МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра «Измерительно-вычислительные комплексы»

КУРСОВАЯ РАБОТА

по дисциплине «Алгоритмы и структуры данных»

Тема: «Компьютерная игра “WWK. Часть 2”»

**Инв.№ подл.**

**Подп. и дата**

**Взам.инв. №**

**Инв.№ дубл.**

**Подп. и дата**

Р.02069337. 23/712 23 ПЗ-01

Листов 86

Руководитель разработки:

к.т.н, доцент кафедры "ИВК"

Шишкин Вадим Викторинович

« » 2024 г.

Исполнитель:

студент гр. ИСТбд-21

Саранцев Семён Александрович

« » 2024 г.

2024

Содержание

[**ТЕХНИЧЕСКОЕ ЗАДАНИЕ** 3](#_Toc185449081)

[**Пояснительная записка** 8](#_Toc185449082)

[**Руководство программиста** 24](#_Toc185449083)

[**Тестовая документация** 29](#_Toc185449084)

[**Листинг кода** 34](#_Toc185449085)

[**Заключение** 79](#_Toc185449086)

[**Источники, использованные при разработке** 85](#_Toc185449087)

[**ПРИЛОЖЕНИЕ 1** 86](#_Toc185449088)

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра «Измерительно-вычислительные комплексы»

ТЕХНИЧЕСКОЕ ЗАДАНИЕ

на курсовую работу

по дисциплине «Алгоритмы и структуры данных»

Тема: «Компьютерная игра “WWK. Часть 2”»

**Инв.№ подл.**

**Подп. и дата**

**Взам.инв. №**

**Инв.№ дубл.**

**Подп. и дата**

Р.02069337. 23/712 23 ТЗ-01

Листов 6

Руководитель разработки:

к.т.н, доцент кафедры "ИВК"

Шишкин Вадим Викторинович

« » 2024 г.

Исполнитель:

студент гр. ИСТбд-21

Саранцев Семён Александрович

« » 2024 г.

2024

|  |  |
| --- | --- |
|  | УТВЕРЖДЕНО  Распоряжением по кафедре ИВК  №\_\_\_ от «\_\_» \_\_\_\_\_\_\_\_\_\_\_\_2024 |

# **ТЕХНИЧЕСКОЕ ЗАДАНИЕ**

на выполнение курсовой работы

по дисциплине «Алгоритмы и структуры данных»

Специальность 09.03.02 – «Информационные системы и технологии»

Тема: «**Разработка компьютерной игры “WWK. Часть 2”**»

**1. Введение**

Целью разработки является создание RPG-игры с процедурной генерацией мира, где игрок может исследовать локации, сражаться с врагами разных типов и боссами.

**2. Основания для разработки**

Разработка осуществляется с использованием игрового движка Godot Engine для создания современной 2D RPG игры с уникальной механикой процедурной генерации и боевой системой.

**3. Назначение разработки**

Программа предназначена для предоставления игрокам увлекательного игрового процесса, сочетающего элементы исследования процедурно-генерируемого мира и сражений с различными противниками.

**4. Требования к программе**

**4.1 Требования к функциональным характеристикам**

**4.1.1 Система генерации мира:**

* Процедурная генерация ландшафта с использованием шумов (высота, деревья, температура, влажность)
* Различные биомы (океан, пляж, тундра, лес, равнины, пустыня)
* Система размещения поселений (малые и средние деревни)
* Система размещения надгробий
* Валидация генерации мира
* Безопасные расстояния между объектами

**4.1.2 Система врагов:**

1. Базовый враг (Enemy):

* Система здоровья и характеристик
* Адаптация к уровню игрока
* Система атаки и защиты
* Система уклонения
* Выпадение предметов
* Анимации движения и атаки

1. Улучшенный враг (Enemy2):

* Улучшенная система боя
* Различные анимации для разных направлений
* Оптимизированная система уклонения
* Улучшенная система выпадения предметов

1. Босс:

* Система фаз (3 фазы)
* Уникальные способности:
* Рывок
* АОЕ атаки
* Огненный след
* Адаптивная сложность
* Система частиц для визуальных эффектов

**4.1.3 Интерфейс**

* Визуальные эффекты при атаках
* Индикация получения урона
* Система подсказок

**4.2 Требования к надежности**

* Корректная обработка коллизий
* Валидация генерации мира
* Обработка краевых случаев в боевой системе
* Система восстановления при ошибках

**4.3 Требования к составу и параметрам технических средств**

Программа должна функционировать на персональных компьютерах с операционными системами Windows 8 и выше, минимальный объем оперативной памяти — 2 ГБ.

**4.4 Требования к информационной и программной совместимости**

* Программа должна быть разработана на Godot Engine 4.0+
* Поддержка кроссплатформенности (Windows, macOS, Linux)
* Поддержка различных разрешений экрана
* Поддержка контроллеров и клавиатуры/мыши

**4.5 Условия и срок хранения**

Обеспечение свободного доступа к проекту в репозитории до окончания срока учебы.

**5. Требования к программной документации**

1. В состав программной документации должны входить:
2. Техническое задание (составленное в соответствии с ГОСТ 19.201-78);
3. Руководство пользователя;
4. Пояснительная записка;
5. Тестовая документация (чек-листы, тест-кейсы, отчет о тестировании).

**6. Стадии и этапы разработки**

1. Базовая генерация мира (3 недели):

* Реализация системы шумов (высота, деревья, температура, влажность)
* Создание системы биомов
* Разработка алгоритмов размещения деревень и объектов
* Внедрение системы валидации генерации
* Тестирование и отладка генерации

1. Реализация базовой боевой системы (2 недели):

* Создание базового класса врага
* Реализация системы здоровья и характеристик
* Внедрение системы атаки и защиты
* Разработка системы выпадения предметов
* Создание анимаций для врагов

1. Разработка улучшенного ИИ врагов (2 недели):

* Улучшение системы поиска пути
* Реализация улучшенных анимаций
* Внедрение системы уклонения
* Оптимизация боевого поведения
* Тестирование ИИ

1. Создание системы босса (3 недели):

* Разработка фазовой системы
* Реализация уникальных способностей
* Создание системы частиц и визуальных эффектов
* Балансировка характеристик

1. Тестирование и отладка (2 недели):

* Комплексное тестирование всех систем
* Оптимизация производительности
* Исправление обнаруженных ошибок
* Финальная балансировка
* Подготовка релизной версии

**6.1 Распределение обязанностей в команде разработчиков**

**•** Дизайнер, Full-Stack разработчик проекта: Саранцев Семён Александрович.

**7. Порядок контроля и приемки**

**7.1 Методы контроля**

* Тестирование генерации мира
* Проверка баланса боевой системы
* Тестирование ИИ врагов
* Проверка производительности
* Тестирование визуальных эффектов

**7.2 Критерии приемки**

* Стабильная работа генерации мира
* Отсутствие критических ошибок в боевой системе
* Сбалансированная сложность врагов
* Корректная работа всех способностей босса

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра «Измерительно-вычислительные комплексы»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

на курсовую работу

по дисциплине «Алгоритмы и структуры данных»

Тема: «Компьютерная игра “WWK. Часть 2”»

**Инв.№ подл.**

**Подп. и дата**

**Взам.инв. №**

**Инв.№ дубл.**

**Подп. и дата**

Р.02069337. 23/712 23 ПЗ-02

Листов 17

Руководитель разработки:

к.т.н, доцент кафедры "ИВК"

Шишкин Вадим Викторинович

« » 2024 г.

