

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»

Профиль подготовки: «Программное обеспечение средств вычислительной техники и автоматизированных систем.»

БАКАЛАВРСКАЯ РАБОТА

на тему

РАЗРАБОТКА ИГРЫ НА GODOT

ENGINE

Студент

30.04.2025 Рузляев Дмитрий Александрович
(подпись, дата) (ФИО полностью)

Руководитель

5.10.2025 Акифьев И. В.
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр

05.06.2025 Бычков А. С.
(подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 22.05.25 г. № 16)

Заведующий кафедрой

М.А. Митрохин
(подпись) (фамилия, инициалы)

Работа защищена с отметкой отлично (протокол заседания ГЭК от

6 июня 2025 г. № 16)

Секретарь ГЭК

Е.И. Гурин
(подпись) (фамилия, инициалы)

Пенза 2025

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

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
М.А. Митрохин
« 06 » 05 2015 г.

**ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ**

1. Студент: Рузляев Дмитрий Александрович гр. 21ВВП2 ФВТ

2. Тема работы: Разработка игры на Godot Engine

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

3. Руководитель бакалаврской работы: к.э.н., доцент кафедры ВТ ПГУ Акифьев И. В.

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

Разработать игровое приложение для ПК на ОС Windows, и устройств на ОС Android в режиме полной совместимости с использованием Godot Engine. Игра должна быть с интуитивно понятным и адаптивным интерфейсом, который корректно отображается на всех поддерживаемых платформах. Интерфейс должен включать в себя главное меню, экраны настроек, паузы, а также HUD (heads-up display) для отображения ключевой игровой информации, такой как здоровье, очки, и другие показатели. Необходимо обеспечить долгосрочный интерес игроков, в проекте должны быть реализованы элементы, повышающие реиграбельность, путем включения в себя систему случайной генерации NPC, скрытые предметы или бонусы. Управление в игре будет должно быть адаптировано под каждую платформу. На ПК должны использоваться клавиатура и мышь. На Android должны быть предусмотрены сенсорные элементы управления, такие как виртуальные джойстики, кнопки и жесты. Важно обеспечить плавность и отзывчивость управления, чтобы игровой процесс оставался комфортным на любом устройстве.

5. Перечень подлежащих разработке вопросов:

Произвести анализ предметной области: выявить достоинства и недостатки существующих игровых проектов, поставить итоговую задачу. Концептуализировать проект. Изучить работу с выбранным игровым движком. Разработать и протестировать итоговое приложение.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Изучение предметной области	32	06.05.25 -08.05.25	
2	Анализ требований	32	09.05.25 -11.05.25	
3	Выбор технологий, применяемых при проектировании и разработке	30	12.05.25 -14.05.25	
4	Разработка архитектуры приложения	40	15.05.25 -18.05.25	
5	Разработка основной части программы	66	19.05.25 -22.05.25	
6	Тестирование проекта	70	23.05.25 -27.05.25	
7	Оформление пояснительной записки	54	28.05.25 -30.05.25	

Дата выдачи задания "6" мая 2025г.

Руководитель бакалаврской работы

 06.05.25г.
(подпись, дата)

Задание к исполнению принял студент

 06.05.25г.
(подпись, дата)

Работу к защите допустить
Декан факультета

 Л.П. Рогова
05.06.2025 (подпись, дата)

Реферат

Бакалаврская работа содержит 72 листа, 13 рисунков, 1 таблица, 20 источников, 4 приложения.

ДЕСКТОПНОЕ ПРИЛОЖЕНИЕ, МОБИЛЬНОЕ ПРИЛОЖЕНИЕ, ИГРА, ТЕСТИРОВАНИЕ, GODOT ENGINE, WINDOWS, ANDROID

Цель работы – разработать игровое приложение на Godot Engine.

Приложение должно работать в режиме полной совместимости на устройствах с операционными системами Windows и Android с использованием Godot Engine. Игра должна быть с интуитивно понятным и адаптивным интерфейсом, который корректно отображается на всех поддерживаемых платформах. Интерфейс должен включать в себя главное меню, паузы, индикацию для отображения ключевой игровой информации, такой как здоровье, очки, и т.д. В проекте должны быть реализованы системы случайной генерации не игровых персонажей и скрытых предметов. Управление в игре должно быть адаптировано под каждую платформу. На ПК должны использоваться клавиатура и мышь. На Android должны быть предусмотрены сенсорные элементы управления, такие как виртуальные кнопки.

В конечном итоге, было разработано игровое приложение на игровом движке Godot Engine с использованием встроенного языка программирования GDScript, которое удовлетворяет всем поставленным задачам. Была реализована система сражений, сбора монет, сохранений. Само приложение работает на устройствах с ОС Android и Windows

					ПГУ1.090301.40.001 ПЗ		
Изм.	Лист	№ докум.	Подпись	Дата			
Разраб.		Рузляев Д. А.		25.05.25	Лит.	Лист	Листов
Провер.		Акифьев И. В.		25.05.25		3	72
Н. Контр		Бычков А. С.		25.05.25	ФВТ, каф. ВТ гр. 21ВВП2		
Утверд.		Митрохин М. А.		25.05.25			

Разработка игры на Godot Engine
Пояснительная записка

Оглавление

Перечень сокращений и обозначений.....	6
Введение	7
1 Анализ предметной области	9
1.1 Описание игр «Сайд - скроллер»	9
1.2 Описание популярных жанров игр «сайд-скроллеров»	9
1.3 Анализ аналогичных решений	11
2 Проектирование	15
2.1 Требования к приложению	15
2.1.1 Функциональные требования	15
2.1.2 Нефункциональные требования	15
2.1.3 Системные требования	16
2.2 Основной концепт игры.....	16
2.3 UML диаграмма вариантов использования.	17
3 Реализация	19
3.1 Выбор инструментов разработки.....	19
3.2 Архитектура приложения	21
3.3 Использование игровых ассетов	23
3.4 Главный уровень игры	23
3.5 Протагонист	25
3.5.1 Техническая реализация главного героя	26
3.6 Реализация врагов	27
3.7 Пользовательский интерфейс.....	29
4 Тестирование	31
5 Безопасность жизнедеятельности	35
5.1 Правила и техника безопасности при работе с ПК.....	35
Заключение	37
Список использованных источников	38

Приложение А – Uml - диаграммы	40
Приложение Б – Листинг программы	45
Приложение В - Презентация	58
Приложение Г – Скриншоты разработанной игры	67

					ПГУ1.090301.40.001 ПЗ	Лист
						5
Изм.	Лист	№ докум.	Подпись	Дата		

Перечень сокращений и обозначений

Ассет – (от англ. Game Asset) – Игровой ресурс

Инди – (от англ. independent) – независимые произведения

ОС – операционная система

ПК – персональный компьютер

ПО – программное обеспечение

C++ – язык программирования

HUD – (от англ. Head Up Display) – часть визуального интерфейса

IDE – (от англ. Integrated Development Environment) – интегрированная среда разработки

iOS – мобильная операционная система от компании Apple

Modern Pixel Art – стиль графики

NPC – (от англ. Non-Player Character) – персонаж в играх, который не находится под контролем игрока

UI – (от англ. User Interface) – пользовательский интерфейс

UML – (от англ. Unified Modeling Language) – унифицированный язык моделирования

VK – Российская социальная сеть

Введение

История игровой индустрии началась в конце двадцатого века благодаря появлению электронных вычислительных машин, которые использовались для расчётов сложных научных задач. Обрести массового потребителя рынку видеоигр удалось только в 70-х годах XX века с появлением первых аркадных автоматов. Первой коммерчески успешной игрой считается симулятор настольного тенниса «Pong», которая была разработана и выпущена компанией «Atari» в 1972 году. На рисунке 1 приведено её изображение.

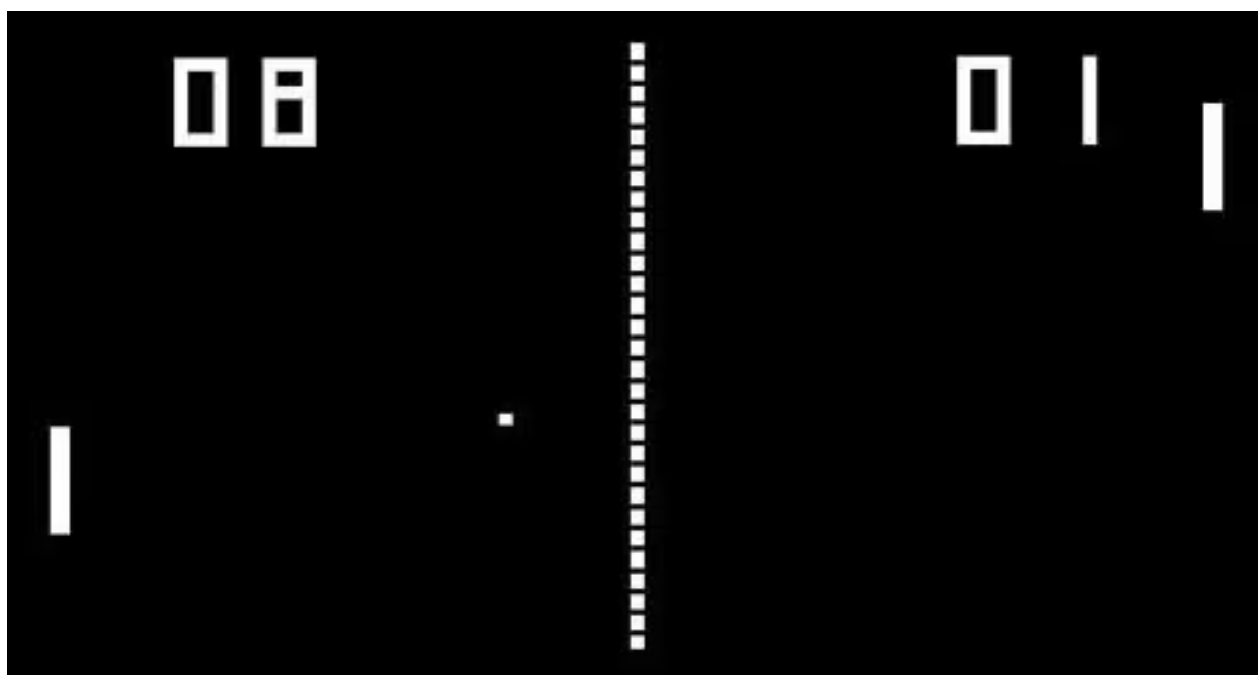


Рисунок 1 – «Atari Pong»

По масштабам игровая индустрия соревнуется с кинопроизводством и значительно опережает его по скорости роста, технологическому развитию, за последние несколько лет. Поэтому создание игр становится все более популярной сферой среди разработчиков по всему миру. Разработка игр требует комплексных знаний и умений, поэтому является хорошей системой для освоения и развития новых навыков.

Игры могут манипулировать восприятием игрока. Так как они незаметно заставляют его поверить во все, что происходит в игре, тем самым создавая

иллюзию реального мира. Таким образом, основной особенностью видеоигр является ассоциативная связь геймера с игровым персонажем [1].

Люди играют в видеоигры по различным причинам. Основными мотивами являются:

- получение эстетического удовольствия от визуальной составляющей игры;
- отвлечение от рутины;
- снятие стресса;
- получение широкого спектра эмоций от положительных до отрицательных;
- развлечение;
- отдых.

По результатам исследования, проводимым игровой платформой и сервисом VK Play, удалось выяснить, как выглядит портрет типичного российского геймера. А также какие гаджеты и игры больше предпочитают в геймсообществе в России.

Большинство респондентов, а именно 85% играет на мобильных устройствах, 63% предпочитают игры на компьютере или ноутбуке, 17% - выбирают игровые консоли. Средний возраст игрока составляет двадцать девять с половиной лет [2].

Целью выпускной квалификационной работы является «Разработка игры на Godot Engine». Для успешного достижения поставленной цели требуется выполнить следующие задачи:

- провести анализ темы, аналогов, постановки задачи;
- спроектировать приложение;
- реализовать и протестировать приложение.

1 Анализ предметной области

1.1 Описание игр «Сайд - скроллер»

Сайд - скроллер (от англ. side-scroller) – это вид компьютерных игр, в которых игровой процесс отслеживается виртуальной камерой, размещенной сбоку по отношению к уровню, а содержащиеся в игре персонажи перемещаются только по горизонтали – влево или вправо. Термин сайд-скроллер обозначает способ визуальной презентации игры, а не жанровую принадлежность. Самыми распространенными жанрами игр, которые имеют данную перспективу, являются «shoot 'em up», а также «beat 'em up».

На рисунке 2 изображен обобщенный пример, выполненный в виде игры сайд - скроллера [3].

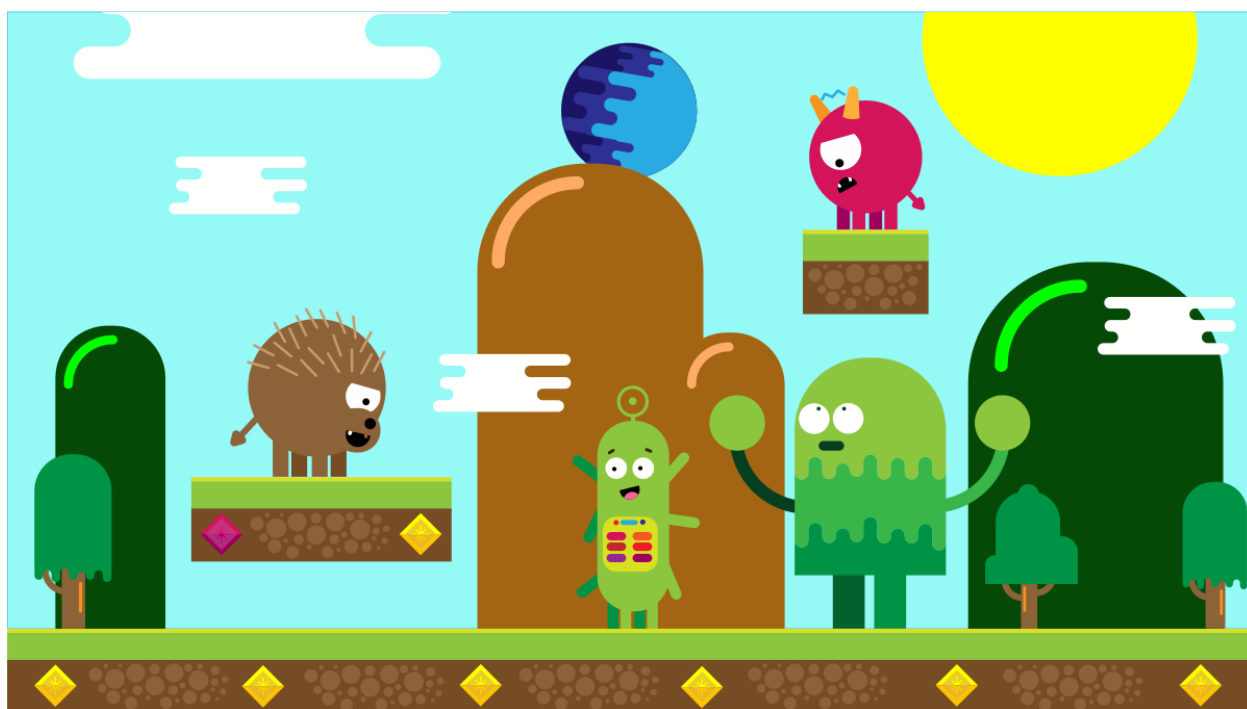


Рисунок 2 – Пример компьютерной игры сайд - скроллера

1.2 Описание популярных жанров игр «сайд-скроллеров»

Жанры «Beat 'em up» («избей их всех») и «Shoot 'em up» («пристрели их всех») — это видеоигры, в которых игрок сражается с большим количеством

врагов, используя при этом холодное или огнестрельное оружие. Это динамичные и простые в освоении жанры, основным различием которых является тип оружия, которое должен использовать игрок.

