафедрой _____ Козлов А.Ю.
(подпись) (фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от _____ № _____)

Секретарь ГЭК _____ Попова Н.А.
(подпись) (фамилия, инициалы)

Пенза, 2023

Федеральное государственное бюджетное образовательное учреждение
высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет

Вычислительной техники

Кафедра

МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
_____ А.Ю. Козлов
«__» _____ 2023 г

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА**

1. Студент Абдраманов Анатолий Талгатович гр. 19ВП2 факультета ВТ
<Фамилия, имя, отчество полностью>

2. Тема работы Программная реализация компьютерной игры «Битва с боссом»

Тема утверждена приказом ПГУ № _____ от " _____ " _____ 20 _____

3. Руководитель работы д.т.н., профессор кафедры «МОиПЭВМ» Кольчугина Е.А.

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

1) Разработать приложение для персонального компьютера, представляющее собой компьютерную игру с видом от третьего лица, содержащую этап битвы с боссом

*2) Среда выполнения приложения: операционная система Windows 10 и выше
Языковые средства разработки: C#, UML*

*Инструментальные средства разработки: Microsoft Visual Studio 2019,
Microsoft Project, DrawIO, Blender, Unity*

Технологии разработки: RUP, ООП

3) Основные функции приложения:

а) предоставление игроку управления главным героем игры, включающем в себя передвижение (включает ходьбу и бег), атаки (легкие и тяжелые) и методы защиты от вражеских атак (блок, пережат, рывок в сторону).

б) предоставление игроку возможности начать новую игру (выбрав оружие и сложность) или продолжить игру с последнего сохранения.

в) сохранение прогресса игрока в контрольных точках игры.

5. Перечень подлежащих разработке вопросов

1) Анализ предметной области и требований

2) Планирование разработки и оценка бюджета

3) Проектирование приложения

4) Реализация

5) Контроль качества программного обеспечения

6) Оформление пояснительной записки

5. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и требований к системе	10%	12.04.2023 – 16.04.2023	
2	Проектирование программных средств	20%	17.04.2023 – 26.04.2023	
3	Реализация программных средств	30%	27.04.2023 – 11.05.2023	
4	Тестирование программных средств	20%	12.05.2023 – 21.05.2023	
5	Оформление пояснительной записки	20%	22.05.2023 – 01.06.2023	

Дата выдачи задания «22» декабря 2023г.

Руководитель бакалаврской работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

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

Реферат

Пояснительная записка содержит 129 листов, 57 рисунков, 21 использованный источник, 2 приложения, 7 таблиц.

ВИДЕОИГРЫ, ACTION, C#, ВИДЕОИГРА С ВИДОМ ОТ ТРЕТЬЕГО ЛИЦА

Объект разработки: видеоигра в жанре Action с видом от третьего лица и трёхмерной графикой.

Цель разработки: изучить предметную область и провести её анализ, провести анализ целевой аудитории, провести планирование разработки и оценку бюджета проекта, разработать и протестировать программное обеспечение.

Средства разработки: C#, Unity, Blender, MS Visual Studio 2019, DrawIO, Microsoft Project.

					ПГУ 09.03.04 – 08BP192.01 ПЗ			
Изм.	Лист	№ докум	Подпись	Дата				
Разраб		Абдраманов А.Т.			«Программная реализация компьютерной игры «Битва с боссом»» Пояснительная записка.	Литера	Лист	Листов
Пров		Кольчугина Е.А.					4	129
Н. Контр.		Такташкин Д.В.				Группа 19ВП2		
Утв		Козлов А.Ю.						

Содержание

Введение.....	7
1 Анализ предметной области и требований.....	9
1.1 Анализ предметной области и постановка задачи.....	9
1.2 Техническое задание.....	10
1.2.1 Функциональные требования к приложению	10
1.2.2 Требования к хранению данных	10
1.2.3 Требования к языкам программирования.....	11
1.2.4 Требования к программному обеспечению.....	11
1.2.5 Требования к аппаратному обеспечению	11
1.2.7 Требования к языковому оформлению	11
1.2.8 Требования к эргономике и технической эстетике	11
1.3 Анализ целевой аудитории.....	11
1.4 Анализ программных продуктов аналогичного назначения	13
1.4.1 Компьютерная игра Furi	15
1.4.2 Компьютерная игра Dark Souls.....	16
1.4.3 Компьютерная игра Risk of Rain 2	18
1.4.4 Сравнение аналогов	19
1.5 Анализ вариантов использования.....	20
1.6 Выбор технологий и инструментальных средств разработки	24
3 Проектирование приложения.....	26
3.1 Проектирование интерфейса пользователя	26
3.2 Проектирование структуры программного обеспечения.....	32
3.3 Проектирование алгоритмов.....	42
3.4 Проектирование структур данных	48
4 Реализация приложения	49
4.1 Кодирование приложения	49
5 Планирование разработки и оценка бюджета.....	52
5.1 Календарное и ресурсное планирование	52
6 Тестирование	55

6.1 Выбор способа тестирования приложения.....	55
6.2 Тестирование приложения методом «чёрного ящика»	55
6.3 Подсчёт метрик кода.....	68
6.3.1 Подсчёт метрики Холстеда	70
6.3.2 Подсчёт метрики Чепина.....	74
Заключение	76
Список использованных источников	77
Приложение А. Код программного обеспечения.	80
Приложение Б. Скриншоты тестов.....	115

Введение

Видеоигры начали свою историю в 1950-х годах и стали набирать популярность в 1970–1980 годы. На данный же момент видеоигры являются одним из самых популярных развлечений и занимают важное место в мировой культуре.

Рынок видеоигр считается одним из наиболее больших и перспективных в сфере развлечений. По состоянию на 2022 год объём рынка видеоигр составил 184,4 миллиардов долларов [1]. В 2022 году лидерами данного рынка по выручке можно назвать США (\$20,8 млрд), Японию (\$14,2 млрд) и Китай (\$13,4 млрд) [2].

Наиболее важными игровыми платформами на данный момент являются смартфоны, консоли и персональные компьютеры (далее ПК). Среди вышеперечисленных платформ наиболее перспективными и распространёнными являются смартфоны. На второе место можно поставить консоли и на третье ПК.

Самым популярным магазином игр по состоянию на 2017 год является Steam и главными его конкурентами являются магазины Origin, Uplay, BattleNet и GOG [3]. В текущее время Steam всё ещё остаётся одним из крупнейших и самых популярных магазинов игр в мире. Также к списку конкурентов Steam на данный момент можно добавить игровые магазины Epic Games Store и Microsoft Store.

Одним из наиболее крупных и популярных игровых жанров является Action. В рамках данного жанра можно выделить, например, такие жанры как Slasher, Shooter и Fighting.

Большая часть крупных игр, созданных в наше время, принадлежит именно к жанру Action. В ряду данных видеоигр можно увидеть серии Grand Theft Auto, Resident Evil, The Witcher, Call of Duty и т. д.

К признакам жанра Action можно отнести [4]:

- активность – признак подразумевающий, что игрок должен совершать действия, результат которых зависит от его реакции, координации и глазомера;
- динамика – признак, который определяет насыщенность самой видеоигры действиями и ситуациями (сюжетами), требующими быстрых решений, а также готовность игроков участвовать в подобных действиях;
- конфликтность – признак, выражающийся как в игровых механиках, так и в сюжете видеоигр, заключается в наличии в сюжете игры конфликта, который является поводом для игрового персонажа/игрока победить игровых соперников с помощью своих умений;
- персонаж – сюжетное действие в Action-играх реализуется через поступки конкретного персонажа (или группы персонажей), через то, как герои достигают своих целей при помощи вышеуказанных умений игрока как управляющего, через их выразительность и зрелищность.

Видеоигры в жанре Action способны стимулировать работу мозга и развивать реакцию. Однако, в первую очередь целью разработки видеоигры является предоставление пользователю возможности развлечься.

На основании всего вышесказанного можно сделать вывод о том, что разработка компьютерной игры в жанре Action является вполне актуальной и перспективной на данный момент.

1 Анализ предметной области и требований

1.1 Анализ предметной области и постановка задачи

Целью данной работы является разработка игры от третьего лица в жанре Action. Данная игра будет работать на персональных компьютерах под управлением операционной системы Windows 10 и выше. Игра будет одиночной, поэтому для её функционирования не потребуется иметь доступ к сети интернет. Игра покажет небольшую историю, которая будет включать в себя несколько сражений между главным героем и его противниками. История закончится битвой с финальным боссом.

К основным функциям игры можно отнести следующие:

- управление главным героем игры;
- начало новой игры;
- выбор сложности игры;
- выбор оружия главного героя;
- сохранение прогресса игрока в контрольных точках игры.

Возможности главного героя будут включать в себя:

- обычное и ускоренное (бег) передвижение по поверхности игровых уровней;
- слабая и сильная атаки;
- блок вражеских ударов;
- перекал, отскок и прыжок.

Основными возможностями противников будут атака цели, преследование цели и блок атак цели.

Босс, появляющийся на финальной стадии прохождения, по сравнению с обычными противниками будет иметь пару специальных атак и две фазы боя. Во второй фазе боя босс получает одну дополнительную атаку в арсенал атак, а также урон его атак увеличивается.

1.2 Техническое задание

Назначение:

- предоставление управления главным героем игры;
- реализация обычных врагов, поведение которых регулируется системой состояний (среди которых можно выделить состояние покоя, состояние преследования, состояние боевой стойки и состояние атаки);
- реализация игрового босса, бой с которым будет включать две фазы (во второй набор атак босса будет меняться) и поведение которого также будет регулироваться системой состояний, аналогичной системе состояний обычных врагов.

Цель создания приложения: предоставление пользователям видеоигры для развлечения.

1.2.1 Функциональные требования к приложению

Разрабатываемая видеоигра должна иметь описанные ниже характеристики и функции.

Должна иметься возможность начала новой игры с выбором уровня сложности и вооружения главного героя.

Должна быть возможность продолжить игру с последнего сохранения, если таковое имеется.

Должно быть реализовано управление главным героем, которое включает управление передвижением и боевыми приёмами (атака, блок, пережат и т. д.).

Игра должна сохраняться автоматически в контрольных точках уровней.

1.2.2 Требования к хранению данных

Прогресс игрока в прохождении игры будет сохраняться в текстовом файле, который будет находиться в папке с игрой. Данный текстовый файл будет содержать информацию о текущих уровне игры, координатах главного героя на уровне игры, уровне сложности прохождения, оружии главного героя.

1.2.3 Требования к языкам программирования

Для кодирования игры будет использоваться язык программирования C#.

1.2.4 Требования к программному обеспечению

Для успешной работы видеоигры требуется операционная система Windows 10 или Windows 11.

1.2.5 Требования к аппаратному обеспечению

Для успешной работы видеоигры потребуется аппаратное обеспечение, обладающее описанными ниже характеристиками.

Процессор: Intel Core i3.

Оперативная память: 4096Мб.

Свободное место на накопителе: 80 Мб.

1.2.7 Требования к языковому оформлению

Интерфейс игры должен быть выполнен полностью на русском языке.

1.2.8 Требования к эргономике и технической эстетике

Игра должна быть оптимизирована для отображения на экранах с соотношением сторон 16:9 при разрешении окна не меньше, чем 1280 на 720.

Главное меню и меню паузы игры должны быть выполнены в едином стиле. Перемещение между экранами меню должно быть удобным и интуитивно понятным.

Управление главным героем должно быть также удобным и интуитивно понятным.

1.3 Анализ целевой аудитории

Целевой аудиторией разрабатываемого проекта являются геймеры, использующие ПК в качестве основной игровой платформы.

Число людей, играющих в видеоигры, в 2020 году достигло 2,894 млрд, в 2022 году – 3,228 млрд, а в 2025 году по прогнозу может достигнуть 3,572 млрд [1]. Ожидается, что рост числа игроков будет равен 4,3% в год на промежутке 2020–2025 годов [1]. Таким образом можно сделать вывод о том, что число геймеров в мире с каждым годом существенно растёт.

На рисунке 1 показана диаграмма, демонстрирующая распределение выручки рынка видеоигр по основным сегментам за 2022 год [1].

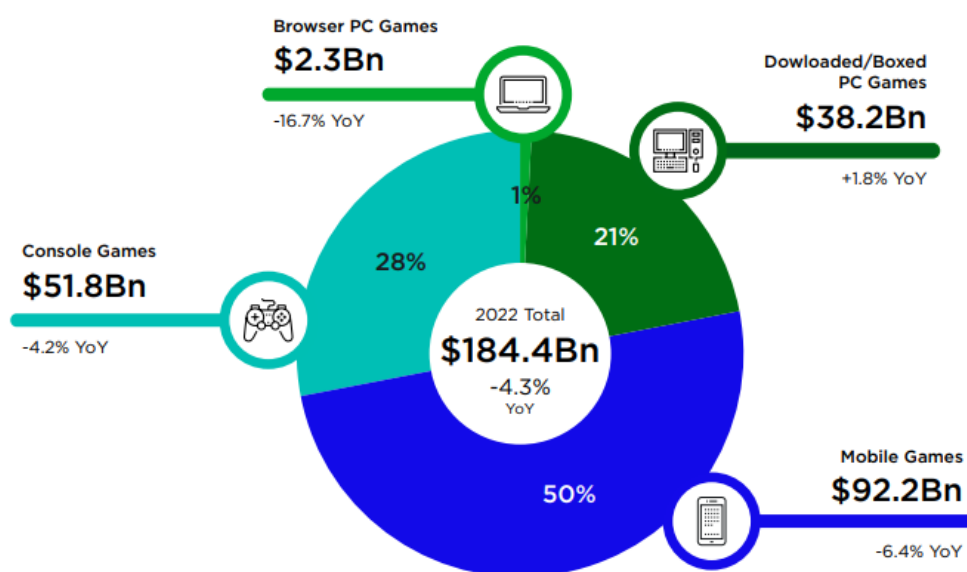


Рисунок 1 – Диаграмма выручки рынка видеоигр по сегментам за 2022 год

Доля объёма рынка видеоигр, приходящаяся на ПК, в 2022 году составляла 21%. Таким образом можно сделать вывод о том, что доля рынка, приходящаяся на ПК, остаётся значительной, хоть и уступает долям смартфонов и консолей.

Данные, представленные на рисунке 2 позволяют сделать вывод о том, что выручка ПК-сегмента видеоигр растёт, хоть и доля данного сегмента в рамках всего рынка видеоигр уменьшается.

Выручка рынка видеоигр в России в 2021 году составила 177,4 млрд. рублей, из которых 48,14% пришлись на сегмент ПК-игр [5]. Таким образом можно сделать вывод об актуальности ПК, как игровой платформы, в мире и особенно в России.

На рисунке 2 показана диаграмма выручки игровой индустрии за 2012–2020 годы [6].



Рисунок 2 – Выручка игровой индустрии в распределении по сегментам

По состоянию на апрель 2023 года 62,65% персональных компьютеров работают на операционной системе Windows [7]. Среди операционных систем семейства Windows наибольшей популярностью на данный момент пользуются Windows 10 и 11, которые в апреле 2023 года использовались на 94,47% компьютеров под управлением данного семейства ОС [8]. Таким образом, следует разработать видеоигру таким образом, чтобы она успешно работала на операционных системах Windows 10 и 11.

1.4 Анализ программных продуктов аналогичного назначения

На сегодняшний день существует большое количество компьютерных игр. Разрабатываемая видеоигра будет относиться к жанру Action с видом от третьего лица. Графика в игре будет трёхмерная. Главной особенностью разрабатываемой игры будет наличие битвы с боссом, состоящей из двух фаз, а кроме того в видеоигре будет возможность выбора оружия и сложности при начале новой игры.

Ниже приведены некоторые термины, используемые в данной работе:

- босс – игровой противник, отличающийся от обычных противников большим запасом здоровья и более высоким уроном;
- игра с видом от третьего лица – игра, в которой действие показывается не из глаз главного героя, а из-за плеча или со стороны;
- софтлок (англ. softlock) – ситуация, когда игра в каком-то месте становится непроходимой (при этом причина непроходимости заключается не в вылете игры с ошибкой, а в непродуманности игры в некотором месте со стороны разработчиков);
- hack and slash – жанр видеоигр, где игровой процесс заключается в истреблении множества противников в ближнем бою с помощью разнообразного оружия;
- shoot'em up – жанр видеоигр, где игровой персонаж продвигается вперёд сам по себе и под управлением игрока он расстреливает надвигающихся на него противников, уклоняясь от их пуль и случайных препятствий (одним из наиболее известных представителей данного жанра можно считать игру Space Invaders, выпущенную в 1978 году);
- RPG (аббревиатура от role playing game) – жанр видеоигр, основанный на элементах классических настольных ролевых игр (одним из наиболее известных представителей таких настольных игр является dungeon and dragons), главной особенностью которого является рост игровых персонажей в способностях (часто игроку дают возможность управлять данным ростом посредством выбора навыков игровых персонажей);
- roguelike – жанр видеоигр, главными особенностями которого являются генерируемые случайным образом уровни, необратимость смерти (в случае смерти игрового персонажа игру придётся начинать заново) и пошаговость игрового процесса;
- shooter – жанр видеоигр, игровой процесс в котором заключается, как правило, в том, чтобы устранять игровых противников посредством игрового оружия (преимущественно стрелкового);

– fps (аббревиатура от frames per second – количество кадров в секунду) – одна из главных характеристик производительности видеоигр (оптимальным значением данной характеристики считаются стабильные 60 кадров в секунду).

Рассмотрим некоторые известные игры, похожие по жанру и стилю на игру, разрабатываемую в рамках данной работы, и проведём для них сравнительный анализ.

1.4.1 Компьютерная игра Furi

Furi – игра французской студии The Game Bakers, вышедшая в 2016 году. Представляет собой игру в жанре action с элементами жанров hack and slash и shoot’em up, а также с видом сверху. Прохождение состоит исключительно из битв с боссами, что довольно необычно для видеоигр.

Скриншот с демонстрацией момента из битвы с первым боссом игры Furi показан на рисунке 3.



Рисунок 3 – Скриншот из игры Furi

Игра получила хорошую оценку у игроков, о чём говорит 91% положительных отзывов в игровом магазине Steam. К сожалению игра не получила большой популярности, подтверждением этому можно считать

максимальное число игроков онлайн на площадке Steam за всё время существования игры равное 640 игрокам [9].

К самым сильным сторонам игры можно отнести реализацию битв с боссами и музыкальное сопровождение. Вышеупомянутые битвы состоят из нескольких фаз, каждая последующая из которых сложнее предыдущей. Furi получила «PING Award for Best OST», что подтверждает качество игровой музыки [10].

Также к достоинствам игры можно отнести возможность выбора сложности, таким образом игрок может настроить игру так, чтобы играть было комфортно.

Графика, анимация и звуковое сопровождение игры (не включая музыку) выполнены на достойном уровне.

К немногочисленным недостаткам данной игры можно отнести неудобное управление в случае игры с использованием клавиатуры и мыши. Разработчики поэтому даже перед игрой выводят на экран предупреждение о том, что лучше использовать для управления геймпад.

Также к недостаткам можно отнести то, что в игре по большому счёту нельзя выбирать навыки и оружие для главного героя. Правда этот недостаток несколько сглаживается возможностью выбрать для прохождения второго персонажа, который, к сожалению, становится доступен только после покупки платного дополнения для игры.

1.4.2 Компьютерная игра Dark Souls

Dark Souls – игра японской компании From Software, вышедшая в 2011 году, на игровые консоли PlayStation 3 и Xbox 360. На компьютерах данная игра вышла в 2012 году. Dark Souls представляет собой смесь жанров action и RPG.

Игра получила положительные оценки от игроков – 91,36% положительных отзывов в магазине Steam. Кроме того, игру можно назвать вполне популярной и на данный момент, о чём говорит то, что в промежуток с

начала по апрель 2023 года её среднее число игроков онлайн колеблется около двух тысяч [11].

Скриншот с моментом из битвы с одним из боссов Dark Souls представлен на рисунке 4.



Рисунок 4 – Скриншот из игры Dark Souls

Игра обладает сложным и интересным игровым процессом. Особенностью Dark Souls является отсутствие карты и маркеров направления движения к конкретной цели. Перед геймером стоит проблема познания неизвестного, сбор по крупицам информации (из описания предметов, врагов, рассказов союзников и собственного воображения). Каждый камень, мост или здание могут поведать целую историю [12].

Кроме того, интересной особенностью игры является её система подсказок для игрока.

В этой игре есть сразу несколько видов подсказок, среди которых наиболее интересны текстовые сообщения, которые игроки могут оставить другим игрокам в практически любой локации игры. Обладая игровым предметом Orange Guidance Soapstone («оранжевый путевой мелок»), пользователь может

оставить до шести сообщений, который случайно высветятся в виде оранжевой надписи на полу у других игроков, проходящих ту же самую локацию. Такие сообщения могут как помочь (подсказать лучший путь, игровой секрет и т. д.), так и навредить (поскольку содержат ложные подсказки, с целью подшутить над слишком доверчивыми) [13].

Также к сильным сторонам игры можно отнести звуковое сопровождение и большое разнообразие игровых стилей, которым может следовать игрок в своём прохождении. Данные игровые стили обеспечиваются возможностью по-разному развивать характеристики персонажа и подбирать ему любое оружие и снаряжение из представленных в игре.

К недостаткам игры можно отнести невозможность выбора сложности и неудобное управление при использовании клавиатуры и мыши (прицеливание из луков и арбалетов в данном случае очень неудобное).

1.4.3 Компьютерная игра Risk of Rain 2

Risk of Rain 2 – игра с видом от третьего лица от студии Hopoo Games, вышедшая в 2020 году и сочетающая в себе жанры roguelike и shooter.

Скриншот с демонстрацией момента из игры Risk of Rain 2 показан на рисунке 5.



Рисунок 5 – Скриншот из игры Risk of Rain 2

В магазине Steam Risk of Rain 2 получила отличные отзывы от игроков и имеет хороший показатель количества игроков онлайн – 96,22% процентов положительных отзывов из около 230,000 отзывов и около семи с половиной тысяч игроков онлайн в среднем на начало мая 2023 года [14].

Одной из главных особенностей игры является то, что сложность игры в рамках одного прохождения бесконечно растёт по ходу времени. Другой особенностью игры является то что, главный герой может практически бесконечно наращивать силу, подбирая случайно раскидываемые по уровням предметы.

Данная игра обладает большим разнообразием рядовых противников и боссов, хорошим звуковым сопровождением и высокой сложностью игрового процесса. Кроме того, Risk of Rain 2 предоставляет игроку возможность играть самыми различными стилями, позволяя выбирать игрового персонажа и его навыки и настраивать модификаторы игрового процесса. К достоинствам игры также можно добавить возможность выбора сложности и удобное управление.

К недостаткам данной игры можно отнести проседание производительности игры при большом количестве врагов на экране. Эта проблема также может проявиться, когда герой игры станет слишком сильным. Например, производительность может упасть из-за того, что герой выпустит очень много ракет.

Ещё одним недостатком является то, что в игре есть моменты, когда случайность может сделать текущее прохождение игры почти невозможным.

1.4.4 Сравнение аналогов

Сравнительный анализ рассмотренных выше игр, похожих на разрабатываемую, представлен в таблице 1.

Теперь рассмотрим подробнее критерии сравнения, используемые в таблице 1:

- выбор сложности – предоставление игроку возможности выбирать сложность прохождения игры;

- производительность – критерий, показывающий стабильность работы игры (которая тем выше, чем реже fps в игре падает ниже 60 кадров в секунду) при соблюдении рекомендуемых разработчиками требований к аппаратному обеспечению;
- разнообразие стилей игры – критерий, показывающий есть ли у игрока возможность выбирать умения или оружие главного героя игры, таким образом выбирая стиль игры;
- удобство управления – критерий, показывающий насколько удобно управление видеоигры при использовании клавиатуры и мыши;
- возможность софтлока – критерий, показывающий насколько высока вероятность софтлока в игре.

Таблица 1 – Сравнение аналогов

Игра	Выбор сложности	Производительность	Разнообразие стилей игры	Удобство управления	Возможность софтлока
Furi	Имеется	Стабильная	Есть, но только с платным дополнением	Не удобное	Крайне низкая
Dark Souls	Отсутствует	Стабильная	Есть	Не удобная стрельба из луков и арбалетов	Крайне низкая
Risk of Rain 2	Имеется	Не стабильная	Есть	Удобное	Возможен
Разрабатываемое приложение	Имеется	Стабильная	Есть	Удобное	Крайне низкая

В результате анализа игр, похожих на разрабатываемую, было принято решение о разработке собственной игры, которая будет лишена недостатков, выявленных у рассмотренных выше игр.

1.5 Анализ вариантов использования

В разрабатываемой системе есть только один вид пользователей – игроки.

Видеоигра должна предоставлять пользователю возможность начать новую игру или продолжить уже начатое прохождение игры с последнего

сохранения, а также должна быть опция выхода из игры. Если пользователь решит начать новую игру, то ему должны предоставить возможности выбора сложности и выбора оружия для главного героя.

Во время прохождения видеоигры пользователь в основном будет управлять главным героем. Управление героем должно позволять совершать следующие действия в игре: лёгкая атака, тяжёлая атака, блок, пережат, рывок, отскок назад, захват врага в фокус камеры, переключение захвата в фокус камеры на ближайшего врага слева или справа, ходьба по земле, бег по земле, выход в меню паузы, выход из меню паузы, выход в главное меню.

Видеоигра должна сохранять прогресс игрока при достижении им контрольных точек.

В результате анализа требований задания была создана диаграмма вариантов использования (рисунок 6).



Рисунок 6 – Диаграмма вариантов использования

Диаграммы вариантов использования – это один из видов диаграмм UML, предназначенных для моделирования динамических аспектов систем. Диаграммы вариантов использования – основной вид диаграмм при моделировании поведения системы, подсистемы или класса. Каждая из них показывает набор вариантов использования и действующих лиц в их взаимодействии [15].

Для варианта использования «Начать новую игру» был составлен сценарий варианта использования (рисунок 7).

Название: Начать новую игру
Назначение: начало нового прохождения игры
Актеры: пользователь
Предусловие: приложение запущено
Основной сценарий: 1. Пользователь нажимает в главном меню кнопку «Новая игра». 2. Пользователь выбирает сложность игры. 3. Пользователь выбирает оружие главного героя.
Постусловие: начинается новая игра

Рисунок 7 – Сценарий варианта использования «Начать новую игру»

В рамках варианта использования «Начать новую игру» пользователь начинает новую игру. Для этого он нажимает кнопку «Новая игра» в главном меню игры, а потом выбирает сложность игры и оружие главного героя.

Для варианта использования «Захват врага в фокус камеры» был составлен сценарий варианта использования (рисунок 8).

Название: Захват врага в фокус камеры
Назначение: контроль режима работы камеры (свободный режим и режим фокусировки)
Актеры: пользователь
Предусловие: приложение запущено и пользователю передано управление главным героем
Основной сценарий: 1. Пользователь нажимает на клавишу, отвечающую за захват врага в фокус камеры. 2. Приложение проверяет, находится ли главный герой в состоянии фокуса на враге. Если находится – переход на шаг 3. Если не находится – камера переходит в свободный режим, а дальше переход на шаг 6. 3. Составляется список врагов, находящихся в определённом радиусе перед главным героем в поле его зрения. 4. Определяется какой из врагов в списке из шага 3 находится к нему ближе всех остальных. 5. Камера фокусируется на враге, найденном на шаге 4. 6. Завершение.
Постусловие: камера переведена в состояние фокусировки или в свободный режим

Рисунок 8 – Сценарий варианта использования «Захват врага в фокус камеры»

В рамках варианта использования «Захват врага в фокус камеры» пользователь отдаёт команду игре захватить ближайшего игрового противника в фокус камеры, чтобы противник всегда находился в поле зрения пользователя и с ним было удобнее вести сражение.

Для варианта использования «Совершить легкую атаку» был составлен сценарий варианта использования (рисунок 9).

Название: Совершить легкую атаку
Назначение: совершение лёгкой атаки
Актеры: пользователь
Предусловие: приложение запущено и пользователю передано управление главным героем
Основной сценарий: 1. Пользователь нажимает на клавишу легкой атаки. 2. Приложение проверяет не занят ли герой другими действиями. Если герой занят другими действиями, то легкая атака не совершается. В обратном случае переход на шаг 3. 3. Приложение проверяет хватает ли герою выносливости на удар. Если выносливости не хватает, то удар не совершается. В обратном случае переход на шаг 4. 4. Герой начинает легкую атаку. 5. Герой тратит выносливость, нужную для совершения легкой атаки, и всем, кто стоит перед героем на дистанции атаки, наносится урон. 6. Герой завершает легкую атаку.
Постусловие: совершена легкая атака

Рисунок 9 – Сценарий варианта использования «Совершить легкую атаку»

В рамках варианта использования «Совершить легкую атаку» пользователь отдаёт главному герою видеоигры команду провести легкую атаку. Для того, чтобы атака была проведена требуется, чтобы главный герой не был занят другими действиями и ему было достаточно выносливости.

Для варианта использования «Совершить перекат» был составлен сценарий варианта использования (рисунок 10).

Название: Совершить перекат
Назначение: совершение переката
Актеры: пользователь
Предусловие: приложение запущено и пользователю передано управление главным героем
Основной сценарий: 1. Пользователь выбирает регистрацию 2. Приложение проверяет не занят ли герой другими действиями. Если герой занят другими действиями, то перека не совершается. В обратном случае переход на шаг 3. 3. Приложение проверяет хватает ли герою выносливости на перека. Если выносливости не хватает, то перека не совершается. В обратном случае переход на шаг 4. 4. Герой начинает перека. 5. Герой тратит на перека выносливость и включается время неуязвимости. 6. Время неуязвимости во время переката заканчивается. 7. Герой заканчивает перека.
Постусловие: перека совершен

Рисунок 10 – Сценарий варианта использования «Совершить перека»

В рамках варианта использования «Совершить перекат» пользователь отдаёт главному герою видеоигры команду провести перекат. Для этого проводятся проверки, аналогичные проверкам из варианта использования «Совершить легкую атаку».

В ходе анализа вариантов использования были выявлены основные функции разрабатываемой видеоигры в виде вариантов использования, которые соответствуют техническому заданию на разработку видеоигры. Для некоторых вариантов использования были составлены сценарии, где варианты использования описаны в общих чертах.

1.6 Выбор технологий и инструментальных средств разработки

Для разработки приложения было решено использовать методологию RAD (аббревиатура от rapid application development). Rapid Application Development – это жизненный цикл процесса проектирования, созданный для достижения более высокой скорости разработки и качества ПО, чем это возможно при традиционном подходе к проектированию [16]. Данная технология разработки включает в себя, как правило, четыре этапа: планирование требований, совместное проектирование приложения, конструирование, приемочные испытания и установка программы.

К основным принципам методологии RAD можно отнести [16]:

- работа ведётся небольшими группами людей (RAD группами);
- создание прототипа для уточнения требований заказчика;
- RAD группа всегда работает только над одним прототипом;
- применение инструментов, которые автоматизируют процесс разработки.