Исполнитель:

студент гр. ИСТбд-21

Саранцев Семён Александрович

« » 2024 г.

2024

# **Пояснительная записка**

на выполнение курсовой работы

по дисциплине «Алгоритмы и структуры данных»

Специальность 09.03.02 – «Информационные системы и технологии»

Тема: «**Разработка компьютерной игры “WWK. Часть 2”**»

**АННОТАЦИЯ**

Проект "World Without a King" представляет собой RPG игру с процедурной генерацией мира, реализованную на игровом движке Godot. Основная цель проекта - создание динамичной ролевой игры с уникальной системой генерации контента и многофазовыми боссами.

Основные функции и возможности:

1. Система процедурной генерации:

* Генерация ландшафта с использованием шумов Перлина
* Различные биомы (океан, пляж, тундра, лес, равнины, пустыня)
* Система размещения поселений и объектов
* Валидация генерации мира

1. Система противников:

* Базовые враги с системой характеристик
* Улучшенные враги с направленными атаками
* Многофазовый босс со специальными способностями
* Система адаптивной сложности

1. Боевая система:

* Реализация различных типов атак
* Система урона и защиты
* Механика уклонения
* Выпадение предметов

1. Система босса:

* Три уникальные фазы
* Специальные способности (рывок, АОЕ атаки, огненный след)
* Система призыва миньонов
* Визуальные эффекты

1. Технические особенности:

* Оптимизированная генерация мира
* Система частиц для визуальных эффектов
* Продвинутая система коллизий
* Валидация игровых состояний

1. Проект демонстрирует навыки работы с:

* Процедурной генерацией контента
* Искусственным интеллектом противников
* Объектно-ориентированным программированием
* Оптимизацией производительности
* Системами частиц и визуальными эффектами
* Разработкой сложных игровых механик

Игра может служить основой для дальнейшего развития через добавление новых биомов, типов противников, боссов и игровых механик. Особое внимание уделено масштабируемости и оптимизации кода для будущих обновлений.

**Введение**

Разрабатываемое приложение представляет собой RPG игру с процедурной генерацией мира, разработанную с использованием игрового движка Godot Engine и языка GDScript. Игра сочетает в себе элементы исследования процедурно генерируемого мира с боевой системой, включающей различные типы врагов и боссов.

**Основания для разработки**

Задание для курсового проекта по учебному плану направления «Информационные системы и технологии».

**Постановка задачи**

Цель разработки заключается в создании RPG игры с процедурной генерацией, которая включает в себя:

1. Реализация системы генерации мира:

* Процедурная генерация ландшафта
* Система биомов
* Размещение объектов и поселений
* Валидация генерации

1. Реализация боевой системы:

* Различные типы врагов
* Система характеристик
* Адаптивная сложность
* Система босса с фазами

**Архитектура проекта (Общая)**

1. Основные сцены (scenes/)

worldgenerator.tscn - Генератор игрового мира

* TileMapLayer (слои тайлов)
* Шумовые текстуры для генерации (шумы)

enemy.tscn - Базовый противник

* CharacterBody2D (физическое тело)
* AnimatedSprite2D (анимации)
* AggroArea (зона агрессии)
* NavigationAgent2D (навигация)

enemy2.tscn - Продвинутый противник

* CharacterBody2D (физическое тело)
* AnimatedSprite2D (анимации)
* AggroArea (зона агрессии)
* NavigationAgent2D (навигация)

boss.tscn – Босс

Наследует от enemy.tscn

• Расширенные анимации

- idle

- run

- hurt\_left

- death

• Увеличенный размер коллизий

• Специальные эффекты

1. Скрипты (scripts/)

Основные системы

* worldgenerator.gd- Генерация игрового мира

Игровые механики

enemy.gd

* Система здоровья
* Базовые атаки
* Система агрессии
* Навигация

enemy2.gd

* Расширенные анимации
* Улучшенная боевая система
* Направленные атаки

boss.gd

* Система фаз
* Специальные атаки
* Визуальные эффекты
* Управление миньонами

1. Конфигурация проекта

project.godot

* Основные настройки
* Автозагружаемые скрипты
* Настройки окна
* Управление

1. Системы и механики

Игровые системы

Система генерации мира

* Процедурная генерация ландшафта
* Система биомов
* Размещение объектов
* Валидация генерации

Боевая система

* Базовые атаки
* Система урона
* Уклонение
* Выпадение предметов

Система босса

* Фазы босса
* Специальные способности
* Система миньонов
* Визуальные эффекты

Технические системы

* Процедурная генерация
* Система навигации
* Управление анимациями
* Обработка коллизий

**Взаимодействие компонентов**

WorldGenerator → Enemy/Enemy2/Boss

Enemy → Enemy2 → Boss

Player → Enemy/Boss

**Структура программы**

**Классы (Скрипты gd):**

1. **WorldGenerator** (worldgenerator.gd): Основной класс для генерации игрового мира:

Основные функции:

• **generate\_world()** - генерация мира с использованием шумов

• **place\_settlements()** - размещение деревень

• **place\_graves()** - размещение надгробий

• **validate\_generation()** - проверка корректности генерации

• **clear\_area()** - очистка области для размещения объектов

2. **Enemy (enemy.gd)**: Базовый класс врага:

Основные функции:

• **take\_damage()** - получение урона

• **perform\_attack()** - выполнение атаки

• **drop\_items()** - выпадение предметов

• **update\_aggro()** - обновление агрессии

• **navigate\_to\_target()** - навигация к цели

3. **Enemy2 (enemy2.gd):** Улучшенный класс врага:

Основные функции:

• **perform\_directional\_attack()** - направленная атака

• **handle\_animations()** - управление анимациями

• **dodge()** - система уклонения

• **enhanced\_pathfinding()** - улучшенный поиск пути

• **special\_abilities()** - особые способности

4. **Boss (boss.gd):** Класс босса:

Основные функции:

• **change\_phase()** - смена фазы босса

• **special\_attack()** - специальная атака

• **summon\_minions()** - призыв миньонов

• **create\_particles()** - создание визуальных эффектов

• **rage\_mode()** - режим ярости

***Функциональное назначение:***

**1. Генерация и управление миром**

• Процедурная генерация ландшафта

- Создание различных биомов

- Размещение деревень и объектов

- Валидация генерации

• Управление окружением

- Размещение надгробий

- Создание безопасных зон

- Обработка коллизий

**2. Система противников**

**Базовый враг (Enemy):**

• Базовая боевая система

- Атаки и защита

- Система здоровья

- Выпадение предметов

• Навигация

- Преследование цели

- Обход препятствий

**Улучшенный враг (Enemy2):**

• Расширенная боевая система

- Направленные атаки

- Система уклонения

- Улучшенные анимации

**Босс:**

• Система фаз

- Смена поведения

- Особые способности

• Специальные механики

- Призыв миньонов

- Визуальные эффекты

- Режим ярости

**3. Визуальное оформление**

• Анимации

- Движение

- Атаки

- Получение урона

- Смерть

• Эффекты

- Частицы

- Визуальные индикаторы

- Подсветка действий

**4. Технические системы**

• Система навигации

- Поиск пути

- Обход препятствий

• Обработка столкновений

- Проверка коллизий

- Валидация позиций

• Управление состояниями

- Фазы босса

- Состояния врагов

***Проектная часть***

***1.1 Математические методы***

Для генерации мира (worldgenerator.gd):

* Алгоритмы шумов Перлина для генерации:

- Высоты ландшафта

- Распределения деревьев

- Температуры и влажности

* Формулы расчета безопасных расстояний между объектами
* Алгоритмы валидации размещения объектов

Для боевой системы (enemy.gd, enemy2.gd):

* Формулы расчета урона с учетом характеристик
* Система адаптации к уровню игрока
* Алгоритмы поиска пути
* Расчет вероятностей выпадения предметов

Для системы лута (boss.gd):

* Алгоритмы расчета специальных атак
* Формулы масштабирования урона по фазам
* Система расчета позиций для спавна миньонов
* Математика визуальных эффектов

***1.2 Архитектура и алгоритмы***

***1.2.1 Архитектура***

Система генерации мира:

* Многослойные тайловые карты
* Система биомов
* Размещение объектов
* Валидация генерации

Система противников:

* Иерархия врагов
* Система характеристик
* Боевые механики
* Система выпадения предметов

Система босса:

* Фазовая система
* Специальные способности
* Система миньонов
* Визуальные эффекты
  + 1. ***Алгоритмы***

**Генерация мира:**

func generate\_world():

# Генерация карты высот с помощью шума Перлина

# Определение типов биомов на основе температуры и влажности

# Размещение деревень и объектов

# Проверка валидности генерации

**Базовый враг:**

func process\_enemy():

# Проверка расстояния до игрока

# Обновление состояния агрессии

# Расчет пути к цели

# Выполнение атаки

**Босс:**

func handle\_boss\_behavior():

# Определение текущей фазы

# Выбор специальной атаки

# Управление миньонами

# Применение визуальных эффектов

**Стадии и этапы разработки**

1. Проектирование базовых систем

- Система генерации мира

- Базовая боевая механика

- Система навигации

2. Разработка системы противников

- Базовый класс врага

- Улучшенный враг

- Система босса

3. Реализация дополнительных механик

- Визуальные эффекты

- Система фаз босса

- Спавн миньонов

4. Тестирование и отладка

**Тестирование**

***Цель тестирования***

Проверка корректности работы всех игровых механик, генерации мира и боевой системы.

***Методика тестирования***

Тестирование методом "черного ящика" с проверкой всех возможных игровых сценариев.

***Тестовые сценарии***

1. Проверка генерации мира:

- Корректность создания ландшафта

- Правильное размещение объектов

- Валидность позиций деревень

- Генерация безопасных зон

1. Тестирование системы противников:

- Поведение базовых врагов

- Работа улучшенных врагов

- Навигация и поиск пути

- Система агрессии

1. Тестирование босса:

- Смена фаз

- Работа специальных атак

- Призыв миньонов

- Визуальные эффекты

- Система урона

1. Тестирование производительности:

- Оптимизация генерации мира

- Проверка FPS при большом количестве врагов

- Использование памяти

- Стабильность при длительной игре

1. Выявленные ошибки были исправлены:

- Оптимизирована система генерации мира

- Исправлены проблемы с навигацией врагов

- Улучшен баланс фаз босса

- Устранены утечки памяти

Программа работает стабильно и соответствует всем требованиям технического задания.

**Использованные методы и алгоритмы**

В данном проекте использованы различные методы и алгоритмы, обеспечивающие функциональность RPG игры с процедурной генерацией мира. Одним из ключевых компонентов является система генерации мира, реализованная в классе WorldGenerator. Этот компонент использует алгоритмы шума Перлина для создания естественно выглядящего ландшафта, включая различные биомы и особенности рельефа. Система использует несколько слоев шума для создания комплексного ландшафта, где noise\_height\_text отвечает за генерацию высот, noise\_tree\_text управляет размещением растительности, а noise\_temp\_text и noise\_moisture\_text определяют биомы. Дополнительный слой noise\_settlement\_text обеспечивает логичное размещение поселений в мире.

Система противников реализована через иерархию классов, начиная с базового класса Enemy. Базовый класс обеспечивает основные механики боя и передвижения, используя NavigationAgent2D для навигации и AggroArea для определения зоны агрессии. Система анимаций включает в себя различные состояния: атака, получение урона, передвижение и смерть. При этом каждое действие сопровождается соответствующими визуальными эффектами, что делает игровой процесс более динамичным и отзывчивым.

Улучшенный класс Enemy2 расширяет базовую функциональность, добавляя более сложные паттерны поведения. В него включены направленные атаки с различными анимациями, улучшенная система навигации и уклонения от атак. Особое внимание уделено визуальной составляющей - каждое действие имеет свой набор анимаций, что делает поведение противника более естественным и предсказуемым для игрока.

Система босса, реализованная в классе Boss, представляет собой наиболее сложную боевую механику в игре. Она включает в себя систему фаз с различными паттернами атак, механику призыва миньонов и специальные визуальные эффекты. Босс способен адаптировать свое поведение в зависимости от оставшегося здоровья, что создает динамичные и сложные боевые столкновения. Каждая фаза босса имеет свои уникальные способности и паттерны атак, что требует от игрока изучения и адаптации стратегии.