Первыми значимыми играми стали «Kung-Fu Master» в 1984 году и «Renegade» позднее в 1986. В данных играх преобладает городская атмосфера, сценарий сюжета описывает месть криминальной среде. Данный сценарий стал основой для множества произведений впоследствии. На изображении 3 показан скриншот из игры Renegade.



Рисунок 3 – Игра Renegade 1986 года.

В основном эти игры имеют пиксельный стиль и двухмерную графику, но более современные экземпляры перешли на трехмерное окружение.

Разрабатываемая игра в жанре Beat 'em up будет содержать локацию с бесконечно появляющимися врагами, которые будут наносить урон игроку, пытаясь его уничтожить.

1.3 Анализ аналогичных решений

Для анализа аналогичных проектов были выбраны следующие игры: «Punch kick duck» [4], «Untill we die» [5], а также «Mother Russia Bleeds» [6].

«Punch kick duck» - игра в жанре «Beat 'em up», разработанная Шоном Коулманом и изданная его компанией «Pipsqueak Games». Изначально появилась на платформе iOS 25 января 2023 года. Позднее, 11 февраля, стала доступна для устройств на системе Android.

Игра имеет простое управление. Задача игрока заключается в погоне за очками, которые начисляются за скорость и количество поверженных врагов. Игра стартует с того, что персонаж бесконечно бежит по горизонтали в одну из сторон, а затем вступает в сражение, используя для каждого врага определенный боевой прием.

На рисунке 4 изображен игровой процесс этой игры.

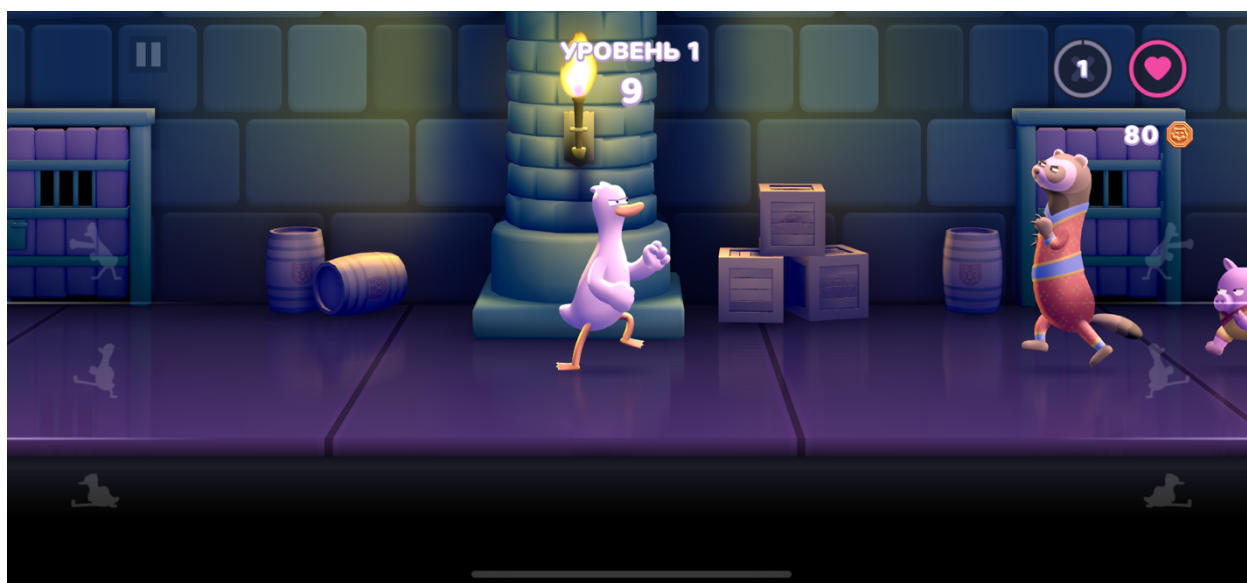


Рисунок 4 – «Punch kick duck»

Главным недостатком игры является её уровень сложности. Неподготовленному игроку будет трудно влиться в игровой процесс, поэтому практически все уровни будут сложным испытанием для него.

«Untill we die» - игра в жанре «shoot 'em up» с элементами из жанра стратегии. Основная цель - защита базы. В роли базы выступает станция метро, а сражаться игроку придется с различными врагами, которые по сценарию занесены метеоритами.

«Untill we die» была разработана и издана Российской студией «Pixeye Games». Изначально, 11 мая 2021 года была выпущена демонстрационная версия игры. Полноценный релиз для платформы Windows состоялся 3 июня 2021 года.

Игра имеет выдающийся визуальный стиль, атмосферу лагеря, а также постоянно нарастающую сложность игры.

На рисунке 5 показан фрагмент из официального трейлера игры.



Рисунок 5 - «Untill we die»

Создателям не удалось заинтересовать аудиторию, из-за чего дальнейшая разработка проекта вскоре был приостановлена.

К недостаткам игры можно отнести:

- отсутствие в релизной версии возможности сохранений;
- плохо реализованная игровая экономика;
- малое количество контента;

- неудовлетворительная реализация логики поведения NPC;
- отсутствие поддержки обновлениями практически сразу после релиза.

«Mother Russia Bleeds» игра в жанре «Beat 'em up» разработанная Французской студией «Le cartel studio» и изданная одним из самых крупнейших издательских агентств для инди-игр - «Devolver Digital».

Игра была выпущена 5 сентября 2016 года для платформ Windows, macOS, Linux. Позднее появилась на игровых консолях, таких как Xbox One, Playstation 4, Nintendo Switch.

Геймплей основывается на множестве боевых столкновений с противниками разной степени сложности на отличных друг от друга уровнях. Стилль игры - СССР альтернативной реальности.

В игре реализована система кооперативного прохождения с другими людьми, что позволяет каждый раз по-новому проходить уровни.

На рисунке 6 представлен скриншот игры с официальной интернет-страницы студии разработчиков [6].



Рисунок 6 - «Mother Russia Bleeds»

Игра имеет противоречивый сюжет, поэтому может показаться для многих игроков слишком жестокой. Игровой процесс, в свою очередь, кажется монотонным несмотря на разнообразие уровней.

Вывод по первому разделу.

В этом разделе представлен анализ аналогов разрабатываемой игры. Рассмотрены три проекта, проведен анализ их преимуществ и недостатков. Все представленные игры получили преимущественно положительные оценки как от профессиональных рецензентов, так и от игроков. Особенно важно учесть их сильные стороны и обратить внимание на моменты, которые пользователи отметили, как требующие улучшения.

					ПГУ1.090301.40.001 ПЗ	Лист
						14
Изм.	Лист	№ докум.	Подпись	Дата		

2 Проектирование

2.1 Требования к приложению

Приложение должно демонстрировать стабильную работу и корректный запуск как на настольных компьютерах, так и на мобильных устройствах.

2.1.1 Функциональные требования

Функциональные требования представляют собой перечень критериев, отражающих ожидания пользователей от программного обеспечения. Они детализируют набор функций и возможностей, необходимых для удовлетворения конечного потребителя. Функциональные требования к разрабатываемому приложению сформированы в результате изучения предметной области.

Приложение должно обеспечивать для пользователя:

- игровой интерфейс;
- игровое меню;
- игровое меню паузы;
- возможность управления персонажем с клавиатуры и мышью, и с сенсорного экрана мобильного устройства;
- возможность взаимодействия с игровыми объектами;
- сохранение игрового прогресса;
- загрузка игрового прогресса из сохранения.

2.1.2 Нефункциональные требования

Нефункциональные требования – это не связанные с поведением приложения характеристики и ограничения, которые приложение должно обеспечивать. Нефункциональные требования охватывают такие аспекты качества продукта как: производительность, безопасность, удобство использования.

Разрабатываемое приложение должно удовлетворять следующим нефункциональным требованиям:

					<i>ПГУ1.090301.40.001 ПЗ</i>	Лист
						15
Изм.	Лист	№ докум.	Подпись	Дата		

- разработка приложения будет осуществлена с использованием игрового движка «Godot Engine»;
- приложение должно работать на операционной системе «Android 12.0» и новее;
- приложение должно работать на операционной системе «Windows 10» и новее.

2.1.3 Системные требования

Системные требования — это спецификации и характеристики, которым должно соответствовать устройство, для обеспечения корректной работы приложения.

Для операционной системы «Android»:

- модель процессора – «MediaTek Helio G36» или аналогичный;
- графический ускоритель – «PoverVR GE8320» или аналогичный;
- оперативная память – 3 гигабайта;
- место на диске – 120 мегабайт.

Для операционной системы «Windows»:

- модель процессора – «AMD Athlon 300GE» или аналогичный;
- интегрированное видеоядро или дискретная видеокарта – «AMD Radeon Vega 3» или аналогичная;
- оперативная память – 4 гигабайта;
- место на диске – 120 мегабайт.

2.2 Основной концепт игры

Жанр игры - «Beat 'em up» с элементами выживания.

Вид игры - «Side – scroller».

Действие разрабатываемой игры будет проходить в средневековом фэнтезийном мире [15]. Общий художественный стиль – «Modern Pixel Art». Выбор пиксельной графики позволяет экономить ресурсы

производительности системы и является простым в исполнении. Игрок управляет персонажем, который должен сражаться с врагами.

Изначально главный герой игры появляется в середине уровня, одновременно с ним постепенно возникают волны врагов с каждой из сторон. Задача игрока продержаться как можно дольше, тем самым повышая количество «прожитых» игровых суток. Так же игрок может собирать монеты с павших врагов.

Проект разрабатывается для операционных систем «Windows» и «Android» на игровом движке «Godot Engine».

2.3 UML диаграмма вариантов использования.

Диаграмма вариантов использования наглядно отображает взаимосвязь между пользователем и функциональностью системы. Она позволяет описать систему на концептуальном уровне и используется для спецификации внешних требований к системе [7].

В представленной диаграмме вариантов использования на рисунке 7 зафиксированы ключевые игровые функции и необходимое количество элементов интерфейса.

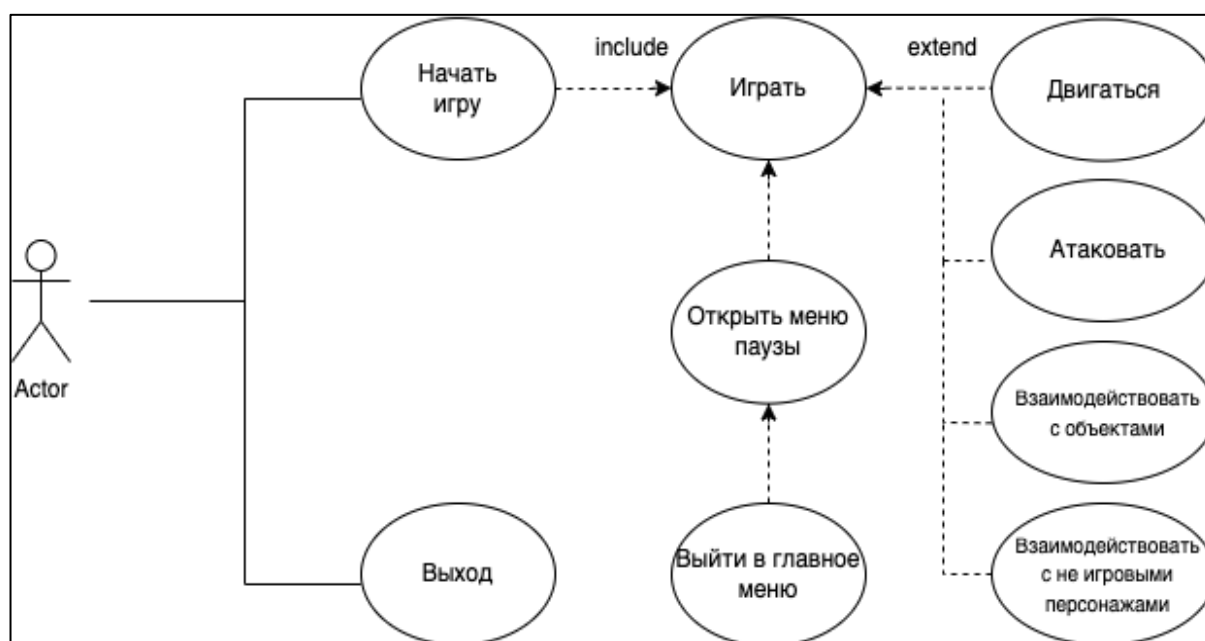


Рисунок 7 - UML диаграмма вариантов использования

Вывод по второму разделу.

Во втором разделе были сформированы ключевые требования к системе. В ней дано описание основного концепта игры и выбран художественный стиль. В соответствии с этими требованиями составлена диаграмма вариантов использования приложения.

					ПГУ1.090301.40.001 ПЗ	Лист
						18
Изм.	Лист	№ докум.	Подпись	Дата		

3 Реализация

3.1 Выбор инструментов разработки

Игровой движок представляет собой программное обеспечение, в состав которого обязательно входит множество компонентов для создания и последующего редактирования игр. Такими компонентами являются комплексные системы редактирования графики, звукового сопровождения, реализации поведения физических объектов, системы анимирования, и интегрированная среда разработки (IDE) для программирования. Основными определяющими признаками игрового движка являются механизмы отрисовки, позволяющие вести разработку сразу на множестве различных платформ и устройств, а также наличие системы управления данными. Благодаря этому значительно упрощается процесс разработки игр.