Методология RAD подходит для разработки видеоигры в данном случае по следующим причинам: бюджет проекта ограничен, сжатые сроки разработки, проект не имеет большой вычислительной сложности.

Для планирования работ проекта решено использовать Microsoft Project. Данное приложение предоставляет все основные функции для планирования

проекта, такие как: составление списка задач, составление календарного плана проекта, задание ресурсов проекта, задание загрузки ресурсов проекта, расчёт материальных, трудовых и финансовых ресурсов, составление диаграмм проекта.

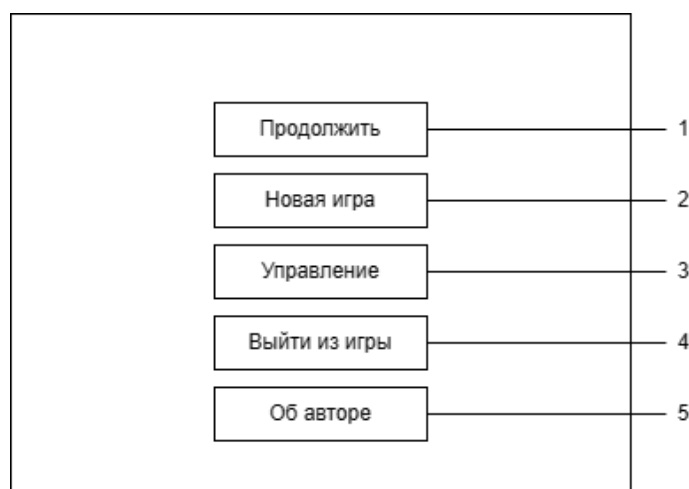
Для реализации приложения было решено использовать игровой движок Unity, ПО для создания трёхмерной графики Blender и язык программирования C#. Программное обеспечение Unity является мощным, чрезвычайно простым в освоении, а также бесплатным (пока проект не начнет приносить большой доход) [17]. К основным преимуществам Unity можно отнести: универсальность, бесплатность и большое количество обучающих материалов в сети Интернет. Blender – свободный пакет для создания трехмерной компьютерной графики, включающий в себя средства моделирования, анимации, рендеринга, пост-обработки видео, а также создание интерактивных игр [18]. Для написания и редактирования кода на C# было решено использовать среду разработки Visual Studio 2019.

3 Проектирование приложения

3.1 Проектирование интерфейса пользователя

Для видеоигры был разработан интерфейс. Он включает в себя интерфейс главного меню, интерфейс меню паузы, экраны с информацией об управлении и авторе, HUD видеоигры при битве с боссом и в остальных случаях. HUD (аббревиатура от heads-up display) – часть визуального интерфейса игры, отображающаяся на фоне виртуального пространства экрана [19]. В HUD выводится наиболее востребованная для пользователя информация, обработка которой не требует большого вовлечения и времени [19].

Схема интерфейса главного меню представлена на рисунке 11.



(1 – кнопка «Продолжить» (отсутствует, если на данный момент в игре нет файла сохранения), 2 – кнопка «Новая игра», 3 – кнопка «Управление», 4 – «Выйти из игры», 5 – «Об авторе»)

Рисунок 11 – Схема интерфейса главного меню игры

При нажатии на кнопку «Продолжить» в интерфейсе главного меню игры происходит загрузка игры на последнюю пройденную игроком контрольную точку. При нажатии на кнопку «Новая игра» произойдёт переход в меню выбора сложности и вооружения главного героя игры. При нажатии на кнопку «Управление» произойдёт переход на экран с информацией об управлении игры. При нажатии на кнопку «Выйти из игры» произойдёт закрытие видеоигры. При

нажатии на кнопку «Об авторе» произойдёт переход на экран с информацией об авторе.

На рисунке 12 показан реализованный интерфейс главного меню игры.

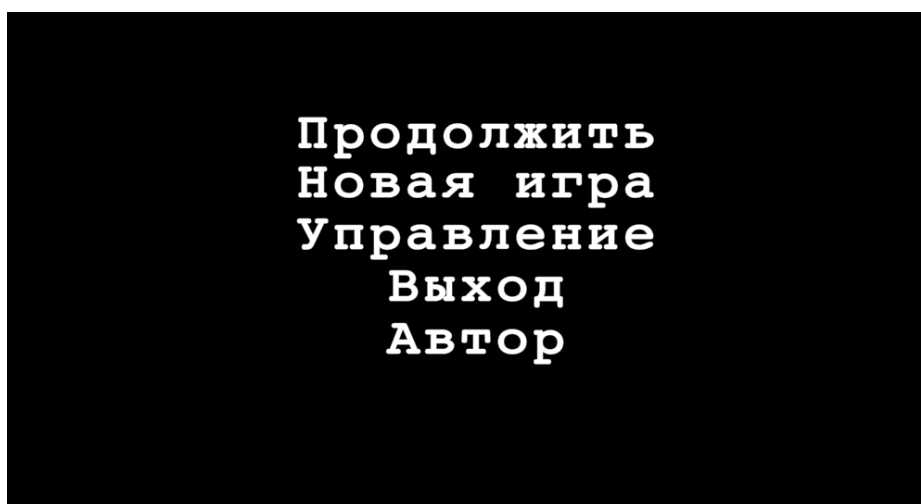
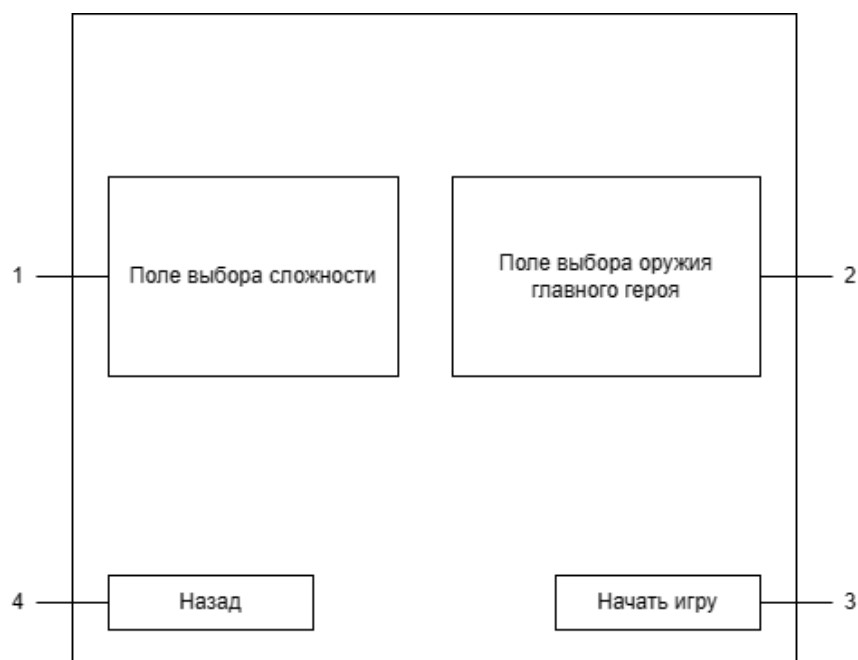


Рисунок 12 – Интерфейс главного меню игры

Схема интерфейса выбора сложности игры и оружия главного героя показана на рисунке 13.



(1 – поле выбора сложности, 2 – поле выбора оружия главного героя, 3 – кнопка «Начать игру», 4 – кнопка «Назад»)

Рисунок 13 – Схема интерфейса меню выбора сложности игры и оружия главного героя

При нажатии на кнопку «Начать игру» будет загружен первый уровень игры, а при нажатии на кнопку «Назад» произойдёт переход в главное меню игры.

На рисунке 14 показана реализация интерфейса меню выбора сложности игры и оружия главного героя.

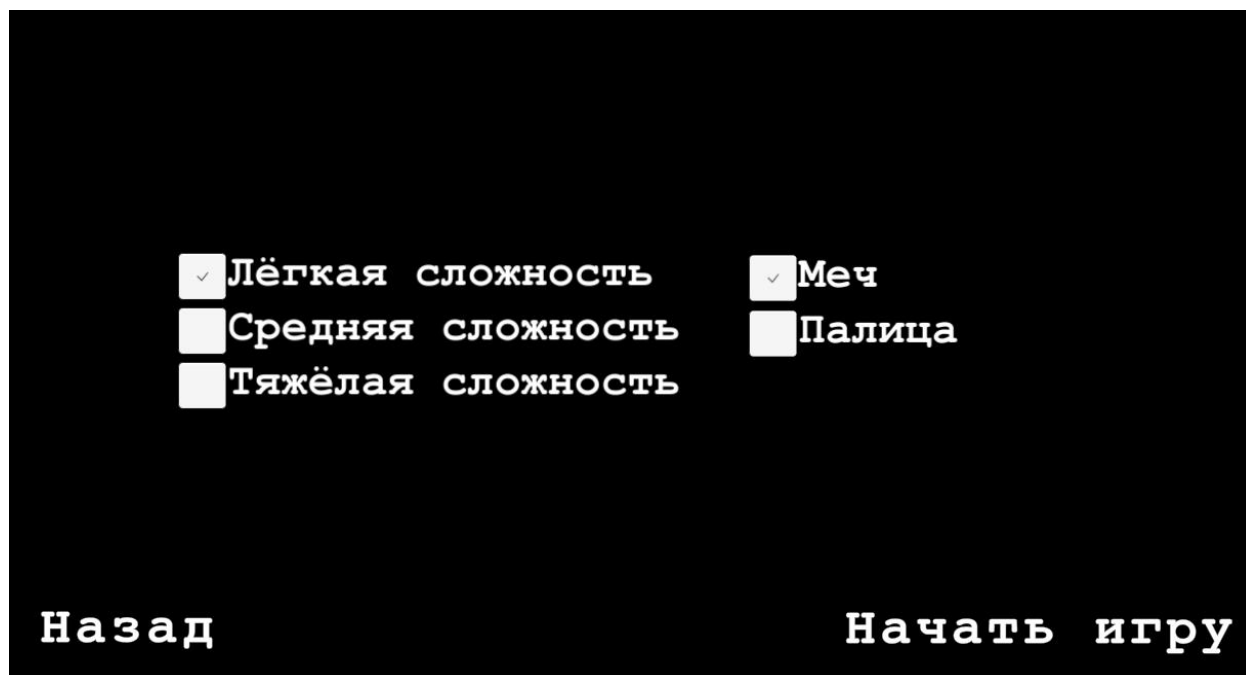


Рисунок 14 – Интерфейс выбора сложности игры и оружия главного героя

Схема экрана с информацией об управлении показана на рисунке 15.

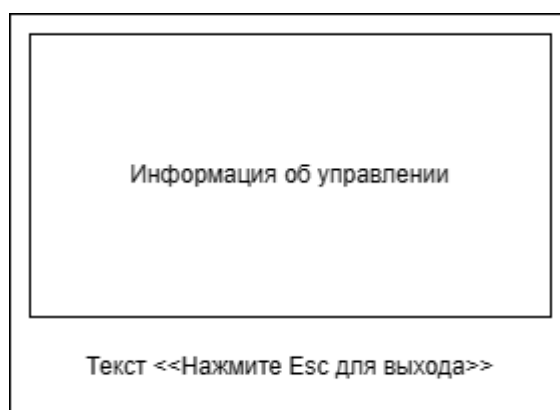


Рисунок 15 – Экран с информацией об управлении

На данном экране показана информация об основных командах управления главным героем игры. Выход в главное меню из данного экрана осуществляется путём нажатия клавиши Escape, о чём говорит также текстовая

подсказка на экране. Экран с информацией об авторе устроен таким же образом, как и экран с информацией об управлении.

Реализация экрана с информацией об управлении показана на рисунке 16.

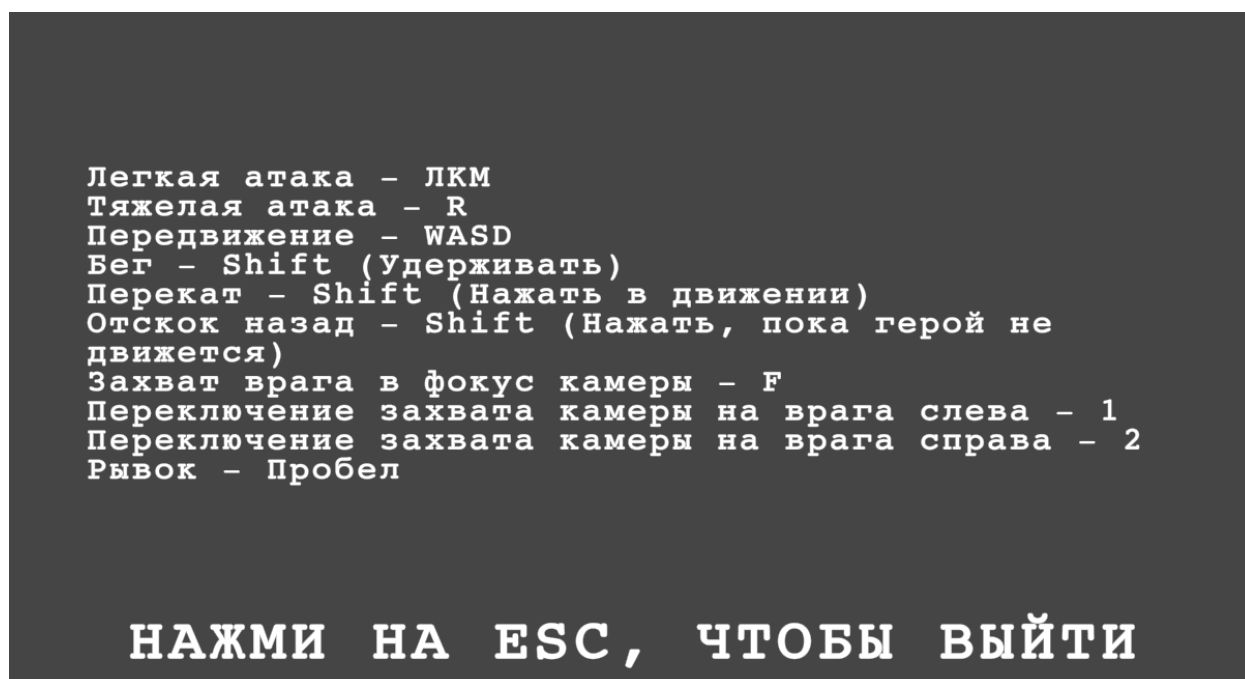
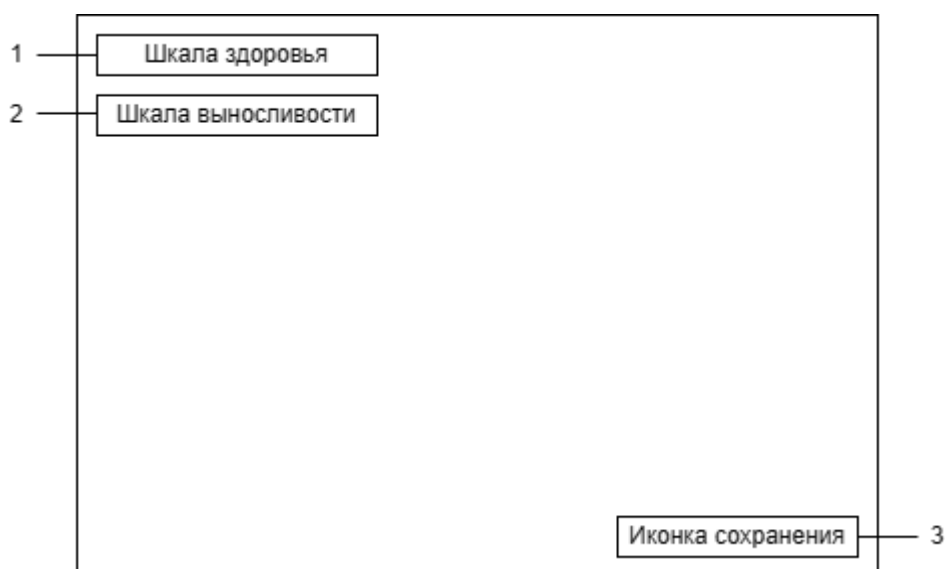


Рисунок 16 – Экран с информацией об управлении

Схема HUD'а видеоигры представлена на рисунке 17.



(1 – шкала здоровья главного героя, 2 – шкала выносливости, 3 – иконка сохранения)

Рисунок 17 – Схема HUD'а

Шкалы выносливости и здоровья, показывают текущие уровни выносливости и здоровья соответственно. Иконка сохранения обычно не видна игроку, но во время достижения главным героем контрольной точки, где игра сохраняется, иконка сохранения несколько раз мигает.

На рисунке 18 показана реализация HUD'а игры.



Рисунок 18 – реализация HUD'а

HUD видеоигры несколько изменяется во время схватки с боссом игры. Схема HUD'а во время битвы с боссом показана на рисунке 19.

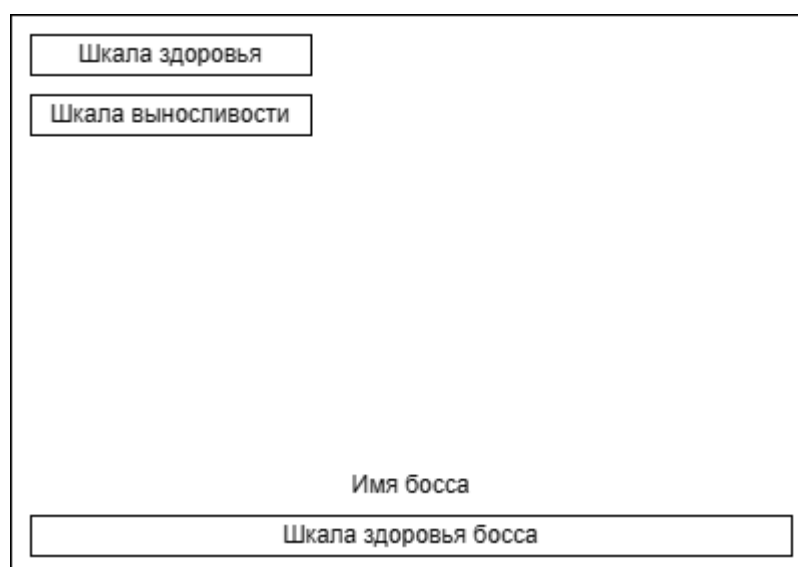


Рисунок 19 – Схема HUD'а во время битвы с боссом

Во время битвы с боссом внизу экрана появляется шкала здоровья босса и его имя.

Реализация HUD'а во время битвы с боссом показана на рисунке 20.



Рисунок 20 – HUD во время битвы с боссом

Схема меню паузы показана на рисунке 21.



(1 – кнопка «Продолжить», 2 – кнопка «Главное меню», 3 – кнопка «Выйти»)

Рисунок 21 – Схема меню паузы

Меню паузы появляется во время игры при нажатии на клавишу Escape, пока оно активно события в видеоигре приостанавливаются. При нажатии на кнопку «Продолжить» происходит возврат в игру. При нажатии на кнопку «Главное меню» происходит переход в главное меню. При нажатии на кнопку «Выйти» происходит выход из игры.

На рисунке 22 показана реализация меню паузы.

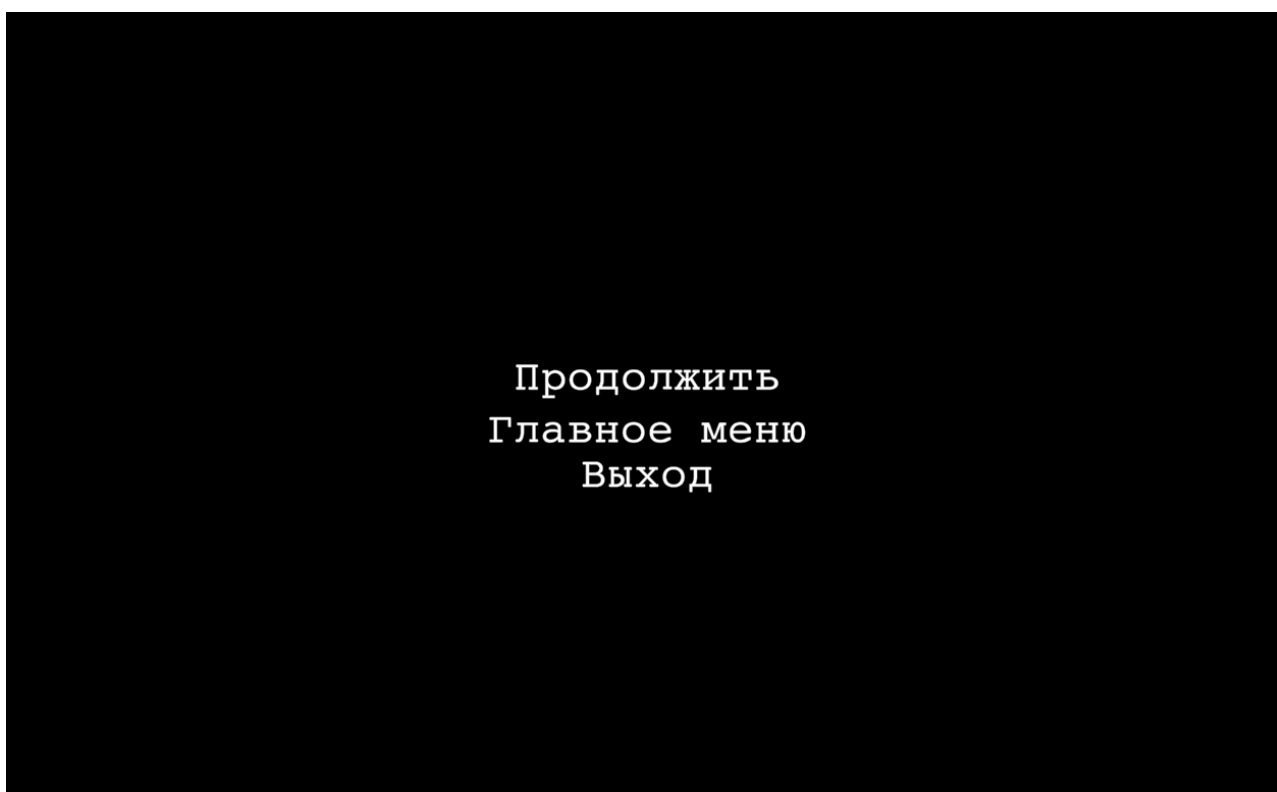


Рисунок 22 – Меню паузы

Таким образом был спроектирован интерфейс видеоигры. Были составлены макеты всех меню и экранов игры с пояснением работы элементов интерфейса, представленных на данных макетах.

3.2 Проектирование структуры программного обеспечения

На рисунке 23 показана диаграмма классов, включающая классы связанные с главным героем игры.

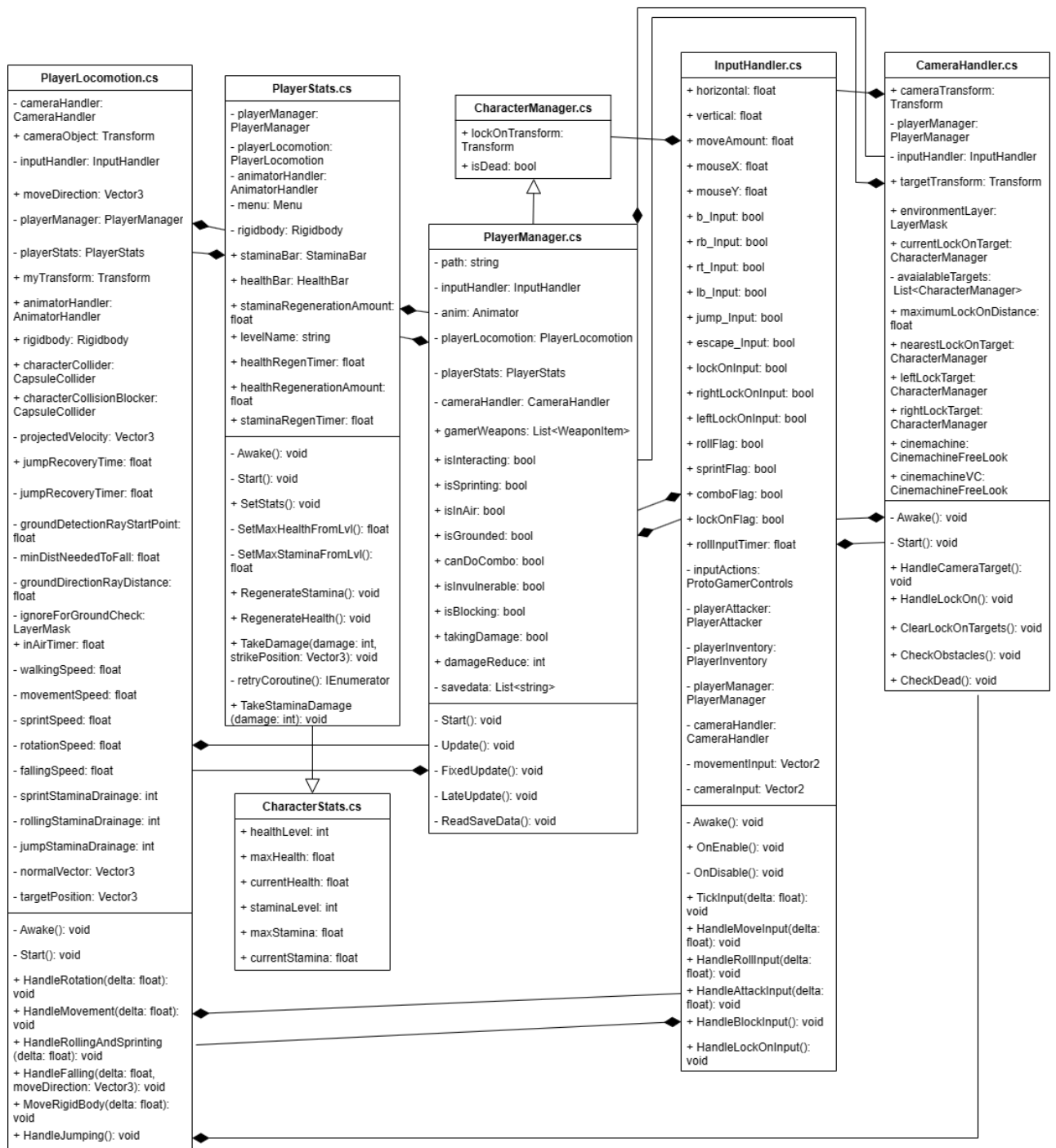


Рисунок 23 – Диаграмма классов (часть 1)

На рисунке 23 представлены классы, связанные с главным героем игры. Данные классы описывают управление главным героем, управление игровой камерой и характеристики главного героя.

На рисунке 24 показана диаграмма классов, связанная с системой вооружения главного героя.

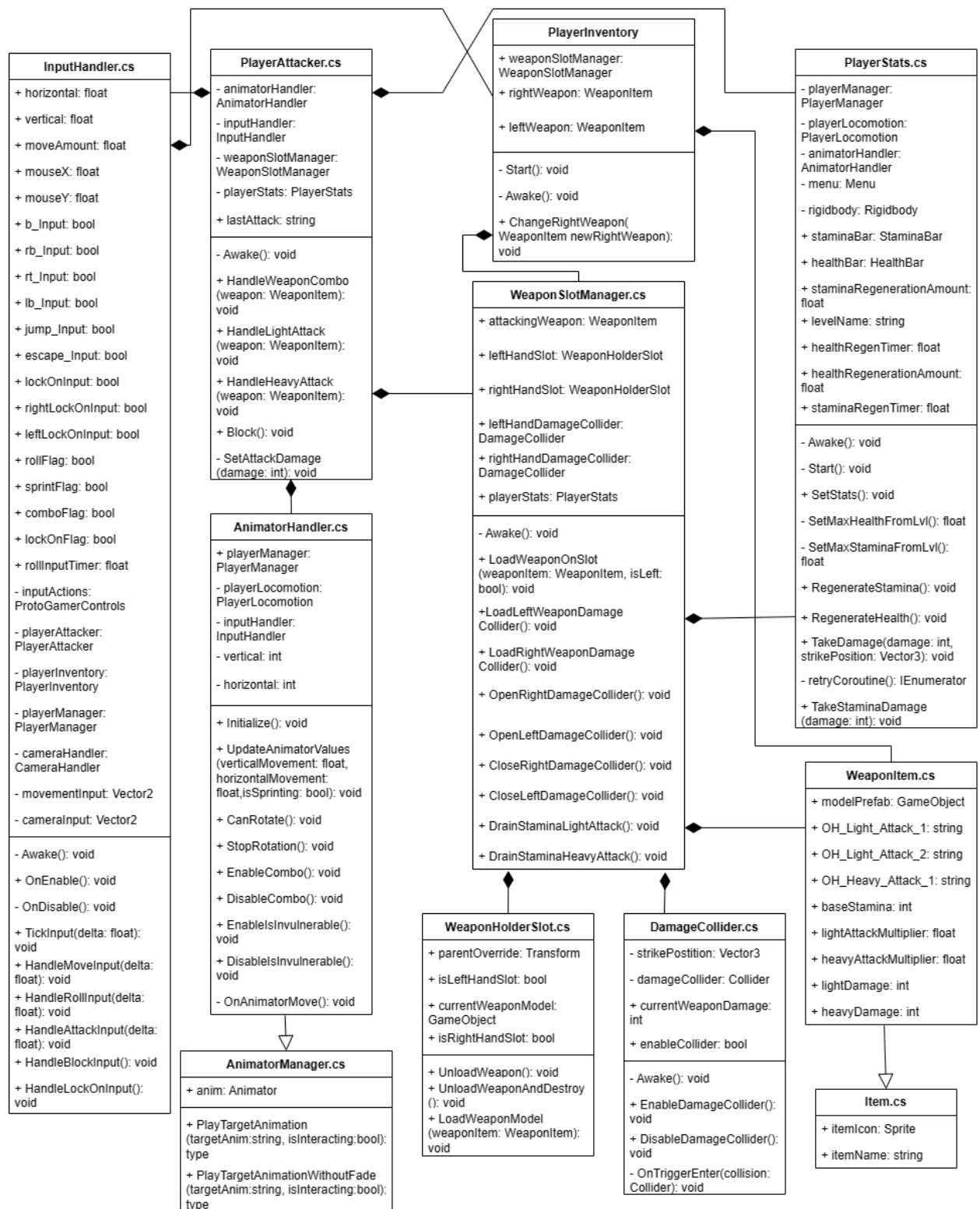


Рисунок 24 – Диаграмма классов (часть 2)

На рисунке 24 представлены классы, связанные с игровой системой вооружения. Данные классы описывают оружие, методы нанесения урона, управление анимациями атак и так далее.

На рисунке 25 показана диаграмма классов, связанная с классами, описывающими противников.