Для оптимизации производительности реализован ряд технических решений. Использование системы тайловых карт с несколькими слоями позволяет эффективно отрисовывать большие территории. Генерация мира оптимизирована с использованием предварительных расчетов, а система валидации размещения объектов предотвращает возможные конфликты. Особое внимание уделено эффективному управлению анимациями и визуальными эффектами, что позволяет поддерживать стабильную производительность даже при большом количестве объектов на экране.

**Заключение**

Разработанная программа полностью соответствует поставленным задачам и представляет собой уникальное RPG с процедурной генерацией мира и продвинутой боевой системой. Программа обладает динамичным игровым процессом, разнообразными противниками и впечатляющими боссами, что создает захватывающий игровой опыт.

Проект демонстрирует успешную интеграцию различных игровых систем: процедурная генерация создает уникальный мир для каждого прохождения, многоуровневая система противников обеспечивает постоянный вызов игроку, а продвинутая система босса с различными фазами и механиками добавляет эпические сражения. Особого внимания заслуживает система генерации мира, которая создает разнообразные биомы и органично размещает объекты и поселения, что значительно повышает реиграбельность.

Техническая реализация проекта демонстрирует высокий уровень оптимизации: эффективное использование тайловых карт, продуманная система навигации противников и оптимизированные визуальные эффекты обеспечивают плавную работу игры даже при большом количестве объектов на экране. Модульная архитектура и четкая иерархия классов противников позволяют легко расширять функционал и добавлять новый контент в будущем.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра «Измерительно-вычислительные комплексы»

РУКОВОДСТВО ПРОГРАММИСТА

на курсовую работу

по дисциплине «Алгоритмы и структуры данных»

Тема: «Компьютерная игра “WWK. Часть 2”»

**Инв.№ подл.**

**Подп. и дата**

**Взам.инв. №**

**Инв.№ дубл.**

**Подп. и дата**

Р.02069337. 23/712 23 РП-01

Листов 6

Руководитель разработки:

к.т.н, доцент кафедры "ИВК"

Шишкин Вадим Викторинович

« » 2024 г.

Исполнитель:

студент гр. ИСТбд-21

Саранцев Семён Александрович

« » 2024 г.

2024

# **Руководство программиста**

на выполнение курсовой работы

по дисциплине «Алгоритмы и структуры данных»

Специальность 09.03.02 – «Информационные системы и технологии»

Тема: «**Разработка компьютерной игры “WWK. Часть 2”**»

**1. Назначение и условия применения программы**

**1.1. Назначение программы**

Программа представляет собой RPG игру с процедурной генерацией мира и развитой боевой системой. Она визуализирует процедурно генерируемый мир, различных противников и боссов.

**1.2. Функции программы**

Программа выполняет следующие функции:

• Процедурная генерация игрового мира

• Система противников разных типов

• Продвинутая боевая система

• Система босса с фазами

• Визуальные эффекты

• Система навигации противников

**1.3. Условия применения**

Для использования программы необходимы:

• Godot Engine версии 4.x

Следующие файлы скриптов:

worldgenerator.gd

enemy.gd

enemy2.gd

boss.gd

Графические ресурсы:

Тайлсеты для генерации мира

Спрайты противников

Спрайты босса

Текстуры эффектов

**2. Характеристика программы**

Программа реализована с использованием объектно-ориентированного подхода в среде Godot Engine.

* Основными классами являются:

WorldGenerator (генерация мира)

Enemy (базовый противник)

Enemy2 (улучшенный противник)

Boss (босс)

* Основные характеристики

Программа использует встроенный движок Godot для отрисовки графики и обработки физики. Игровой процесс происходит в реальном времени с процедурно генерируемым миром, динамической боевой системой и сложными противниками.

**3. Обращение к программе**

**3.1. Класс WorldGenerator (worldgenerator.gd)**

**3.1.1. Метод generate\_world()**

Назначение: генерация игрового мира.

**3.1.2. Метод place\_settlements()**

Назначение: размещение поселений.

**3.1.3. Метод validate\_generation()**

Назначение: проверка корректности генерации.

**3.2. Класс Enemy (enemy.gd)**

**3.2.1. Метод take\_damage()**

Назначение: обработка получения урона.

**3.2.2. Метод perform\_attack()**

Назначение: выполнение атаки.

**3.2.3. Метод update\_aggro()**

Назначение: обновление состояния агрессии.

**3.3. Класс Enemy2 (enemy2.gd)**

**3.3.1. Метод perform\_directional\_attack()**

Назначение: выполнение направленной атаки.

**3.3.2. Метод handle\_animations()**

Назначение: управление анимациями.

**3.3.3. Метод dodge()**

Назначение: выполнение уклонения.

**3.3.4. Метод enhanced\_pathfinding()**

Назначение: улучшенный поиск пути.

**3.4. Класс Boss (boss.gd)**

**3.4.1. Метод change\_phase()**

Назначение: смена фазы босса.

**3.4.2. Метод special\_attack()**

Назначение: выполнение специальной атаки.

**3.4.3. Метод summon\_minions()**

Назначение: призыв миньонов.

**3.4.4. Метод rage\_mode()**

Назначение: активация режима ярости.

**4. Входные и выходные данные**

**4.1. Входные данные**

• Параметры генерации мира (сиды для шумов Перлина)

• Пользовательский ввод (клавиатура, мышь)

• Конфигурационные файлы противников

• Параметры боссов и их фаз

* Настройки визуальных эффектов

**4.2. Выходные данные**

• Процедурно сгенерированный мир

• Визуализация противников и их состояний

• Эффекты атак и способностей

• Системные сообщения

• Состояние босса и его фаз

**5. Сообщения**

Программа выводит следующие типы сообщений:

• Игровые события (появление противников, смена фаз босса)

• Системные уведомления

• Предупреждения о близости противников

• Индикаторы урона

• Сообщения об ошибках генерации

**6. Используемые технические средства**

**6.1. Программное обеспечение**

• Godot Engine 4.x

• GDScript

• Встроенные модули Godot:

TileMap

CharacterBody2D

NavigationAgent2D

AnimatedSprite2D

Area2D

**6.2. Ресурсы**

• Тайлсеты для генерации мира

• Спрайты и анимации противников

• Визуальные эффекты

• Звуковые эффекты

• Шейдеры

**7. Особенности реализации**

7.1. Архитектурные решения

• Модульная система генерации мира

• Иерархия классов противников

• Система состояний для босса

• Компонентный подход к визуальным эффектам

• Событийно-ориентированная модель взаимодействия

7.2. Алгоритмические особенности

• Алгоритмы шума Перлина для генерации мира

• Система поиска пути для противников

• Алгоритмы определения фаз босса

• Расчет позиций для спавна миньонов

• Система весов для визуальных эффектов

7.3. Оптимизация

• Эффективная генерация тайловых карт

• Оптимизация навигации противников

• Система пулинга для визуальных эффектов

• Кэширование результатов генерации

• Оптимизация анимаций

7.4. Расширяемость

• Возможность добавления новых типов биомов

• Расширение системы противников

• Добавление новых фаз босса

• Создание новых визуальных эффектов

• Модификация параметров генерации мира

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра «Измерительно-вычислительные комплексы»

ТЕСТОВАЯ ДОКУМЕНТАЦИЯ

на курсовую работу

по дисциплине «Алгоритмы и структуры данных»

Тема: «Компьютерная игра “WWK. Часть 2”»

**Инв.№ подл.**

**Подп. и дата**

**Взам.инв. №**

**Инв.№ дубл.**

**Подп. и дата**

Р.02069337. 23/712 23 ТД-01

Листов 6

Руководитель разработки:

к.т.н, доцент кафедры "ИВК"

Шишкин Вадим Викторинович

« » 2024 г.

Исполнитель:

студент гр. ИСТбд-21

Саранцев Семён Александрович

« » 2024 г.

2024

# **Тестовая документация**

на выполнение курсовой работы

по дисциплине «Алгоритмы и структуры данных»

Специальность 09.03.02 – «Информационные системы и технологии»

Тема: «**Разработка компьютерной игры “WWK. Часть 2”**»

**1. Описание тестируемых функций**

В процессе тестирования игры необходимо проверить следующие функции:

**worldgenerator.gd:**

1. generate\_world():

* Описание: Генерирует игровой мир с использованием шумов Перлина.
* Тестируемые моменты:
* Корректная генерация ландшафта
* Правильное размещение биомов
* Валидация генерации
* Размещение объектов

1. place\_settlements():

* Описание: Размещает поселения в мире.
* Тестируемые моменты:
* Корректное размещение
* Проверка расстояний
* Валидация позиций
* Генерация безопасных зон

**enemy.gd:**

1. take\_damage():

* Описание: Обработка получения урона противником.
* Тестируемые моменты:
* Корректный расчет урона
* Анимации получения урона
* Обработка смерти
* Выпадение предметов

1. perform\_attack():

* Описание: Выполнение атаки противником.
* Тестируемые моменты:
* Расчет урона
* Анимации атаки
* Проверка дистанции
* Коллизии атаки

**enemy2.gd:**

1. perform\_directional\_attack():

* Описание: Выполнение направленной атаки.
* Тестируемые моменты:
* Направление атаки
* Специальные эффекты
* Улучшенные анимации
* Расчет урона

**boss.gd:**

1. change\_phase():

* Описание: Смена фазы босса.
* Тестируемые моменты:
* Изменение паттернов
* Новые способности
* Визуальные эффекты

**2. Mind map**

Для тестирования игры можно организовать карту разума (mind map), чтобы структурировать процесс (Приложение 1)

**3. Чеклист**

|  | **Тестируемая функция** | **Шаги выполнения** | **Ожидаемый результат** | **Статус** |
| --- | --- | --- | --- | --- |
| 1 | generate\_world() | Запуск генерации мира | Мир сгенерирован корректно | Пройдено/Не пройдено |
| 2 | place\_settlements() | Размещение поселений | Поселения размещены правильно | Пройдено/Не пройдено |
| 3 | enemy.take\_damage() | Нанесение урона противнику | Урон обработан корректно | Пройдено/Не пройдено |
| 4 | enemy.perform\_attack() | Атака противника | Атака выполнена правильно | Пройдено/Не пройдено |
| 5 | enemy2.perform\_directional\_attack() | Направленная атака | Атака выполнена с правильным направлением | Пройдено/Не пройдено |
| 6 | boss.change\_phase() | Смена фазы босса | Фаза изменена корректно | Пройдено/Не пройдено |
| 7 | boss.summon\_minions() | Призыв миньонов | Миньоны призваны правильно | Пройдено/Не пройдено |

**4. Набор тест-кейсов**

**Тест-кейс 1: Генерация мира**

**Цель:** Проверка корректности генерации игрового мира

**Предусловия:** Игра запущена

**Шаги:**

1. Вызвать generate\_world()

2. Проверить генерацию ландшафта

3. Проверить размещение биомов

4. Проверить валидацию генерации

Ожидаемый результат: Мир сгенерирован корректно, все биомы и объекты размещены правильно

**Тест-кейс 2:** Базовый противник

**Цель:** Проверка функционала базового противника

**Предусловия:** Противник размещен в мире

**Шаги:**

1. Проверить систему агрессии

2. Выполнить базовую атаку

3. Нанести урон противнику

4. Проверить смерть противника

Ожидаемый результат: Противник корректно реагирует на все действия

**Тест-кейс 3:** Улучшенный противник

**Цель:** Проверка функционала улучшенного противника

**Предусловия:** Улучшенный противник размещен в мире

**Шаги:**

1. Проверить направленные атаки

2. Проверить систему уклонения

3. Проверить улучшенную навигацию

Ожидаемый результат: Улучшенный противник демонстрирует расширенный функционал

**Тест-кейс 4:** Система босса

**Цель:** Проверка функционала босса

**Предусловия:** Босс активирован

**Шаги:**

1. Проверить смену фаз

2. Проверить специальные атаки

3. Проверить призыв миньонов

4. Проверить визуальные эффекты

Ожидаемый результат: Босс корректно меняет фазы и использует способности.