Game Maker (изначально Animo, Game Maker (до 2011) и GameMaker: Studio), представляет собой игровой движок, созданный Марком Овермарсом в 1999 году. После шестой версии конструктора развитие продолжилось в компании YoYo Games [8]. Он предназначен для разработки игр и предлагает способ оперирования элементами интерфейса, который является уникальным сочетанием простоты визуального программирования (Drag & Drop) [9] и расширенных возможностей, предоставляемых встроенным собственным языком сценариев Game Maker Language. Собственный язык является главным недостатком данного решения из-за своей сложности. Движок специализируется на 2D-проектах, но также поддерживает базовые 3D-функции, хотя их возможности не сравнятся с такими известными платформами, как Unity или Unreal Engine.

Game Maker предоставляет бесплатную версию с определенными ограничениями, например, разработка игры только под платформу Opera GX.games [10], но для расширенных возможностей и профессионального уровня работы доступны платные подписки.

Unity, по статистике использования, популярнейший движок для разработки инди-игр [11], представленный и выпущенный компанией Unity Technologies в июне 2005 года [12]. Несмотря на противоречивую политику компании, движок является одним из лидирующих в мире. Он постоянно развивается, обладает широкой функциональностью и подходит для создания игр различного масштаба, начиная от простых мобильных приложений и заканчивая крупными проектами. В качестве основного языка программирования Unity использует C#, а также предлагает интуитивно понятный визуальный редактор с компонентной системой. Основной проблемой данного решения являются высокие затраты производительности.

Unreal Engine – второй по популярности использования игровой движок, разработанный компанией Epic Games в 1998 году [13]. Первоначально создавался для разработки шутеров от первого лица, но со временем стал универсальным инструментом для игр самых разных жанров. Базируется на C++. Работает на множестве операционных систем и устройств. Его модульная архитектура на основе взаимосвязанных компонентов упрощает портирование проектов между платформами. Кроме того, Unreal Engine предлагает широкие возможности в области графики, звука и сетевых технологий, что делает его одним из самых популярных движков в индустрии. С точки зрения практического применения в разрезе данного проекта, барьером служат избыточность, сложность и требовательность к системе.

Godot engine - это современный игровой движок с открытым исходным кодом, который сочетает простоту освоения данного инструмента с мощными возможностями для создания 2D и 3D игр [14]. Его главное преимущество в том, что он полностью бесплатен.

Благодаря продуманной архитектуре, он обеспечивает эффективную работу с двухмерной графикой без лишних затрат производительности.

Godot уверенно занимает свою нишу как оптимальный инструмент для инди - разработки и экспериментальных проектов.

Для реализации задуманного проекта был выбран игровой движок Godot со встроенным в него языком программирования GDScript. На рисунке 8 показан его интерфейс.

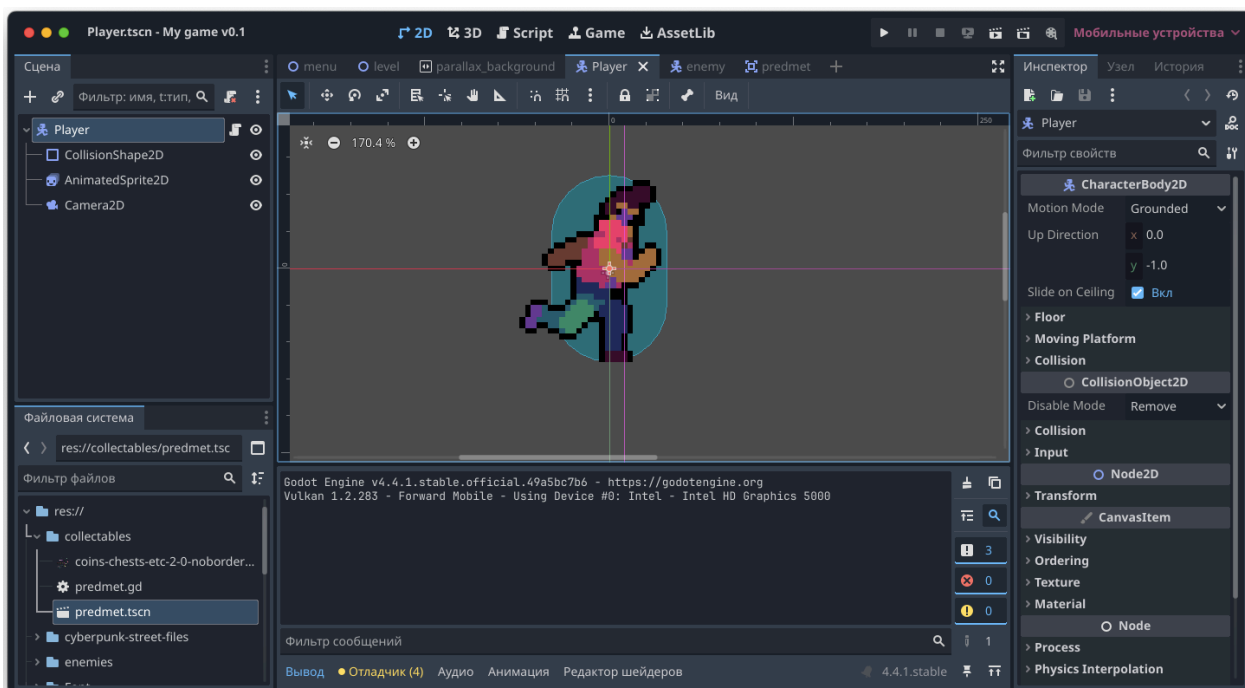


Рисунок 8 – Интерфейс Godot Engine

3.2 Архитектура приложения

После запуска приложения игрок попадает в меню с двумя вариантами выбора: начать игру и выход. При нажатии на кнопку «выход» приложение закроется. При нажатии на кнопку «начать игру» сцена меню меняется на сцену уровня, тем самым запуская игровой процесс, реализованный с помощью скриптов. Скрипт представляет собой компактную программу, предназначенную для выполнения определенной задачи. Скрипт активируется по запросу, обрабатывает данные, затем реализует необходимые операции и завершает свою работу. Ниже приведен перечень используемых в игре скриптов с их кратким описанием.

Скрипт `Player.gd` – скрипт для игрового персонажа с механикой движения и системой анимаций.

Скрипт `Bg.gd` – скрипт для работы подвижного фона.

Скрипт Collectibles.gd – скрипт для создания игровых предметов.

Скрипт Coin.gd – скрипт для создания и воспроизведения движений игрового предмета.

Скрипт Stats.gd – скрипт для обработки и вывода в интерфейс пользователя показателей урона, здоровья и выносливости.

Скрипт Enemy.gd – родительский скрипт, в котором описано поведение врагов.

Скрипт Mushroom.gd – наследуемый скрипт первого врага, в котором хранится сигнал для передачи данных и состояния получение урона, бездействия и смерти.

Скрипт Flying_eye.gd – наследуемый скрипт второго врага, в котором хранится сигнал для передачи данных и состояния получение урона, бездействия и смерти.

Скрипт Mob_health.gd – скрипт для обработки и вывода в UI показателей здоровья врагов.

Скрипт Menu.gd – скрипт для отображения и реализации функциональности пунктов главного меню.

Скрипт Manager.gd – скрипт для работы паузы и вызова ее меню.

Скрипт Lantern2.gd – скрипт для работы искусственного динамического освещения.

Скрипт Global.gd – скрипт для обработки глобальных переменных внутренних систем игры.

Скрипт Day_night.gd – скрипт для смены времени суток по таймеру.

Скрипт Damage_text.gd – скрипт для показа количества единиц урона после нанесения удара.

Скрипт Damage_box.gd – скрипт для работы коллизии во время атаки.

Скрипт Signals.gd – механизм для получения и нанесения урона.

Скрипт Spawner.gd – скрипт для появления врагов на игровом уровне по определенным правилам.

Скрипт Resources.gd – скрипт для обработки глобальной переменной «Монет», и вывода их количества на интерфейс пользователя.

Скрипт При разработке игры будет использован объектно – ориентированный подход программирования.

3.3 Использование игровых ассетов

Современная игровая индустрия широко применяет готовые ассеты (предварительно созданные ресурсы) с целью оптимизации и ускорения процесса разработки игровых проектов. К ним относятся: модели, текстуры, аудиофайлы, а также программные скрипты. Использование подобных решений приобретает особую актуальность в условиях бюджетных ограничений и сжатых сроков разработки

При реализации проекта был применен дифференцированный подход к интеграции готовых решений. Для визуальных компонентов преимущественно использовались ресурсы с сайта itch.io [15]. В частности, авторский комплект, содержащий спрайты - растровые графические объекты [16], игрового персонажа, задние фоны и различные декоративные элементы. Звуковое сопровождение формировалось с помощью библиотек pixabay.com [17].

Ключевым этапом работы с ассетами стал тщательный отбор по двум основным критериям: стилистическая согласованность с художественной концепцией проекта и прозрачность лицензионных условий использования.

3.4 Главный уровень игры

После нажатия на кнопку играть, игрок попадает на главный уровень. Он представляет собой платформу, ландшафт которой состоит из тайловой графики, на которой расположен игровой персонаж, задний фон, а также иные декоративные элементы. Принцип тайловой графики основан на составлении целостных изображений из повторяющихся и стандартизированных фрагментов, что позволяет оптимизировать использование ресурсов.

Для создания иллюзии глубины и устранения монотонности фона применяется эффект параллакса – смещения видимого положения объекта. Реализован через асинхронное перемещение трех фоновых слоев: чем дальше слой, тем медленнее он движется. Рисунок 9 демонстрирует создание эффекта.

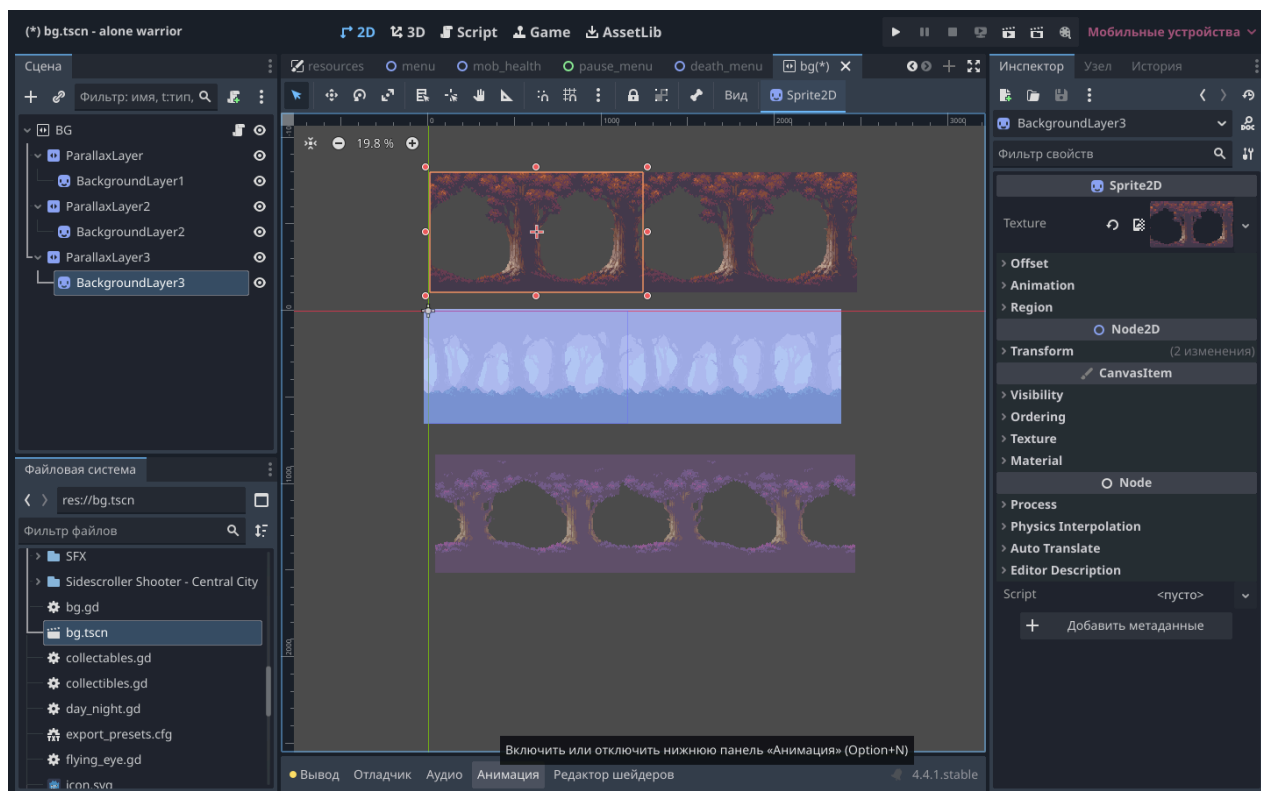


Рисунок 9 – Реализация эффекта параллакса

В целях усиления эстетической составляющей проекта, на уровень были добавлены декоративные элементы, такие как здание с анимацией, портал, из которого появляются враги и фонари, которые зажигаются при наступлении темного времени суток. Так же в игре реализована система смены дня и ночи. В листинге 1 показана программная реализация смены дня и ночи.

Листинг 1. Функции реализации смены времени суток.

```
func _on_day_night_timeout():
    match state:
        MORNING:
            morning_state()
        EVENING:
            evening_state()
    if state < 3:
        state += 1
```



```

else:
    state = MORNING
    day_count += 1
    set_day_text()
    day_text_fade()

func morning_state ():
    var tween = get_tree().create_tween()
    tween.tween_property(light, "energy", 0.2, 20)
    var tween1 = get_tree().create_tween()
    tween1.tween_property(pointlight, "energy", 0, 20)

func evening_state():
    var tween = get_tree().create_tween()
    tween.tween_property(light, "energy", 0.9, 20)
    var tween1 = get_tree().create_tween()
    tween1.tween_property(pointlight, "energy", 1.5,
20)

func day_text_fade():
    animPlayer.play("day_text_fade_in")
    await get_tree().create_timer(3).timeout
    animPlayer.play("day_text_fade_out")

```

3.5 Протагонист

Главный герой проекта выполняет ключевую роль полностью управляемого игроком персонажа являясь центральным элементом в рамках системного игрового процесса. Как основной игровой аватар, он обеспечивает непосредственную связь между игроком и игровым миром через систему взаимодействия. Персонаж относится к архетипу «классического героя» [19], что определяет его базовые характеристики и возможности.