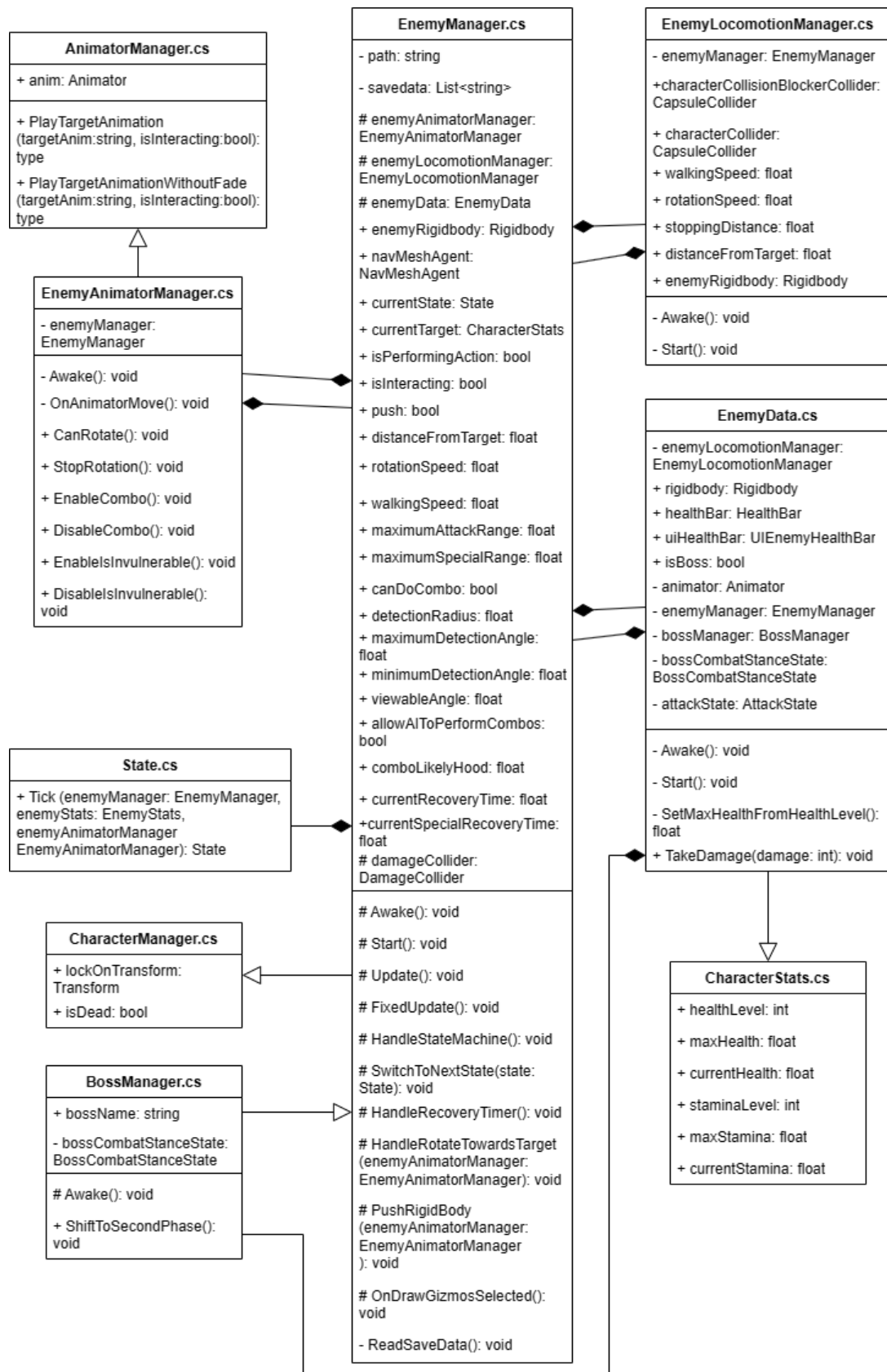


Рисунок 25 – Диаграмма классов (часть 3)

На рисунке 25 показаны классы, описывающие противников. Данные классы описывают характеристики врага, передвижение врага.

Противники в игре делятся на два типа: обычные враги и босс. Классы, описывающие босса, являются расширенными версиями классов обычных врагов, поэтому классы босса реализованы как классы-наследники по отношению к классам обычных врагов.

На рисунке 26 показана диаграмма классов, описывающих поведение противников.

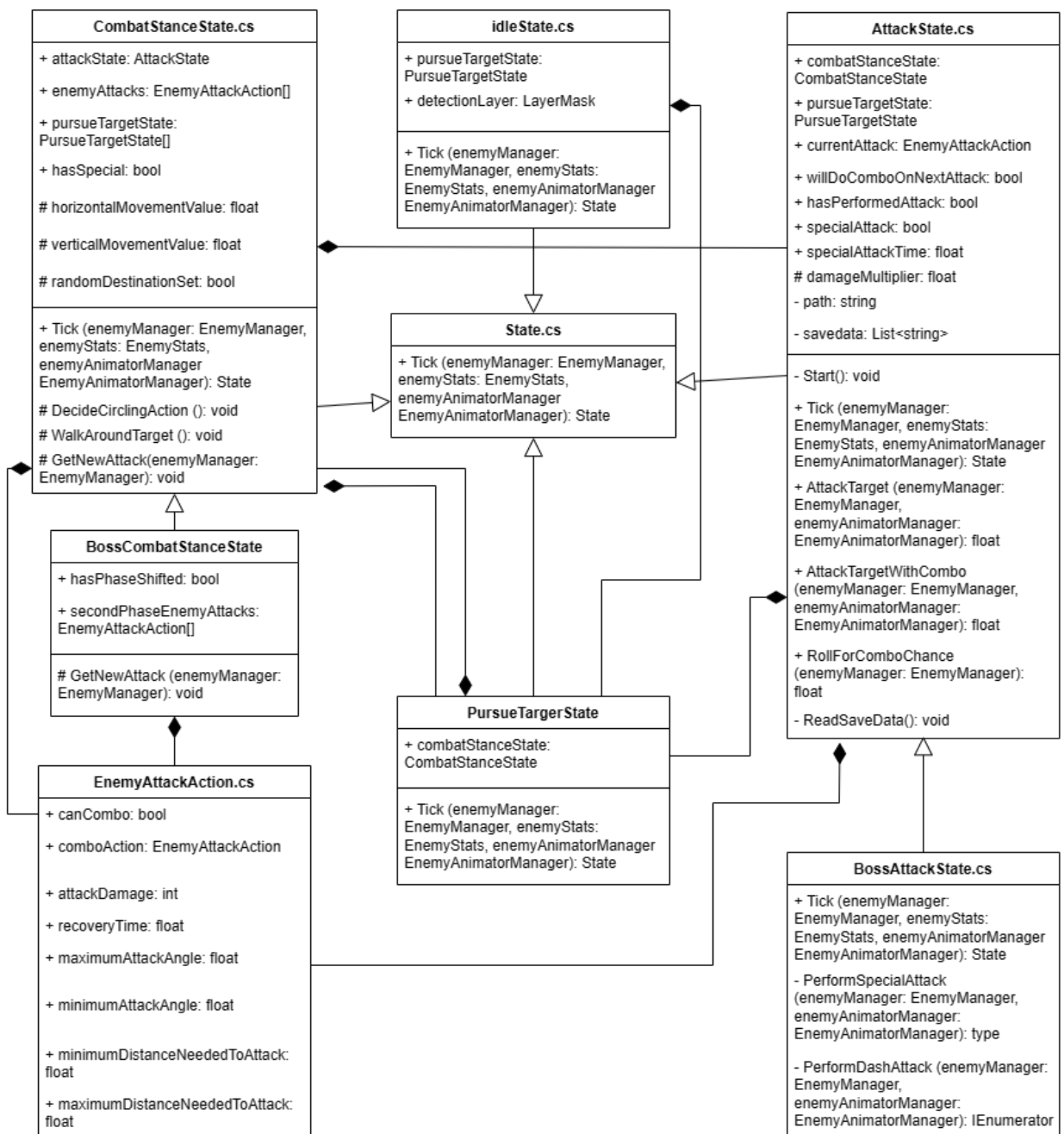


Рисунок 26 – Диаграмма классов (часть 4)

На рисунке 26 показаны классы, описывающие состояния игровых противников. Поведение врагов в игре, управляется машиной состояний, которая включает 4 состояния: состояние покоя, состояние преследования, состояние боевой стойки, состояние атаки. Набор состояний идентичен у обычных врагов и у босса. Состояния босса отличаются от состояний обычных врагов тем, что описывают две фазы боя с разными атаками, а не одну.

На рисунке 27 показана диаграмма классов, описывающих систему сохранений игры и переходы между уровнями.

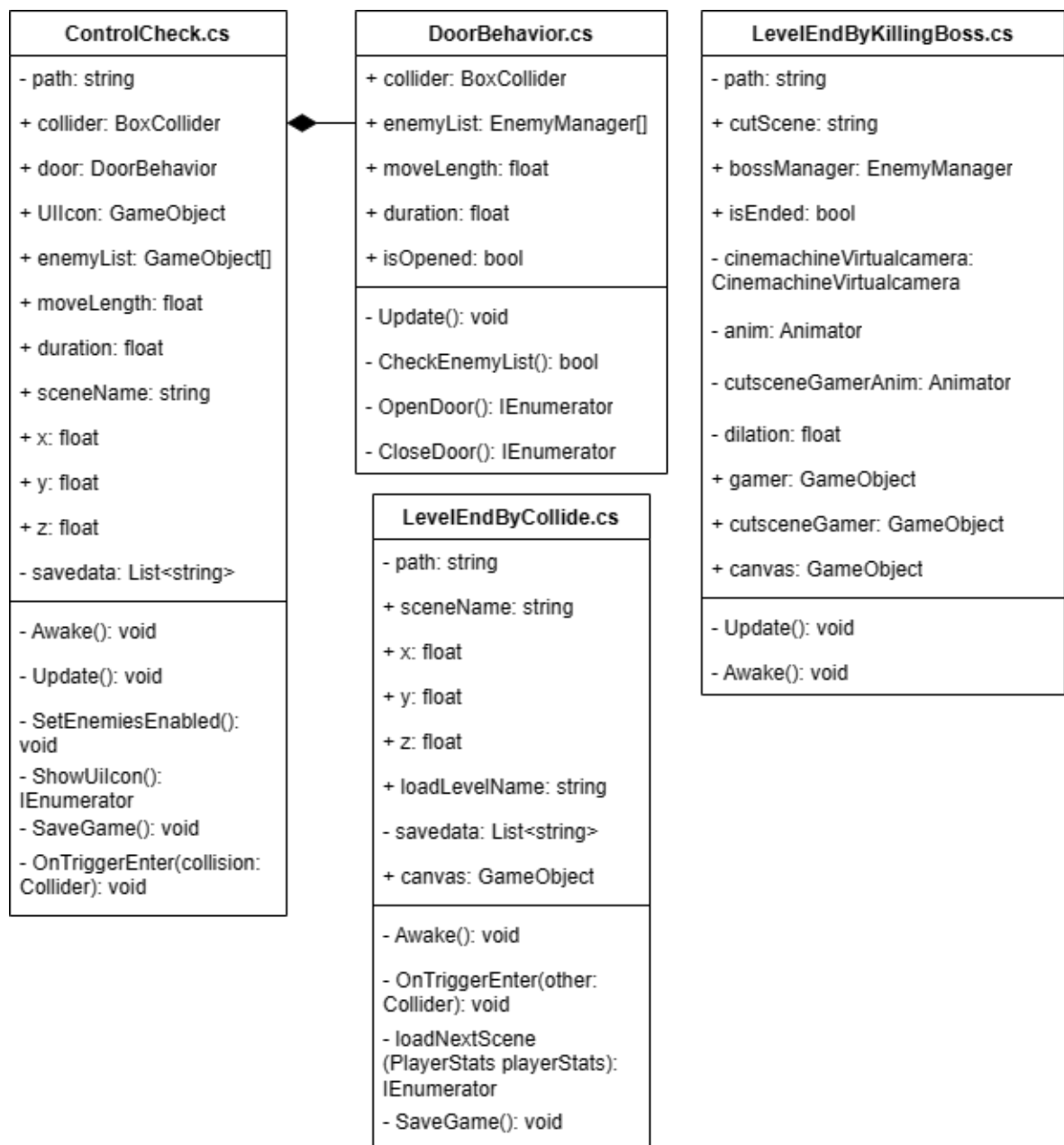


Рисунок 27 – Диаграмма классов (часть 5)

На рисунке 27 показаны классы, связанные с системой сохранений и переходов между игровыми уровнями.

Теперь опишем классы, приведенные на диаграммах классов:

- класс `CharacterManager` является родительским классом для классов `PlayerManager` и `EnemyManager` и содержит пару полей, описывающих то, жив ли персонаж и точку для фокусировки камеры в режиме захвата в случае, если камера будет зафиксирована на данном персонаже;
- класс `PlayerManager` содержит почти всю информацию о текущем состоянии главного героя, координирует работу других связанных с ним классов и в отличие от `EnemyManager` отвечает за расположение героя на игровом уровне в координатах, указанных в файле сохранения;
- класс `CharacterStats` является родительским классом для классов `PlayerStats` и `EnemyStats`, а также его поля содержат информацию о здоровье и выносливости персонажа;
- класс `PlayerLocomotion` описывает передвижение главного героя;
- класс `InputHandler` отвечает за обработку сигналов от клавиатуры и мыши;
- класс `CameraHandler` отвечает за управление режимами работы камеры (свободным режимом и режимом захвата врага в фокус камеры);
- класс `PlayerAttacker` описывает атаки главного героя и режим блока ударов;
- класс `AnimatorManager` используется для запуска различных анимаций, а также является родительским для классов `AnimatorHandler` и `EnemyAnimatorManager`;
- класс `AnimatorHandler` описывает функции, которые можно вызывать из анимаций главного героя (например, если в анимации вызвать функцию `StopRotate()`, то персонаж не сможет поворачиваться во время данной анимации), а также данный класс регулирует перемещение героя во время анимаций (например, перемещение героя во время переката);

- класс `PlayerInventory` содержит информацию о том, какие вещи держит главный герой в левой и правой руке;
- класс `WeaponSlotManager` содержит функции, вызываемые во время атак (например, одна из функций активирует способность оружия в правой руке наносить урон);
- класс `WeaponHolderSlot` описывает слот для оружия и включает сведения о том, в какой руке находится оружие и какое это оружие;
- класс `DamageCollider` описывает такие свойства коллайдера (зона, которой оружие наносит урон) оружия, как текущий урон, позиция ударяющего персонажа, текущее состояние коллайдера (включен или нет);
- класс `Item` описывает предмет и является родительским для класса `WeaponItem`;
- класс `WeaponItem` описывает оружие;
- класс `EnemyAnimatorManager` схож по смыслу с классом `AnimatorHandler`, только предназначен для противников;
- класс `EnemyManager` схож по смыслу с `PlayerManager`, но описывает не главного героя видеоигры, а врага;
- класс `BossManager` является дочерним по отношению к `EnemyManager` и относится к боссу игры;
- класс `EnemyLocomotionManager` инициализирует компоненты врага, ответственные за передвижение;
- класс `EnemyData` схож по смыслу с классом `PlayerStats`, но описывает обычных врагов и босса;
- класс `State` является родительским для других классов состояний и содержит абстрактный метод `Tick`, который отвечает за логику состояния и переопределяется в классах наследниках;
- класс `idleState` описывает состояние покоя;
- класс `PursueTargetState` описывает состояние погони за главным героем;
- класс `CombatStanceState` описывает состояние боевой стойки;

- класс `BossCombatStanceState` описывает состояние боевой стойки босса;
- класс `EnemyAttackAction` описывает атаку врага;
- класс `AttackState` описывает состояние атаки;
- класс `BossAttackState` описывает состояние атаки босса;
- класс `ControlCheck` описывает контрольную точку;
- класс `DoorBehavior` описывает поведение дверей, используется в классе `ControlCheck` для закрытия дверей за главным героем, после достижения контрольной точки;
- класс `LevelEndByCollide` описывает переход на следующий уровень;
- класс `CheckPointAfterBossDeath` описывает переход на финальную сцену после победы над боссом.

Далее приведён пример полного описания одного из классов.

Название класса: `PlayerManager`.

Назначение: содержит почти всю информацию о текущем состоянии главного героя, координирует работу других связанных с ним классов и отвечает за расположение героя на игровом уровне в координатах, указанных в файле сохранения.

Члены класса:

- `path: string` – путь к файлу сохранения;
- `inputHandler: InputHandler` – ссылка на скрипт, ответственный за обработку сигналов с клавиатуры и мыши;
- `anim: Animator` – компонент ответственный за анимации главного героя;
- `playerLocomotion: PlayerLocomotion` – ссылка на скрипт, отвечающий за перемещение и действия (вроде переката) главного героя;
- `playerStats: PlayerStats` – ссылка на скрипт, хранящий характеристики главного героя;
- `cameraHandler: CameraHandler` – ссылка на скрипт, отвечающий за управление режимами работы камеры;

- `gamerWeapons: List<WeaponItem>` – список, хранящий оружие главного героя (игрок может выбрать только одно из оружий в начале игры);
- `isInteracting: bool` – флаг, показывающий занят ли главный герой каким либо действием;
- `isSprinting: bool` – флаг, показывающий бежит ли главный герой;
- `isInAir: bool` – флаг, показывающий находится ли главный герой в воздухе;
- `isGrounded` – флаг, показывающий находится ли главный герой на земле;
- `canDoCombo` – флаг, показывающий может ли главный герой в данный момент продолжить комбо ударов;
- `isInvulnerable` – флаг, показывающий неуязвим ли главный герой в данный момент;
- `isBlocking: bool` – флаг, показывающий находится ли главный герой в состоянии блока ударов;
- `takingDamage: bool` – флаг, показывающий получает ли главный герой урон в данный момент;
- `damageReduce: int` – процент уменьшения урона в состоянии блока;
- `saveData: List<string>` – список данных из файла сохранения.

Функции класса:

- `Start()` – метод, отвечающий за инициализацию полей данного класса в момент его появления, а также загружающий сведения о главном герое из файла сохранения (например, характеристики героя подстраиваются под уровень сложности);
- `Update()` – метод, отвечающий за выполнение действий во время каждого такта работы игры;
- `FixedUpdate()` – метод схожий по смыслу с `Update`, но если `Update` срабатывает один раз за один кадр игры, то `FixedUpdate` может не сработать ни разу, либо сразу несколько раз в течение одного кадра игры;

– LateUpdate() – метод схожий по смыслу с другими Update – методами, но в данном случае он вызывается один раз за один кадр игры после всех остальных видов Update-методов;

– ReadSaveData() – метод для считывания данных из файла сохранения.

Таким образом с помощью диаграмм классов и описания классов была спроектирована структура приложения.

3.3 Проектирование алгоритмов

На рисунках ниже показаны примеры некоторых алгоритмов разрабатываемой видеоигры. На рисунке 28 представлена блок схема для алгоритма захвата врага в фокус камеры.

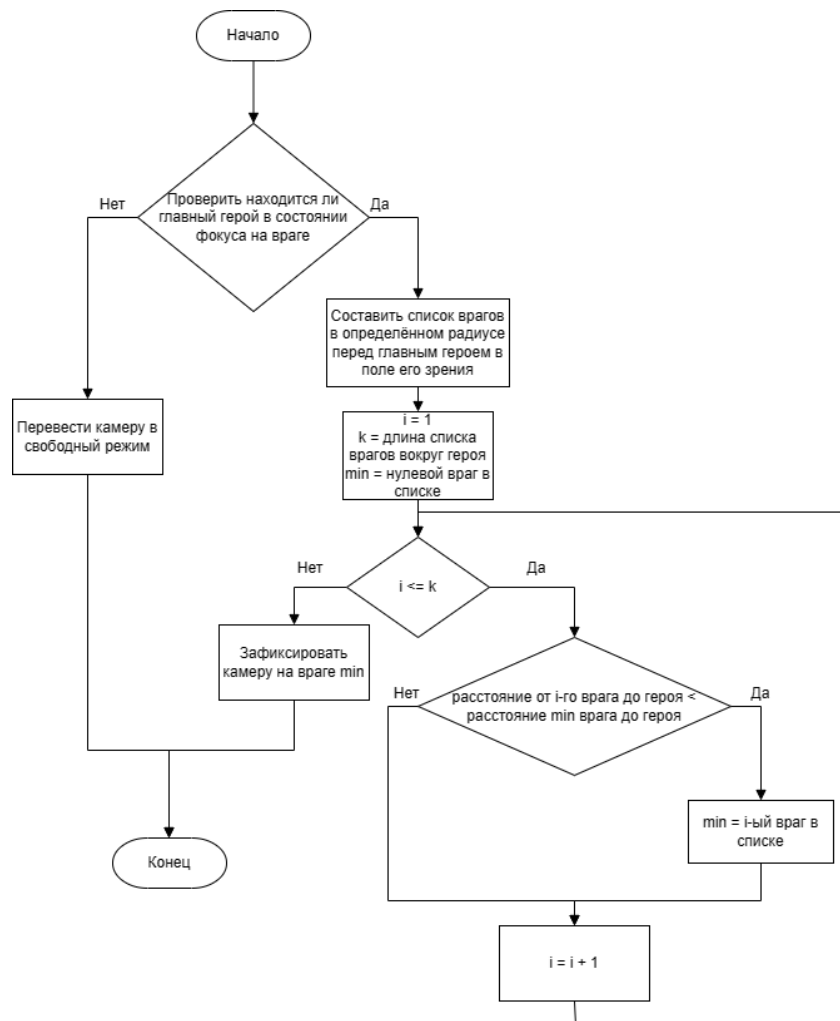


Рисунок 28 – Блок схема для алгоритма захвата врага в фокус камеры

На рисунке 29 представлена диаграмма последовательности для алгоритма совершения лёгкой атаки.

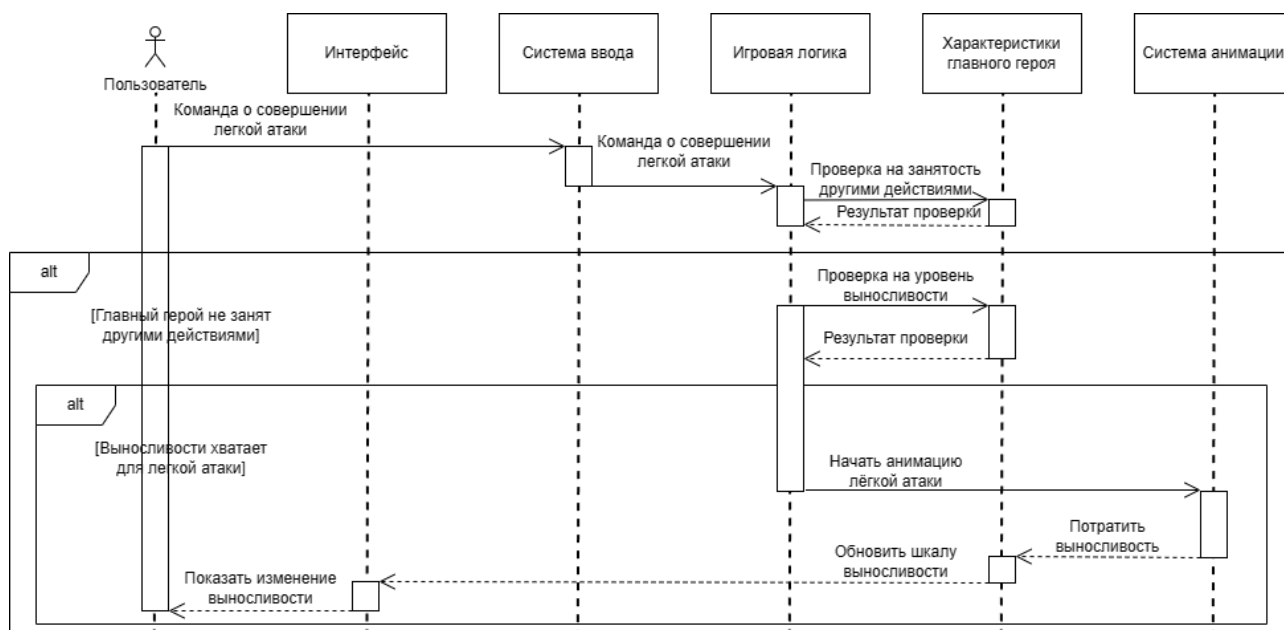


Рисунок 29 – Диаграмма последовательности для алгоритма совершения легкой атаки

Поведение противников и босса регулируется машинами состояний. Для босса и обычных противников эти машины состояний отличаются незначительно. Поведение босса характеризуется следующими состояниями:

- состояние покоя;
- состояние преследования;
- состояние атаки;
- состояние боевой стойки.

Поведение босса отличается от поведения обычных противников тем, что босс переходит во вторую фазу после того, как потеряет половину своих очков здоровья. Во второй фазе список обычных атак босса меняется, а ещё он может время от времени совершать специальную атаку (рывок в сторону главного героя с последующим ударом).

Для отображения данных состояний и переходов была построена схема состояний и переходов для обычного противника (рисунок 30).

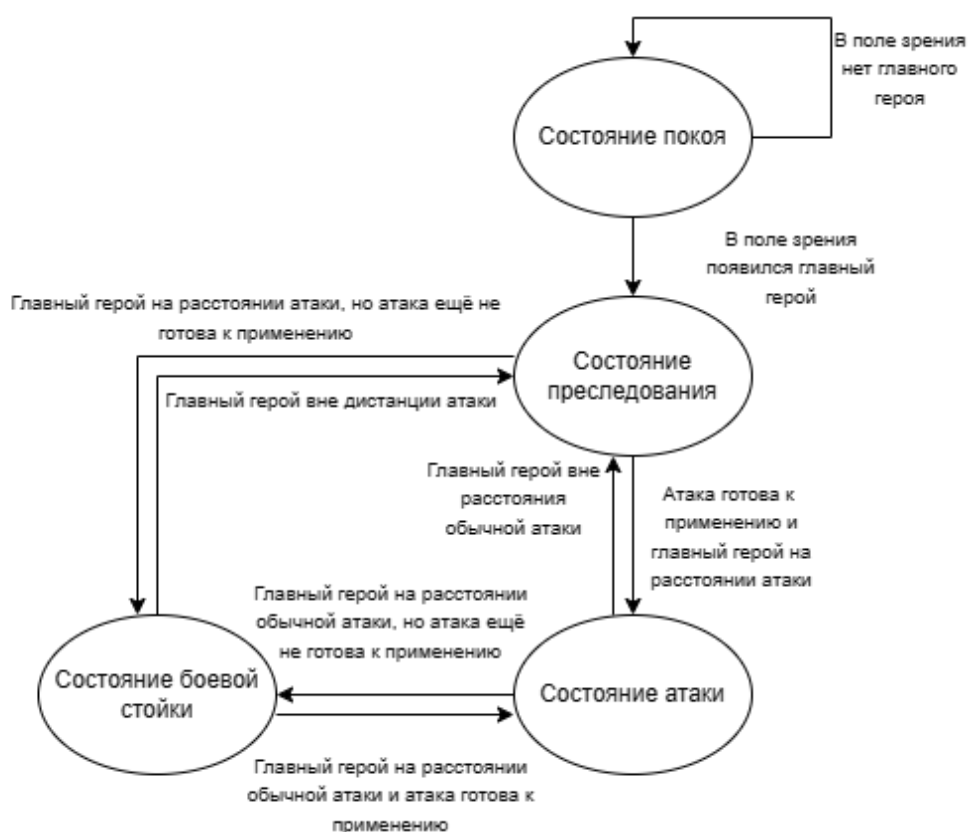


Рисунок 30 – Схема состояний и переходов для противника

В первой фазе машина состояний босса идентична машине состояний обычных противников. Схема состояний и переходов для второй фазы битвы с боссом показана на рисунке 31.



Рисунок 31 – Схема состояний и переходов для босса (вторая фаза боя)

На рисунках 32, 33 показаны блок схемы, демонстрирующие порядок действий в рамках состояния покоя и состояния преследования босса.



Рисунок 32 – Блок схема для состояния покоя

В состоянии покоя босс ожидает главного героя до тех пор, пока не увидит его.

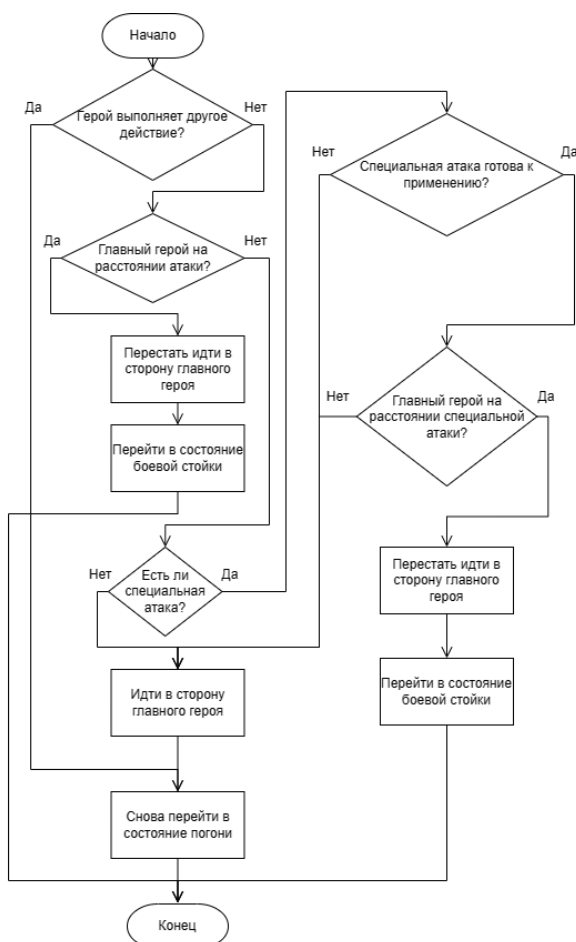


Рисунок 33 – Блок схема для состояния преследования

В рамках состояния преследования босс приближается к главному герою и переходит в состояние боевой стойки в случаях, когда возможно совершить специальную атаку или обычную атаку.

На рисунке 34 показана блок схема, описывающая состояние боевой стойки босса.