5**. Критерии тестирования**

Генерация мира:

- Плавность переходов между биомами

- Отсутствие визуальных артефактов

- Корректность размещения объектов

- Производительность генерации

**Боевая система:**

- Точность попадания

- Баланс урона

- Корректность анимаций

- Отзывчивость управления

**Искусственный интеллект:**

- Поведение противников

- Точность навигации

- Реакция на действия игрока

- Сложность боя

**6. Отчет о тестировании**

Для каждого теста указывается:

- Дата проведения

- Версия сборки

- Результат тестирования

- Обнаруженные ошибки

- Скриншоты (при необходимости)

**7. Окружение тестирования**

Технические требования:

- Версия Godot Engine: 4.x

- Операционная система

- Минимальные системные требования

- Разрешение экрана

# 

# **Листинг кода**

на выполнение курсовой работы

по дисциплине «Алгоритмы и структуры данных»

Специальность 09.03.02 – «Информационные системы и технологии»

Тема: «**Разработка компьютерной игры “WWK. Часть 2”**»

worldgenerator.gd:

extends Node2D

const SAFETY\_WATER\_WEIGHT = 2.0

const SAFETY\_WALKABLE\_WEIGHT = 1.0

const SAFETY\_CENTER\_PENALTY = 0.5

const CHECK\_RADIUS = 5

const WALLS\_WIDTH = 140

const WALLS\_HEIGHT = 140

const WALLS\_MIN\_DISTANCE = 200

const WALLS\_REQUIRED\_COUNT = 1

const BORDER\_SAFE\_DISTANCE = 40

const VILLAGE\_SAFE\_DISTANCE = 100

# Экспорт текстур шума для настройки через редактор

@export var noise\_height\_text:NoiseTexture2D # Шум высоты для рельефа - определяет высоту местности

@export var noise\_tree\_text:NoiseTexture2D # Шум для деревьев - определяет где будут расти деревья

@export var noise\_temp\_text:NoiseTexture2D # Шум температуры - влияет на тип биома

@export var noise\_moisture\_text:NoiseTexture2D # Шум влажности - влияет на тип биома

@export var noise\_settlement\_text:NoiseTexture2D # Шум для поселений

@export var grave\_scene:PackedScene # Сцена надгробия

@export var walls\_scene: PackedScene # Сцена с боссом

# Сцена поселения и игрока

@export var village\_scene:PackedScene

@export var village\_middle\_scene:PackedScene # Добавляем экспорт сцены средней деревни

@onready var player = get\_tree().get\_root().get\_node("Game/Player") # Получаем ссылку на игрока

@onready var camera = get\_tree().get\_root().get\_node("Game/Player/Camera2D") # Получаем ссылку на камеру

# Настройки размещения поселений

const VILLAGE\_MIN\_DISTANCE = 100 # Расстояние между деревнями

# Размеры маленькой деревни

const SMALL\_VILLAGE\_WIDTH = 38

const SMALL\_VILLAGE\_HEIGHT = 35

# Размеры средней деревни

const MIDDLE\_VILLAGE\_WIDTH = 50 # Увеличенная ширина для средней деревни

const MIDDLE\_VILLAGE\_HEIGHT = 35 # Увеличенная высота для средней деревни

const VILLAGE\_BORDER = 20 # Отступ от края карты

const WATER\_SAFE\_DISTANCE = 80 # Безопасное расстояние от воды

const TREE\_SAFE\_DISTANCE = 80 # Безопасное расстояние от деревьев

const MAX\_GENERATION\_ATTEMPTS = 20 # Максималное количество попыток генерации мира

const WATER\_BORDER\_WIDTH = 32 # Ширина водной границы в тайлах

# Размеры области надгробия

const GRAVE\_AREA\_WIDTH = 5

const GRAVE\_AREA\_HEIGHT = 5

# В начале фала доавим константы для количества надгробий

const GRAVES\_PER\_AREA = 400 # Увеличиваем с 200 до 400

enum Biome {OCEAN, BEACH, TUNDRA, FOREST, PLAINS, DESERT}

enum SettlementType {VILLAGE, GRAVE, WALLS}

# Настройки шумов

var noise:Noise # Шум для генерации высоты

var tree\_noise:Noise # Шум для генерации деревьев

var temp\_noise:Noise # Шум для температуры

var moisture\_noise:Noise # Шум для влажности

var settlement\_noise:Noise # Шум для поселений

# Размеры карты

var width:int=1024 # Увеличиваем размер карты для больших деревень

var height:int=1024 # Увеличиваем размер карты для больших деревень

# Тайлы

var water\_atlas=Vector2i(0,2) # Координаты тайла воды

var grass\_tiles\_arr=[] # Массив для тайлов травы

var terrain\_grass\_int=0 # Индекс текущего тайла травы

var grass\_atlas\_arr=[Vector2i(1,1),Vector2i(5,1),Vector2i(5,0),Vector2i(5,2)]

# Массив тайлов для пустыни

var desert\_atlas\_arr=[

Vector2i(2,6), # Обычный песок

Vector2i(3,6), # Песок с кактсом

Vector2i(0,4) # Псок с камнями

]

var thread: Thread

var cell\_map:Array = []

var settlements:Array = []

# Структура для хранения информации о поселении

class Settlement:

var type: int

var position: Vector2i

var scene: Node2D

func \_init(t: int, pos: Vector2i, s: Node2D):

type = t

position = pos

scene = s

# В начале файла добавим новые константы

const MAX\_ATTEMPTS\_PER\_GENERATION = 20 # Максимальное число попыток для одной генерации

const GENERATION\_TIMEOUT = 30.0 # Таймаут в секундах для всего процесса генерации

# Добавляем новые константы для контроля генерации

const MAX\_TOTAL\_ATTEMPTS = 100 # Максимальное общее количество попыток

const RETRY\_DELAY = 0.05 # Задержка между попытками в секундах

const MIN\_REQUIRED\_GRAVES = 10 # Минимальное количество надгробий для успешной генерации

# Добавляем новые константы для безопасных расстояний

const SMALL\_VILLAGE\_WATER\_SAFE\_DISTANCE = 80 # Для маленькой деревни

const MIDDLE\_VILLAGE\_WATER\_SAFE\_DISTANCE = 120 # Увеличенное расстояние для средней деревни

# Добавьте переменные для хранения позиций

var small\_village\_position: Vector2

var middle\_village\_position: Vector2

var walls\_position: Vector2 = Vector2(512, 512) \* 32 # Устанавливаем позицию по умолчанию в центре карты

signal locations\_updated(small\_pos: Vector2, middle\_pos: Vector2, walls\_pos: Vector2)

func \_ready() -> void:

if !validate\_requirements():

return

emit\_signal("locations\_updated", small\_village\_position, middle\_village\_position, walls\_position)

await generate\_valid\_world()

func validate\_requirements() -> bool:

if !player or !camera:

push\_error("КРИТИЧЕСКАЯ ОШИБКА: Не удалось найти игрока или камеру!")

return false

if !village\_scene or !village\_middle\_scene or !grave\_scene:

push\_error("КРИТИЧЕСКАЯ ОШИБКА: Отсутствуют необходимые сцены!")

return false

if !$water or !$terrain or !$grass or !$plants:

push\_error("КРИТИЧЕСКАЯ ОШИБКА: Отсутствуют необходимые тайлмапы!")