Игровые характеристики персонажа:

- ходьба;
- бег;
- кратковременное ускоряющее скольжение;
- три варианта атаки;
- блокирование атакующих ударов врагов.

Визуальное исполнение протагониста, а именно дизайн и цветовая палитра костюма, наличие щита и клинка соответствуют фэнтезийной

стилистике игры и подчеркивает его. На рисунке 11 представлена модель протагониста.

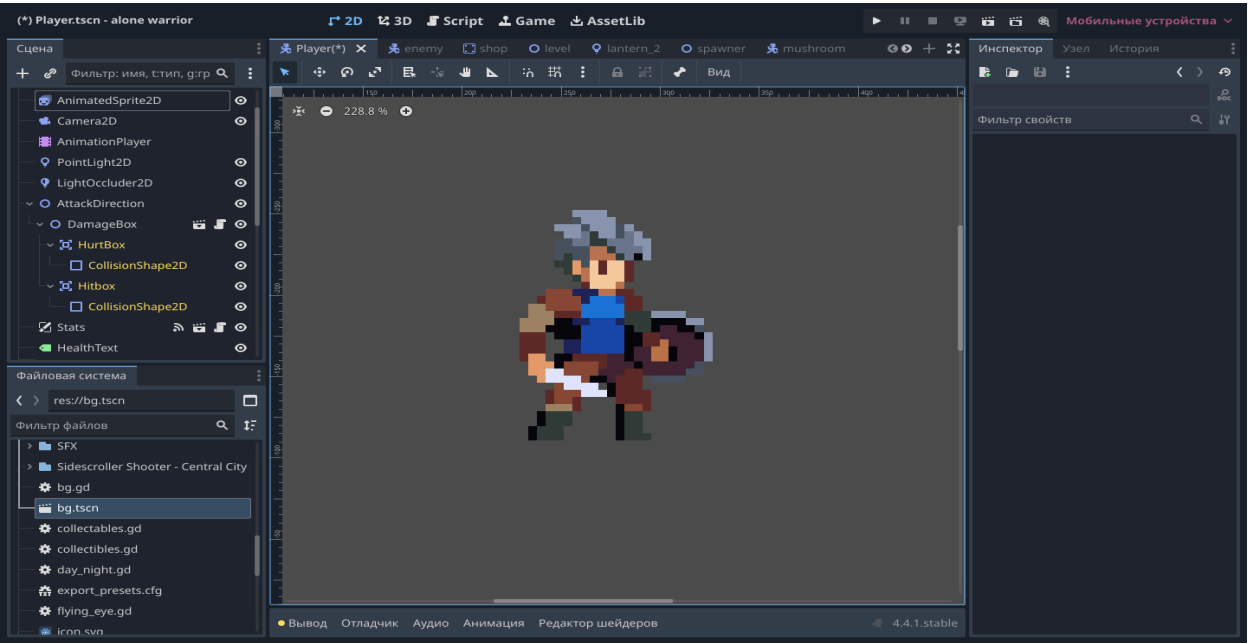


Рисунок 11 – Модель игрового персонажа.

3.5.1 Техническая реализация главного героя

При создании протагониста было решено множество задач. Например, реализация системы физического взаимодействия между объектами, основанная на специализированных зонах коллизий. Данная система включает три типа взаимодействующих областей: столкновения, получения и нанесения урона.

Особое внимание уделено механизму нанесения и получения урона, который реализован через систему взаимодействующих триггерных зон. Все персонажи в игре оснащены данными зонами, которые постоянно активны и отслеживают пересечения с областями других персонажей. Нанесение урона происходит исключительно в момент пересечения областей, что исключает ложные срабатывания и обеспечивает точность игрового процесса. На рисунке 12 показан процесс создания коллизий.

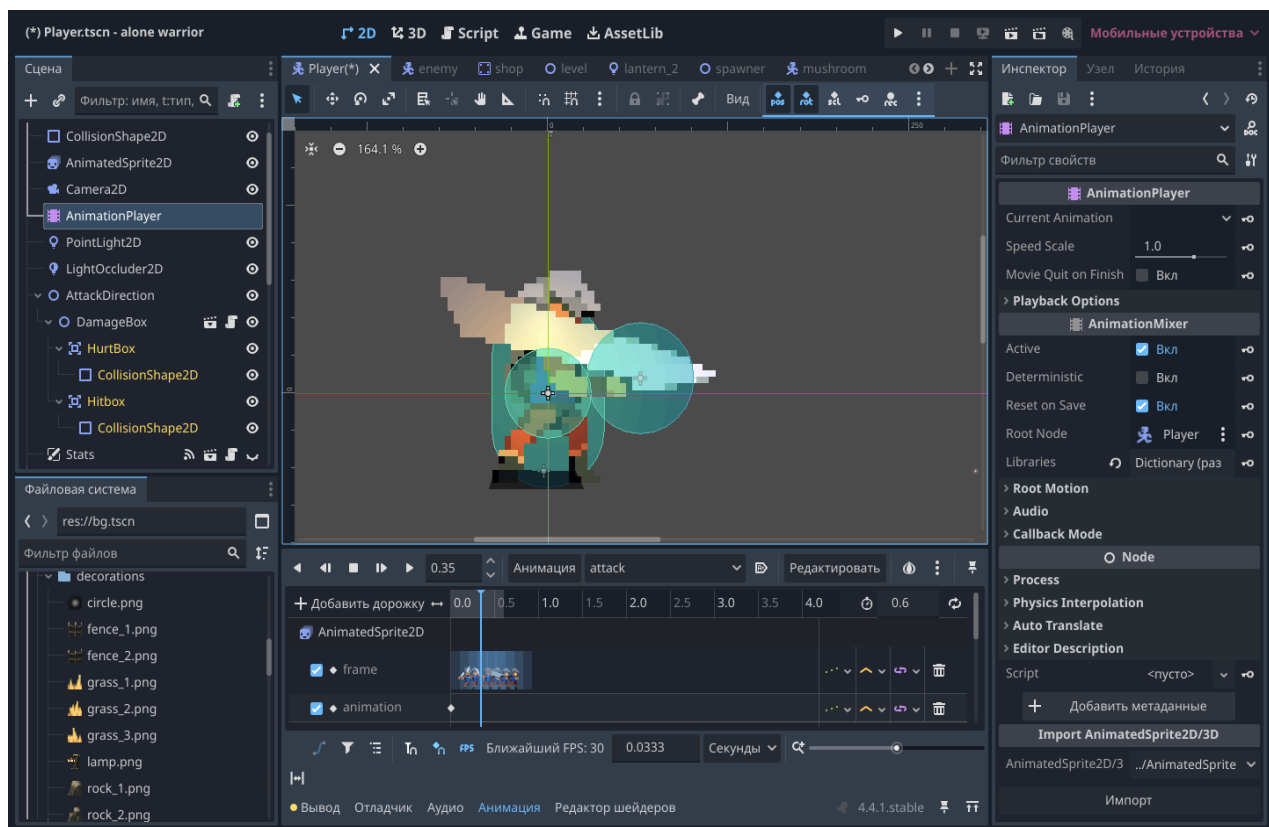


Рисунок 12 – Работа с коллизиями игрока.

3.6 Реализация врагов

Современные игры требуют продуманного поведения неигровых персонажей (NPC), в частности врагов [20]. От них зависит динамика игрового процесса. В данном игровом проекте для реализации поведения врагов используется система состояний, встроенная в Godot Engine. Основу составляет конечный автомат - математическая абстракция, модель, которая может находиться только в одном из конечного числа состояний в каждый конкретный момент времени. В разрабатываемом приложении используется шесть возможных состояний:

- бездействие;
- атака;
- преследование;
- получение урона;
- смерть;

– восстановление.

Механизм переключения состояний реализован с помощью сеттера - переменной `state`, которая автоматически вызывает соответствующие методы – обработчики при изменении значения. В начальном состоянии преследования, персонаж непрерывно отслеживает позицию главного героя через глобальную переменную, вычисляет направление движения и соответствующим образом ориентирует модель врага и точку атаки. При обнаружении игрока в зоне нанесения урона происходит переход в состояние атаки с воспроизведением анимации.

При получении урона активируется состояние получения урона с эффектом отбрасывания, реализованным через твин – анимацию, которая постепенно уменьшает скорость отскока до нуля.

Для добавления непредсказуемости поведению противника была реализована случайная вариативность скорости передвижения, вызываемая через периодический вызов `_on_run_timeout`.

Система взаимодействия с игроком организована через сигналы.

Сигналы — это встроенный в Godot механизм делегирования, позволяющий одному игровому объекту отслеживать изменения в другом, не устанавливая между ними прямой зависимости. При вхождении в зону атаки генерируется сигнал `enemy_attack` с передачей значения урона.

Удаление модели врага с уровня происходит после обязательного завершения анимации смерти через вызов `queue_free`. В листинге 2 показана программная реализация состояний врага.

Листинг 2. Функции состояний врага

```
func idle_state():
    velocity.x = 0
    animPlayer.play("Idle")
    state = CHASE

func attack_state():
    velocity.x = 0
    animPlayer.play("Attack")
    await animPlayer.animation_finished
    state = RECOVER
```

```

func chase_state():
    animPlayer.play("Run")
    direction = (player - self.position).normalized()
    if direction.x < 0:
        sprite.flip_h = true
        $AttackDirection.rotation_degrees = 180
    else:
        sprite.flip_h = false
        $AttackDirection.rotation_degrees = 0
    velocity.x = direction.x * move_speed

func damage_state():
    velocity.x = 0
    damage_anim()
    animPlayer.stop()
    animPlayer.play("Damage")
    await animPlayer.animation_finished
    state = IDLE

func death_state():
    velocity.x = 0
    animPlayer.play("Death")
    await animPlayer.animation_finished
    queue_free()

func recover_state():
    velocity.x = 0
    animPlayer.play("Recover")
    await animPlayer.animation_finished
    state = IDLE

```

3.7 Пользовательский интерфейс

Разработка пользовательского интерфейса является критически важным аспектом создания видеоигр, непосредственно влияющим на удобство взаимодействия игрока с продуктом. На рисунке 13 показан процесс создания интерфейса для игрока, который создан с помощью нодов. Ноды или узлы – фундаментальная единица для создания сцен в игровом движке. Узлы могут выполнять различные функции: хранить данные или предоставлять компонент внутри сцен.

Составляющие интерфейса:

- основное меню (кнопки старт и выход) с анимацией перехода между сценами;

- игровой HUD (счетчик количества дней, монет, показатели выносливости, здоровья игрока и врагов, а также кнопки управления для мобильных устройств);
- систему паузы (кнопки продолжения игры, сохранения и загрузки, а также выхода).

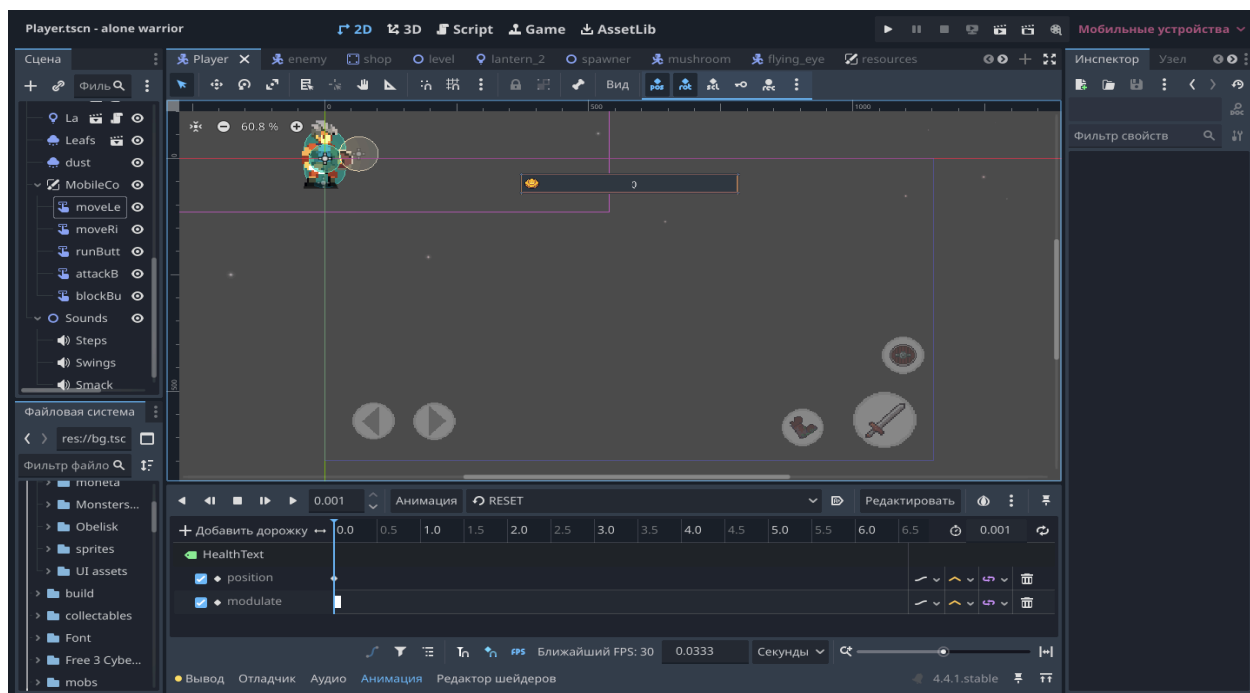


Рисунок 13 – HUD для игрока.

Вывод по третьему разделу.

Проведен анализ различных инструментов разработки и в соответствии с заданием создана 2D игра в жанре «Beat 'em up». Игровой процесс включает управление персонажем, с возможностью атаки и взаимодействия с окружением, а также сохранением и загрузкой прогресса. Противники реализованы с помощью функций преследования протагониста и нанесения урона при контакте, который в свою очередь отражается на соответствующем элементе интерфейса.

4 Тестирование

Тестирование является обязательным компонентом жизненного цикла разработки любого программного обеспечения. Оно показывает соответствие продукта установленным требованиям. Представляет собой систематический процесс верификации и валидации функциональных характеристик разрабатываемого продукта. Набор тестов представлен в таблице 1.

Таблица 1 – функциональное тестирование игры.