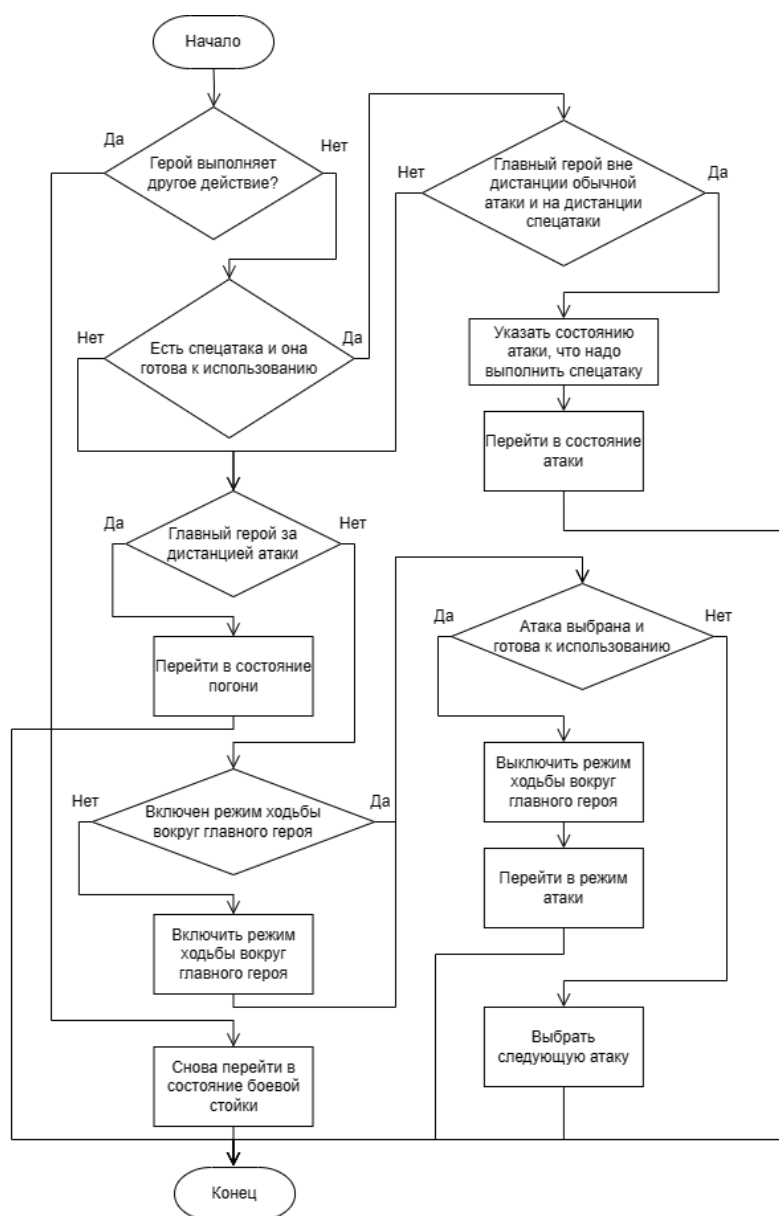


Рисунок 34 – Блок схема для состояния боевой стойки

В рамках состояния боевой стойки босс может либо перейти в состояние преследования, либо в состояние атаки. Переход в состояние преследования происходит в случае, когда дистанция до героя больше дистанции атаки и спецатака не готова к использованию (также переход возможен в ситуации, когда

расстояние до главного героя больше дистанции спецатаки). Переход в состояние атаки происходит, когда главный герой в зоне досягаемости обычной атаки или главный герой в зоне досягаемости спецатаки (при условии, что спецатака готова к использованию). Также в случае, когда ни одна из атак не доступна и босс находится на расстоянии обычной атаки от главного героя, босс может оставаться в состоянии боевой стойки и кружить вокруг главного героя в случайном направлении.

На рисунке 35 показана блок схема, описывающая состояние атаки босса.

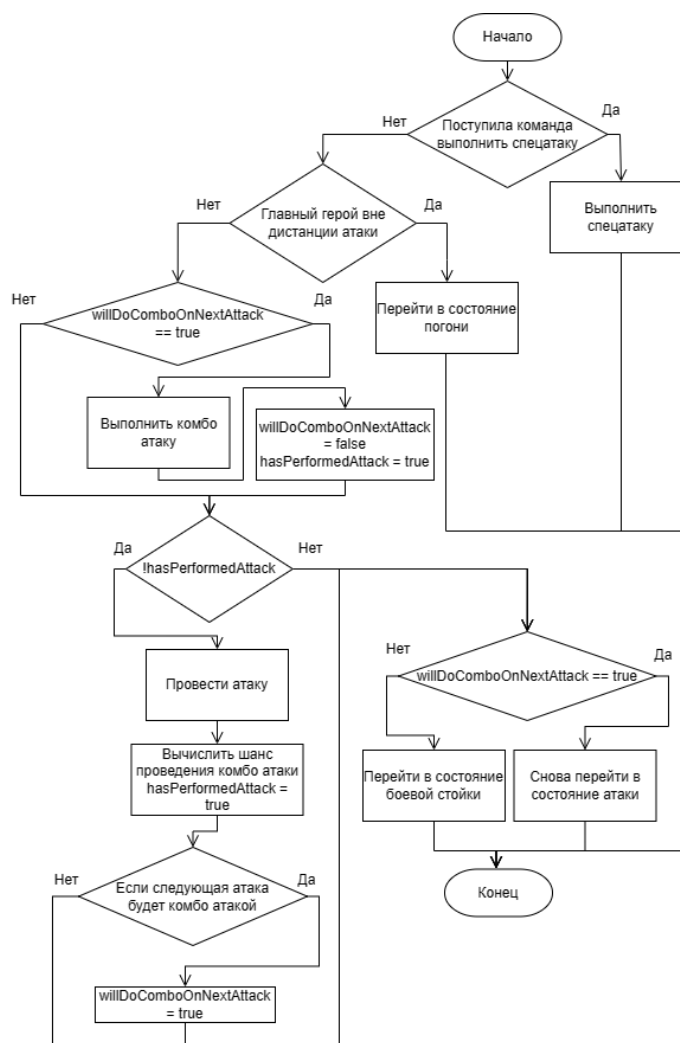


Рисунок 35 – Блок схема для состояния атаки

В рамках состояния атаки босс может совершить спецатаку или обычную атаку, а также из может произойти переход в состояние боевой стойки или состояние преследования. Переход в состояние боевой стойки происходит, когда

атака совершена и расстояние до главного героя не больше расстояния обычной атаки. Переход в состояние преследования происходит, когда расстояние до главного героя больше расстояния обычной атаки и не совершается спецатака.

3.4 Проектирование структур данных

Для сохранения прогресса игрока в видеоигре используется текстовый файл. На рисунке 37 показан пример файла сохранения.

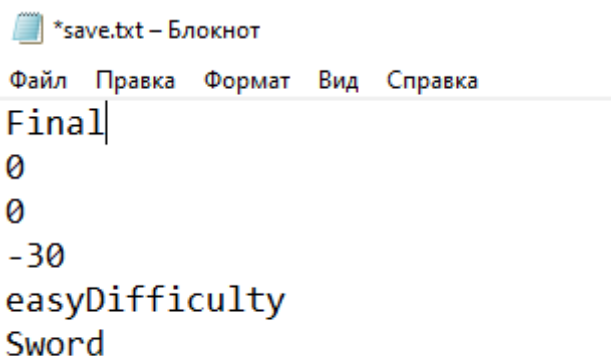


Рисунок 37 – Файл сохранения

Теперь опишем содержимое файла сохранения, показанного на рисунке 37. В первой строке файла хранится название текущего уровня игры. Во второй – четвёртой строках записаны координаты игрока на уровне. В пятой строке хранится информация о сложности игры. В шестой строке хранится название оружия главного героя, которое выбрал игрок.

Все остальные данные в игре хранятся в динамически созданных объектах. Когда пользователь запускает игру динамические объекты генерируются с учётом данных в файле сохранения (например, здоровье и урон врагов будут заданы с учётом сложности игры, записанной в файле сохранения). Когда пользователь закрывает игру, то динамические объекты удаляются вместе с данными, которые в них хранились.

4 Реализация приложения

4.1 Кодирование приложения

Видеоигра была реализована на игровом движке Unity с использованием языка программирования C#. Примеры кода видеоигры представлены в приложении А.

Для отображения основных компонентов и связей между ними была составлена диаграмма компонентов, показанная на рисунке 38.

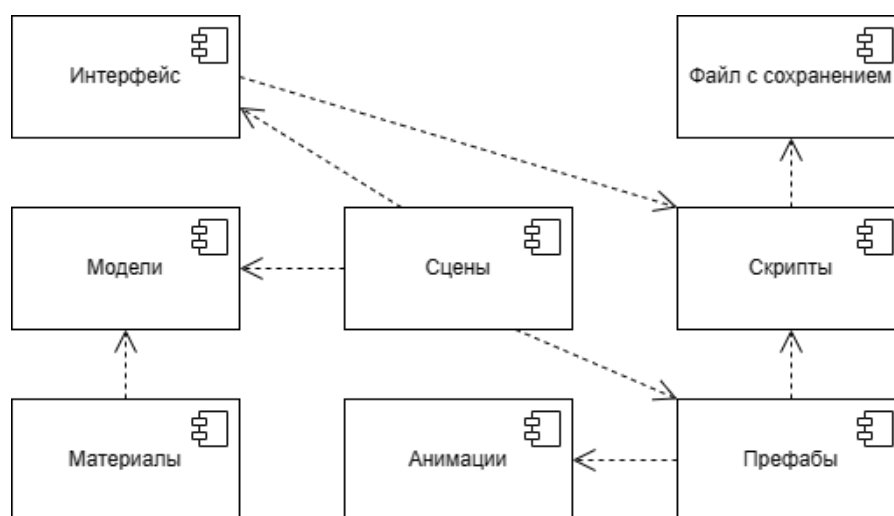


Рисунок 38 – Диаграмма компонентов

Основными компонентами видеоигры являются сцены. Вся игра является последовательностью сцен, которые в том или ином виде включают в себя все остальные компоненты игры (анимации, скрипты, префабы, элементы интерфейса и т. д.).

Описание показанных на рисунке 38 компонентов содержится в таблице 2.

Таблица 2 – Описание компонентов видеоигры

Комонент	Назначение
Сцены	Данный компонент включает в себя все сцены видеоигры. Сцена – это уровень игры со всеми нужными объектами (к таким объектам можно отнести элементы интерфейса, главного героя, игровых противников и т. д.).

Комонент	Назначение
Модели	Компонент включает в себя все трёхмерные модели игры. Трёхмерная модель состоит из полигональной сетки (меша) и текстуры.
Скрипты	Компонент включает в себя все скрипты видеоигры. Скрипт – это код на языке C#.
Анимации	Компонент включает в себя все анимации, использующиеся в игре.
Материалы	Компонент включает в себя все материалы видеоигры. Материал – информация о зрительных свойствах модели (прозрачность, блики, цвет, текстура и т. д.).
Префабы	Компонент включает в себя все префабы видеоигры. Префаб – это шаблонный игровой объект, который может многократно использоваться в игре. К префабам можно отнести, например, рядовых противников и двери.
Интерфейс	Компонент включает в себя все элементы интерфейса видеоигры. К компонентам интерфейса можно отнести текстовые поля, кнопки, шкалы здоровья и выносливости и т. д.
Файл с сохранением	Текстовый файл, содержащий данные о последнем сохранении пользователя. Сохранения делаются автоматически в моменты достижения контрольных точек видеоигры.

Для разработанной видеоигры также была составлена диаграмма развертывания, представленная на рисунке 39.

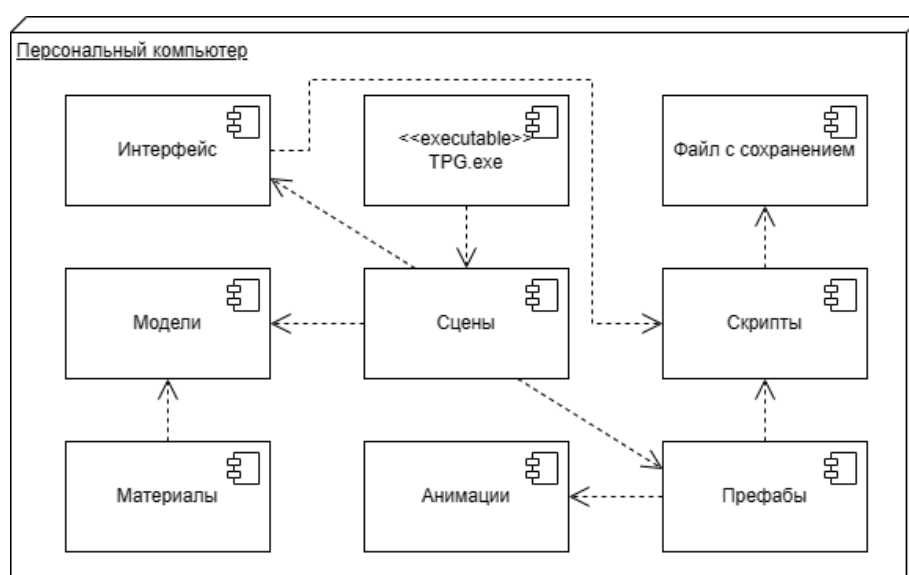
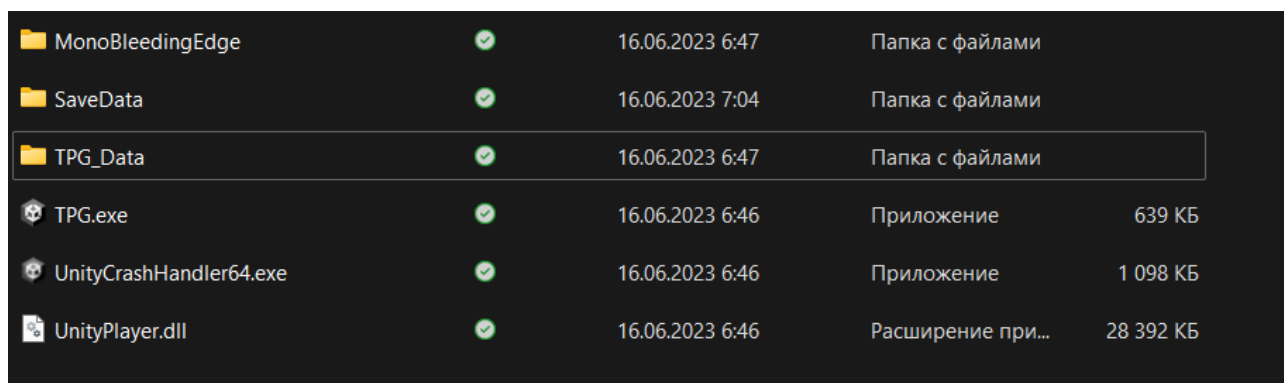


Рисунок 39 – Диаграмма развертывания

На рисунке 40 показана папка с игрой, полученная в результате сборки игры на движке Unity.



Folder	MonoBleedingEdge	✓	16.06.2023 6:47	Папка с файлами	
Folder	SaveData	✓	16.06.2023 7:04	Папка с файлами	
Folder	TPG_Data	✓	16.06.2023 6:47	Папка с файлами	
File	TPG.exe	✓	16.06.2023 6:46	Приложение	639 КБ
File	UnityCrashHandler64.exe	✓	16.06.2023 6:46	Приложение	1 098 КБ
File	UnityPlayer.dll	✓	16.06.2023 6:46	Расширение при...	28 392 КБ

Рисунок 40 – Скриншот папки с игрой

Так как видеоигра была реализована на движке Unity, то её сборка производилась автоматически. Полученная в результате сборки папка с видеоигрой имеет трудную в плане подробного описания структуру, поэтому диаграмма развертывания (смотри рисунок 39) более абстрактно описывает структуру, установленной видеоигры.

5 Планирование разработки и оценка бюджета

5.1 Календарное и ресурсное планирование

Задачи проекта приведены на рисунке 41.

Название задачи	Длительность	Предшественники
Начало проекта	0 дней	
▢ Анализ требований	6 дней	
Анализ приложений аналогов	4 дней	1
Анализ функциональных требований	2 дней	3
Анализ нефункциональных требований	1 день	3
Завершение анализа требований	0 дней	4;5
▢ Выбор средств реализации проекта	3 дней	
Составление списка возможных средств	1 день	6
Анализ средств из списка	2 дней	8
Выбор средств реализации проекта завер	0 дней	9
▢ Проектирование приложения	7 дней	
Проектирование интерфейсов приложения	2,5 дней	10
Проектирование классов приложения	7 дней	10
Завершение проектирования приложения	0 дней	12;13
▢ Реализация приложения	31 дней	
Создание игровых ассетов	25 дней	14
Реализация интерфейсов видеоигры	7 дней	14
Кодирование игровой логики	6 дней	16;17
Завершение реализации приложения	0 дней	18
▢ Тестирование и отладка приложения	7 дней	
Тестирование системы сохранений	1 день	19
Тестирование управления главным героем	3 дней	19
Тестирование игровых врагов	3 дней	19
Исправление ошибок	4 дней	21;22;23
Завершение тестирования и отладки	0 дней	24
▢ Составление документации	4,82 дней	
Составление документации для разработч	4 дней	25
Составление руководства пользователя	0,82 дней	27
Конец проекта	0 дней	28

Рисунок 41 – Задачи проекта

На рисунке 42 представлен лист ресурсов проекта.

Название ресурса	Тип	Единицы измерения материалов	Стандартная ставка	Ставка сверхурочных
Руководитель проекта	Трудовой		70 000,00 Р/мес	437,50 Р/ч
Программист 1	Трудовой		50 000,00 Р/мес	312,50 Р/ч
Программист 2	Трудовой		50 000,00 Р/мес	312,50 Р/ч
Художник	Трудовой		45 000,00 Р/мес	281,25 Р/ч
Аниматор	Трудовой		45 000,00 Р/мес	281,25 Р/ч
Тестировщик 1	Трудовой		30 000,00 Р/мес	187,50 Р/ч
Тестировщик 2	Трудовой		30 000,00 Р/мес	187,50 Р/ч
Тестировщик 3	Трудовой		30 000,00 Р/мес	187,50 Р/ч
Писатель	Трудовой		45 000,00 Р/мес	281,25 Р/ч
Доступ к сети Интернет	Затраты			
Компьютер	Материальный	штука	50 000,00 Р	
Принтер	Материальный	штука	10 000,00 Р	
Бумага	Материальный	штука	1 000,00 Р	

Рисунок 42 – Лист ресурсов проекта

На рисунке 43 представлена диаграмма Ганта, составленная для данного проекта.

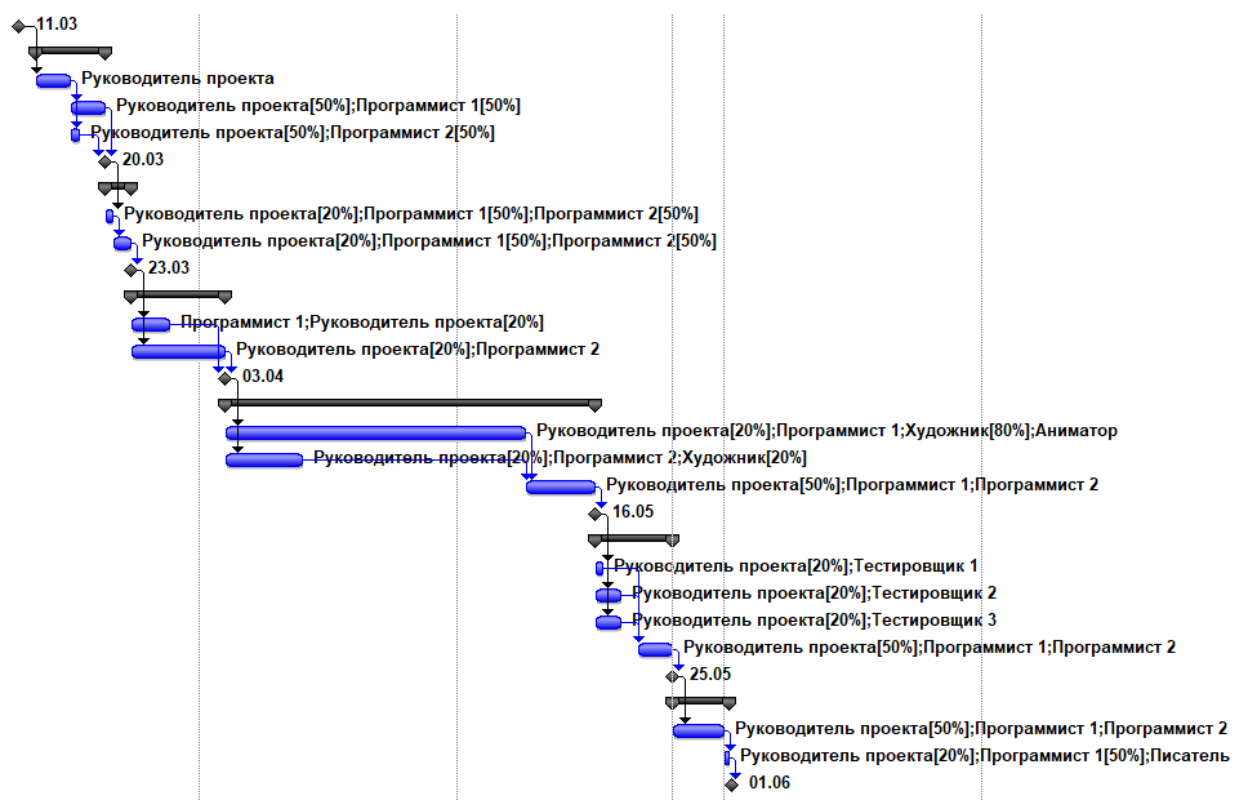


Рисунок 43 – Диаграмма Ганта

По расчётам, полученным в программе Microsoft Project, продолжительность проекта составит 59 дней (без учёта выходных дней) и зарплатный фонд сотрудников будет составлять 355 908,82 рубля.

Полный бюджет проекта можно посчитать по формуле:

$$Б = 3Ф + 4К + 1П + 1Б + 2И + Е, \quad (1)$$

где Б – бюджет проекта (руб.);

3Ф – зарплатный фонд (руб.);

К – стоимость компьютера (руб.);

П – стоимость принтера (руб.);

Б – стоимость пачки бумаги (руб.);

И – месячная плата за доступ к сети Интернет (руб.);

Е – затраты на электроэнергию (руб.).

В нашем случае пример месячную плату за Интернет равной 500 рублям. А для расчёта затрат на электроэнергию, потребуется использовать формулу:

$$E = C * (4 * ПФВ1 * ПМ1 + ПФВ2 * ПМ2), \quad (2)$$

где C – стоимость 1 киловатт-часа электроэнергии (руб.);

ПФВ1 – полезный фонд работы одного компьютера (часы);

ПМ1 – потребляемая мощность одного компьютера (кВт);

ПФВ2 – полезный фонд работы принтера (часы);

ПМ2 – потребляемая мощность принтера (кВт).

Примем потребляемую мощность компьютера за 0,25 кВт. Полезный фонд времени работы компьютера примем за 472 часа (59 дней по 8 часов). Потребляемую мощность принтера будем считать равной 0,31 кВт, а полезный фонд времени работы принтера посчитаем равным 4,91 часа (59 дней по 5 минут). Стоимость 1 кВт*ч энергии примем равной 2,8 рубля.

Расчёт затрат на электроэнергию по формуле (2) будет в итоге выглядеть следующим образом:

$$E = 2,8 * (4 * 472 * 0,25 + 4,91 * 0,31) = 1325,9.$$

Теперь может быть осуществлён расчёт бюджета проекта по формуле (1), который будет выглядеть так:

$$B = 355\,908,82 + 4 * 50\,000 + 10\,000 + 1\,000 + \\ + 2 * 500 + 1325,9 = 569\,234,72.$$

Итоговый бюджет проекта равен пятиста шестидесяти девяти тысячам двести тридцати четырём рублям и семидесяти двум копейкам. Таким образом был составлен план выполнения работ проекта и были рассчитаны затраты на реализацию проекта.

6 Тестирование

6.1 Выбор способа тестирования приложения

Для разработанного игрового проекта будет проведено тестирование методом «чёрного ящика», так как оно в достаточной мере позволяет протестировать разработанного приложение и не требует больших затрат для реализации.

При тестировании «чёрного ящика» программа рассматривается как объект, внутренняя структура которого неизвестна. Тестирущик вводит данные и анализирует результат, но, как именно работает программа, он не знает [20].

Тестирование методом «чёрного ящика» удобно использовать в данном случае благодаря тому, что данный подход позволяет сложную и большую программу рассматривать, как абстрактный «чёрный ящик», который на выходе выдаёт результат, соответствующий поступившим на входе действиям пользователя.

В рамках выбранного вида тестирования для разработанного приложения будут составлены классы эквивалентности в табличном и графическом видах. В конечном итоге будут составлены таблицы с примерами тестов для разработанного ПО.

6.2 Тестирование приложения методом «чёрного ящика»

В разработанном приложении есть два основных объекта: главный герой и противник. В таблице 3 представлены классы эквивалентности (далее они будут называться КЭ) для главного героя и противника.

Таблица 3 – Табличное представление классов эквивалентности для главного героя и противника

	Правильные КЭ	Неправильные КЭ
Клавиша переката	Shift (1)	Все остальные клавиши (2)

	Правильные КЭ	Неправильные КЭ
Уровень выносливости достаточный для переката	20 – 100 (3)	Все значения от нуля до 20, не включая 20 (4)
Персонаж уже выполняет другое действие (значение флага isInteracting)	False (5)	True (6)
Персонаж на земле (значение флага isGrounded)	True (7)	False (8)
Персонаж оправляется после получения урона (проигрывается анимация Damage_01)	Анимация не проигрывается (9)	Анимация проигрывается (10)
Персонаж не двигается и стоит на месте (значение флага MoveAmount > 0)	Персонаж движется (11)	Персонаж стоит на месте (12)
Клавиши передвижения	w,a,s,d (13)	Все остальные клавиши (14)
Клавиша удара	ЛКМ (15)	Все остальные клавиши (16)
Уровень выносливости достаточный для удара	20 – 100 (17)	Все значения от нуля до 20, не включая 20 (18)
Обнаружение цели (главного героя)	В поле зрения врага попал главный герой (19)	В поле зрения врага нет главного героя(20)
Цель	Есть (21)	Нет (22)
Расстояние до цели	Меньше или равно дальности атаки (23)	Больше дальности атаки (24)
Время восстановления после атаки	Равно нулю (25)	Больше нуля (26)
Состояние покоя	Да (27)	Нет (28)
Состояние преследования	Да (29)	Нет (30)
Состояние атаки	Да (31)	Нет (32)
Состояние боевой стойки	Да (33)	Нет (34)
Наличие противников в поле зрения главного героя	Да (35)	Нет (36)
Клавиша блока	ПКМ (37)	Все остальные клавиши (38)

Окончание таблицы 3

	Правильные КЭ	Неправильные КЭ
Клавиша захвата противника в фокус камеры	F (39)	Все остальные клавиши (40)

На рисунке 44 показаны КЭ для клавиши переката в графическом виде. Белым цветом выделен правильный КЭ, а синим выделен неправильный (так будет и на остальных подобных рисунках в данной работе).



Рисунок 44 – Клавиша переката

Правильным КЭ для клавиши переката является клавиша Shift, все остальные клавиши относятся к неправильным КЭ.

На рисунке 45 показаны КЭ для уровня выносливости, достаточного для переката или удара, в графическом виде.

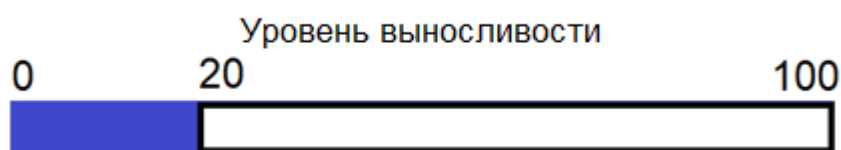


Рисунок 45 – Уровень выносливости достаточный для переката или удара

Правильными КЭ для уровня выносливости достаточного для переката или удара являются значения выносливости от 20 до 100 единиц, все значения выносливости, которые меньше 20 единиц, относятся к неправильным КЭ.

На рисунке 46 показаны КЭ для ситуации, когда персонаж уже занят другими действиями, в графическом виде.

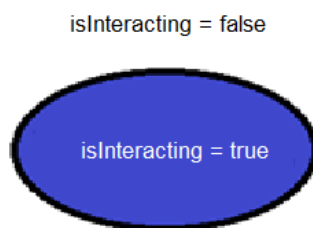


Рисунок 46 – Персонаж уже выполняет другое действие (значение флага isInteracting)

Для значения флага isInteracting правильным КЭ является ложное значение (false), а неправильным КЭ является истинное значение (true).

На рисунке 47 показаны КЭ для ситуации, когда персонаж находится на земле, в графическом виде.

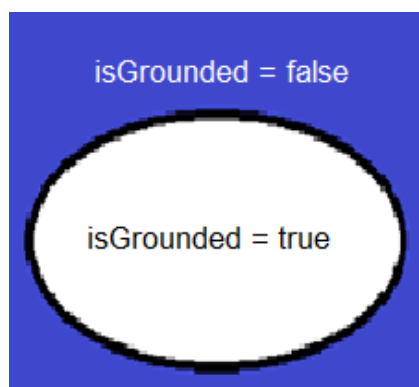


Рисунок 47 – Персонаж на земле (значение флага isGrounded)

Для значения флага isGrounded правильным КЭ является истинное значение (true), а неправильным КЭ является ложное значение (false).

На рисунке 48 показаны КЭ для ситуации, когда персонаж оправляется после получения урона, в графическом виде.

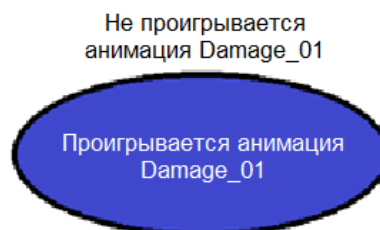


Рисунок 48 – Персонаж оправляется после получения урона (проигрывается анимация Damage_01)

Правильным КЭ является ситуация, когда главный герой (персонаж) не оправляется после получения урона, а неправильным КЭ является ситуация, когда главный герой оправляется после получения урона.

На рисунке 49 показаны КЭ для ситуации, когда персонаж не двигается, графическом виде.

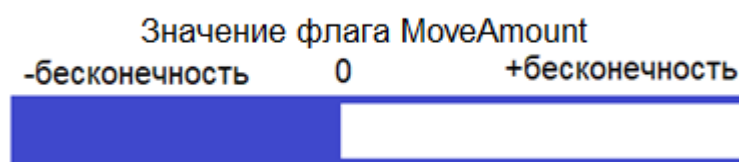


Рисунок 49 – Персонаж не двигается и стоит на месте (значение флага $\text{MoveAmount} > 0$)

Правильным КЭ является ситуация, когда главный герой движется, а неправильным КЭ является ситуация, когда главный герой стоит на месте.

На рисунке 50 показаны КЭ для клавиш передвижения в графическом виде.



Рисунок 50 – Клавиши передвижения

Для клавиш передвижения правильными КЭ являются клавиши w, a, s, d. Любые другие клавиши относятся к неправильным КЭ.

На рисунке 51 показаны КЭ для клавиши удара в графическом виде.



Рисунок 51 – Клавиша удара

Для клавиши удара правильным КЭ является левая кнопка мыши, а все остальные клавиши относятся к неправильным КЭ.

На рисунке 52 показаны КЭ для клавиши блока в графическом виде.



Рисунок 52 – Клавиша блока

Для клавиши блока правильным КЭ является правая кнопка мыши, а все остальные клавиши относятся к неправильным КЭ.

На рисунке 53 показаны КЭ для обнаружения игрока противником в графическом виде.

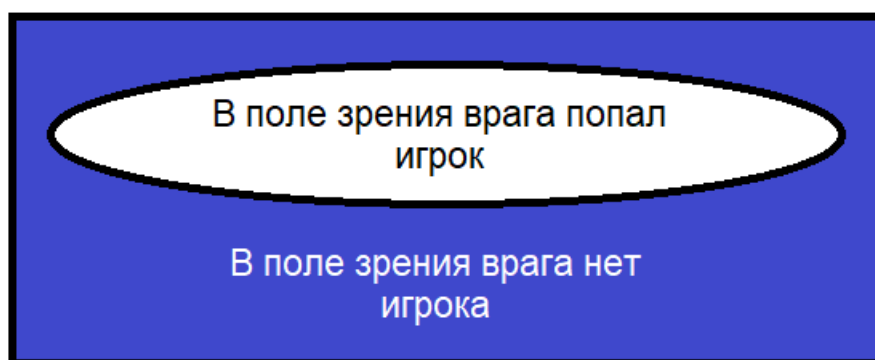


Рисунок 53 – Обнаружение цели (игрока)

Для обнаружения цели (игрока) противником правильным КЭ является ситуация, когда в поле зрения попал игрок, а неправильным КЭ является ситуация, когда в поле зрения нет игрока.