return false

return true

func generate\_valid\_world() -> void:

var total\_attempts = 0

var world\_generated = false

while !world\_generated and total\_attempts < MAX\_TOTAL\_ATTEMPTS:

total\_attempts += 1

print("Попытка полной генерации мира #", total\_attempts)

if await try\_generate\_world():

if validate\_world\_state():

world\_generated = true

print("Мир успешно сгенерирован на попытке ", total\_attempts)

break

await get\_tree().create\_timer(RETRY\_DELAY).timeout

if !world\_generated:

push\_error("Не удалось сгенерировать валидный мир после " + str(total\_attempts) + " попыток")

func try\_generate\_world() -> bool:

await clear\_previous\_generation()

var current\_seed = randi()

setup\_noise(current\_seed)

print("Используется сид: ", current\_seed)

generate\_world\_data()

generate\_water\_borders()

if !await place\_settlements():

return false

setup\_camera\_and\_player()

return true

func validate\_world\_state() -> bool:

# Проверяем наличие маленькой деревни

var small\_village = find\_small\_village()

if !small\_village:

print("Валидация: отсутствует маленькая деревня")

return false

# Проверяем наличие средней деревни

var has\_middle\_village = false

for settlement in settlements:

if settlement.type == SettlementType.VILLAGE and settlement.scene.scene\_file\_path == village\_middle\_scene.resource\_path:

has\_middle\_village = true

break

if !has\_middle\_village:

print("Валидация: отсутствует средняя деревня")

return false

# Проверяем количество надгробий

var grave\_count = 0

for settlement in settlements:

if settlement.type == SettlementType.GRAVE:

grave\_count += 1

if grave\_count < MIN\_REQUIRED\_GRAVES:

print("Валидация: недостаточно надгробий (", grave\_count, "/", MIN\_REQUIRED\_GRAVES, ")")

return false

# Проверяем доступность всех деревень (нет окружения водой)

if !validate\_settlements\_accessibility():

print("Валидация: проблемы с доступностью деревень")

return false

return true

func validate\_settlements\_accessibility() -> bool:

for settlement in settlements:

if settlement.type != SettlementType.VILLAGE:

continue

var pos = settlement.position

var width = SMALL\_VILLAGE\_WIDTH

var height = SMALL\_VILLAGE\_HEIGHT

if settlement.scene.scene\_file\_path == village\_middle\_scene.resource\_path:

width = MIDDLE\_VILLAGE\_WIDTH

height = MIDDLE\_VILLAGE\_HEIGHT

# Проверяем наличие прохода к деревне

var has\_access = false

for check\_dir in [Vector2i(1,0), Vector2i(-1,0), Vector2i(0,1), Vector2i(0,-1)]:

var check\_pos = pos + check\_dir \* (width/2)

if is\_valid\_settlement\_point(check\_pos):

has\_access = true

break

if !has\_access:

return false

return true

func clear\_previous\_generation() -> void:

for settlement in settlements:

if is\_instance\_valid(settlement.scene):

settlement.scene.queue\_free()

await get\_tree().process\_frame

settlements.clear()

# Очищаем карту и тайлмапы

cell\_map.clear()

cell\_map.resize(width)

for x in width:

cell\_map[x] = []

cell\_map[x].resize(height)

$water.clear()

$terrain.clear()

$grass.clear()

$plants.clear()

func place\_settlements() -> bool:

# Сначала размещаем стены

if !place\_walls():

print("Не удалось разместить стены")

return false

# Добавляем задержку для обработки размещения стен

await get\_tree().create\_timer(0.1).timeout

var middle\_locations = find\_suitable\_locations(village\_middle\_scene, WALLS\_MIN\_DISTANCE \* 2)

print("Найдено подходящих мест для средней деревни: ", middle\_locations.size())

if middle\_locations.is\_empty():

print("Не удалось найти место для средней деревни")

return false

var middle\_pos = middle\_locations.pick\_random()

if !spawn\_settlement(village\_middle\_scene, middle\_pos, SettlementType.VILLAGE, MIDDLE\_VILLAGE\_WIDTH, MIDDLE\_VILLAGE\_HEIGHT):

return false

print("Создана средняя деревня в позиции: ", middle\_pos)

var small\_locations = find\_suitable\_locations(village\_scene)

print("Найдено подходящих мест для маленькой деревни: ", small\_locations.size())

if small\_locations.is\_empty():

print("Не удалось найти место для маленькой деревни")

return false

var best\_small\_pos = find\_safest\_location(small\_locations)

if !spawn\_settlement(village\_scene, best\_small\_pos, SettlementType.VILLAGE, SMALL\_VILLAGE\_WIDTH, SMALL\_VILLAGE\_HEIGHT):

return false

print("Создана маленькая деревня в позиции: ", best\_small\_pos)

place\_graves()

return true

func find\_safest\_location(locations: Array) -> Vector2i:

var best\_location = locations[0]

var max\_safety\_score = -1

for pos in locations:

var safety\_score = calculate\_safety\_score(pos)

if safety\_score > max\_safety\_score:

max\_safety\_score = safety\_score

best\_location = pos

return best\_location

func calculate\_safety\_score(pos: Vector2i) -> float:

var score = 0.0

# Проверяем расстояние до воды

var water\_distance = get\_min\_water\_distance(pos)

score += water\_distance \* 2

# Проверяем количество проходимой территории вокруг

var walkable\_tiles = count\_walkable\_tiles(pos)

score += walkable\_tiles

# Проверяем центральность позиции

var center\_distance = pos.distance\_to(Vector2i(width/2, height/2))

score -= center\_distance \* 0.5 # Штраф за удаленность от центра

return score

func get\_min\_water\_distance(pos: Vector2i) -> float:

var min\_distance = 999999.0

for x in range(max(0, pos.x - WATER\_SAFE\_DISTANCE), min(width, pos.x + SMALL\_VILLAGE\_WIDTH + WATER\_SAFE\_DISTANCE)):

for y in range(max(0, pos.y - WATER\_SAFE\_DISTANCE), min(height, pos.y + SMALL\_VILLAGE\_HEIGHT + WATER\_SAFE\_DISTANCE)):

if cell\_map[x][y] == Biome.OCEAN:

var dist = pos.distance\_to(Vector2i(x, y))

min\_distance = min(min\_distance, dist)

return min\_distance

func count\_walkable\_tiles(pos: Vector2i) -> int:

var count = 0

var check\_radius = CHECK\_RADIUS

for x in range(max(0, pos.x - check\_radius), min(width, pos.x + SMALL\_VILLAGE\_WIDTH + check\_radius)):

for y in range(max(0, pos.y - check\_radius), min(height, pos.y + SMALL\_VILLAGE\_HEIGHT + check\_radius)):

if x >= width or y >= height:

continue

if cell\_map[x][y] != Biome.OCEAN:

count += 1

return count

func find\_suitable\_locations(scene: PackedScene, safe\_distance: int = VILLAGE\_MIN\_DISTANCE) -> Array:

var locations = []

var village\_width = SMALL\_VILLAGE\_WIDTH

var village\_height = SMALL\_VILLAGE\_HEIGHT

if scene == village\_middle\_scene:

village\_width = MIDDLE\_VILLAGE\_WIDTH

village\_height = MIDDLE\_VILLAGE\_HEIGHT

# Адаптивный шаг в зависимости от размера карты

var step = max(4, min(village\_width, village\_height) / 4)