№	Название	Действия	Ожидаемый результат
1	Движение.	Для ПК: нажатие клавиш «А», «D», стрелки влево, стрелки вправо. Для мобильных устройств: нажатие соответствующих кнопок на тачскрине.	Главный герой перемещается в заданном направлении, автоматически разворачиваясь в сторону движения.
2	Атака.	Для ПК: нажатие клавиши «Space». Для мобильных устройств: нажатие соответствующих кнопок на тачскрине.	Персонаж проигрывает анимацию атаки.
3	Нанесение урона	Подойти к врагу и атаковать.	Персонаж проигрывает анимацию атаки и наносит урон врагу.

№	Название	Действия	Ожидаемый результат
4	Получение урона	Подойти к врагу и получить удар	Персонаж проигрывает анимацию получения урона. От количества здоровья отнимается численное значение урона.
5	Блокирование ударов	Для ПК: нажатие клавиши «S» Для мобильных устройств: нажатие соответствующих кнопок на тачскрине.	Персонаж проигрывает анимацию блокирования. От количества здоровья отнимается численное значение урона, разделенное на два.
6	Скольжение	Для ПК: нажатие клавиши «S» + «A» или «D». Для мобильных устройств: нажатие соответствующих кнопок на тачскрине.	Персонаж проигрывает анимацию скольжения и кратковременно ускоряется.

№	Название	Действия	Ожидаемый результат
7	Бег	Для ПК: нажатие клавиш «A», «D», стрелки влево, стрелки вправо + клавиши «Left Shift». Для мобильных устройств: нажатие соответствующих кнопок на тачскрине.	Персонаж проигрывает анимацию бега и ускоряется.
8	Смерть игрока	Получить от врага 120 единиц урона	Персонаж проигрывает анимацию смерти, и игра переходит в главное меню
9	Пауза	Для ПК: нажатие клавиши «esc». Для мобильных устройств: нажатие соответствующих кнопок на тачскрине.	Игра останавливается и показывает меню паузы
10	Сохранение и загрузка	Для ПК: нажатие клавиши «esc». Затем нажатие кнопки «сохранить». После чего снова зайти в меню паузы и нажать кнопку «загрузить». Для мобильных устройств: нажатие соответствующих кнопок на тачскрине.	Создается файл с данными, а затем эти данные загружаются.

№	Название	Действия	Ожидаемый результат
11	Подбирание монеты	Подойти вплотную к монете.	Монета подскакивает, а затем плавно исчезает. Монета записывается в соответствующий элемент интерфейса.
12	Убийство врага	Нанести врагу смертельный удар	Враг проигрывает анимацию смерти и исчезает с уровня
13	Выход в главное меню	Нажать кнопку «выход» в меню паузы	Открывается главное меню.
14	Выход из игры	Нажать кнопку «выход» в главном меню	Приложение закрывается.

Вывод по четвертому разделу.

В результате проведения функционального тестирования установлено полное соответствие между ожидаемым и реальным поведением программы. Подтвердилась работоспособность всех элементов проекта. Все тестовые сценарии были пройдены, критических ошибок не выявлено.

5 Безопасность жизнедеятельности

Для обеспечения безопасности жизни и здоровья специалистов, включая разработчиков и прочих категорий работников, проводящих значительное количество времени за средствами вычислительной техники, разработаны и внедрены соответствующие нормативные документы и правила по охране труда. Несоблюдение данных рекомендаций может привести к негативным последствиям как физического характера, так и к ущербу имуществу.

5.1 Правила и техника безопасности при работе с ПК

При взаимодействии с компьютерной техникой пользователь может подвергаться воздействию различных факторов производственной среды, включая:

- потенциальную опасность возгорания оборудования;
- риск поражения электрическим током;
- вероятность механических повреждений;
- шумовое воздействие;
- электромагнитное воздействие.

Для минимизации указанных рисков необходимо соблюдение установленных правил эксплуатации и организации рабочего места.

Одним из важных аспектов является организация освещённости рабочей зоны. Уровень освещения должен быть достаточным, исключая появление бликов на экране монитора и попадание светового потока непосредственно в глаза работающего. Предпочтительно сочетание естественного и искусственного освещения, при этом яркость не должна вызывать дискомфорта.

Рабочее место должно быть оборудовано эргономичной мебелью. Стул должен иметь возможность регулировки по высоте и углу наклона спинки, что позволяет сохранять оптимальную позу:

- выпрямленные ноги;

- ступни, полностью контактирующие с полом;
- спина плотно прилегает к спинке сиденья.

Монитор следует размещать на расстоянии вытянутой руки, с углом наклона экрана порядка 30 градусов. Такое расположение способствует снижению нагрузки на позвоночный столб, мышцы шеи и верхних конечностей, а также помогает предотвратить развитие хронических заболеваний опорно-двигательного аппарата.

Категорически запрещено использовать технические устройства в случае наличия видимых повреждений. Работник обязан систематически проверять исправность оборудования и не оставлять его в рабочем состоянии без присмотра. Для уменьшения уровня электромагнитного излучения мониторы рекомендуется устанавливать задней стороной к стене, располагая рабочие места вдоль перегородок.

Для профилактики переутомления и стрессовых состояний необходимо организовывать регулярные перерывы в течение рабочего дня. При стандартной восьмичасовой смене суммарная продолжительность перерывов должна составлять от тридцати минут до полутора часов. Кроме того, рекомендуется проведение кратковременных физкультурных пауз, направленных на расслабление мышц, профилактику заболеваний глаз и улучшение общего самочувствия.

Вывод по пятому разделу.

В данном разделе были рассмотрены основные рекомендации и инструкции, регулирующие безопасную эксплуатацию персональных компьютеров и мобильных устройств.

Заключение

В результате выполнения работы была успешно реализована поставленная цель – разработка игры на Godot Engine.

В процессе разработки:

- проведён анализ предметной области и существующих аналогов;
- выполнено проектирование структуры приложения;
- осуществлена программная реализация;
- проведено комплексное тестирование.

В ходе программной реализации проекта были освоены:

- технологии разработки игр в Godot Engine;
- основы программирования на GDScript;
- принципы работы с 2D - графикой;
- основы игрового дизайна.

В перспективе планируется усовершенствовать механизм сохранения и загрузки игры, добавить новые эффекты, расширить аудио сопровождение и визуальную составляющую, за счет использования собственных уникальных ресурсов. Особое внимание будет уделено процессу распространения игры в различных магазинах приложений.

					<i>ПГУ1.090301.40.001 ПЗ</i>	Лист
						37
Изм.	Лист	№ докум.	Подпись	Дата		

Список использованных источников

1. Учёные ННГУ впервые исследовали связь типа личности геймера с его персонажем [Электронный ресурс] URL: <http://www.unn.ru/site/about/news/uchjonye-nngu-vpervye-issledovali-svyaz-tipa-lichnosti-gejmera-s-ego-personazhem> (дата обращения: 02.05.2025 г.).
2. Портрет российского геймера [Электронный ресурс] URL: <https://vk.com/@play-portret-rossiiskogo-geimera-v-2022-godu> (дата обращения: 04.05.2025 г.).
3. Банк стоковых изображений [Электронный ресурс] URL: <https://pixabay.com/en/illustration-videogame-graphics-1736462/> (дата обращения: 04.05.2025 г.).
4. Сайт игры «Punch Kick Duck» [Электронный ресурс] URL: <https://pipsqueakgames.com> (дата обращения: 04.05.2025 г.).
5. Сайт игры «Until We Die» [Электронный ресурс] URL: <https://pixeye.games> (дата обращения: 05.05.2025 г.).
6. Сайт игры «Mother Russia Bleeds» [Электронный ресурс] URL: <https://lecartelstudio.fr> (дата обращения: 06.05.2025 г.).
7. Фаулер М. UML. Основы. // Символ Плюс, 2004. – 192 с.
8. Сайт игрового движка «GameMaker» [Электронный ресурс] URL: <https://gamemaker.io/ru> (дата обращения: 08.05.2025 г.).
9. Болгов А. Е., Перетыкина К. Р. Создание Drag-And-Drop элемента объектной модели документа для загрузки файлов на сервер // Научно-практические исследования. — 2020. — 8-11 с.
10. Сайт игрового ресурса Opera GX Games [Электронный ресурс] URL: <https://gx.games/ru/> (дата обращения: 10.05.2025 г.).
11. Сайт игрового ресурса SteamDB [Электронный ресурс] URL: <https://steamdb.info/tech/> (дата обращения: 12.05.2025 г.).
12. Сайт игрового движка Unity [Электронный ресурс] URL: <https://unity.com/ru> (дата обращения: 14.05.2025 г.).

13. Сайт игрового движка Unreal Engine [Электронный ресурс]
URL: <https://www.unrealengine.com/en-US> (дата обращения: 15.05.2025 г.).

14. Сайт игрового движка Godot Engine [Электронный ресурс]
URL: <https://godotengine.org> (дата обращения: 15.05.2025 г.).

15. Банк игровых ассетов [Электронный ресурс] URL: <https://itch.io>
(дата обращения: 19.05.2025 г.).

16. Сайт игрового ассета [Электронный ресурс] URL:
<https://brullov.itch.io/oak-woods> (дата обращения: 16.05.2025 г.).

17. Банк стокового аудиосопровождения [Электронный ресурс]
URL: <https://pixabay.com/> (дата обращения: 17.05.2025 г.).

18. Банк игровых ассетов [Электронный ресурс] URL: <https://itch.io>
(дата обращения: 19.05.2025 г.).

19. Николина Н. А., Литовская М. А., Купина Н. А. Массовая литература сегодня // Флинта, 2009. — 425 с.

20. Джейсон Шрейер, Кровь, пот и пиксели. Обратная сторона индустрии видеоигр // Бомбора, 2024 — 226-254 с.

Приложение А

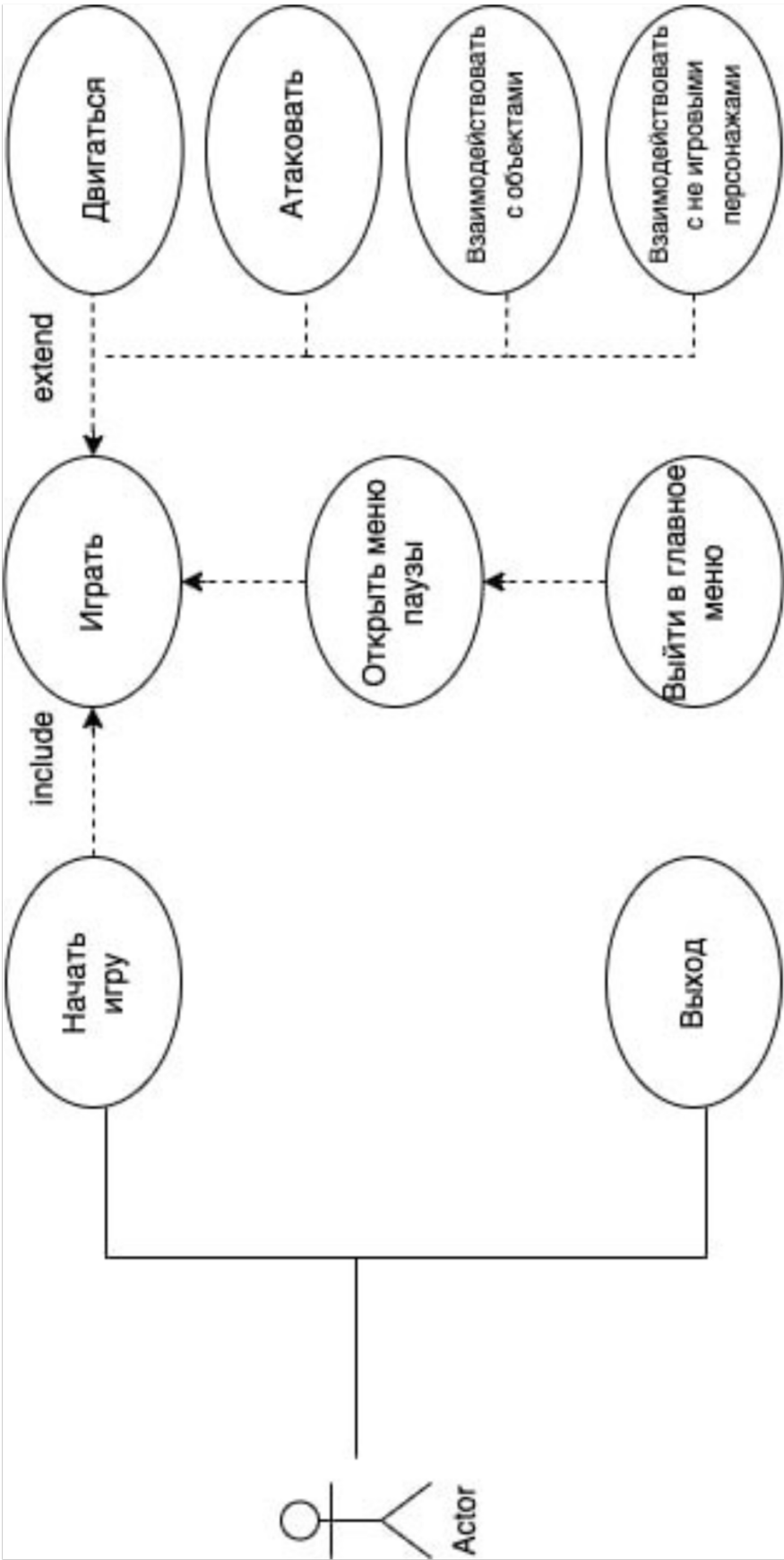
(Обязательное)