На рисунке 54 показаны КЭ для наличия текущей цели у противника в графическом виде.

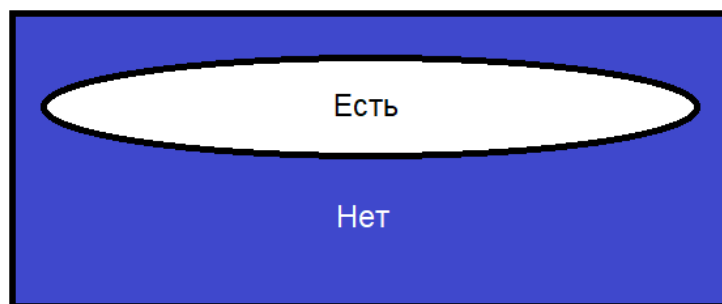


Рисунок 54 – Наличие текущей цели

Для текущей цели противника правильным КЭ является ситуация, когда текущая цель есть, а неправильным КЭ является ситуация, когда текущей цели нет.

На рисунке 55 показаны КЭ для расстояния до цели противника в графическом виде.

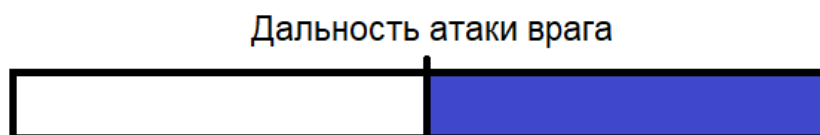


Рисунок 55 – Расстояние до цели

В случае расстояния до цели правильным КЭ является расстояние не больше дальности атаки, а неправильным КЭ является расстояние больше дальности атаки.

На рисунке 56 показаны КЭ для времени восстановления после атаки противника в графическом виде.

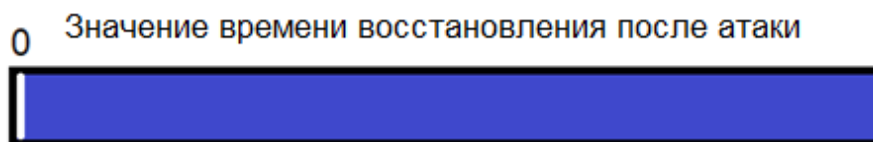


Рисунок 56 – Время восстановления после атаки

В случае времени восстановления после атаки правильным КЭ является время равное нулю, а к неправильному КЭ относится любое время больше нуля.

На рисунке 57 показаны КЭ для состояний покоя, преследования, атаки, боевой стойки в графическом виде.

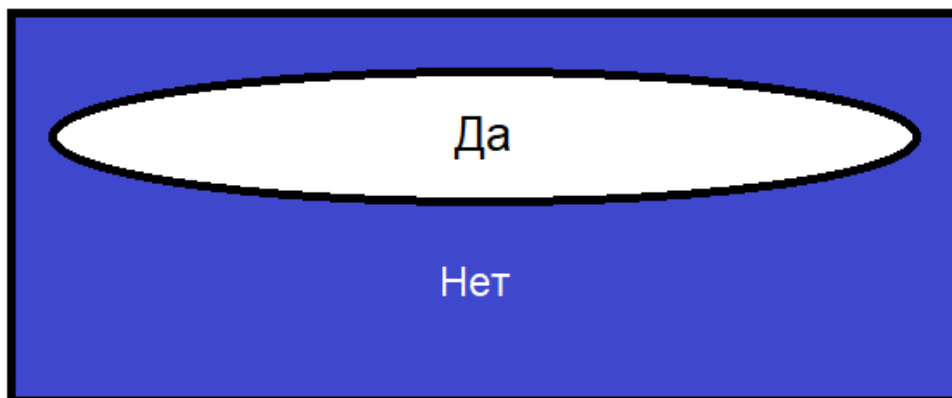


Рисунок 57 – Состояния покоя, преследования, атаки, боевой стойки

В случае состояния покоя к правильному КЭ будет относиться ситуация, когда противник находится в состоянии покоя, а к неправильному КЭ будет относиться ситуация, когда противник находится в любом другом состоянии. Аналогичным образом можно описать КЭ для состояний преследования, атаки и боевой стойки.

В таблице 4 показаны примеры тестов для главного героя и противника.

Таблица 4 – Тесты для главного героя и противника

№	КЭ	Входные данные	Выходные данные
1	Все правильные 1,3,5,7,9,11	100 – выносливости, нажатие клавиши Shift, isInteracting = false, isGrounded = true, Анимация Damage_01 не проигрывается, Персонаж движется.	Персонаж совершает пережат (см. рисунок Б.1)
2	2,3,5,7,9,11	100 – выносливости, нажатие клавиши E, isInteracting = false, isGrounded = true, Анимация Damage_01 не проигрывается, Персонаж движется.	Персонаж не делает пережат из за нажатия неверной клавиши действия (см. рисунок Б.2)
3	1,4,5,7,9,11	10 – выносливости, нажатие клавиши Shift, isInteracting = false, isGrounded = true, Анимация Damage_01 не проигрывается, Персонаж движется.	Персонаж не делает пережат из за недостаточного количества выносливости (см. рисунок Б.3)

№	КЭ	Входные данные	Выходные данные
4	1,3,6,7,9,12	100 – выносливости, нажатие клавиши Shift, isInteracting = true, isGrounded = true, Анимация Damage_01 не проигрывается, Персонаж не двигается.	Персонаж не делает перекат из-за того, что он делает другое действие в данный момент (см. рисунок Б.4)
5	1,3,5,8,9,12	100 – выносливости, нажатие клавиши Shift, isInteracting = false, isGrounded = false, Анимация Damage_01 не проигрывается, Персонаж не двигается.	Персонаж не делает перекат из-за того, что он не на земле (см. рисунок Б.5)
6	1,3,5,7,10,12	100 – выносливости, нажатие клавиши Shift, isInteracting = false, isGrounded = true, Анимация Damage_01 проигрывается, Персонаж не двигается.	Персонаж не делает перекат из-за того, что он оправляется после получения урона (см. рисунок Б.6)
7	1,3,5,7,9,12	100 – выносливости, нажатие клавиши Shift, isInteracting = false, isGrounded = true, Анимация Damage_01 не проигрывается, Персонаж не двигается.	Персонаж не совершает перекат, потому что не движется (см. рисунок Б.7)
8	13,5,7,9	Нажатие клавиши w, isInteracting = false, isGrounded = true, Анимация Damage_01 не проигрывается.	Персонаж движется (см. рисунок Б.8)
9	14,5,7,9	Нажатие клавиши m, isInteracting = false, isGrounded = true, Анимация Damage_01 не проигрывается.	Персонаж не движется из-за того, что нажата неверная клавиша действия (см. рисунок Б.9)
10	13,6,7,9	Нажатие клавиши w, isInteracting = true, isGrounded = true, Анимация Damage_01 не проигрывается.	Персонаж не движется из-за того, что он занят другими действиями (см. рисунок Б.10)
11	13,5,8,9	Нажатие клавиши w, isInteracting = false, isGrounded = false, Анимация Damage_01 не проигрывается.	Персонаж не движется из-за того, что он не на земле (см. рисунок Б.11)
12	13,5,7,10	Нажатие клавиши w, isInteracting = false, isGrounded = true, Анимация Damage_01 проигрывается.	Персонаж не движется из-за того, что он оправляется после получения урона (см. рисунок Б.12)
13	15,5,7,9,17	Нажатие клавиши ЛКМ, isInteracting = false, isGrounded = true, Анимация Damage_01 не проигрывается, 100 – выносливости.	Персонаж делает легкий удар (см. рисунок Б.13)

№	КЭ	Входные данные	Выходные данные
14	16,5,7,9,17	Нажатие клавиши g, isInteracting = false, isGrounded = true, Анимация Damage_01 не проигрывается, 100 – выносливости.	Персонаж не делает удар из за того, что нажата не верная клавиша действия (см. рисунок Б.14)
15	15,6,7,9,17	Нажатие клавиши ЛКМ, isInteracting = true, isGrounded = true, Анимация Damage_01 не проигрывается, 100 – выносливости.	Персонаж не делает удар из за того, что персонаж занят другими действиями (см. рисунок Б.15)
16	15,5,8,9,17	Нажатие клавиши ЛКМ, isInteracting = false, isGrounded = false, Анимация Damage_01 не проигрывается, 100 – выносливости.	Персонаж не делает удар из за того, что персонаж не на земле (см. рисунок Б.16)
17	15,5,7,10,17	Нажатие клавиши ЛКМ, isInteracting = false, isGrounded = true, Анимация Damage_01 проигрывается, 100 – выносливости.	Персонаж не делает удар из за того, что оправляется после получения урона (см. рисунок Б.17)
18	15,5,7,9,18	Нажатие клавиши ЛКМ, isInteracting = false, isGrounded = true, Анимация Damage_01 не проигрывается, 10 – выносливости.	Персонаж не делает удар из за того, что персонажу не хватает выносливости (см. рисунок Б.18)
19	27,30,32,34,19,22	Состояние покоя – да, Состояние преследования – нет, Состояние атаки – нет, Состояние боевой стойки – нет, В поле зрения попал главный герой, Цель – нет	Враг переходит из состояния покоя в состояние преследования, а также в качестве цели устанавливается игрок (см. рисунок Б.19)
20	27,30,32,34,20,22	Состояние покоя – да, Состояние преследования – нет, Состояние атаки – нет, Состояние боевой стойки – нет, В поле зрения нет главного героя, Цель – нет	Противник остаётся в состоянии покоя (см. рисунок Б.20)
21	28,29,32,34,21,24	Состояние покоя – нет, Состояние преследования – да, Состояние атаки – нет, Состояние боевой стойки – нет, Цель есть, Расстояние до цели больше дальности атаки	Противник преследует главного героя (см. рисунок Б.21)
22	28,29,32,34,21,23,25	Состояние покоя – нет, Состояние преследования – да, Состояние атаки – нет, Состояние боевой стойки – нет, Цель есть, Расстояние до цели меньше дальности атаки Время восстановления после атаки равно нулю	Противник заканчивает преследование главного героя и переходит в состояние атаки (см. рисунок Б.22)

№	КЭ	Входные данные	Выходные данные
23	28,29,32,34,21,23,26	Состояние покоя – нет, Состояние преследования – да, Состояние атаки – нет, Состояние боевой стойки – нет, Цель есть, Расстояние до цели меньше дальности атаки Время восстановления после атаки больше нуля	Противник заканчивает преследование главного героя и переходит в состояние боевой стойки (см. рисунок Б.23)
24	28,30,31,34,21,23,26	Состояние покоя – нет, Состояние преследования – нет, Состояние атаки – да, Состояние боевой стойки – нет, Цель есть, Расстояние до цели меньше дальности атаки, Время восстановления после атаки больше нуля.	Противник переходит из состояния атаки в состояние боевой стойки (см. рисунок Б.24)
25	28,30,31,34,21,24,26	Состояние покоя – нет, Состояние преследования – нет, Состояние атаки – да, Состояние боевой стойки – нет, Цель есть, Расстояние до цели больше дальности атаки, Время восстановления после атаки больше нуля.	Противник переходит из состояния атаки в состояние преследования (см. рисунок Б.25)
26	28,30,32,33,21,24	Состояние покоя – нет, Состояние преследования – нет, Состояние атаки – нет, Состояние боевой стойки – да, Цель – есть, Расстояние до цели – больше дальности атаки.	Противник переходит из состояния боевой стойки в состояние преследования (см. рисунок Б.26)
27	28,30, 32,33,21,23,25	Состояние покоя – нет, Состояние преследования – нет, Состояние атаки – нет, Состояние боевой стойки – да, Цель – есть, Расстояние до цели – меньше или равно дальности атаки, Время восстановления после атаки – равно нулю	Противник переходит из состояния боевой стойки в состояние атаки (см. рисунок Б.27)
28	5, 7, 9, 37	isInteracting = false, isGrounded = true, Анимация Damage_01 не проигрывается, Нажатие клавиши ПКМ	Главный герой переходит в состояние блока (см. рисунок Б.28)
29	5, 7, 9, 38	isInteracting = false, isGrounded = true, Анимация Damage_01 не проигрывается, Нажатие клавиши H	Главный герой ничего не делает (см. рисунок Б.29)
30	5,7,9, 35, 39	isInteracting = false, isGrounded = true, Анимация Damage_01 не проигрывается, В поле зрения главного героя есть противники, Нажатие клавиши F	Ближайший противник к главному герою будет захвачен в фокус камеры (см. рисунок Б.30)

№	КЭ	Входные данные	Выходные данные
31	5,7,9, 36, 39	isInteracting = false, isGrounded = true, Анимация Damage_01 не проигрывается, В поле зрения главного героя нет противников, Нажатие клавиши F	Ничего не произойдёт (см. рисунок Б.31)
32	5,7,9, 35, 40	isInteracting = false, isGrounded = true, Анимация Damage_01 не проигрывается, В поле зрения главного героя есть противники, Нажатие клавиши H	Ничего не произойдёт (см. рисунок Б.32)

При проведении тестов из таблицы 4 результат совпал с ожидаемым, что говорит о том, что игровое приложение работает корректно.

Кроме главного героя и противника был подвержен тестированию и интерфейс видеоигры (в данном случае классы эквивалентности для интерфейса уже не составлялись).

Интерфейс должен выполнять все необходимые функции и при этом быть удобным и желательно приятным с эстетической точки зрения. Будет проверена работоспособность всех компонентов интерфейса и корректность отображения интерфейса в обычном окне и при изменении размеров окна игры.

Таким образом будет проверено качество, разработанного для видеоигры интерфейса.

В таблице 5 приведены примеры тестов интерфейса игры.

Таблица 5 – Тесты интерфейса игры

Номер теста	Описание теста	Входные данные	Ожидаемый результат	Фактический результат
1	Проверка работы кнопки Новая игра в главном меню	Нажать на кнопку Новая игра в главном меню	Откроется экран выбора оружия и сложности	Соответствует ожидаемому (см. рисунок Б.33)
2	Проверка работы кнопки Продолжить	В игре есть файл сохранения и нажата кнопка Продолжить	Игра загрузится на последней достигнутой контрольной точке	Соответствует ожидаемому (см. рисунок Б.34)
3	Проверка работы кнопки Назад на экране выбора оружия и сложности	Нажата кнопка Назад	Произойдет переход из экрана выбора оружия и сложности в главное меню игры	Соответствует ожидаемому (см. рисунок Б.35)
4	Проверка работы кнопки Начать игру на экране выбора оружия и сложности	После выбора легкой сложности и меча нажата кнопка Начать игру	Произойдет загрузка первого уровня игры (сложность будет легкой, а герой будет вооружен мечом)	Соответствует ожидаемому (см. рисунок Б.36)

Номер теста	Описание теста	Входные данные	Ожидаемый результат	Фактический результат
5	Проверка работы кнопки Управление в главном меню	Нажата кнопка Управление	Произойдет переход на экран с информацией об управлении игры	Соответствует ожидаемому (см. рисунок Б.37)
6	Проверка закрытия экрана с данными об управлении	Нажать на кнопку escape	Экран с информацией об управлении закроется и откроется главное меню	Соответствует ожидаемому (см. рисунок Б.38)
7	Проверка работы кнопки Выход в главном меню	Нажать на кнопку Выход в главном меню	Игра завершит свою работу	Соответствует ожидаемому
8	Проверка работы кнопки Автор в главном меню	Нажать на кнопку Автор в главном меню	Появится экран с данными об авторе	Соответствует ожидаемому (см. рисунок Б.39)
9	Проверка закрытия экрана с данными об авторе	Нажать на кнопку escape	Экран с данными об авторе закроется и откроется главное меню	Соответствует ожидаемому (см. рисунок Б. 40)
10	Запуск меню паузы в игре	Нажать на кнопку escape на стартовом уровне игры	Игра будет поставлена на паузу и появится меню паузы	Соответствует ожидаемому (см. рисунок Б.41)
11	Проверка работы кнопки Продолжить в меню паузы	Нажать на кнопку Продолжить	Меню паузы закроется и игра будет снята с паузы	Соответствует ожидаемому (см. рисунок Б.42)
12	Проверка работы кнопки Главное меню в меню паузы	Нажать на кнопку Главное меню	Меню паузы закроется и будет открыто главное меню	Соответствует ожидаемому (см. рисунок Б.43)
13	Проверка работы кнопки Выход в меню паузы	Нажать на кнопку Выход	Игра завершит свою работу	Соответствует ожидаемому
14	Проверка размещения элементов в главном меню при изменении размера окна игры	Находясь в главном меню, уменьшить размер окна игры	Кнопки Продолжить, Новая игра, Управление, Выход и Автор будут располагаться, как при обычном размере окна	Соответствует ожидаемому (см. рисунок Б.44)
15	Проверка размещения элементов в меню паузы при изменении размера окна игры	Находясь в меню паузы, уменьшить размер окна игры	Кнопки Продолжить, Главное меню и Выход будут располагаться друг под другом, как и при обычном размере окна	Соответствует ожидаемому (см. рисунок Б.45)
16	Проверка цветов элементов главного меню	Войти в главное меню игры	Фон главного меню – чёрный, Фон окна с данными об авторе – серый, Цвет букв – белый, Кнопки прозрачные, видны только буквы с надписями на них	Соответствует ожидаемому (см. рисунок Б.44)
17	Проверка цветов элементов меню паузы	Войти в меню паузы игры	Фон меню паузы – чёрный, Цвет букв – белый	Соответствует ожидаемому (см. рисунок Б.45)
18	Проверка положения элементов игрового HUD	Запустить стартовый уровень	Слева сверху в окне игры находятся полоски здоровья и выносливости, а справа снизу находится иконка сохранения игры	Соответствует ожидаемому (см. рисунок Б.46)

Окончание таблицы 5

Номер теста	Описание теста	Входные данные	Ожидаемый результат	Фактический результат
19	Проверка положения элементов игрового HUD при уменьшении размера окна игры	Запустить стартовый уровень и уменьшить размер окна игры	Полоски здоровья и выносливости окажутся слева сверху в окне игры, а иконка сохранения окажется в правом нижнем углу экрана	Соответствует ожидаемому (см. рисунок Б.47)
20	Проверка цветов элементов игрового HUD	Запустить стартовый уровень	Полоска здоровья красная, а полоска выносливости зелёная. Пустые части данных полосок помечаются чёрным цветом. Иконка сохранения – фиолетовая.	Соответствует ожидаемому (см. рисунок Б.46)
21	Проверка расположения элементов на экране смерти	Погибнуть в игре и таким образом открыть данный экран	В центре экрана появится надпись «Ты погиб перезагрузка через 5 секунд»	Соответствует ожидаемому (см. рисунок Б.48)
22	Проверка расположения элементов на экране смерти при изменении размеров окна игры	Погибнуть в игре и таким образом открыть данный экран и изменить размер окна	В центре экрана появится надпись «Ты погиб перезагрузка через 5 секунд»	Соответствует ожидаемому (см. рисунок Б.49)
23	Проверка цветов элементов на экране смерти	Погибнуть в игре и таким образом открыть данный экран	Фон красный-прозрачный, Буквы – чёрные	Соответствует ожидаемому (см. рисунок Б.48)

Результаты тестов интерфейса из таблицы 5 совпали с ожидаемыми. Это говорит о том, что интерфейс выполняет все свои функции и корректно отображается в разных масштабах.

6.3 Подсчёт метрик кода

Одним из способов оценки качества кода является подсчёт различных метрик кода. Для примера подсчёт метрик кода будет осуществлён на коде функций Update, HandleMovement, HandleJumping, приведённых в листинге 1. Будут посчитаны метрики Холстеда и Чепина.

Листинг 1 – Методы Update, HandleMovement, HandleJumping

```
void Update()
{
    float delta = Time.deltaTime;

    isInteracting = anim.GetBool("isInteracting");
    canDoCombo = anim.GetBool("canDoCombo");
```

```

        isInvulnerable = anim.GetBool("isInvulnerable");
        anim.SetBool("isInAir", isInAir);

        inputHandler.TickInput(delta);
        playerStats.RegenerateStamina();
        cameraHandler.HandleCameraTarget();
        playerLocomotion.HandleJumping();
        playerLocomotion.HandleMovement(delta);
    }
    public void HandleMovement(float delta)
    {
        projectedVelocity = Vector3.zero;
        if (inputHandler.rollFlag)
            return;

        if (playerManager.isInteracting)
        {
            return;
        }

        moveDirection = cameraObject.forward * inputHandler.vertical;
        moveDirection += cameraObject.right * inputHandler.horizontal;
        moveDirection.y = 0;
        moveDirection.Normalize();

        float speed = movementSpeed;

        if (inputHandler.sprintFlag && inputHandler.moveAmount > 0.5 &&
playerStats.currentStamina > 0)
        {
            playerStats.TakeStaminaDamage(sprintStaminaDrainage *
Time.deltaTime);
            speed = sprintSpeed;
            isSprinting = true;
            moveDirection *= speed;
        }
        else
        {
            if (inputHandler.moveAmount < 0.5)
            {
                moveDirection *= walkingSpeed;
                isSprinting = false;
            }
            else
            {
                moveDirection *= speed;
                isSprinting = false;
            }
        }

        projectedVelocity = Vector3.ProjectOnPlane(moveDirection,
normalVector);

        if (inputHandler.lockOnFlag && inputHandler.sprintFlag == false)
        {
            animatorHandler.UpdateAnimatorValues(inputHandler.vertical,
inputHandler.horizontal, playerManager.isSprinting);
        }
        else
        {
            animatorHandler.UpdateAnimatorValues(inputHandler.moveAmount, 0,
playerManager.isSprinting);
        }
    }

```

```

    }

public void HandleJumping()
{
    if (isInteracting)
        return;
    if (inputHandler.jump_Input)
    {
        if (inputHandler.moveAmount > 0 && playerStats.currentStamina >=
jumpStaminaDrainage)
        {
            playerStats.TakeStaminaDamage(jumpStaminaDrainage);
            moveDirection = cameraObject.forward *
inputHandler.vertical;
            moveDirection += cameraObject.right *
inputHandler.horizontal;
            moveDirection.y = 0;
            animatorHandler.PlayTargetAnimation("jump", true);
            Quaternion jumpRotation =
Quaternion.LookRotation(moveDirection);
            myTransform.rotation = jumpRotation;
        }
    }
}

```

На примере кода выше будут приведены примеры расчёта метрик кода.

6.3.1 Подсчёт метрики Холстеда

Метрика Холстеда относится к метрикам, вычисляемым на основании анализа числа строк и синтаксических исходного кода программы [21].

Для подсчёта оценок (например, таких как словарь программы или объём программы) в рамках данной метрики используются следующие характеристики: число уникальных операторов программы ($n1$), число уникальных операндов программы ($n2$), общее число операторов программы ($N1$), общее число операндов программы ($N2$).

Метрика Холстеда отлично подходит для оценки сложности кода и прогнозирования частоты ошибок в коде. Данная метрика применима к программам на разных языках и относительно легко подсчитывается. Для большей точности надо считать данную метрику для всего кода проекта, но в данном случае для упрощения вычислений будет приведён подсчёт метрики для примера кода из листинга 1.

Характеристики, связанные с операторами ($n1$, $N1$), приведены в таблице 6.

Таблица 6 – Операторы

Операторы	Число вхождений
=	15
==	1
*=	3
+=	2
>	3
<	1
>=	1
&&	4
return	3
anim.GetBool	3
anim.SetBool	1
inputHandler.TickInput	1
cameraHandler.HandleCamera Target	1
playerStats.RegenerateStamina	1
playerLocomotion.HandleJumping	1
playerLocomotion.HandleMovement	1
moveDirection.Normalize	2
playerStats.TakeStaminaDamage	2
Vector3.ProjectOnPlane	1

Продолжение таблицы 6

Операторы	Число вхождений
animatorHandler.UpdateAnimatorValues	2
animatorHandler.PlayTargetAnimation	1
Quaternion.LookRotation	1
;	37
,	7
+	1
*	5
n1 = 26	N1 = 101

Характеристики, связанные с операндами (n2, N2), приведены в таблице 7.

Таблица 7 – Операнды

Операнды	Число вхождений
delta	3
Time.deltaTime	2
isInteracting	4
canDoCombo	1
isInvulnerable	1
isInAir	1
projectedVelocity	2
inputHandler.rollFlag	3
moveDirection	13
moveDirection.y	2

Операнды	Число вхождений
cameraObject.forward	2
cameraObject.right	2
inputHandler.vertical	2
inputHandler.horizontal	2
Speed	2
movementSpeed	1
inputHandler.sprintFlag	2
inputHandler.moveAmount	4
playerStats.currentStamina	2
sprintStaminaDrainage	1
sprintSpeed	1
isSprinting	5
walkingSpeed	1
inputHandler.lockOnFlag	1
inputHandler.vertical	1
inputHandler.horizontal	1
inputHandler.jump_Input	1
jumpStaminaDrainage	2
jumpRotation	2
myTransform.rotation	1
0	4
0.5	2
false	3

Операнды	Число вхождений
True	2
“jump”	1
“isInteracting”	1
“canDoCombo”	1
“isInvulnerable”	1
n2 = 38	N2 = 83

Длина программы равна:

$$N = N1 + N2 = 101 + 83 = 184.$$

Словарь программы равен:

$$n = n1 + n2 = 26 + 38 = 64.$$

Объём программы:

$$HV = N * \log_2(n) = 184 * 6 = 1104.$$

Оценка уровня качества программирования:

$$L^{\wedge} = 2 * \frac{n2}{n1 * N2} = 2 * \frac{38}{26 * 83} = 0,035.$$

Сложность программы:

$$HD = \left(\frac{n1}{2}\right) * \left(\frac{N2}{n2}\right) = \left(\frac{26}{2}\right) * \left(\frac{83}{38}\right) = 28,39.$$

Для рассматриваемого кода оценки сложности программы и её объёма получились довольно высокими, а оценка уровня качества программирования получилась довольно невысокой. Всё это говорит об относительной громоздкости кода. Однако найденные значения лежат в пределах нормы для видеоигр, разрабатываемых на движке Unity.

6.3.2 Подсчёт метрики Чепина

Метрика Чепина состоит в оценке информационной прочности отдельно взятого программного модуля с помощью анализа характера использования переменных из списка ввода-вывода [21].

Для подсчёта метрики Чепина требуется посчитать значения следующих множеств [21]:

- множество вводимых переменных для расчётов и обеспечения вывода (P);
- множество модифицируемых или создаваемых внутри программы переменных (M);
- множество переменных, участвующих в управлении работой программного модуля (C);
- множество неиспользуемых в программе (паразитных) переменных (T).

Данная метрика так же, как и метрика Холстеда, применима к разным языкам программирования и относительно легка в плане расчётов, используемых в ней. Чем выше значение метрики Чепина, тем менее устойчив анализируемый блок кода и тем он менее качественен.

Метрика Чепина рассчитывается по следующей формуле:

$$Q = P + 2M + 3C + 0,5T.$$

В рассматриваемом примере кода (смотри листинг 1) значения множеств P, M, C, T равны соответственно значениям 6, 11, 6, 0. В таком случае расчёт значения метрики Чепина будет выглядеть так:

$$Q = 6 + 2 * 11 + 3 * 6 + 0,5 * 0 = 46.$$

Значение метрики Чепина для рассматриваемого кода равно 46. Если в результате изменения кода рассматриваемых функций значение метрики значительно увеличится, это будет означать низкое качество изменений.

Для значений данной метрики нет какой-то универсальной нормы. Значение метрики Чепина, полученное для кода из листинга 1, является приемлемым для разработанной видеоигры, так как оно сопоставимо со средними значениями данной метрики для других модулей кода данной видеоигры.

Заключение

В процессе выполнения выпускной квалификационной работы была разработана видеоигра в жанре Action с видом от третьего лица, в которой была реализована битва с игровым боссом, включающая две стадии.

В ходе работы были пройдены этапы: анализа предметной области и требований, проектирования приложения, реализации приложения, планирования разработки и оценки бюджета, тестирования приложения.

На этапе анализа предметной области и требований были проведены: анализ предметной области и постановка задачи, формирование технического задания, анализ целевой аудитории, анализ и сравнения программных продуктов аналогичного назначения, анализ вариантов использования, выбор технологий и инструментальных средств разработки.

На этапе проектирования приложения были осуществлены: проектирование интерфейсов, проектирование структуры программного обеспечения, проектирование алгоритмов, проектирование структур данных.

На этапе реализации было проведено кодирование приложения, а также была составлена диаграмма компонентов приложения.

На этапе планирования разработки и оценки бюджета был составлен календарный план проекта, построена диаграмма Ганта, посчитаны затраты на реализацию проекта.

На этапе тестирования были составлены классы эквивалентности и примеры тестов «чёрного ящика», а также на данном этапе был приведён пример подсчёта метрик Холстеда и Чепина для фрагмента кода разработанной видеоигры.

В итоге, была создана видеоигра, соответствующая составленным требованиям.

Список использованных источников

1. Newzoo Global Games Market Report 2022 | Free Version // newzoo. [Электронный ресурс]. URL: <https://newzoo.com/resources/trend-reports/newzoo-global-games-market-report-2022-free-version> (дата обращения: 24.05.2023)
2. 2022-й в цифрах: самые продаваемые игры и рост ПК-гейминга // stopgame. [Электронный ресурс]. URL: https://stopgame.ru/newsdata/56350/2022_y_v_cifrah_samye_prodavaemye_igry_i_rost_pk_geyminga?ysclid=lhfb0hovoo854337038 (дата обращения: 24.05.2023)
3. Олепенкова, Е. М. Проблема лидерства на российском рынке цифровой дистрибуции компьютерных игр / Е. М. Олепенкова, А. М. Лаврова, А. В. Ламкова, Н. С. Полякова // Современная конкуренция. – 2019. – №2 (74). – С. 20 – 36.
4. Крутов, Н. Ю. Специфические признаки жанра экшен в видеоиграх конца XX - начала XXI века / Н. Ю. Крутов, Н. А. Федоровская // Культура и искусство. – 2022. – №12. – С. 55 – 63.
5. Ильянов, Д. С. Источники роста экономики в XXI веке: индустрия видеоигр / Д. С. Ильянов, Т. К. Чернышева, М. А. Юревич // Теоретическая и прикладная экономика. – 2020. – №3. – С. 78 – 89.
6. Овдин, А. Е. Игровая индустрия России в условиях санкционного давления / А. Е. Овдин // Студент года 2022: Сборник статей XXIII Международного научно-исследовательского конкурса, Пенза, 05 ноября 2022 года. – Пенза: Наука и Просвещение (ИП Гуляев Г.Ю.), 2022. – С. 124 – 126.
7. Desktop Operating System Market Share Worldwide // statcounter. [Электронный ресурс]. URL: <https://gs.statcounter.com/os-market-share/desktop/worldwide> (дата обращения: 24.05.2023)
8. Desktop Windows Version Market Share Worldwide // statcounter. [Электронный ресурс]. URL: <https://gs.statcounter.com/os-version-market-share/windows/desktop/worldwide#monthly-202204-202304> (дата обращения: 24.05.2023)

9. Страница с информацией об игре Furi // thegamebakers – официальный сайт разработчика игры Furi. [Электронный ресурс]. URL: <https://www.thegamebakers.com/press/sheet.php?p=furi> (дата обращения: 24.05.2023)
10. Furi Steam Charts · SteamDB // steamdb. [Электронный ресурс]. URL: <https://steamdb.info/app/423230/charts/> (дата обращения: 24.05.2023)
11. DARK SOULS™: REMASTERED Steam Charts · SteamDB // steamdb. [Электронный ресурс]. URL: <https://steamdb.info/app/570940/charts/> (дата обращения: 24.05.2023)
12. Бабинович, В. С. Идентичность Геймера в виртуальном пространстве видеоигр / В. С. Бабинович // Вестник Томского государственного университета. – 2019. – №446. – С. 64 – 67.
13. Кудряшов, И. С. Семиотика «подсказки» в видеоиграх / И. С. Кудряшов // e-FORUM. – 2019. – №3(8). – С. 9.
14. Risk of Rain 2 Steam Charts · SteamDB // steamdb. [Электронный ресурс]. URL: <https://steamdb.info/app/632360/charts/> (дата обращения: 24.05.2023)
15. Буч, Г. Язык UML. Руководство пользователя. 2-е изд. / Г. Буч, Д. Рамбо, И. Якобсон. – М.: ДМК Пресс, 2006. – 496 с.
16. Габдрашитова, А. М. Технология быстрой разработки приложений RAD / А. М. Габдрашитова // Качество в производственных и социально-экономических системах: сборник научных трудов 2-ой Международной научно-практической конференции, посвященной 50-летию Юго-Западного государственного университета: в 2-х томах. Том 2. – Курск: Закрытое акционерное общество «Университетская книга», 2014. – С. 106 – 108.
17. Ершов, Т. А. Обзор игрового движка Unity / Т. А. Ершов // Modern Science. – 2020. – №8 – 2. – С. 288 – 291.
18. Камалидинова, Э. Р. Программы построения трехмерных графических изображений / Э. Р. Камалидинова, И. В. Рожина // Актуальные вопросы преподавания математики, информатики и информационных технологий. – 2015. – №1. – С. 82 – 90.

19. Марченко, М. Н. Дизайн игровых интерфейсов: история и направления развития на современном этапе / М. Н. Марченко, Н. И. Никитина // Бизнес и дизайн ревю. – 2021. – №3. – С. 23.

20. Канер, С. Тестирование программного обеспечения. Фундаментальные концепции менеджмента бизнес-приложений /С. Канер, Д. Фолк, Е. К. Нгуен. – К.: Издательство «ДиаСофт», 2001. – 544 с.

21. Евсеев, В. В. Применение программных метрик кода на раннем этапе жизненного цикла программного обеспечения / В. В. Евсеев // Восточно-Европейский журнал передовых технологий. – 2011. – Т. 1, №2(49). – С. 19 – 21.

КОД ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Приложение А
(рекомендуемое)

Листинг А.1 – Код класса AttackState.cs

```

public class AttackState : State
{
    public CombatStanceState combatStanceState;
    public PursueTargetState pursueTargetState;
    public EnemyAttackAction currentAttack;

    public bool willDoComboOnNextAttack = false;
    public bool hasPerformedAttack = false;
    public bool specialAttack;
    public float specialAttackTime = 0.5f;

    protected float damageMultiplier;
    string path = @"Assets\SaveData\save.txt";
    private List<string> savedata = new List<string>();

    protected void Start()
    {
        ReadSaveData();
        if (savedata[4] == "easyDifficulty")
        {
            damageMultiplier = 1;
        }
        if (savedata[4] == "middleDifficulty")
        {
            damageMultiplier = 1.5f;
        }
        if (savedata[4] == "hardDifficulty")
        {
            damageMultiplier = 2;
        }
    }

    public override State Tick(EnemyManager enemyManager, EnemyData enemyStats,
    EnemyAnimatorManager enemyAnimatorManager)
    {
        float distanceFromTarget =
        Vector3.Distance(enemyManager.currentTarget.transform.position,
        enemyManager.transform.position);
        if (distanceFromTarget > enemyManager.maximumAttackRange)
        {
            return pursueTargetState;
        }

        if (willDoComboOnNextAttack && enemyManager.canDoCombo)
        {
            AttackTargetWithCombo(enemyManager, enemyAnimatorManager);
        }

        if (!hasPerformedAttack)
        {
            AttackTarget(enemyManager, enemyAnimatorManager);
            RollForComboChance(enemyManager);
        }

        if (willDoComboOnNextAttack && hasPerformedAttack)
        {
            return this;
        }

        return combatStanceState;
    }
}

```

```

    }

    private void ReadSaveData()
    {
        using (StreamReader sr = File.OpenText(path))
        {
            string s = "";
            while ((s = sr.ReadLine()) != null)
            {
                savedata.Add(s);
            }
        }
    }

    protected void AttackTarget(EnemyManager enemyManager, EnemyAnimatorManager enemyAnimatorManager)
    {
        enemyManager.GetComponentInChildren<DamageCollider>().currentWeaponDamage =
        Mathf.RoundToInt(currentAttack.attackDamage * damageMultiplier);
        enemyAnimatorManager.PlayTargetAnimation(currentAttack.actionAnimation,
        true);
        enemyManager.currentRecoveryTime = currentAttack.recoveryTime;
        hasPerformedAttack = true;
    }

    protected void AttackTargetWithCombo(EnemyManager enemyManager,
    EnemyAnimatorManager enemyAnimatorManager)
    {
        enemyManager.GetComponentInChildren<DamageCollider>().currentWeaponDamage =
        Mathf.RoundToInt(currentAttack.attackDamage * damageMultiplier);
        willDoComboOnNextAttack = false;
        enemyAnimatorManager.PlayTargetAnimation(currentAttack.actionAnimation,
        true);
        enemyManager.currentRecoveryTime = currentAttack.recoveryTime;
        currentAttack = null;
        hasPerformedAttack = true;
    }

    protected void RollForComboChance(EnemyManager enemyManager)
    {
        float comboChance = Random.Range(0, 100);
        if (enemyManager.allowAIToPerformCombos && comboChance <=
        enemyManager.comboLikelyHood)
        {
            if (currentAttack.comboAction != null)
            {
                willDoComboOnNextAttack = true;
                currentAttack = currentAttack.comboAction;
            }
            else
            {
                willDoComboOnNextAttack = false;
                currentAttack = null;
            }
        }
        else
        {
            currentAttack = null;
        }
    }
}

```

Листинг A.2 – Код класса BossAttackState.cs

```

public class BossAttackState : AttackState
{
    public int specialAttackDamage = 200;

    private void Start()
    {
        base.Start();
    }

    public override State Tick(EnemyManager enemyManager, EnemyData enemyStats,
EnemyAnimatorManager enemyAnimatorManager)
    {
        if (enemyAnimatorManager.anim.GetBool("isInvulnerable"))
            return this;
        if (!specialAttack)
        {
            float distanceFromTarget =
Vector3.Distance(enemyManager.currentTarget.transform.position,
enemyManager.transform.position);
            return base.Tick(enemyManager, enemyStats, enemyAnimatorManager);
        }
        else
        {
            float distanceFromTarget =
Vector3.Distance(enemyManager.currentTarget.transform.position,
enemyManager.transform.position);
            if (!hasPerformedAttack)
            {
                PerformSpecialAttack(enemyManager, enemyAnimatorManager);
            }
            return combatStanceState;
        }
    }

    private void PerformSpecialAttack(EnemyManager enemyManager,
EnemyAnimatorManager enemyAnimatorManager)
    {
        StartCoroutine(PerformDashAttack(enemyManager, enemyAnimatorManager));
    }

    IEnumerator PerformDashAttack(EnemyManager enemyManager,
EnemyAnimatorManager enemyAnimatorManager)
    {
        Vector3 DashDirection = enemyManager.currentTarget.transform.position -
enemyManager.transform.position;
        DashDirection.y = 0;
        float length = DashDirection.magnitude;
        float scaling = (length - enemyManager.maximumAttackRange) / length;
        DashDirection.Set(DashDirection.x * scaling, DashDirection.y * scaling,
DashDirection.z * scaling);
        float timeElapsed = 0;
        float duration = 0.2f;
        Vector3 startPosition = enemyManager.transform.position;
        enemyAnimatorManager.PlayTargetAnimation("Boss_jump", true);
        while (timeElapsed < duration)
        {
            enemyManager.GetComponent<Rigidbody>().MovePosition(Vector3.Lerp(startPosition,
startPosition + DashDirection, timeElapsed / duration));
            timeElapsed += Time.deltaTime;
        }
    }
}

```

```

        yield return null;
    }

    enemyManager.GetComponentInChildren<DamageCollider>().currentWeaponDamage =
    Mathf.RoundToInt(specialAttackDamage * damageMultiplier);
    enemyAnimatorManager.PlayTargetAnimation("Boss_OH_heavy_attack_1",
    true);
    enemyManager.currentSpecialRecoveryTime = 10f;
    hasPerformedAttack = true;
    }
}

```

Листинг A.3 – Код класса BossCombatStanceState.cs

```

public class BossCombatStanceState : CombatStanceState
{
    [Header("Second Phase Attacks")]
    public bool hasPhaseShifted;
    public EnemyAttackAction[] secondPhaseEnemyAttacks;

    protected override void GetNewAttack(EnemyManager enemyManager)
    {
        if (hasPhaseShifted)
        {
            int randomValue = Random.Range(0, secondPhaseEnemyAttacks.Length);
            attackState.currentAttack = secondPhaseEnemyAttacks[randomValue];
        }
        else
        {
            base.GetNewAttack(enemyManager);
        }
    }
}

```

Листинг A.4 – Код класса CombatStanceState.cs

```

public class CombatStanceState : State
{
    public AttackState attackState;
    public EnemyAttackAction[] enemyAttacks;
    public PursueTargetState pursueTargetState;

    public bool hasSpecial = false;
    protected bool randomDestinationSet = false;
    protected float verticalMovementValue = 0;
    protected float horizontalMovementValue = 0;

    public override State Tick(EnemyManager enemyManager, EnemyData enemyStats,
    EnemyAnimatorManager enemyAnimatorManager)
    {
        enemyManager.distanceFromTarget =
        Vector3.Distance(enemyManager.currentTarget.transform.position,
        enemyManager.transform.position);
        //enemyAnimatorManager.anim.SetFloat("Vertical", 0, 0.1f,
        Time.deltaTime);
        enemyAnimatorManager.anim.SetFloat("Vertical", verticalMovementValue,
        0.2f, Time.deltaTime);
        enemyAnimatorManager.anim.SetFloat("Horizontal",
        horizontalMovementValue, 0.2f, Time.deltaTime);
        attackState.hasPerformedAttack = false;
        attackState.specialAttack = false;
    }
}

```

```

        if (enemyManager.isInteracting)
        {
            enemyAnimatorManager.anim.SetFloat("Vertical", 0);
            enemyAnimatorManager.anim.SetFloat("Horizontal", 0);
            return this;
        }
        if (hasSpecial && enemyManager.distanceFromTarget <=
enemyManager.maximumSpecialRange && enemyManager.currentSpecialRecoveryTime <= 0
&& enemyManager.distanceFromTarget > enemyManager.maximumAttackRange)
        {
            attackState.specialAttack = true;
            enemyAnimatorManager.anim.SetFloat("Vertical", 0);
            enemyAnimatorManager.anim.SetFloat("Horizontal", 0);
            return attackState;
        }

        if (enemyManager.distanceFromTarget > enemyManager.maximumAttackRange)
        {
            enemyAnimatorManager.StopRotation();
            return pursueTargetState;
        }

        if (!randomDestinationSet)
        {
            randomDestinationSet = true;
            DecideCirclingAction();
            enemyAnimatorManager.CanRotate();
        }

        if (enemyManager.currentRecoveryTime <= 0 && attackState.currentAttack
!= null)
        {
            randomDestinationSet = false;
            enemyAnimatorManager.anim.SetFloat("Vertical", 0);
            enemyAnimatorManager.anim.SetFloat("Horizontal", 0);
            return attackState;
        }
        else
        {
            GetNewAttack(enemyManager);
        }
        return this;
    }

    protected void DecideCirclingAction()
    {
        WalkAroundTarget();
    }

    protected void WalkAroundTarget()
    {
        verticalMovementValue = 1f;

        horizontalMovementValue = Random.Range(-1, 1);

        if (horizontalMovementValue <= 1 && horizontalMovementValue >= 0)
        {
            horizontalMovementValue = 1f;
        }
        else if (horizontalMovementValue >= -1 && horizontalMovementValue < 0)
        {

```

```

        horizontalMovementValue = -1f;
    }
}

protected virtual void GetNewAttack(EnemyManager enemyManager)
{
    int randomValue = Random.Range(0, enemyAttacks.Length);
    attackState.currentAttack = enemyAttacks[randomValue];
}
}

```

Листинг A.5 – Код класса IdleState.cs

```

public class IdleState : State
{
    public PursueTargetState pursueTargetState;

    public LayerMask detectionLayer;

    public override State Tick(EnemyManager enemyManager, EnemyData enemyStats,
    EnemyAnimatorManager enemyAnimatorManager)
    {
        if (enemyManager.isInteracting)
        {
            return this;
        }

        #region Handle Enemy Target Detection
        Collider[] colliders = Physics.OverlapSphere(transform.position,
    enemyManager.detectionRadius, detectionLayer);

        for (int i = 0; i < colliders.Length; i++)
        {
            CharacterStats characterStats =
    colliders[i].transform.GetComponent<CharacterStats>();

            if (characterStats != null)
            {
                Vector3 targetDirection = characterStats.transform.position -
    transform.position;
                float viewableAngle = Vector3.Angle(targetDirection,
    transform.forward);

                if (viewableAngle > enemyManager.minimumDetectionAngle &&
    viewableAngle < enemyManager.maximumDetectionAngle)
                {
                    enemyManager.currentTarget = characterStats;
                }
            }
        }
        #endregion

        #region Handle Switching To Next State
        if (enemyManager.currentTarget != null)
        {
            return pursueTargetState;
        }
        else
        {
            return this;
        }
        #endregion
    }
}

```

```

    }
}

```

Листинг А.6 – Код класса PursueTargetState.cs

```

public class PursueTargetState : State
{
    public CombatStanceState combatStanceState;

    public override State Tick(EnemyManager enemyManager, EnemyData enemyStats,
    EnemyAnimatorManager enemyAnimatorManager)
    {
        enemyManager.distanceFromTarget =
        Vector3.Distance(enemyManager.currentTarget.transform.position,
        transform.position);

        if (enemyManager.isInteracting)
            return this;

        if (enemyManager.isPerformingAction)
        {
            Debug.Log("isPerformingAction == true");
            enemyAnimatorManager.anim.SetFloat("Vertical", 0, 0.1f,
            Time.deltaTime);
            enemyAnimatorManager.anim.SetFloat("Horizontal", 0, 0.1f,
            Time.deltaTime);
            return this;
        }

        if (combatStanceState.hasSpecial && enemyManager.distanceFromTarget <=
        enemyManager.maximumSpecialRange && enemyManager.currentSpecialRecoveryTime <=
        0)
        {
            enemyManager.moveForward = false;
            return combatStanceState;
        }

        enemyManager.moveForward = false;
        if (enemyManager.distanceFromTarget > enemyManager.maximumAttackRange)
        {
            enemyManager.moveForward = true;
        }

        if (enemyManager.distanceFromTarget <= enemyManager.maximumAttackRange)
        {
            return combatStanceState;
        }
        else
        {
            return this;
        }
    }
}

```

Листинг А.7 – Код класса EnemyAttackAction.cs

```

public class EnemyAttackAction : EnemyAction
{
    public bool canCombo;
    public EnemyAttackAction comboAction;
}

```

```

public int attackDamage = 10;
public float recoveryTime = 2;

public float maximumAttackAngle = 35;
public float minimumAttackAngle = -35;

public float minimumDistanceNeededToAttack = 0;
public float maximumDistanceNeededToAttack = 3;
}

```

Листинг А.8 – Код класса EnemyData.cs

```

public class EnemyData : CharacterStats
{
    EnemyLocomotionManager enemyLocomotion;
    public new Rigidbody rigidbody;
    public HealthBar healthBar;
    public UIEnemyHealthBar uiHealthBar;

    public bool isBoss = false;

    Animator animator;
    EnemyManager enemyManager;
    BossManager bossManager;
    BossCombatStanceState bossCombatStanceState;
    AttackState attackState;

    private void Awake()
    {
        animator = GetComponentInChildren<Animator>();
        enemyLocomotion = GetComponent<EnemyLocomotionManager>();
        TryGetComponent<EnemyManager>(out enemyManager);
        TryGetComponent<BossManager>(out bossManager);
        rigidbody = GetComponent<Rigidbody>();
    }

    void Start()
    {
        maxHealth = SetMaxHealthFromLvl();
        currentHealth = maxHealth;
        uiHealthBar = GetComponentInChildren<UIEnemyHealthBar>();
        if (healthBar != null)
            healthBar.SetMaxHealth(Mathf.RoundToInt(maxHealth));
        if (uiHealthBar != null)
            uiHealthBar.SetMaxHealth(Mathf.RoundToInt(maxHealth));

        if (isBoss)
        {
            bossCombatStanceState =
GetComponentInChildren<BossCombatStanceState>();
            attackState = GetComponentInChildren<AttackState>();
        }
    }

    public void SetStats()
    {
        maxHealth = SetMaxHealthFromLvl();
        currentHealth = maxHealth;
        if (healthBar != null)
            healthBar.SetMaxHealth(Mathf.RoundToInt(maxHealth));
        if (uiHealthBar != null)

```

```

        uiHealthBar.SetMaxHealth(Mathf.RoundToInt(maxHealth));
    }

    private float SetMaxHealthFromLvl()
    {
        maxHealth = healthLevel * 10;
        return maxHealth;
    }

    public void TakeDamage(int damage)
    {
        if (animator.GetBool("isInvulnerable"))
        {
            return;
        }
        currentHealth = currentHealth - damage;
        if (healthBar != null)
            healthBar.SetCurrentHealth(Mathf.RoundToInt(currentHealth));
        if (uiHealthBar != null)
            uiHealthBar.SetCurrentHealth(Mathf.RoundToInt(currentHealth));
        animator.Play("Damage_01");

        if (isBoss && currentHealth <= maxHealth / 2 &&
!bossCombatStanceState.hasPhaseShifted)
        {
            bossManager.ShiftToSecondPhase();
            attackState.willDoComboOnNextAttack = false;
        }
        if (!isBoss)
            enemyManager.currentRecoveryTime += 0.2f;
        else
            bossManager.currentRecoveryTime += 0.2f;
        if (currentHealth <= 0)
        {
            currentHealth = 0;
            if (!isBoss)
                enemyManager.isDead = true;
            else
                bossManager.isDead = true;
            rigidbody.useGravity = false;
            animator.Play("Dead_01");
            enemyLocomotion.characterCollider.enabled = false;
            enemyLocomotion.characterCollisionBlockerCollider.enabled = false;
        }
    }
}

```

Листинг А.9 – Код класса EnemyManager.cs

```

public class EnemyManager : CharacterManager
{
    string path = @"Assets\SaveData\save.txt";
    private List<string> savedata = new List<string>();

    protected EnemyAnimatorManager enemyAnimatorManager;
    protected EnemyLocomotionManager enemyLocomotionManager;
    protected EnemyData enemyData;

    public Rigidbody enemyRigidbody;
    public NavMeshAgent navMeshAgent;
    public State currentState;
    public CharacterStats currentTarget;
}

```

```

public bool isPerformingAction;
public bool isInteracting;
public bool moveForward;

public float distanceFromTarget;
public float rotationSpeed = 15;
public float walkingSpeed = 2;
public float maximumAttackRange = 1.5f;
public float maximumSpecialRange = 40f;

[Header("Combat Flags")]
public bool canDoCombo;

[Header("AI settings")]
public float detectionRadius = 20;
public float maximumDetectionAngle = 50;
public float minimumDetectionAngle = -50;
public float viewableAngle;

[Header("A.I. Combat Settings")]
public bool allowAIToPerformCombos;
public float comboLikelyHood = 70;

public float currentRecoveryTime = 0;
public float currentSpecialRecoveryTime = 0;
protected DamageCollider damageCollider;

protected void Awake()
{
    enemyLocomotionManager = GetComponent<EnemyLocomotionManager>();
    enemyAnimatorManager = GetComponentInChildren<EnemyAnimatorManager>();
    enemyData = GetComponent<EnemyData>();
    navMeshAgent = GetComponentInChildren<NavMeshAgent>();
    navMeshAgent.enabled = false;
    enemyRigidbody = GetComponent<Rigidbody>();
}

protected void Start()
{
    enemyRigidbody.isKinematic = false;

    ReadSaveData();
    if (savedata[4] == "easyDifficulty")
    {
        enemyData.healthLevel *= 1;
        enemyData.SetStats();
    }
    if (savedata[4] == "middleDifficulty")
    {
        enemyData.healthLevel = Mathf.RoundToInt(1.5f *
enemyData.healthLevel);
        enemyData.SetStats();
    }
    if (savedata[4] == "hardDifficulty")
    {
        enemyData.healthLevel *= 2;
        enemyData.SetStats();
    }
}

private void ReadSaveData()
{

```

```

        using (StreamReader sr = File.OpenText(path))
        {
            string s = "";
            while ((s = sr.ReadLine()) != null)
            {
                savedata.Add(s);
            }
        }
    }

    protected void Update()
    {
        HandleRecoveryTimer();
        canDoCombo = enemyAnimatorManager.anim.GetBool("canDoCombo");
        isInteracting = enemyAnimatorManager.anim.GetBool("isInteracting");
        if (enemyData.uiHealthBar)
            enemyData.uiHealthBar.ControlUIHealthBar();
        if (isDead)
            return;
        HandleStateMachine();
        HandleRotateTowardsTarget(enemyAnimatorManager);
    }

    protected void FixedUpdate()
    {
        if (isDead)
            return;
        PushRigidBody(enemyAnimatorManager);
        navMeshAgent.transform.localPosition = Vector3.zero;
        navMeshAgent.transform.localRotation = Quaternion.identity;
    }

    protected void HandleStateMachine()
    {
        if (currentState != null)
        {
            State nextState = currentState.Tick(this, enemyData,
            enemyAnimatorManager);

            if (nextState != null)
            {
                SwitchToNextState(nextState);
            }
        }
    }

    protected void SwitchToNextState(State state)
    {
        currentState = state;
    }

    protected void HandleRecoveryTimer()
    {
        if (currentRecoveryTime > 0)
        {
            currentRecoveryTime -= Time.deltaTime;
        }

        if (currentSpecialRecoveryTime > 0)
        {
            currentSpecialRecoveryTime -= Time.deltaTime;
        }
    }

```

```

        if (isPerformingAction)
        {
            if (currentRecoveryTime <= 0)
            {
                isPerformingAction = false;
            }
        }
    }

    protected void HandleRotateTowardsTarget (EnemyAnimatorManager
enemyAnimatorManager)
    {
        if (currentTarget == null)
            return;
        if (enemyAnimatorManager.anim.GetBool("canRotate"))
        {
            Vector3 direction =
                currentTarget.transform.position -
                transform.position;
            direction.y = 0;
            direction.Normalize();

            if (direction == Vector3.zero)
            {
                direction = transform.forward;
            }

            Quaternion targetRotation = Quaternion.LookRotation(direction);
            enemyRigidbody.MoveRotation(targetRotation);
        }
        else
        {
            Vector3 relativeDirection =
transform.InverseTransformDirection(navMeshAgent.desiredVelocity);
            Vector3 targetVelocity = enemyRigidbody.velocity;

            navMeshAgent.enabled = true;
            navMeshAgent.SetDestination(currentTarget.transform.position);
            enemyRigidbody.velocity = targetVelocity;
            enemyRigidbody.MoveRotation(Quaternion.Slerp(transform.rotation,
navMeshAgent.transform.rotation, rotationSpeed / Time.deltaTime));
        }
    }

    protected void PushRigidBody (EnemyAnimatorManager enemyAnimatorManager)
    {
        if (!moveForward)
            return;
        enemyAnimatorManager.anim.SetFloat("Vertical", 1, 0.1f, Time.deltaTime);
        enemyAnimatorManager.anim.SetFloat("Horizontal", 0, 0.1f,
Time.deltaTime);
        Vector3 targetVelocity = transform.forward;
        targetVelocity.y = 0;
        targetVelocity.Normalize();
        targetVelocity *= walkingSpeed;
        Vector3 projectedVelocity = Vector3.ProjectOnPlane(targetVelocity,
Vector3.up);
        enemyRigidbody.velocity = targetVelocity;
    }

    protected void OnDrawGizmosSelected()
    {
        Gizmos.color = Color.red;
    }

```

```

        Gizmos.DrawWireSphere(transform.position, detectionRadius);
    }
}

```

Листинг А.10 – Код класса PlayerManager.cs

```

public class PlayerManager : CharacterManager
{
    string path = @"Assets\SaveData\save.txt";
    InputHandler inputHandler;
    Animator anim;
    PlayerLocomotion playerLocomotion;
    PlayerStats playerStats;
    CameraHandler cameraHandler;
    public List<WeaponItem> gamerWeapons = new List<WeaponItem>();

    [Header("Player Flags")]
    public bool isInteracting;
    public bool isSprinting;
    public bool isInAir;
    public bool isGrounded;
    public bool canDoCombo;
    public bool isInvulnerable;
    public bool isBlocking;
    public bool takingDamage;

    public int damageReduce = 75;
    private List<string> savedata = new List<string>();

    private void Awake()
    {
    }

    void Start()
    {
        playerLocomotion = GetComponent<PlayerLocomotion>();
        inputHandler = GetComponent<InputHandler>();
        anim = GetComponentInChildren<Animator>();
        playerStats = GetComponent<PlayerStats>();
        cameraHandler = GetComponent<CameraHandler>();
        ReadSaveData();
        this.GetComponentInParent<Transform>().position = new
Vector3(float.Parse(savedata[1]),
        float.Parse(savedata[2]), float.Parse(savedata[3]));
        if (savedata[4] == "easyDifficulty")
        {
            playerStats.healthLevel *= 2;
            playerStats.healthRegenerationAmount *= 2;
            playerStats.staminaLevel *= 2;
            playerStats.staminaRegenerationAmount *= 2;
            playerStats.SetStats();
        }
        if (savedata[4] == "middleDifficulty")
        {
            playerStats.healthLevel = Mathf.RoundToInt(1.5f *
playerStats.healthLevel);
            playerStats.healthRegenerationAmount = 1.5f *
playerStats.healthRegenerationAmount;
            playerStats.staminaLevel = Mathf.RoundToInt(1.5f *
playerStats.staminaLevel);

```

```

        playerStats.staminaRegenerationAmount = 1.5f *
playerStats.staminaRegenerationAmount;
        playerStats.SetStats();
    }
    if (savedata[4] == "hardDifficulty")
    {
        playerStats.healthLevel *= 1;
        playerStats.healthRegenerationAmount *= 1;
        playerStats.staminaLevel *= 1;
        playerStats.staminaRegenerationAmount *= 1;
        playerStats.SetStats();
    }
    if (savedata[5] == "Sword")
    {

this.GetComponent<PlayerInventory>().ChangeRightWeapon(gamerWeapons[0]);
    }
    if (savedata[5] == "Mace")
    {

this.GetComponent<PlayerInventory>().ChangeRightWeapon(gamerWeapons[1]);
    }
    }

    void Update()
    {
        float delta = Time.deltaTime;

        isInteracting = anim.GetBool("isInteracting");
        canDoCombo = anim.GetBool("canDoCombo");
        isInvulnerable = anim.GetBool("isInvulnerable");
        anim.SetBool("isBlocking", isBlocking);
        anim.SetBool("isInAir", isInAir);
        anim.SetBool("isDead", isDead);
        anim.SetBool("takingDamage", takingDamage);
        inputHandler.TickInput(delta);
        playerStats.RegenerateStamina();
        playerStats.RegenerateHealth();
        cameraHandler.HandleCameraTarget();
        cameraHandler.CheckObstacles();
        cameraHandler.CheckDead();
        playerLocomotion.HandleRollingAndSprinting(delta);
        playerLocomotion.HandleJumping();
        playerLocomotion.HandleMovement(delta);
        if (anim.GetBool("canRotate"))
            playerLocomotion.HandleRotation(delta);
    }

    private void FixedUpdate()
    {
        float delta = Time.fixedDeltaTime;

        playerLocomotion.HandleFalling(delta, playerLocomotion.moveDirection);
        playerLocomotion.MoveRigidBody(delta);
    }

    private void LateUpdate()
    {
        inputHandler.rollFlag = false;
        inputHandler.sprintFlag = false;
        inputHandler.rb_Input = false;
        inputHandler.rt_Input = false;
        inputHandler.jump_Input = false;
    }

```

```

        inputHandler.escape_Input = false;
        takingDamage = false;

        if (isInAir)
        {
            playerLocomotion.inAirTimer = playerLocomotion.inAirTimer +
Time.deltaTime;
        }
    }

    private void ReadSaveData()
    {
        using (StreamReader sr = File.OpenText(path))
        {
            string s = "";
            while ((s = sr.ReadLine()) != null)
            {
                savedata.Add(s);
            }
        }
    }
}

```

Листинг А.11 – Код класса CharacterManager.cs

```

public class CharacterManager : MonoBehaviour
{
    public Transform lockOnTransform;
    public bool isDead = false;
}

```

Листинг А.12 – Код класса BossManager.cs

```

public class BossManager : EnemyManager
{
    public string bossName;
    BossCombatStanceState bossCombatStanceState;

    protected new void Awake()
    {
        base.Awake();
        bossCombatStanceState = GetComponentInChildren<BossCombatStanceState>();
        enemyData.isBoss = true;
    }

    public void ShiftToSecondPhase()
    {
        enemyAnimatorManager.PlayTargetAnimation("Phase Shift", true);
        bossCombatStanceState.hasPhaseShifted = true;
        bossCombatStanceState.hasSpecial = true;
    }
}

```

Листинг А.13 – Код класса CameraHandler.cs

```

public class CameraHandler : MonoBehaviour
{
    public Transform cameraTransform;
    PlayerManager playerManager;
}