Разработка игры на Godot Engine

UML – диаграммы

					<i>ПГУ1.090301.40.001 ПЗ</i>	Лист
						40
Изм.	Лист	№ докум.	Подпись	Дата		

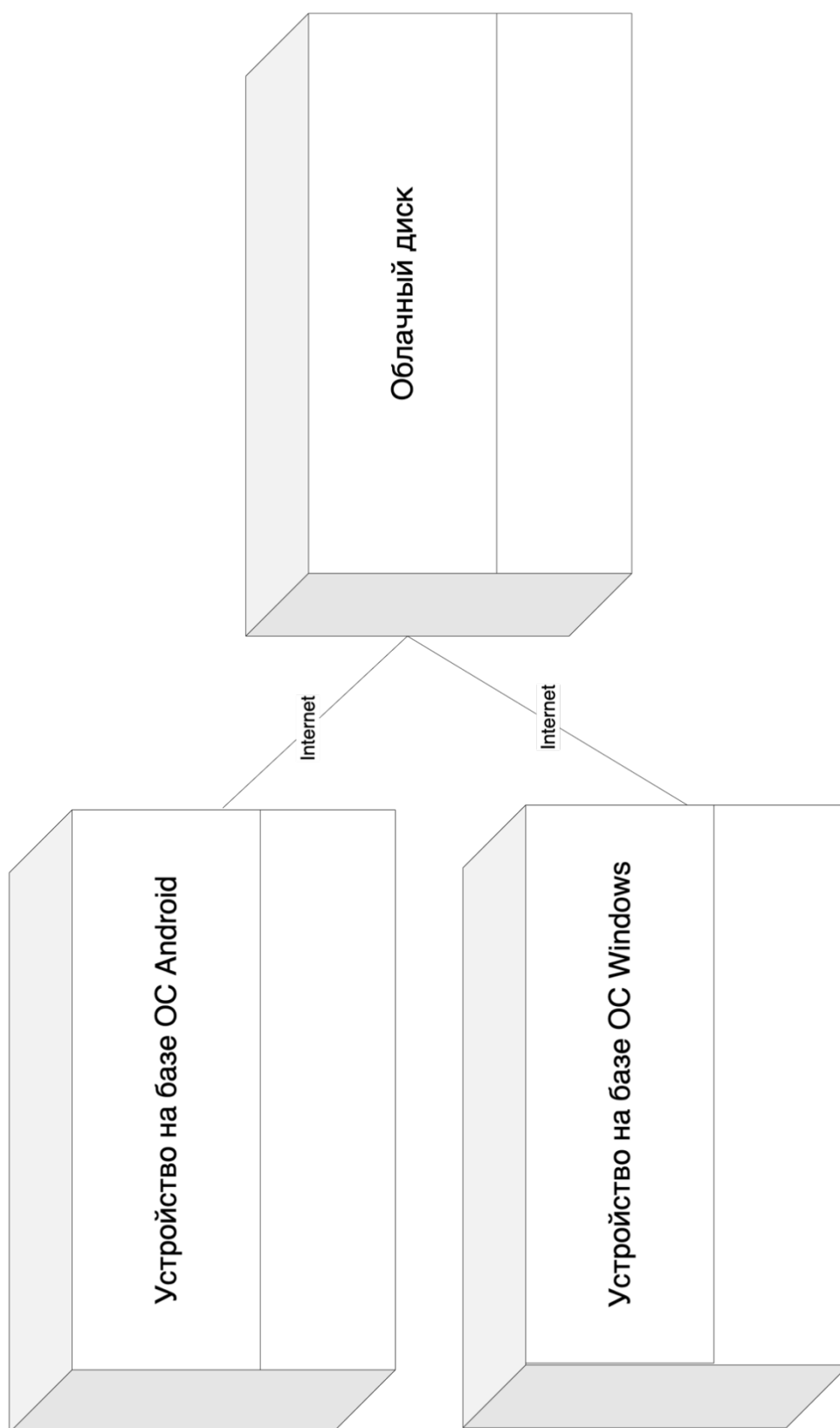
Приложение А.1 - Диаграмма вариантов использования



ПГУ 1.090301.40.002 ПЗ									
Разработка игры на Godot Engine									
Имя	Имя	Имя	Имя	Имя	Имя	Имя	Имя	Имя	Имя
Имя	Имя	Имя	Имя	Имя	Имя	Имя	Имя	Имя	Имя
Имя	Имя	Имя	Имя	Имя	Имя	Имя	Имя	Имя	Имя
Имя	Имя	Имя	Имя	Имя	Имя	Имя	Имя	Имя	Имя
Имя	Имя	Имя	Имя	Имя	Имя	Имя	Имя	Имя	Имя
Имя	Имя	Имя	Имя	Имя	Имя	Имя	Имя	Имя	Имя
Имя	Имя	Имя	Имя	Имя	Имя	Имя	Имя	Имя	Имя
Имя	Имя	Имя	Имя	Имя	Имя	Имя	Имя	Имя	Имя
ФОРМ. № 2.00012									

Изм.	Лист	№ докум.	Подпись	Дата

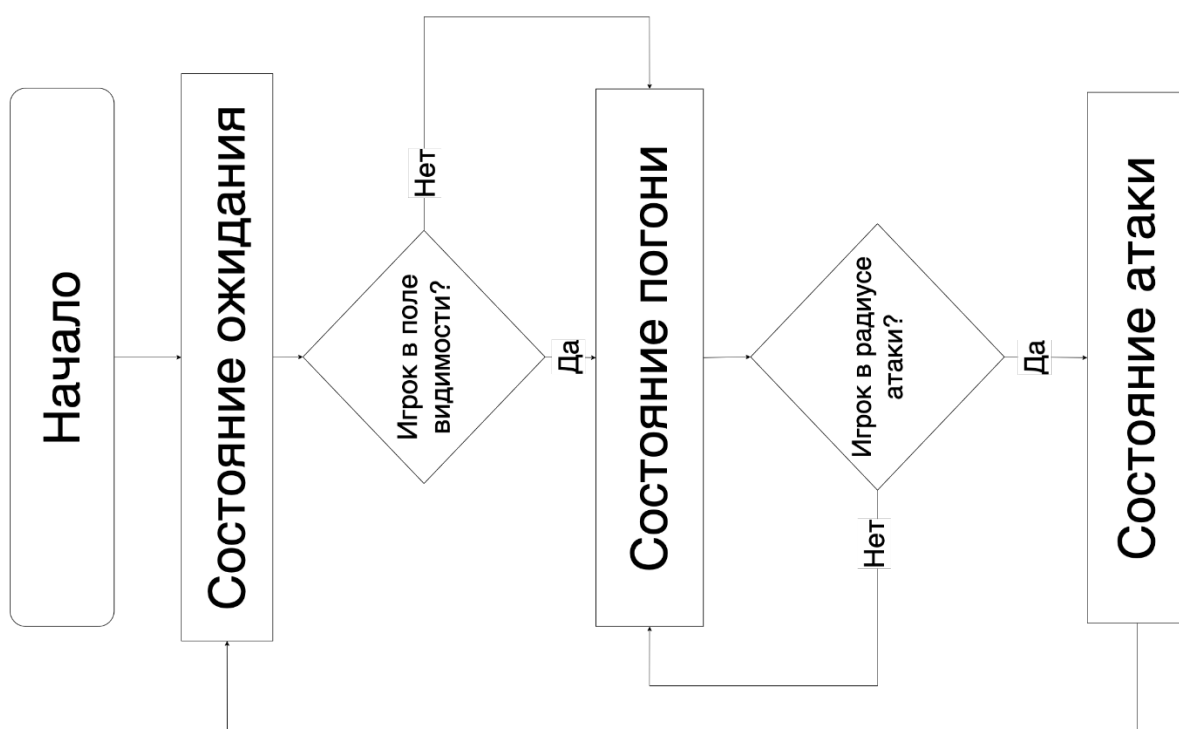
Приложение А.2 - Диаграмма разворачивания

[illegible]

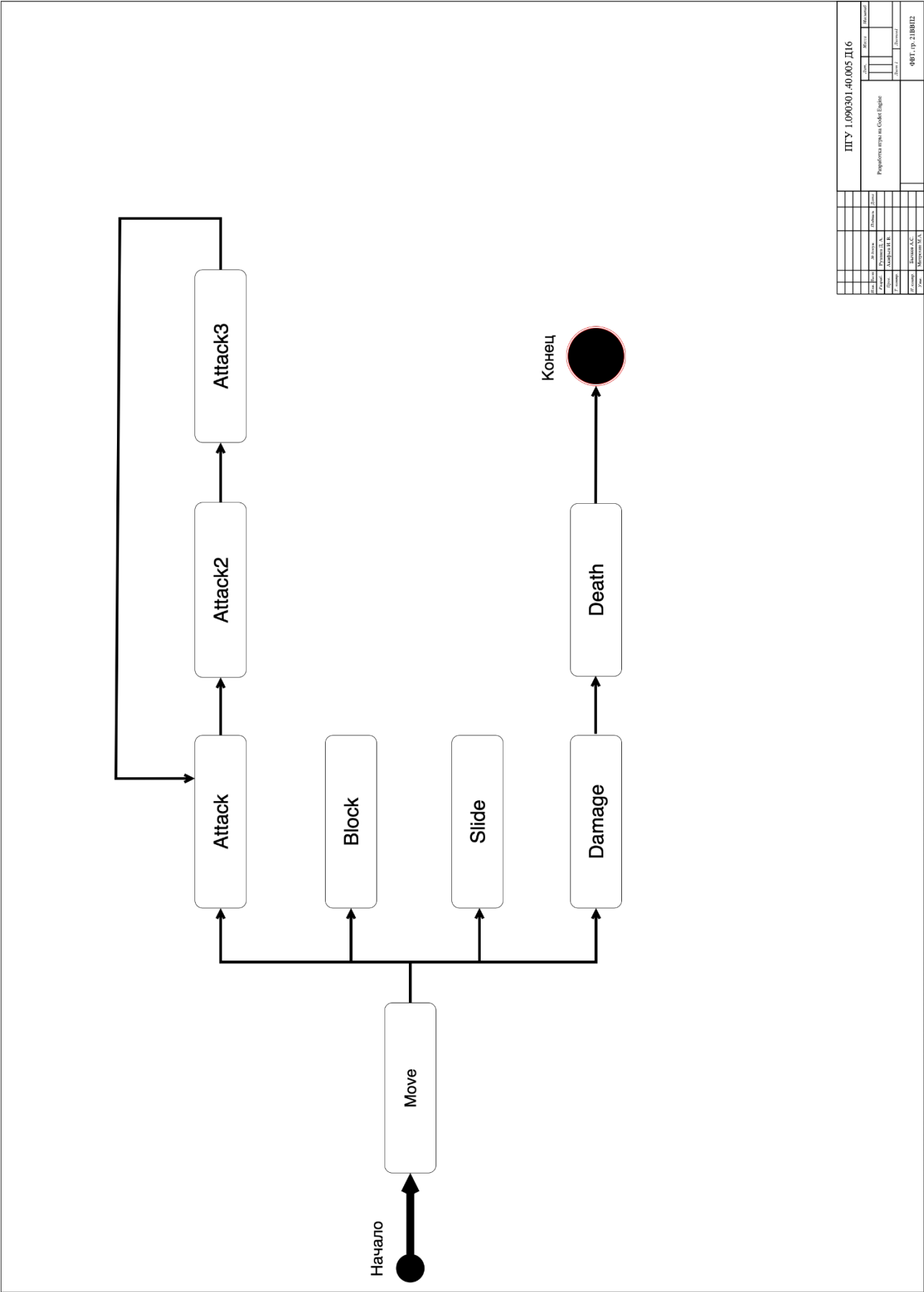
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.40.001 ПЗ

Приложение А.3 - Диаграмма деятельности НРС

[illegible]

Приложение А.4 - Диаграмма состояний игрока



ПГУ 1.090301.40.005 Д1/6									
Разработка игры на Godot Engine									
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Александр М. В.	Романов	И. А.	Романов	И. А.	Романов	И. А.	Романов	И. А.	Романов
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Иванов	И. А.	Иванов	И. А.	Иванов	И. А.	Иванов	И. А.	Иванов	И. А.
9847, стр. 2188112									

Изм.	Лист	№ докум.	Подпись	Дата

Приложение Б

(Обязательное)

Разработка игры на Godot Engine

Листинг программы

					ПГУ1.090301.40.001 ПЗ	Лист
						45
Изм.	Лист	№ докум.	Подпись	Дата		

Файл bg.gd

```
extends ParallaxBackground
var speed = 100
func _process(delta):
    scroll_offset.x -= speed * delta
```

Файл coin.gd

```
extends CharacterBody2D
var gravity = ProjectSettings.get_setting("physics/2d/default_gravity")
func _ready() -> void:
    var tween = get_tree().create_tween()
    tween.parallel().tween_property(self, "velocity",
    Vector2(randi_range(-200, 200), -300), 0.5)
func _physics_process(delta: float) -> void:
    if not is_on_floor():
        velocity.y += gravity * delta
    else:
        velocity.x = 0
    move_and_slide()
func _on_detector_body_entered(_body):
    if is_on_floor():
        var tween = get_tree().create_tween()
        tween.parallel().tween_property(self, "velocity",
        Vector2(0, -300), 0.3)
        tween.parallel().tween_property(self, "modulate:a", 0,
        0.5)
        await get_tree().create_timer(0.5).timeout
        Global.gold += 1
        queue_free()
```

Файл collectibles.gd

```
extends Node2D
var coin_preload = preload("res://scn/collectibles/coin.tscn")
func _ready():
    Signals.connect("enemy_died",Callable(self, "_on_enemy_died"))
func _on_enemy_died (enemy_position, state):
    if state != 4:
        for i in randi_range(1,5):
            coin_spawn(enemy_position)
            await get_tree().create_timer(0.2).timeout
func coin_spawn (pos):
    var coin = coin_preload.instantiate()
    coin.position = pos
    call_deferred("add_child", coin)
```

Файл enemy.gd

```

extends CharacterBody2D
class_name Enemy
enum {
    IDLE,
    ATTACK,
    CHASE,
    DAMAGE,
    DEATH,
    RECOVER
}
var state: int = 0:
    set(value):
        state = value
        match state:
            IDLE:
                idle_state()
            ATTACK:
                attack_state()
            DAMAGE:
                damage_state()
            DEATH:
                death_state()
            RECOVER:
                recover_state()

@onready var animPlayer = $AnimationPlayer
@onready var sprite = $AnimatedSprite2D
var gravity = ProjectSettings.get_setting("physics/2d/default_gravity")
var player = Vector2.ZERO
var direction = Vector2.ZERO
var damage = 20
var move_speed = 150
func _ready() -> void:
    state = CHASE
func _physics_process(delta):
    if not is_on_floor():
        velocity += get_gravity() * delta
    if state == CHASE:
        chase_state()
    move_and_slide()
    player = Global.player_pos
func _on_attack_range_body_entered(_body: Node2D) -> void:
    state = ATTACK
func idle_state():
    velocity.x = 0
    animPlayer.play("Idle")
    state = CHASE
func attack_state():