```

```

    InputHandler inputHandler;
    public Transform targetTransform;
    public LayerMask environmentLayer;

    public CharacterManager currentLockOnTarget;

    List<CharacterManager> availableTargets = new List<CharacterManager>();
    public float maximumLockOnDistance = 30;
    public CharacterManager nearestLockOnTarget;
    public CharacterManager leftLockTarget;
    public CharacterManager rightLockTarget;

    public CinemachineFreeLook cinemachine;
    public CinemachineFreeLook cinemachineVC;

    private void Awake()
    {
        targetTransform = FindObjectOfType<PlayerManager>().lockOnTransform;
        inputHandler = FindObjectOfType<InputHandler>();
        cameraTransform = FindObjectOfType<Camera>().transform;
        playerManager = GetComponent<PlayerManager>();
    }

    private void Start()
    {
        environmentLayer = LayerMask.NameToLayer("Environment");
    }

    public void HandleCameraTarget()
    {
        if (inputHandler.lockOnFlag == true && currentLockOnTarget != null)
        {
            cinemachine.Priority = 0;
            cinemachineVC.Priority = 10;
            cinemachineVC.LookAt =
currentLockOnTarget.lockOnTransform.transform;
            Vector3 targetdirection = currentLockOnTarget.transform.position -
targetTransform.position;
            targetdirection.y = 0;
            float angle = Vector3.SignedAngle(targetdirection, Vector3.forward,
Vector3.up);
            cinemachineVC.m_YAxis.Value = 0.65f;
            cinemachineVC.m_XAxis.Value = -angle;
        }
        else
        {
            cinemachineVC.Priority = 0;
            cinemachine.Priority = 10;
            cinemachine.LookAt = targetTransform;
        }
    }

    public void HandleLockOn()
    {
        float shortestDistance = Mathf.Infinity;
        float shortestDistanceOfLeftTarget = -Mathf.Infinity;
        float shortestDistanceOfRightTarget = Mathf.Infinity;

        Collider[] colliders = Physics.OverlapSphere(targetTransform.position,
26);

        for (int i = 0; i < colliders.Length; i++)

```

```

    {
        CharacterManager character =
colliders[i].GetComponent<CharacterManager>();

        if (character != null)
        {
            Vector3 lockTargetDirection = character.transform.position -
targetTransform.position;
            float distanceFromTarget =
Vector3.Distance(targetTransform.position, character.transform.position);
            float viewableAngle = Vector3.Angle(lockTargetDirection,
cameraTransform.forward);
            RaycastHit hit;

            if (character.transform.root != targetTransform.transform.root
                && viewableAngle > -50 && viewableAngle < 50
                && distanceFromTarget <= maximumLockOnDistance)
            {
                if (Physics.Linecast(playerManager.lockOnTransform.position,
character.lockOnTransform.position, out hit))
                {
                    Debug.DrawLine(playerManager.lockOnTransform.position,
character.lockOnTransform.position);
                    if (hit.transform.gameObject.layer == environmentLayer)
                    {
                        }
                    else
                    {
                        avaialableTargets.Add(character);
                    }
                }
            }
        }

        for (int k = 0; k < avaialableTargets.Count; k++)
        {
            float distanceFromTarget =
Vector3.Distance(targetTransform.position,
avaialableTargets[k].transform.position);

            if (distanceFromTarget < shortestDistance)
            {
                shortestDistance = distanceFromTarget;
                nearestLockOnTarget = avaialableTargets[k];
            }

            if (inputHandler.lockOnFlag)
            {
                Vector3 relativeEnemyPosition =
inputHandler.transform.InverseTransformPoint(avaialableTargets[k].transform.posi
tion);

                var distanceFromLeftTarget = relativeEnemyPosition.x;
                var distanceFromRightTarget = relativeEnemyPosition.x;

                if (relativeEnemyPosition.x <= 0.00 && distanceFromLeftTarget >
shortestDistanceOfLeftTarget
                    && avaialableTargets[k] != currentLockOnTarget)
                {
                    shortestDistanceOfLeftTarget = distanceFromLeftTarget;
                    leftLockTarget = avaialableTargets[k];
                }
            }
        }
    }
}

```

```

        if (relativeEnemyPosition.x >= 0.00 && distanceFromRightTarget <
shortestDistanceOfRightTarget
            && avaialableTargets[k] != currentLockOnTarget)
        {
            shortestDistanceOfRightTarget = distanceFromRightTarget;
            rightLockTarget = avaialableTargets[k];
        }
    }
}

public void ClearLockOnTargets()
{
    avaialableTargets.Clear();
    nearestLockOnTarget = null;
    currentLockOnTarget = null;
}

public void CheckObstacles()
{
    if (!currentLockOnTarget) return;
    Vector3 lockTargetDirection = currentLockOnTarget.transform.position -
targetTransform.position;
    float distanceFromTarget = Vector3.Distance(targetTransform.position,
currentLockOnTarget.transform.position);
    float viewableAngle = Vector3.Angle(lockTargetDirection,
cameraTransform.forward);
    RaycastHit hit;

    if (currentLockOnTarget.transform.root != targetTransform.transform.root
        && viewableAngle > -50 && viewableAngle < 50
        && distanceFromTarget <= maximumLockOnDistance)
    {
        if (Physics.Linecast(playerManager.lockOnTransform.position,
currentLockOnTarget.lockOnTransform.position, out hit))
        {
            Debug.DrawLine(playerManager.lockOnTransform.position,
currentLockOnTarget.lockOnTransform.position);
            if (hit.transform.gameObject.layer == environmentLayer)
            {
                inputHandler.lockOnFlag = false;
                currentLockOnTarget = null;
            }
        }
    }
}

public void CheckDead()
{
    if (playerManager.isDead)
    {
        inputHandler.lockOnFlag = false;
        currentLockOnTarget = null;
    }
    if (currentLockOnTarget != null)
    {
        if (currentLockOnTarget.isDead)
        {
            currentLockOnTarget = null;
            inputHandler.lockOnFlag = false;
        }
    }
}

```

```

    }
}

```

Листинг А.14 – Код класса PlayerAttacker.cs

```

public class PlayerAttacker : MonoBehaviour
{
    AnimatorHandler animatorHandler;
    InputHandler inputHandler;
    WeaponSlotManager weaponSlotManager;
    PlayerStats playerStats;
    //private DamageCollider damageCollider;
    public string lastAttack;

    private void Awake()
    {
        animatorHandler = GetComponentInChildren<AnimatorHandler>();
        inputHandler = GetComponent<InputHandler>();
        weaponSlotManager = GetComponentInChildren<WeaponSlotManager>();
        playerStats = GetComponent<PlayerStats>();
    }

    public void HandleWeaponCombo(WeaponItem weapon)
    {
        if (playerStats.currentStamina <= 0 || playerStats.currentHealth <= 0)
            return;
        if (inputHandler.comboFlag)
        {
            animatorHandler.anim.SetBool("canDoCombo", false);

            if (lastAttack == weapon.OH_Light_Attack_1)
            {
                SetAttackDamage(weapon.lightDamage);
                animatorHandler.anim.SetFloat("Vertical", 0);
                animatorHandler.anim.SetFloat("Horizontal", 0);
                animatorHandler.PlayTargetAnimation(weapon.OH_Light_Attack_2,
true);
                lastAttack = weapon.OH_Light_Attack_2;
            }
            else if (lastAttack == weapon.OH_Light_Attack_2)
            {
                SetAttackDamage(weapon.lightDamage);
                animatorHandler.anim.SetFloat("Vertical", 0);
                animatorHandler.anim.SetFloat("Horizontal", 0);
                animatorHandler.PlayTargetAnimation(weapon.OH_Light_Attack_1,
true);
                lastAttack = weapon.OH_Light_Attack_1;
            }
        }
    }

    public void HandleLightAttack(WeaponItem weapon)
    {
        if (playerStats.currentStamina <= 0 || playerStats.currentHealth <= 0)
            return;
        weaponSlotManager.attackingWeapon = weapon;
        SetAttackDamage(weapon.lightDamage);
        animatorHandler.anim.SetFloat("Vertical", 0);
        animatorHandler.anim.SetFloat("Horizontal", 0);
        animatorHandler.PlayTargetAnimation(weapon.OH_Light_Attack_1, true);
        lastAttack = weapon.OH_Light_Attack_1;
    }
}

```

```

public void HandleHeavyAttack(WeaponItem weapon)
{
    if (playerStats.currentStamina <= 0 || playerStats.currentHealth <= 0)
        return;
    weaponSlotManager.attackingWeapon = weapon;
    SetAttackDamage(weapon.heavyDamage);
    animatorHandler.PlayTargetAnimation(weapon.OH_Heavy_Attack_1, true);
    lastAttack = weapon.OH_Heavy_Attack_1;
}

public void Block()
{
    animatorHandler.PlayTargetAnimation("block_start", false);
}

private void SetAttackDamage(int damage)
{
    this.GetComponentInChildren<DamageCollider>().currentWeaponDamage =
damage;
}
}

```

Листинг А.15 – Код класса PlayerInventory.cs

```

public class PlayerInventory : MonoBehaviour
{
    WeaponSlotManager weaponSlotManager;

    public WeaponItem rightWeapon;
    public WeaponItem leftWeapon;

    private void Awake()
    {
        weaponSlotManager = GetComponentInChildren<WeaponSlotManager>();
    }

    private void Start()
    {
        weaponSlotManager.LoadWeaponOnSlot(rightWeapon, false);
        weaponSlotManager.LoadWeaponOnSlot(leftWeapon, true);
    }

    public void ChangeRightWeapon(WeaponItem newRightWeapon)
    {
        rightWeapon = newRightWeapon;
        weaponSlotManager.LoadWeaponOnSlot(newRightWeapon, false);
    }
}

```

Листинг А.16 – Код класса PlayerLocomotion.cs

```

public class PlayerLocomotion : MonoBehaviour
{
    CameraHandler cameraHandler;
    public Transform cameraObject;
    InputHandler inputHandler;
    public Vector3 moveDirection;
    PlayerManager playerManager;
    PlayerStats playerStats;
}

```

```

[HideInInspector]
public Transform myTransform;
[HideInInspector]
public AnimatorHandler animatorHandler;

public new Rigidbody rigidbody;
public CapsuleCollider characterCollider;
public CapsuleCollider characterCollisionBlocker;

private Vector3 projectedVelocity;
public float jumpRecoveryTime = 1;
private float jumpRecoveryTimer = 4;

[Header("Ground & Air Detection Stats")]
[SerializeField]
float groundDetectionRayStartPoint = 0.5f;
[SerializeField]
float minDistNeededToFall = 1f;
[SerializeField]
float groundDirectionRayDistance = 0.2f;
LayerMask ignoreForGroundCheck;
public float inAirTimer;

[Header("Unrealized movement stats")]
[SerializeField]
float walkingSpeed = 2;
[Header("Movement Stats")]
[SerializeField]
float movementSpeed = 5;
[SerializeField]
float sprintSpeed = 8;
[SerializeField]
float rotationSpeed = 10;
[SerializeField]
float fallingSpeed = 45;
[SerializeField]
int sprintStaminaDrainage = 20;
[SerializeField]
int rollingStaminaDrainage = 20;
[SerializeField]
int jumpStaminaDrainage = 20;

private void Awake()
{
    cameraHandler = FindObjectOfType<CameraHandler>();
}

void Start()
{
    playerManager = GetComponent<PlayerManager>();
    rigidbody = GetComponent<Rigidbody>();
    inputHandler = GetComponent<InputHandler>();
    animatorHandler = GetComponentInChildren<AnimatorHandler>();
    myTransform = GetComponent<Transform>();
    playerStats = GetComponent<PlayerStats>();
    animatorHandler.Initialize();
    characterCollider = GetComponent<CapsuleCollider>();

    playerManager.isGrounded = true;
    ignoreForGroundCheck = ~(1 << 8 | 1 << 11);
    Physics.IgnoreCollision(characterCollider, characterCollisionBlocker,
true);

```

```

    }

    #region Movement
    Vector3 normalVector;
    Vector3 targetPosition;

    public void HandleRotation(float delta)
    {
        if (inputHandler.lockOnFlag)
        {
            if (inputHandler.sprintFlag || inputHandler.rollFlag)
            {
                Vector3 targetDirection = Vector3.zero;
                targetDirection = cameraHandler.cameraTransform.forward *
inputHandler.vertical;
                targetDirection = cameraHandler.cameraTransform.right *
inputHandler.horizontal;
                targetDirection.y = 0;
                targetDirection.Normalize();

                if (targetDirection == Vector3.zero)
                {
                    targetDirection = transform.forward;
                }

                Quaternion tr = Quaternion.LookRotation(targetDirection);
                Quaternion targetRotation = Quaternion.Slerp(transform.rotation,
tr, rotationSpeed * Time.deltaTime);

                //transform.rotation = targetRotation;
                rigidbody.MoveRotation(targetRotation);
            }
            else
            {
                Vector3 rotationDirection = moveDirection;
                rotationDirection =
cameraHandler.currentLockOnTarget.transform.position - transform.position;
                rotationDirection.y = 0;
                rotationDirection.Normalize();
                Quaternion tr = Quaternion.LookRotation(rotationDirection);
                Quaternion targetRotation = Quaternion.Slerp(transform.rotation,
tr, rotationSpeed * delta);
                rigidbody.MoveRotation(targetRotation);
            }
        }
        else
        {
            Vector3 targetDir = Vector3.zero;
            float moveOverride = inputHandler.moveAmount;

            targetDir = cameraObject.forward * inputHandler.vertical;
            targetDir += cameraObject.right * inputHandler.horizontal;

            targetDir.y = 0;
            targetDir.Normalize();

            if (targetDir == Vector3.zero)
                targetDir = myTransform.forward;

            Quaternion tr = Quaternion.LookRotation(targetDir);
            Quaternion targetRotation = Quaternion.Slerp(myTransform.rotation,
tr, rotationSpeed * delta);

```

```

        rigidbody.MoveRotation(targetRotation);
    }
}

public void HandleMovement(float delta)
{
    projectedVelocity = Vector3.zero;
    if (inputHandler.rollFlag)
        return;

    if (playerManager.isInteracting)
    {
        return;
    }

    moveDirection = cameraObject.forward * inputHandler.vertical;
    moveDirection += cameraObject.right * inputHandler.horizontal;
    moveDirection.y = 0;
    moveDirection.Normalize();

    float speed = movementSpeed;

    if (inputHandler.sprintFlag && inputHandler.moveAmount > 0.5 &&
playerStats.currentStamina > 0)
    {
        playerStats.TakeStaminaDamage(sprintStaminaDrainage *
Time.deltaTime);
        speed = sprintSpeed;
        playerManager.isSprinting = true;
        moveDirection *= speed;
    }
    else
    {
        if (inputHandler.moveAmount < 0.5)
        {
            moveDirection *= walkingSpeed;
            playerManager.isSprinting = false;
        }
        else
        {
            moveDirection *= speed;
            playerManager.isSprinting = false;
        }
    }

    projectedVelocity = Vector3.ProjectOnPlane(moveDirection, normalVector);

    if (inputHandler.lockOnFlag && inputHandler.sprintFlag == false)
    {
        animatorHandler.UpdateAnimatorValues(inputHandler.vertical,
inputHandler.horizontal, playerManager.isSprinting);
    }
    else
    {
        animatorHandler.UpdateAnimatorValues(inputHandler.moveAmount, 0,
playerManager.isSprinting);
    }
}

public void MoveRigidBody(float delta)
{

```

```

        rigidbody.velocity = projectedVelocity;
    }

    public void HandleRollingAndSprinting(float delta)
    {
        if (animatorHandler.anim.GetBool("isInteracting"))
            return;

        if (inputHandler.rollFlag)
        {
            moveDirection = cameraObject.forward * inputHandler.vertical;
            moveDirection += cameraObject.right * inputHandler.horizontal;
            moveDirection.y = 0;
            moveDirection.Normalize();

            if (inputHandler.moveAmount > 0 && playerStats.currentStamina >=
rollingStaminaDrainage)
            {
                playerStats.TakeStaminaDamage(rollingStaminaDrainage);
                animatorHandler.PlayTargetAnimation("rolling", true);

                Quaternion rollRotation =
Quaternion.LookRotation(moveDirection);
                myTransform.rotation = rollRotation;
            }
            else if (inputHandler.moveAmount == 0)
            {
                animatorHandler.PlayTargetAnimation("backStep", true);
            }
        }
    }

    public void HandleFalling(float delta, Vector3 moveDirection)
    {
        playerManager.isGrounded = false;
        RaycastHit hit;
        Vector3 origin = myTransform.position;
        origin.y += groundDetectionRayStartPoint;

        if (Physics.Raycast(origin, myTransform.forward, out hit, 0.4f))
        {
            moveDirection = Vector3.zero;
        }

        if (playerManager.isInAir)
        {
            rigidbody.AddForce(-Vector3.up * fallingSpeed);
            rigidbody.AddForce(moveDirection * fallingSpeed / 10f);
        }

        Vector3 dir = moveDirection;
        dir.Normalize();
        origin = origin + dir * groundDirectionRayDistance;

        targetPosition = myTransform.position;

        Debug.DrawRay(origin, -Vector3.up * minDistNeededToFall, Color.red,
0.1f, false);
        if (Physics.Raycast(origin, -Vector3.up, out hit, minDistNeededToFall,
ignoreForGroundCheck))
        {
            normalVector = hit.normal;
            Vector3 tp = hit.point;

```

```

playerManager.isGrounded = true;
targetPosition.y = tp.y;

if (playerManager.isInAir)
{
    if (inAirTimer > 0.5f)
    {
        Debug.Log("You were in the air for " + inAirTimer);
        animatorHandler.PlayTargetAnimation("land", true);
        inAirTimer = 0;
    }
    else
    {
        animatorHandler.PlayTargetAnimation("Empty", false);
        inAirTimer = 0;
    }

    playerManager.isInAir = false;
}
else
{
    if (playerManager.isGrounded)
    {
        playerManager.isGrounded = false;
    }

    if (playerManager.isInAir == false)
    {
        if (playerManager.isInteracting == false)
        {
            animatorHandler.PlayTargetAnimation("falling", true);
        }

        Vector3 vel = rigidbody.velocity;
        vel.Normalize();
        rigidbody.velocity = vel * (movementSpeed / 2);
        playerManager.isInAir = true;
    }
}

if (playerManager.isGrounded)
{
    if (playerManager.isInteracting || inputHandler.moveAmount > 0)
    {
        myTransform.position = Vector3.Lerp(myTransform.position,
targetPosition, Time.deltaTime / 0.1f);
    }
    else
    {
        myTransform.position = targetPosition;
    }
}

public void HandleJumping()
{
    if (playerManager.isInteracting)
        return;

    jumpRecoveryTimer += Time.deltaTime;

    if (inputHandler.jump_Input)

```

```

        {
            if (inputHandler.moveAmount > 0 && playerStats.currentStamina >=
jumpStaminaDrainage && jumpRecoveryTimer > jumpRecoveryTime)
            {
                jumpRecoveryTimer = 0;
                playerStats.TakeStaminaDamage(jumpStaminaDrainage);
                moveDirection = cameraObject.forward * inputHandler.vertical;
                moveDirection += cameraObject.right * inputHandler.horizontal;
                moveDirection.y = 0;
                animatorHandler.PlayTargetAnimation("jump", true);
                Quaternion jumpRotation =
Quaternion.LookRotation(moveDirection);
                myTransform.rotation = jumpRotation;
            }
        }
    }
    #endregion
}

```

Листинг А.17 – Код класса PlayerStats.cs

```

public class PlayerStats : CharacterStats
{
    PlayerManager playerManager;
    PlayerLocomotion playerLocomotion;
    AnimatorHandler animatorHandler;
    Menu menu;
    new Rigidbody rigidbody;
    public HealthBar healthBar;
    public StaminaBar staminaBar;
    public float staminaRegenerationAmount = 1;
    public float staminaRegenTimer = 0;
    public float healthRegenerationAmount = 1;
    public float healthRegenTimer = 0;
    public string levelName = "SAMPLE";

    private void Awake()
    {
        menu = GetComponent<Menu>();
        animatorHandler = GetComponentInChildren<AnimatorHandler>();
        playerManager = GetComponent<PlayerManager>();
        rigidbody = GetComponent<Rigidbody>();
        playerLocomotion = GetComponent<PlayerLocomotion>();
    }

    void Start()
    {
        SetStats();
    }

    public void SetStats()
    {
        maxHealth = SetMaxHealthFromLvl();
        currentHealth = maxHealth;
        healthBar.SetMaxHealth(Mathf.RoundToInt(maxHealth));

        maxStamina = SetMaxStaminaFromLvl();
        currentStamina = maxStamina;
        staminaBar.SetMaxStamina(Mathf.RoundToInt(maxStamina));
    }

    private float SetMaxHealthFromLvl()

```

```

{
    maxHealth = healthLevel * 10;
    return maxHealth;
}

private float SetMaxStaminaFromLvl()
{
    maxStamina = staminaLevel * 10;
    return maxStamina;
}

public void RegenerateStamina()
{
    if (playerManager.isInteracting || playerManager.isSprinting)
    {
        staminaRegenTimer = 0;
    }
    else
    {
        staminaRegenTimer += Time.deltaTime;
        if (currentStamina < maxStamina && staminaRegenTimer > 1f)
        {
            currentStamina += staminaRegenerationAmount * Time.deltaTime;
            staminaBar.SetCurrentStamina(Mathf.RoundToInt(currentStamina));
        }
    }
}

public void RegenerateHealth()
{
    if (playerManager.takingDamage || playerManager.isDead)
    {
        healthRegenTimer = 0;
    }
    else
    {
        healthRegenTimer += Time.deltaTime;
        if (currentHealth < maxHealth && healthRegenTimer > 2f)
        {
            currentHealth += healthRegenerationAmount * Time.deltaTime;
            healthBar.SetCurrentHealth(Mathf.RoundToInt(currentHealth));
        }
    }
}

public void TakeDamage(int damage, Vector3 strikePostition)
{
    if (playerManager.isInvulnerable)
    {
        return;
    }

    playerManager.takingDamage = true;
    if (playerManager.isBlocking)
    {
        Vector3 blockDirection = strikePostition -
playerManager.transform.position;
        float angle = Vector3.Angle(blockDirection,
playerManager.transform.forward);
        if (angle >= 0 && angle <= 45)
        {
            currentHealth = currentHealth - Mathf.RoundToInt(damage * (100 -
playerManager.damageReduce) / 100);

```

```

    }
    else
    {
        currentHealth = currentHealth - damage;
    }
    healthBar.SetCurrentHealth(Mathf.RoundToInt(currentHealth));
    animatorHandler.PlayTargetAnimation("block_damage", true);
}
else
{
    currentHealth = currentHealth - damage;
    healthBar.SetCurrentHealth(Mathf.RoundToInt(currentHealth));
    animatorHandler.PlayTargetAnimation("damage", true);
}

if (currentHealth <= 0)
{
    currentHealth = 0;
    rigidbody.useGravity = false;
    playerLocomotion.characterCollider.enabled = false;
    animatorHandler.StopAllCoroutines();
    playerManager.isDead = true;
    if (playerManager.isInAir)
        animatorHandler.PlayTargetAnimation("deathInAir", true);
    else
        animatorHandler.PlayTargetAnimation("death", true);
    menu.ShowDeathScreen();
    StartCoroutine(retryCoroutine());
}

IEnumerator retryCoroutine()
{
    menu.deathScreenText.SetText("YOU DIED \n restart in 5 seconds");
    yield return new WaitForSeconds(1);
    menu.deathScreenText.SetText("YOU DIED \n restart in 4 seconds");
    yield return new WaitForSeconds(1);
    menu.deathScreenText.SetText("YOU DIED \n restart in 3 seconds");
    yield return new WaitForSeconds(1);
    menu.deathScreenText.SetText("YOU DIED \n restart in 2 seconds");
    yield return new WaitForSeconds(1);
    menu.deathScreenText.SetText("YOU DIED \n restart in 1 seconds");
    yield return new WaitForSeconds(1);
    SceneManager.LoadScene(levelName);
}

public void TakeStaminaDamage(float damage)
{
    currentStamina = currentStamina - damage;
    staminaBar.SetCurrentStamina(Mathf.RoundToInt(currentStamina));
}
}

```

Листинг А.18 – Код класса WeaponHolderSlot.cs

```

public class WeaponHolderSlot : MonoBehaviour
{
    public Transform parentOverride;
    public bool isLeftHandSlot;
    public bool isRightHandSlot;

    public GameObject currentWeaponModel;
}

```

```

public void UnloadWeapon()
{
    if (currentWeaponModel != null)
    {
        currentWeaponModel.SetActive(false);
    }
}

public void UnloadWeaponAndDestroy()
{
    if (currentWeaponModel != null)
    {
        Destroy(currentWeaponModel);
    }
}

public void LoadWeaponModel(WeaponItem weaponItem)
{
    UnloadWeaponAndDestroy();

    if (weaponItem == null)
    {
        UnloadWeapon();
        return;
    }

    GameObject model = Instantiate(weaponItem.modelPrefab) as GameObject;
    if (model != null)
    {
        if (parentOverride != null)
        {
            model.transform.parent = parentOverride;
        }
        else
        {
            model.transform.parent = transform;
        }

        model.transform.localPosition = Vector3.zero;
        model.transform.localRotation = Quaternion.identity;
        model.transform.localScale = Vector3.one;
    }

    currentWeaponModel = model;
}
}

```

Листинг А.19 – Код класса DamageCollider.cs

```

public class DamageCollider : MonoBehaviour
{
    Vector3 strikePosition = Vector3.zero;
    Collider damageCollider;
    public int currentWeaponDamage = 25;
    public bool enableCollider = false;

    private void Awake()
    {
        damageCollider = GetComponent<Collider>();
        damageCollider.gameObject.SetActive(true);
        damageCollider.isTrigger = true;
    }
}

```

```

        damageCollider.enabled = enableCollider;
    }

    public void EnableDamageCollider()
    {
        damageCollider.enabled = true;
    }

    public void DisableDamageCollider()
    {
        damageCollider.enabled = false;
    }

    private void OnTriggerEnter(Collider collision)
    {
        if (collision.tag == "Player")
        {
            PlayerStats playerStats = collision.GetComponent<PlayerStats>();
            if (this.GetComponentInParent<CharacterManager>() != null)
                strikePostition =
this.GetComponentInParent<CharacterManager>().gameObject.transform.position;
            if (playerStats != null)
            {
                playerStats.TakeDamage(currentWeaponDamage, strikePostition);
            }
        }

        if (collision.tag == "Enemy")
        {
            EnemyData enemyStats = collision.GetComponent<EnemyData>();

            if (enemyStats != null)
            {
                enemyStats.TakeDamage(currentWeaponDamage);
            }
        }
    }
}

```

Листинг А.20 – Код класса EnemyWeaponSlotManager.cs

```

public class EnemyWeaponSlotManager : MonoBehaviour
{
    public WeaponItem rightHandWeapon;
    public WeaponItem leftHandWeapon;

    WeaponHolderSlot rightHandSlot;
    WeaponHolderSlot leftHandSlot;

    DamageCollider rightHandDamageCollider;
    DamageCollider leftHandDamageCollider;

    private void Awake()
    {
        WeaponHolderSlot[] weaponHolderSlots =
GetComponentInChildren<WeaponHolderSlot>();
        foreach (WeaponHolderSlot weaponSlot in weaponHolderSlots)
        {
            if (weaponSlot.isLeftHandSlot)
            {
                leftHandSlot = weaponSlot;
            }
        }
    }
}

```

```

        else if (weaponSlot.isRightHandSlot)
        {
            rightHandSlot = weaponSlot;
        }
    }
}

private void Start()
{
    if (rightHandWeapon != null)
    {
        LoadWeaponOnSlot(rightHandWeapon, false);
    }
    if (leftHandWeapon != null)
    {
        LoadWeaponOnSlot(leftHandWeapon, false);
    }
}

public void LoadWeaponOnSlot(WeaponItem weapon, bool isLeft)
{
    if (isLeft)
    {
        leftHandSlot.LoadWeaponModel(weapon);
        LoadWeaponsDamageCollider(true);
    }
    else
    {
        rightHandSlot.LoadWeaponModel(weapon);
        LoadWeaponsDamageCollider(false);
    }
}

public void LoadWeaponsDamageCollider(bool isLeft)
{
    if (isLeft)
    {
        leftHandDamageCollider =
leftHandSlot.currentWeaponModel.GetComponentInChildren<DamageCollider>();
    }
    else
    {
        rightHandDamageCollider =
rightHandSlot.currentWeaponModel.GetComponentInChildren<DamageCollider>();
    }
}

public void OpenRightDamageCollider()
{
    rightHandDamageCollider.EnableDamageCollider();
}

public void OpenLeftDamageCollider()
{
    leftHandDamageCollider.EnableDamageCollider();
}

public void CloseRightDamageCollider()
{
    rightHandDamageCollider.DisableDamageCollider();
}

public void CloseLeftDamageCollider()

```

```

    {
        leftHandDamageCollider.DisableDamageCollider();
    }
}

```

Листинг А.21 – Код класса WeaponItem.cs

```

public class WeaponItem : Item
{
    public GameObject modelPrefab;
    public bool isUnarmed;

    [Header("One Handed Attack Animations")]
    public string OH_Light_Attack_1;
    public string OH_Light_Attack_2;
    public string OH_Heavy_Attack_1;

    [Header("Stamina costs")]
    public int baseStamina;
    public float lightAttackMultiplier;
    public float heavyAttackMultiplier;

    [Header("Damage Values")]
    public int lightDamage = 25;
    public int heavyDamage = 50;
}

```

Листинг А.22 – Код класса WeaponSlotManager.cs

```

public class WeaponSlotManager : MonoBehaviour
{
    public WeaponItem attackingWeapon;

    WeaponHolderSlot leftHandSlot;
    WeaponHolderSlot rightHandSlot;

    DamageCollider leftHandDamageCollider;
    DamageCollider rightHandDamageCollider;

    PlayerStats playerStats;

    private void Awake()
    {
        playerStats = GetComponentInParent<PlayerStats>();

        WeaponHolderSlot[] weaponHolderSlots =
        GetComponentsInChildren<WeaponHolderSlot>();
        foreach (WeaponHolderSlot weaponSlot in weaponHolderSlots)
        {
            if (weaponSlot.isLeftHandSlot)
            {
                leftHandSlot = weaponSlot;
            }
            else if (weaponSlot.isRightHandSlot)
            {
                rightHandSlot = weaponSlot;
            }
        }
    }
}

```

```

    }

    public void LoadWeaponOnSlot(WeaponItem weaponItem, bool isLeft)
    {
        if (isLeft)
        {
            leftHandSlot.LoadWeaponModel(weaponItem);
            LoadLeftWeaponDamageCollider();
        }
        else
        {
            rightHandSlot.LoadWeaponModel(weaponItem);
            LoadRightWeaponDamageCollider();
        }
    }

    #region Handle Weapon's Damage Collider

    private void LoadLeftWeaponDamageCollider()
    {
        leftHandDamageCollider =
        leftHandSlot.currentWeaponModel.GetComponentInChildren<DamageCollider>();
    }

    private void LoadRightWeaponDamageCollider()
    {
        rightHandDamageCollider =
        rightHandSlot.currentWeaponModel.GetComponentInChildren<DamageCollider>();
    }

    public void OpenRightDamageCollider()
    {
        rightHandDamageCollider.EnableDamageCollider();
    }

    public void OpenLeftDamageCollider()
    {
        leftHandDamageCollider.EnableDamageCollider();
    }

    public void CloseRightDamageCollider()
    {
        rightHandDamageCollider.DisableDamageCollider();
    }

    public void CloseLeftDamageCollider()
    {
        leftHandDamageCollider.DisableDamageCollider();
    }

    #endregion

    #region Handle Weapon's Stamina Drainage
    public void DrainStaminaLightAttack()
    {
        playerStats.TakeStaminaDamage(
            Mathf.RoundToInt(
                attackingWeapon.baseStamina *
                attackingWeapon.lightAttackMultiplier)
        );
    }

    public void DrainStaminaHeavyAttack()