```



```

velocity.x = 0
animPlayer.play("Attack")
await animPlayer.animation_finished
state = RECOVER
func chase_state():
animPlayer.play("Run")
direction = (player - self.position).normalized()
if direction.x < 0:
    sprite.flip_h = true
    $AttackDirection.rotation_degrees = 180
else:
    sprite.flip_h = false
    $AttackDirection.rotation_degrees = 0
velocity.x = direction.x * move_speed
func damage_state():
velocity.x = 0
damage_anim()
animPlayer.stop()
animPlayer.play("Damage")
await animPlayer.animation_finished
state = IDLE
func death_state():
velocity.x = 0
animPlayer.play("Death")
await animPlayer.animation_finished
queue_free()
func recover_state():
velocity.x = 0
animPlayer.play("Recover")
await animPlayer.animation_finished
state = IDLE
func _on_hitbox_area_entered(_area: Area2D) -> void:
Signals.emit_signal("enemy_attack", damage)
func damage_anim():
direction = (player - self.position).normalized()
velocity.x = 0
if direction.x < 0:
    velocity.x += 200
elif direction.x > 0:
    velocity.x -= 200
var tween = get_tree().create_tween()
tween.tween_property(self, "velocity", Vector2.ZERO, 1)
func _on_run_timeout() -> void:
move_speed = move_toward(move_speed, randi_range(120, 170),
100)

```

Файл flying_eye.gd

```

extends Enemy
func _on_mob_health_no_health() -> void:
Signals.emit_signal("enemy_died", position, state)
state = DEATH

```

```
func _on_mob_health_damage_recieved() -> void:
    state = IDLE
    state = DAMAGE
```

Файл lantern_2.gd

```
extends PointLight2D
@onready var timer = $Timer
var day_state = 0
func _ready():
    Signals.connect("day_time", Callable(self,
    "_on_time_changed"))
    light_off ()
func _on_timer_timeout():
    if day_state == 3:
        var rng = randf_range(0.5, 2)
        var tween = get_tree().create_tween()
        tween.parallel().tween_property(self, "texture_scale",
rng, timer.wait_time)
        tween.parallel().tween_property(self, "energy", rng,
timer.wait_time)
        timer.wait_time = randf_range(0.5, 3)
func _on_time_changed (state, _day_count):
    day_state = state
    if state == 0:
        light_off()
    elif state == 2:
        light_on()
func light_on ():
    var tween = get_tree().create_tween()
    tween.tween_property(self, "energy", 1.5,
randi_range(10,20))
func light_off ():
    var tween = get_tree().create_tween()
    tween.tween_property(self, "energy", 0, randi_range(10,20))
```

Файл level.gd

```
extends Node2D
@onready var light = $DirectionalLight2D
@onready var pointlight = $PointLight2D
@onready var day_text = $CanvasLayer/DayText
@onready var animPlayer = $CanvasLayer/AnimationPlayer
@onready var player = $Player/Player
enum {
    MORNING,
    DAY,
    EVENING,
    NIGHT
}
var state: int = MORNING
var day_count: int
```

```

func _ready():
    Global.gold = 0
    light.enabled = true
    day_count = 0
    set_day_text()
    day_text_fade()
    morning_state()
func _on_day_night_timeout():
    match state:
        MORNING:
            morning_state()
        EVENING:
            evening_state()
    if state < 3:
        state += 1
    else:
        state = MORNING
        day_count += 1
        set_day_text()
        day_text_fade()
    Signals.emit_signal("day_time", state, day_count)
func morning_state():
    var tween = get_tree().create_tween()
    tween.tween_property(light, "energy", 0.2, 20)
    var tween1 = get_tree().create_tween()
    tween1.tween_property(pointlight, "energy", 0, 20)
func evening_state():
    var tween = get_tree().create_tween()
    tween.tween_property(light, "energy", 0.9, 20)
    var tween1 = get_tree().create_tween()
    tween1.tween_property(pointlight, "energy", 1.5, 20)
func day_text_fade():
    animPlayer.play("day_text_fade_in")
    await get_tree().create_timer(3).timeout
    animPlayer.play("day_text_fade_out")
func set_day_text():
    day_text.text = "DAY " + str(day_count + 1)

```

Файл manager.gd

```

extends Node
@onready var pause_menu = $"../CanvasLayer/PauseMenu"
@onready var player = $"../Player/Player"
var game_paused: bool = false
var save_path = "user://savegame.save"
func _process(_delta: float) -> void:
    if Input.is_action_just_pressed("ui_cancel"):
        game_paused = !game_paused
    if game_paused == true:
        get_tree().paused = true
        pause_menu.show()
    else:

```

```

        get_tree().paused = false
        pause_menu.hide()
func _on_resume_pressed() -> void:
    game_paused = !game_paused
func _on_quit_pressed() -> void:
    get_tree().paused = false
    get_tree().change_scene_to_file("res://menu.tscn")
func _on_menu_button_pressed() -> void:
    game_paused = !game_paused
func save_game() -> void:
    var file = FileAccess.open(save_path, FileAccess.WRITE)
    file.store_var(Global.gold)
    file.store_var(player.position.x)
    file.store_var(player.position.y)
func load_game() -> void:
    var file = FileAccess.open(save_path, FileAccess.READ)
    Global.gold = file.get_var(Global.gold)
    player.position.x = file.get_var(player.position.x)
    player.position.y = file.get_var(player.position.y)
func _on_save_pressed() -> void:
    save_game()
    game_paused = !game_paused
func _on_load_pressed() -> void:
    load_game()
    game_paused = !game_paused

```

Файл mob_health.gd

```

extends Node2D
signal no_health ()
signal damage_recieved ()
@onready var health_bar = $HealthBar
@onready var damage_text = $DamageText
@onready var animPlayer = $AnimationPlayer
@export var max_health = 100
var health = 100:
    set (value):
        health = value
        health_bar.value = health
        if health <= 0:
            health_bar.visible = false
        else:
            health_bar.visible = true
func _ready():
    damage_text.modulate.a = 0
    health_bar.max_value = max_health
    health = max_health
    health_bar.visible = false
func _on_hurt_box_area_entered(_area: Area2D) -> void:
    await get_tree().create_timer(0.05).timeout
    health -= Global.player_damage
    damage_text.text = str (Global.player_damage)

```

```

animPlayer.stop()
animPlayer.play("damage_text")
if health <= 0:
    emit_signal("no_health")
else:
    emit_signal("damage_recieved")

```

Файл mushroom.gd

```

extends Enemy
func _on_mob_health_no_health() -> void:
    Signals.emit_signal("enemy_died", position, state)
    state = DEATH
func _on_mob_health_damage_recieved() -> void:
    state = IDLE
    state = DAMAGE

```

Файл player.gd

```

extends CharacterBody2D
enum {
    MOVE,
    ATTACK,
    ATTACK2,
    ATTACK3,
    BLOCK,
    SLIDE,
    DAMAGE,
    CAST,
    DEATH
}
const SPEED = 300.0
const JUMP_VELOCITY = -600.0
var gravity =
ProjectSettings.get_setting("physics/2d/default_gravity")
@onready var anim = $AnimatedSprite2D
@onready var animPlayer = $AnimationPlayer
@onready var stats = $Stats
@onready var Leafs = $Leafs
@onready var smack = $Sounds/Smack
var state = MOVE
var run = 1
var combo = false
var attack_cooldown = false
var damage_basic = 10
var damage_multiplier = 1
var damage_current
var recovery = false
func _ready() -> void:
    Signals.connect("enemy_attack", Callable(self,
    "_on_damage_recieved"))
func _physics_process(delta):

```

```

if not is_on_floor():
    velocity += get_gravity() * delta
if velocity.y > 0:
    animPlayer.play("fall and doublejump")
Global.player_damage = damage_basic * damage_multiplier
match state:
    MOVE:
        move_state()
    ATTACK:
        attack_state()
    ATTACK2:
        attack2_state()
    ATTACK3:
        attack3_state()
    BLOCK:
        block_state()
    SLIDE:
        slide_state()
    DAMAGE:
        damage_state()
    CAST:
        cast_state()
    DEATH:
        death_state()
move_and_slide()
Global.player_pos = self.position
func move_state():
    var direction = Input.get_axis("left", "right")
    if direction:
        velocity.x = direction * SPEED * run
        if velocity.y == 0:
            if run == 1:
                animPlayer.play("walk")
            else:
                animPlayer.play("run")
    else:
        velocity.x = move_toward(velocity.x, 0, SPEED)
        if velocity.y == 0:
            animPlayer.play("idle")
    if direction == -1:
        $AnimatedSprite2D.flip_h = true
        $AttackDirection.rotation_degrees = 180
    elif direction == 1:
        $AnimatedSprite2D.flip_h = false
        $AttackDirection.rotation_degrees = 0
    if Input.is_action_pressed("run") and not recovery:
        run = 1.5
        stats.stamina -= stats.run_cost
    else:
        run = 1
    if Input.is_action_pressed("attack"):
        if not recovery:

```

```

        stats.stamina_cost = stats.attack_cost
        if attack_cooldown == false and stats.stamina >
stats.stamina_cost:
            Input.action_release("attack")
            state = ATTACK
    if Input.is_action_pressed("slide") and velocity.x != 0:
        if not recovery:
            stats.stamina_cost = stats.slide_cost
            if stats.stamina > stats.stamina_cost:
                Input.action_release("slide")
                state = SLIDE
    elif Input.is_action_pressed("slide") and velocity.x == 0:
        if not recovery:
            if stats.stamina > 1:
                state = BLOCK
func attack_state ():
    stats.stamina_cost = stats.attack_cost
    damage_multiplier = 1
    if Input.is_action_pressed("attack") and combo == true and
stats.stamina > stats.stamina_cost:
        Input.action_release("attack")
        state = ATTACK2
    velocity.x = 0
    animPlayer.play("attack")
    await animPlayer.animation_finished
    attack_freeze()
    state = MOVE
func block_state ():
    stats.stamina -= stats.block_cost
    velocity.x = move_toward(velocity.x, 0, SPEED)
    animPlayer.play("block")
    if Input.is_action_just_released("slide") or recovery ==
true:
        state = MOVE
func slide_state():
    animPlayer.play("slide")
    await animPlayer.animation_finished
    state = MOVE
func attack2 ():
    combo = true
    await animPlayer.animation_finished
    combo = false
func attack2_state ():
    stats.stamina_cost = stats.attack_cost
    damage_multiplier = 2
    if Input.is_action_pressed("attack") and combo == true and
stats.stamina > stats.stamina_cost:
        Input.action_release("attack")
        state = ATTACK3
    animPlayer.play("attack2")
    await animPlayer.animation_finished
    state = MOVE

```

```

func attack3 ():
    combo = true
    await animPlayer.animation_finished
    combo = false
func attack3_state():
    damage_multiplier = 3
    animPlayer.play("attack3")
    await animPlayer.animation_finished
    state = MOVE
func attack_freeze ():
    attack_cooldown = true
    await get_tree().create_timer(0).timeout
    attack_cooldown = false
func damage_state ():
    animPlayer.play("damage")
    await animPlayer.animation_finished
    state = MOVE
func cast_state():
    pass
func death_state ():
    velocity.x = 0
    animPlayer.play("death")
    await animPlayer.animation_finished
    queue_free()
    get_tree().change_scene_to_file.bind("res://menu.tscn").call
    _deferred()
func _on_damage_recieved (enemy_damage):
    smack.play()
    if state == BLOCK:
        enemy_damage /= 2
    elif state == SLIDE:
        enemy_damage = 0
    else:
        state = DAMAGE
        damage_anim()
    stats.health -= enemy_damage
    if stats.health <= 0:
        stats.health = 0
        state = DEATH
func _on_stats_no_stamina() -> void:
    recovery = true
    await get_tree().create_timer(2).timeout
    recovery = false
func damage_anim ():
    velocity.x = 0
    self.modulate = Color (1, 0, 0, 1)
    if $AnimatedSprite2D.flip_h == true:
        velocity.x += 200
    else:
        velocity.x -= 200
    var tween = get_tree().create_tween()

```



```

        tween.parallel().tween_property(self, "velocity",
Vector2.ZERO, 1)
        tween.parallel().tween_property(self, "modulate", Color(1,
1, 1, 1), 0.4)
func steps():
    Leafs.emitting = true
    Leafs.one_shot = true

```

Файл predmet.gd

```

extends Area2D
func _on_body_entered(body: Node2D) -> void:
    if body.name == "Player":
        var tween = get_tree().create_tween()
        var tween1 = get_tree().create_tween()
        tween.tween_property(self, "position", position-
Vector2(0, 25), 0.3)
        tween1.tween_property(self, "modulate:a", 0, 0.3)
        tween.tween_callback(queue_free)
        body.predmet += 1

```

Файл spawner.gd

```

extends Node2D
@onready var mobs = $".."
@onready var animation_player = $AnimationPlayer
var spawn_count = 0
var mushroom_preload = preload("res://mobs/mushroom.tscn")
var flying_eye_preload = preload("res://mobs/flying_eye.tscn")
func _ready():
    Signals.connect("day_time",Callable(self,
"_on_time_changed"))
func _on_time_changed (state, day_count):
    spawn_count = 2
    var rng = randi_range(2, 5)
    if state == 1:
        for i in (day_count + rng):
            animation_player.play("spawn")
            await animation_player.animation_finished
            spawn_count += 2
    if spawn_count == day_count + rng:
        animation_player.play("idle")
func enemy_spawn():
    var rng = randi_range(1,2)
    if rng == 1:
        mushroom_spawn()
    elif rng == 2:
        flying_eye_spawn()
func flying_eye_spawn():
    var flying_eye = flying_eye_preload.instantiate()
    flying_eye.position = Vector2 (self.position.x,591.78)
    mobs.add_child(flying_eye)

```

```

func mushroom_spawn():
    var mushroom = mushroom_preload.instantiate()
    mushroom.position = Vector2 (self.position.x, 591.78)
    mobs.add_child(mushroom)

```

Файл stats.gd

```

extends CanvasLayer
signal no_stamina ()
@onready var health_bar = $HealthBar
@onready var stamina_bar = $Stamina
@onready var health_text = $"../HealthText"
@onready var health_anim = $"../HealthAnim"
var stamina_cost
var attack_cost = 10
var block_cost = 0.5
var slide_cost = 20
var run_cost = 0.3
var max_health = 120
var old_health = max_health
var stamina = 50:
    set(value):
        stamina = value
        if stamina < 1:
            emit_signal("no_stamina")
var health:
    set(value):
        health = clamp(value, 0 ,max_health)
        health_bar.value = health
        var difference = health - old_health
        health_text.text = str(difference)
        old_health = health
        if difference < 0:
            health_anim.play("damage_recieved")
        elif difference > 0:
            health_anim.play("health_recieved")
func _ready():
    health_text.modulate.a = 0
    health = max_health
    health_bar.max_value = health
    health_bar.value = health
func _process(delta):
    stamina_bar.value = stamina
    if stamina < 100:
        stamina += 10 * delta
func stamina_consumption ():
    stamina -= stamina_co
func _on_health_regen_timeout() -> void:
    health += 1

```

Приложение В

(Обязательное)

Разработка игры на Godot Engine

Презентация

					<i>ПГУ1.090301.40.001 ПЗ</i>	Лист
						58
Изм.	Лист	№ докум.	Подпись	Дата		

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА НА ТЕМУ: РАЗРАБОТКА ИГРЫ НА GODOT ENGINE

Студент гр. 21ВВП2: Дмитрий Александрович Рузляев
Руководитель: к. э. н., доцент кафедры «Вычислительная Техника» -
Илья Владимирович Акифьев

Пгу 2025

ВВЕДЕНИЕ

Цель работы – разработка игры на Godot Engine.

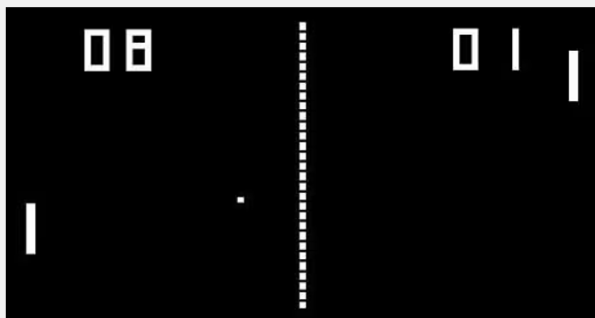
Задачи:

- Запуск в режиме полной совместимости на Android и Windows
- Создание интерфейса
- Адаптация управления под различные типы операционных систем
- Создание системы случайной генерации NPC

Объект исследования — разработка игр

Предмет исследования — готовая игра разработанная на Godot Engine

ВВЕДЕНИЕ



История игровой индустрии:

- Зародилась в XX веке с аркадными автоматами.
- Коммерческий прорыв — игра Pong (Atari, 1972).
- Современные игры соперничают с кино по масштабам.

Потребности геймеров:

- Эстетическое удовольствие.
- Отвлечение от рутины.
- Снятие стресса.
- Получение эмоций.

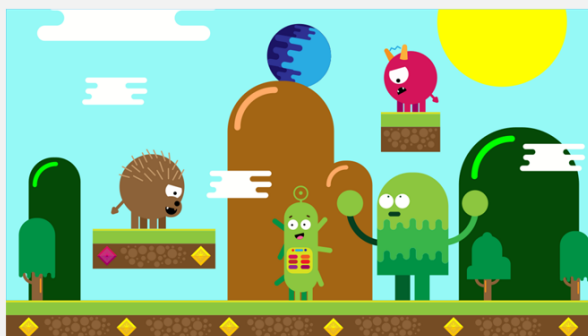
ТЕОРЕТИЧЕСКИЕ ОСНОВЫ

Сайд-скроллер (Side-Scroller)

Игра с боковым видом камеры, где персонажи двигаются только по горизонтали (влево/вправо).

Популярные жанры:

- Shoot 'em up (шутеры).
- Beat 'em up (файтинги).



АНАЛИЗ АНАЛОГИЧНЫХ РЕШЕНИЙ



Until We Die

Жанр: shoot 'em up с элементами стратегии

Разработчик/издатель: Российская студия Pixeye Games

Дата выхода: 3 июня 2021

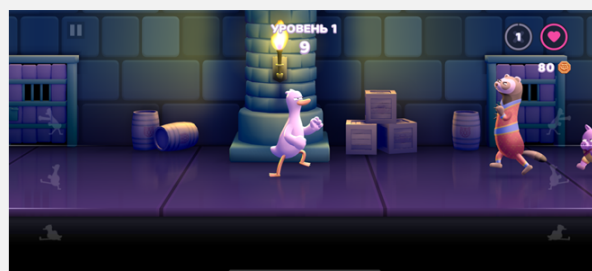
АНАЛИЗ АНАЛОГИЧНЫХ РЕШЕНИЙ

Punch Kick Duck

Жанр: Beat 'em up

Разработчик/издатель: Pipsqueak Games

Дата выхода 25 января 2023



Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.40.001 ПЗ

Лист

61

АНАЛИЗ АНАЛОГИЧНЫХ РЕШЕНИЙ



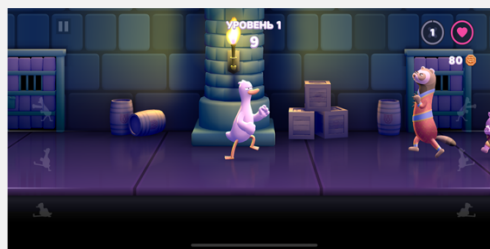
Mother Russia Bleeds

Жанр: Beat 'em up

Разработчик: Le Cartel Studio / издатель: Devolver Digital

Дата выхода: 5 сентября 2016

ВЫВОДЫ



- Плохой баланс.
- Недостаточная поддержка после релиза.
- Малое количество контента.

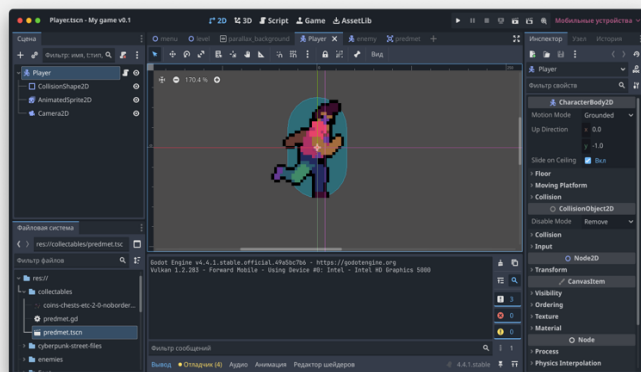
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.40.001 ПЗ

Лист

62

ВЫБОР ИНСТРУМЕНТОВ РАЗРАБОТКИ



- Бесплатный
- Оптимизирован для работы на маломощных устройствах.
- Прост в освоении.

КОНЦЕПЦИЯ РАЗРАБАТЫВАЕМОГО ПРОЕКТА

Действие разрабатываемой игры – средневековый фэнтезийный мир.

Общий художественный стиль – «Modern Pixel Art».



Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.40.001 ПЗ

Лист

63

UML ДИАГРАММА ВАРИАНТОВ ИСПОЛЬЗОВАНИЯ.



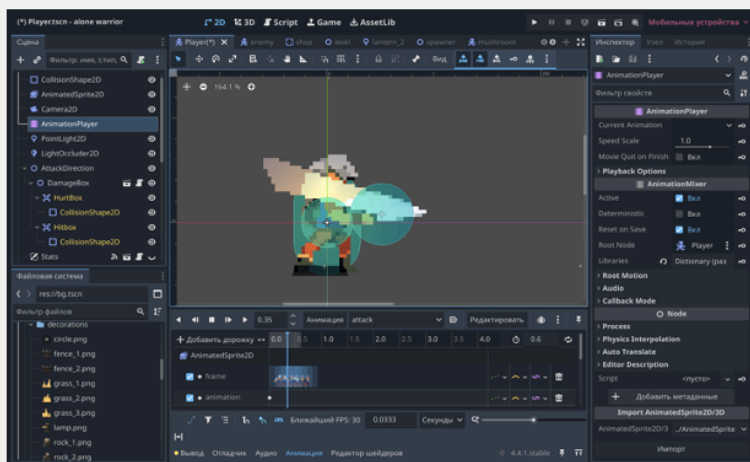
ПРОЦЕСС РАЗРАБОТКИ

Игровые характеристики персонажа:

- Ходьба.
- Бег.
- Кратковременное ускоряющее скольжение.
- Три варианта атаки.
- Блокирование атакующих ударов врагов.



ПРОЦЕСС РАЗРАБОТКИ



Типы взаимодействующих коллизий

- Столкновения.
- Получение урона.
- Нанесение урона.

ПРОЦЕСС РАЗРАБОТКИ

Состояния:

- Бездействие.
- Атака.
- Преследование.
- Получение урона.
- Смерть.



Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.40.001 ПЗ

Лист

65

ЗАКЛЮЧЕНИЕ

- ✓ Проведён анализ предметной области и существующих аналогов.
- ✓ Выполнено проектирование структуры приложения.
- ✓ Осуществлена программная реализация.
- ✓ Проведено комплексное тестирование.

					ПГУ1.090301.40.001 ПЗ	Лист
						66
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение Г

Разработка игры на Godot Engine

Скриншоты разработанной игры

					ПГУ1.090301.40.001 ПЗ	Лист
						67
Изм.	Лист	№ докум.	Подпись	Дата		



Рисунок 1 – Главное меню

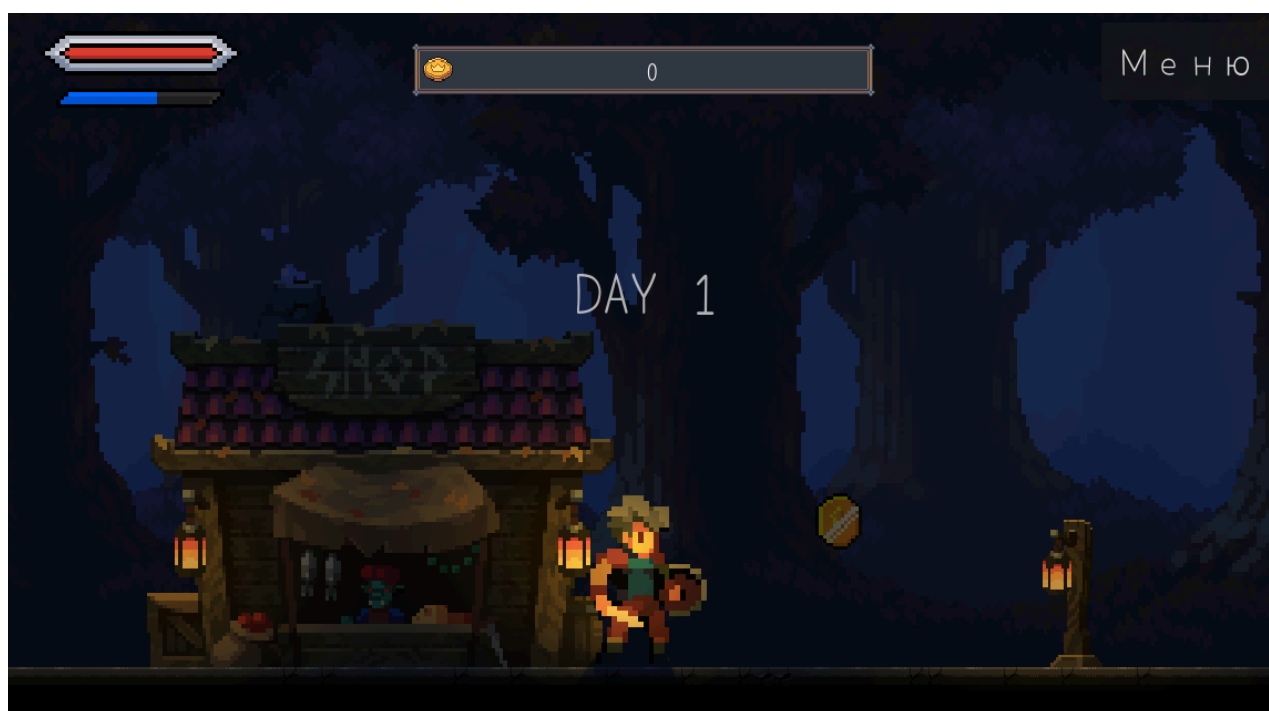


Рисунок 2 – Начало игры

					ПГУ1.090301.40.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		68

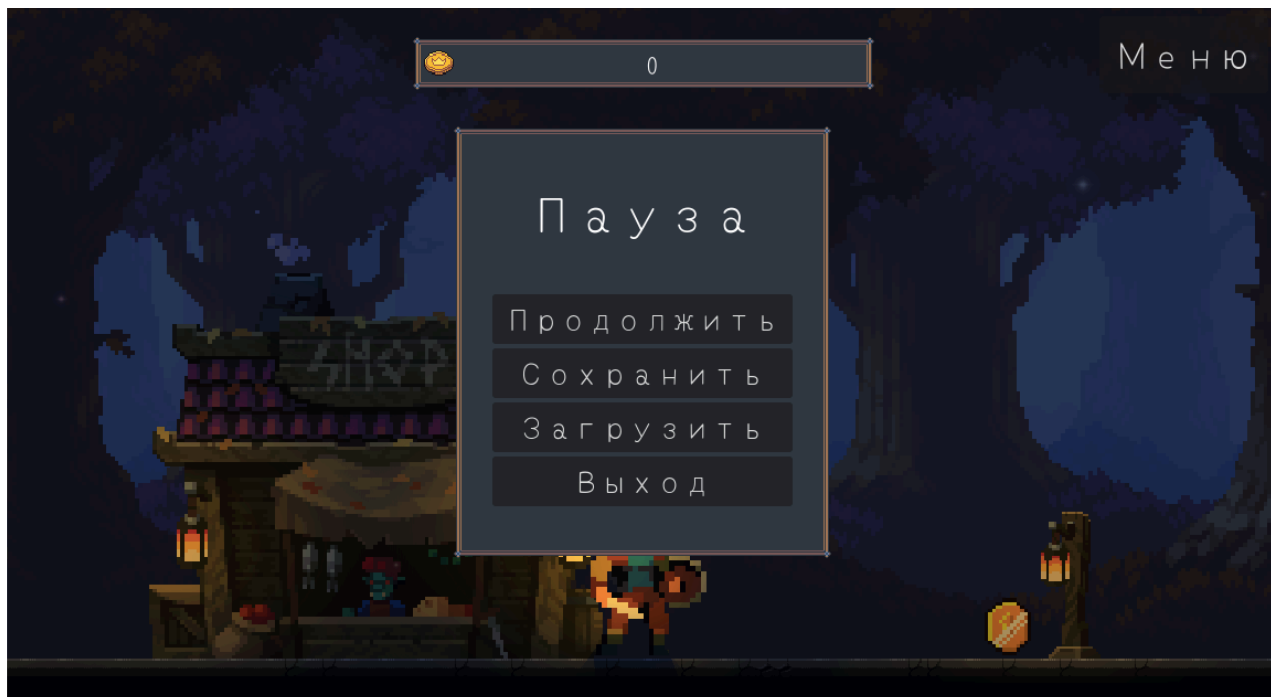


Рисунок 3 – Меню паузы



Рисунок 4 – Появление врага гриба



Рисунок 5 – Блокирование атаки врага



Рисунок 6 - Скольжение



Рисунок 7 – Появление второго врага



Рисунок 8 – Атака



Рисунок 9 – Бег



Рисунок 10 – Смерть