```

```

    {
playerStats.TakeStaminaDamage(Mathf.RoundToInt(attackingWeapon.baseStamina *
attackingWeapon.heavyAttackMultiplier));
    }
    #endregion
}

```

Листинг А.23 – Код класса CharacterStats.cs

```

public class CharacterStats : MonoBehaviour
{
    public int healthLevel = 10;
    public float maxHealth;
    public float currentHealth;

    public int staminaLevel = 10;
    public float maxStamina;
    public float currentStamina;
}

```

СКРИНШОТЫ ТЕСТОВ

Приложение Б

(рекомендуемое)

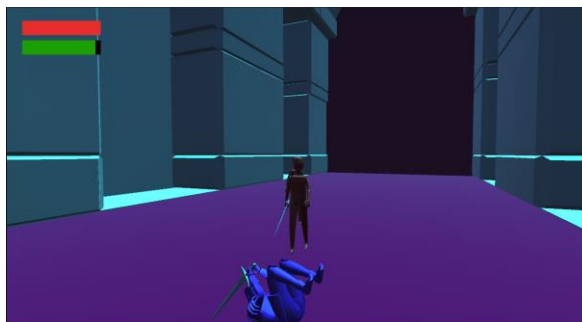


Рисунок Б.1 – Персонаж совершает ерекат



Рисунок Б.2 – Персонаж не делает перекач из-за неверной кнопки действия



Рисунок Б.3 – Персонаж не делает перекач из-за недостатка выносливости



Рисунок Б.4 – Персонаж не делает перекач из-за того, что делает в данный момент другое действие



Рисунок Б.5 – Персонаж не делает перекат из-за того, что не находится на земле



Рисунок Б.6 – Персонаж не делает перекат из-за того, что оправляется после получения урона



Рисунок Б.7 – Персонаж не делает перекат из-за того, что не движется в момент нажатия клавиши переката



Рисунок Б.8 – Персонаж движется



Рисунок Б.9 – Персонаж не движется из-за того, что нажата не верная клавиша действия



Рисунок Б.10 – Персонаж не движется из-за того, что занят другими действиями

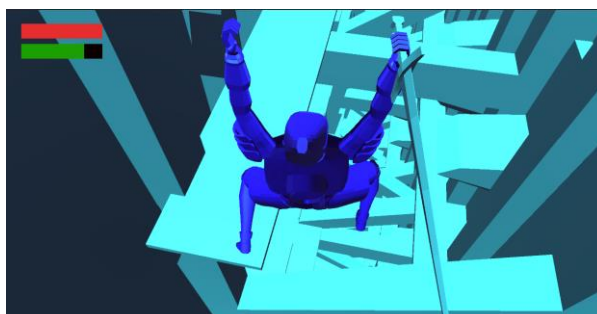


Рисунок Б.11 – Персонаж не движется из-за того, что он не на земле

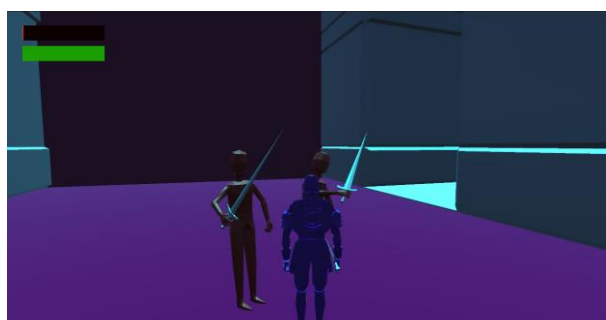


Рисунок Б.12 – Персонаж не движется из-за того, что управляется после получения урона



Рисунок Б.13 – Персонаж делает легкий удар

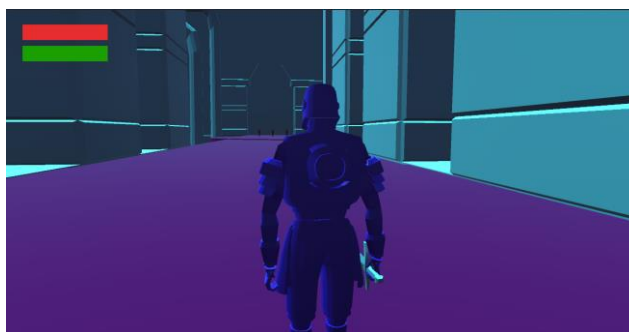


Рисунок Б.14 – Персонаж не делает легкий удар из-за нажатия неверной клавиши действия



Рисунок Б.15 – Персонаж не делает легкий удар из-за того, что он занят другими действиями



Рисунок Б.16 – Персонаж не делает легкий удар из-за того, что он не на земле



Рисунок Б.17 – Персонаж не делает удар из-за того, что он оправляется после получения урона

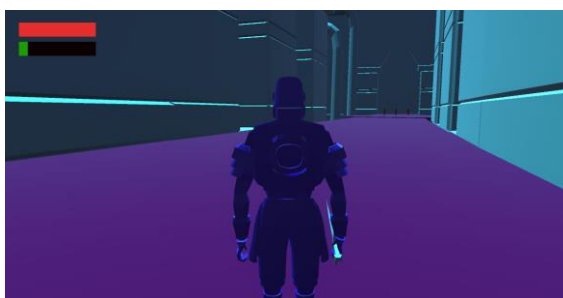


Рисунок Б.18 – Персонаж не делает легкий удар из-за того, что ему недостаточно выносливости

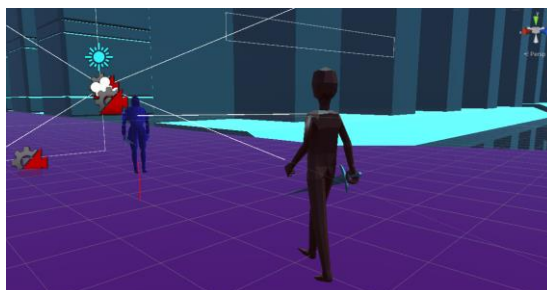


Рисунок Б.19 – Противник перешёл из состояния покоя в состояние преследования, когда главный герой попал в его поле зрения

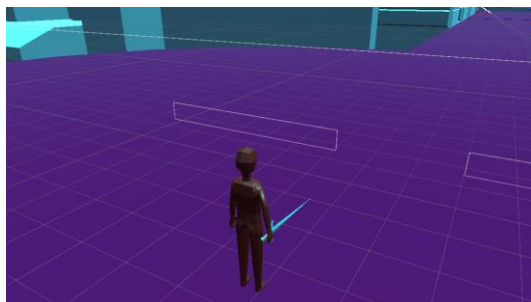


Рисунок Б.20 – Противник остаётся в состоянии покоя, так как пока в его поле зрения не попал главный герой

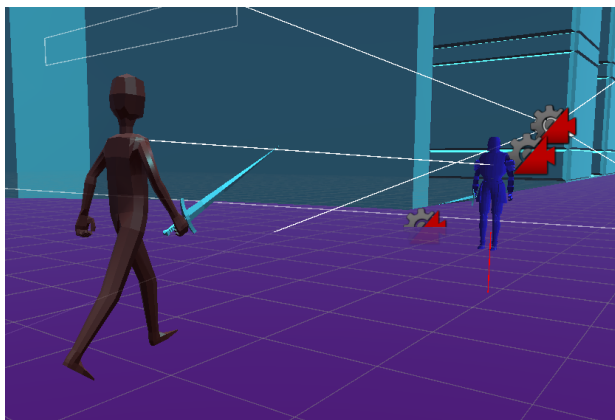


Рисунок Б.21 – Противник преследует главного героя

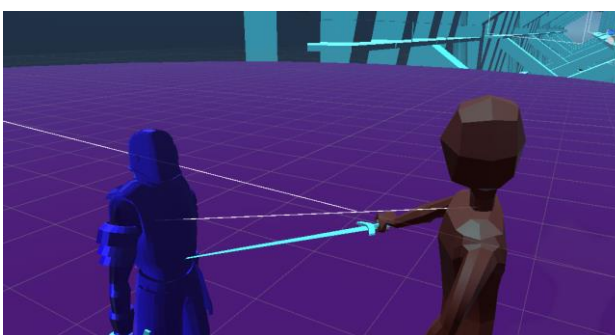


Рисунок Б.22 – Противник закончил преследование главного героя и перешёл в состояние атаки

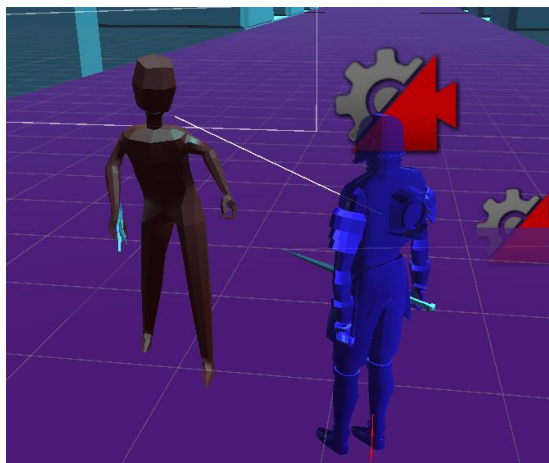


Рисунок Б.23 – Противник закончил преследование главного героя и переходит в состояние боевой стойки

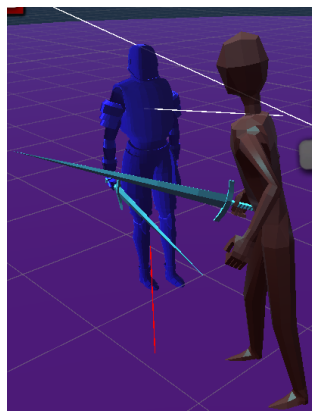


Рисунок Б.24 – Противник переходит из состояния атаки в состояние боевой стойки

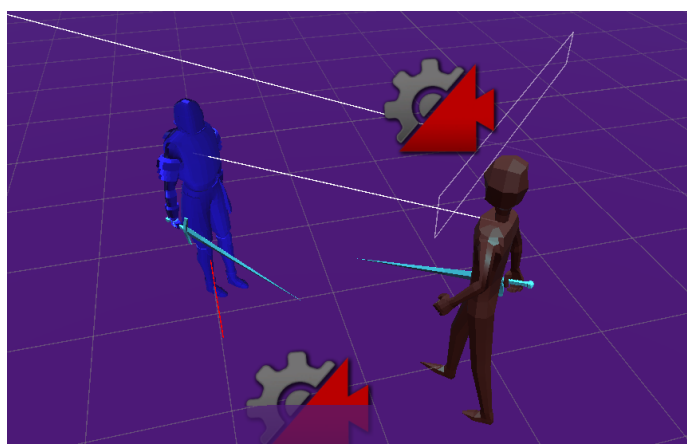


Рисунок Б.25 – Противник переходит из состояния атаки в состояние преследования

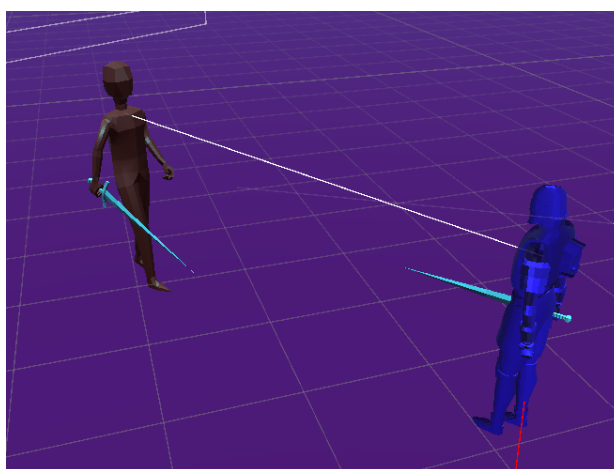


Рисунок Б.26 – Противник переходит из состояния боевой стойки в состояние преследования

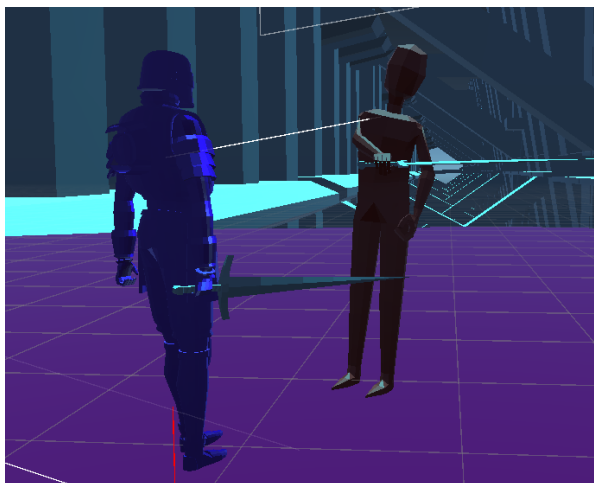


Рисунок Б.27 – Противник переходит из состояния боевой стойки в состояние атаки

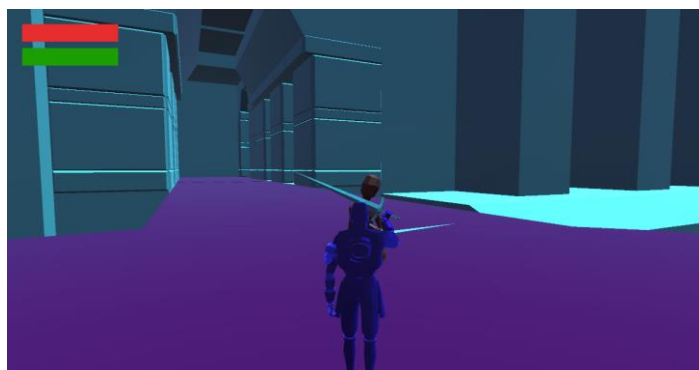


Рисунок Б.28 – Главный герой в состоянии блока

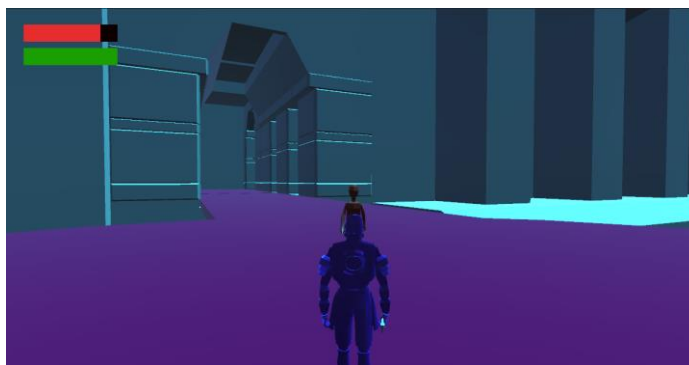


Рисунок Б.29 – Главный герой не ставит блок из-за нажатия неправильной клавиши действия

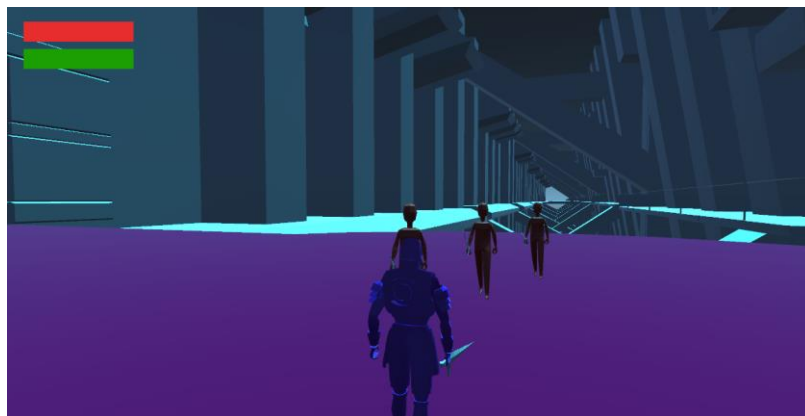


Рисунок Б.30 – Ближайший к главному герою противник захвачен в фокус камеры



Рисунок Б.31 – Никто не захвачен в фокус камеры, так как в поле зрения главного героя нет противников



Рисунок Б.32 – Никто не захвачен в фокус камеры из-за нажатия неверной клавиши действия

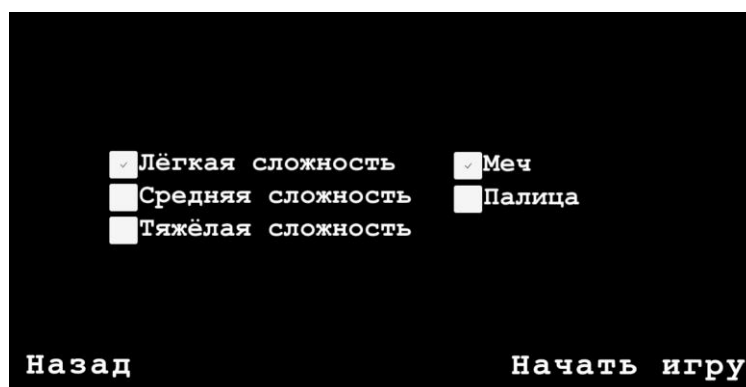


Рисунок Б.33 – После нажатия на кнопку «Новая игра» открылся экран выбора сложности и оружия

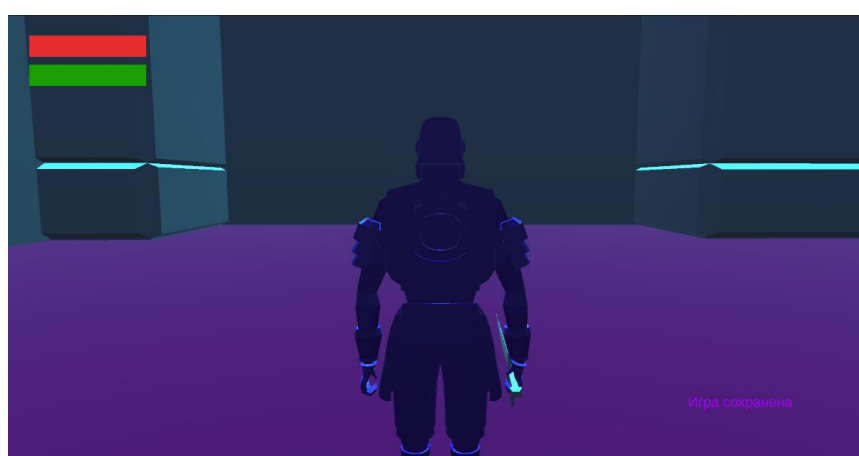


Рисунок Б.34 – После нажатия на кнопку «Продолжить» в главном меню игра загрузилась на последней достигнутой контрольной точке

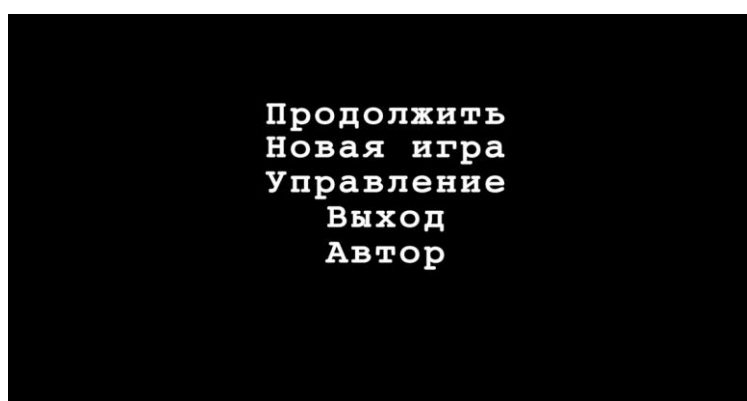


Рисунок Б.35 – После нажатия на кнопку «Назад» на экране выбора оружия и сложности произошёл переход в главное меню игры



Рисунок Б.36 После нажатия на кнопку «Начать игру» на экране выбора оружия и сложности был загружен первый уровень видеоигры

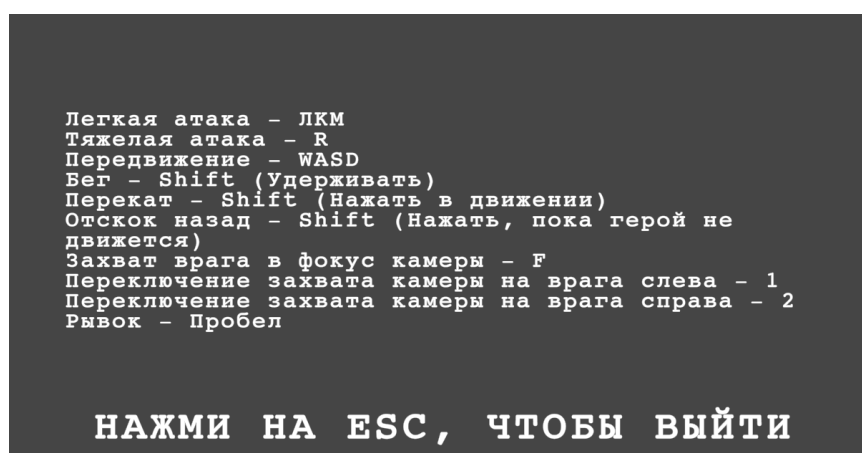


Рисунок Б.37 – После нажатия на кнопку «Управление» в главном меню был открыт экран с информацией об управлении видеоигры

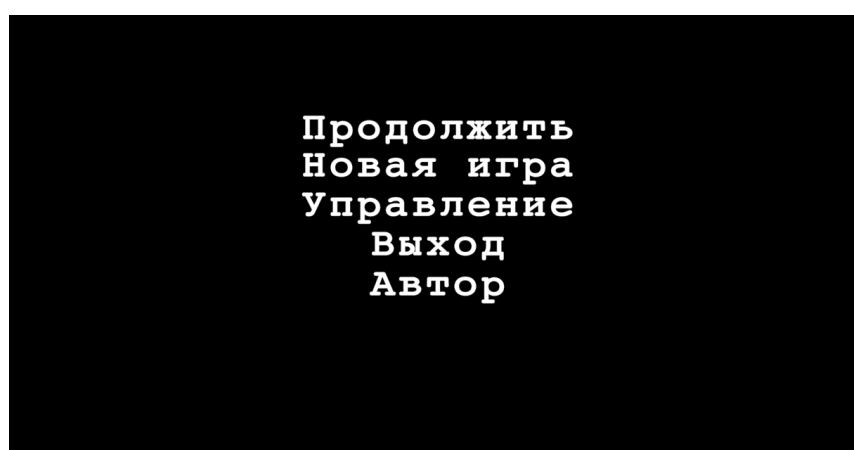


Рисунок Б.38 – После нажатия на клавишу «Escape» на экране с информацией об управлении произошёл переход в главное меню

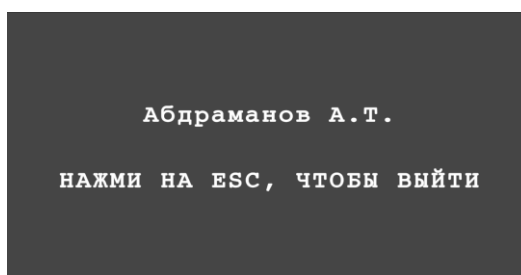


Рисунок Б.39 – После нажатия кнопки «Автор» открылся экран с информацией об авторе

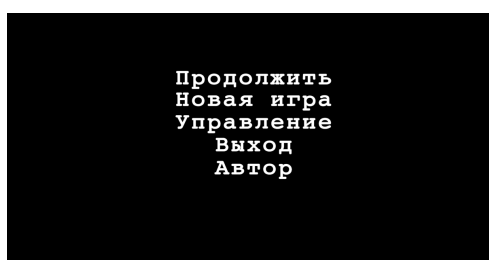


Рисунок Б.40 – После нажатия на клавишу «Escape» на экране с информацией об авторе произошёл переход в главное меню

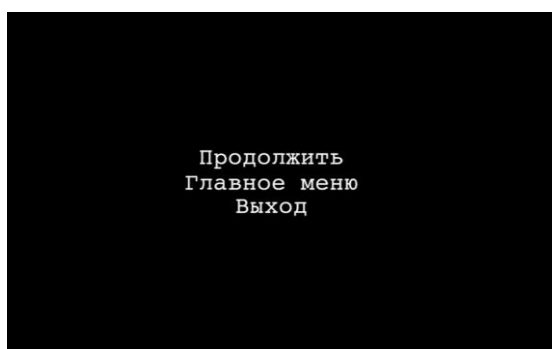


Рисунок Б.41 – После нажатия на клавишу «Escape» на стартовом уровне игры было запущено меню паузы

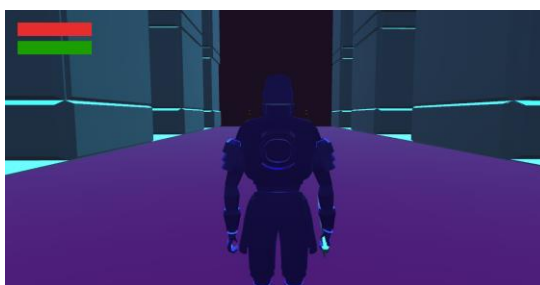


Рисунок Б.42 – После нажатия на кнопку «Продолжить» в меню паузы игра продолжилась

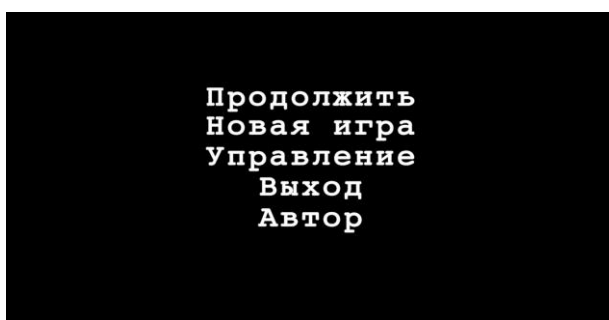


Рисунок Б.43 – После нажатия кнопки «Главное меню» в меню паузы произошёл переход в главное меню

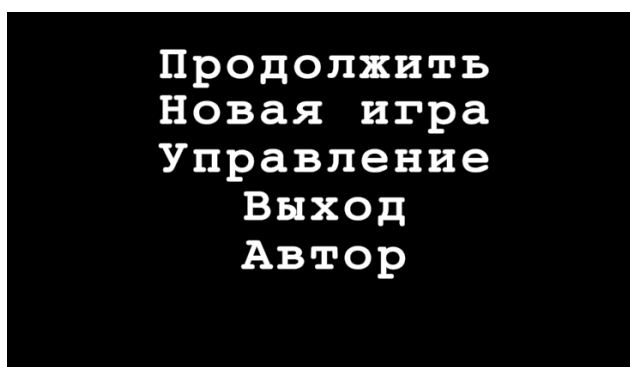


Рисунок Б.44 – Вид главного меню при уменьшении окна игры

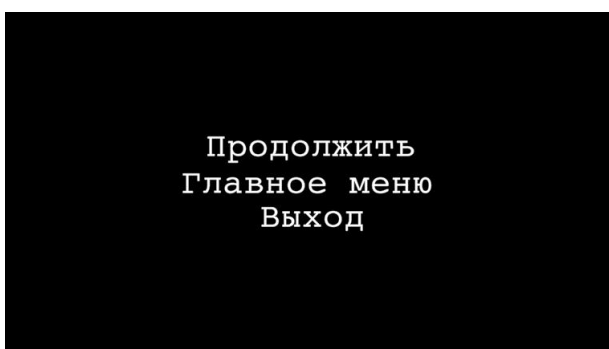


Рисунок Б.45 – Вид меню паузы при уменьшении окна игры

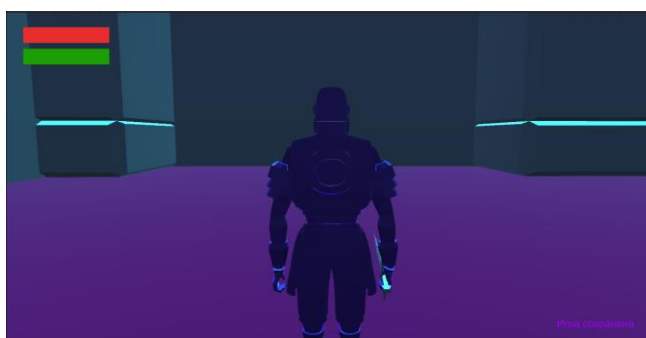


Рисунок Б.46 – Вид игрового HUD



Рисунок Б.47 – Вид игрового HUD при уменьшении окна игры

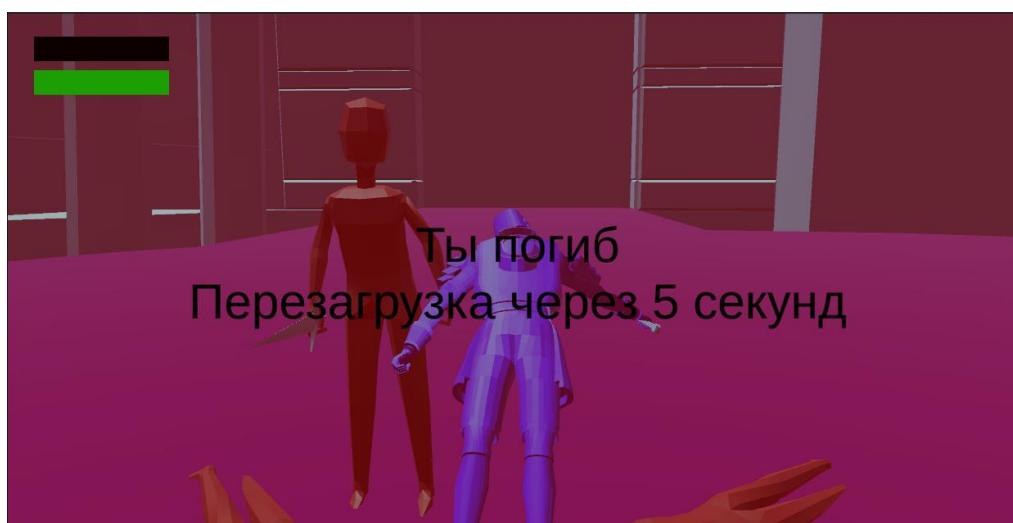


Рисунок Б.48 – Вид экрана смерти

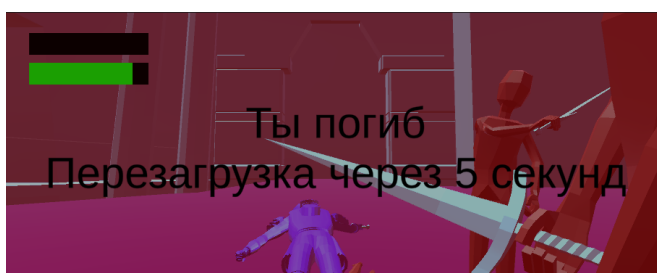


Рисунок Б.49 – Вид экрана смерти при уменьшении окна игры