# Используем пул точек для оптимизации

var check\_points = []

for x in range(VILLAGE\_BORDER, width - VILLAGE\_BORDER - village\_width, step):

for y in range(VILLAGE\_BORDER, height - VILLAGE\_BORDER - village\_height, step):

check\_points.append(Vector2i(x,y))

# Перемешиваем точки для более равномерного распределения

check\_points.shuffle()

# Проверяем только необходимое количество точек

for point in check\_points:

if is\_suitable\_for\_settlement(point, scene):

locations.append(point)

if locations.size() >= 10:

break

return locations

func is\_suitable\_for\_settlement(pos: Vector2i, scene: PackedScene) -> bool:

var village\_width = SMALL\_VILLAGE\_WIDTH

var village\_height = SMALL\_VILLAGE\_HEIGHT

if scene == village\_middle\_scene:

village\_width = MIDDLE\_VILLAGE\_WIDTH

village\_height = MIDDLE\_VILLAGE\_HEIGHT

# Проверяем границы

if pos.x < VILLAGE\_BORDER or pos.x + village\_width >= width - VILLAGE\_BORDER:

return false

if pos.y < VILLAGE\_BORDER or pos.y + village\_height >= height - VILLAGE\_BORDER:

return false

# Проверяем расстояние до других поселений

for settlement in settlements:

var distance = pos.distance\_to(settlement.position)

if settlement.type == SettlementType.WALLS:

if distance < WALLS\_MIN\_DISTANCE: # Используем увеличенное расстояние до стен

return false

elif settlement.type == SettlementType.VILLAGE:

if distance < VILLAGE\_MIN\_DISTANCE:

return false

# Проверяем область на наличие воды

for x in range(pos.x, pos.x + village\_width):

for y in range(pos.y, pos.y + village\_height):

if x >= width or y >= height:

return false

if cell\_map[x][y] == Biome.OCEAN:

return false

return true

func is\_valid\_settlement\_point(point: Vector2i) -> bool:

if point.x < 0 or point.x >= width or point.y < 0 or point.y >= height:

return false

if cell\_map[point.x][point.y] == Biome.OCEAN:

return false

if $plants.get\_cell\_source\_id(point) == 1:

return false

return true

func generate\_world\_data() -> void:

var plains\_count = 0

for x in width:

for y in height:

var height\_val = noise.get\_noise\_2d(x,y)

var temp\_val = temp\_noise.get\_noise\_2d(x,y)

var moisture\_val = moisture\_noise.get\_noise\_2d(x,y)

var tree\_val = tree\_noise.get\_noise\_2d(x,y)

var biome = get\_biome(height\_val, temp\_val, moisture\_val)

cell\_map[x][y] = biome

if biome == Biome.PLAINS:

plains\_count += 1

match biome:

Biome.OCEAN:

$water.set\_cell(Vector2i(x,y), 0, water\_atlas)

Biome.BEACH:

$terrain.set\_cell(Vector2i(x,y), 3, Vector2i(4,4))

Biome.TUNDRA:

$terrain.set\_cell(Vector2i(x,y), 8, Vector2i(0,0))

Biome.FOREST:

$grass.set\_cell(Vector2i(x,y), 0, grass\_atlas\_arr.pick\_random())

if tree\_val > 0.4:

$plants.set\_cell(Vector2i(x,y), 1, Vector2i(0,2))

Biome.PLAINS:

$grass.set\_cell(Vector2i(x,y), 0, grass\_atlas\_arr.pick\_random())

Biome.DESERT:

$terrain.set\_cell(Vector2i(x,y), 5, desert\_atlas\_arr.pick\_random())

print("Количество равнин на карте: ", plains\_count)

func generate\_water\_borders() -> void:

for x in width:

for y in height:

# Проверяем, находится ли точка в пределах границы

if x < WATER\_BORDER\_WIDTH or x >= width - WATER\_BORDER\_WIDTH or y < WATER\_BORDER\_WIDTH or y >= height - WATER\_BORDER\_WIDTH:

# Устанавливаем воду

cell\_map[x][y] = Biome.OCEAN

$terrain.erase\_cell(Vector2i(x,y))

$grass.erase\_cell(Vector2i(x,y))

$plants.erase\_cell(Vector2i(x,y))

$water.set\_cell(Vector2i(x,y), 0, water\_atlas)

func setup\_noise(generation\_seed: int) -> void:

# Настройка шума высот

noise = noise\_height\_text.noise

noise.seed = generation\_seed

noise.frequency = 0.005

# Настройка шума деревьев

tree\_noise = noise\_tree\_text.noise

tree\_noise.seed = generation\_seed + 1 # Разные сид для разных шумов

tree\_noise.frequency = 0.1

# Настройка шума температуры

temp\_noise = noise\_temp\_text.noise

temp\_noise.seed = generation\_seed + 2

temp\_noise.frequency = 0.005

# Настройка шума влажности

moisture\_noise = noise\_moisture\_text.noise

moisture\_noise.seed = generation\_seed + 3

moisture\_noise.frequency = 0.005

# Настройка шума поселений

settlement\_noise = noise\_settlement\_text.noise

settlement\_noise.seed = generation\_seed + 4

settlement\_noise.frequency = 0.01

func get\_biome(elevation:float, temp:float, moisture:float) -> int:

if elevation < -0.3: # Увеличиваем порог для океана

return Biome.OCEAN

if elevation < -0.2: # Увеличиваем порог для пляжа

return Biome.BEACH

if temp < -0.2:

return Biome.TUNDRA

elif moisture > 0:

return Biome.FOREST

elif moisture < -0.5:

return Biome.DESERT

else: # Увеличиваем шанс появления равнин

return Biome.PLAINS

func spawn\_settlement(scene: PackedScene, pos: Vector2i, type: int, village\_width: int, village\_height: int) -> bool:

if scene == null:

push\_error("ОШИБКА: scene не установлена!")

return false

var settlement\_instance = scene.instantiate()

if settlement\_instance == null:

push\_error("ОШИБКА: Не удалось создать экземпляр!")

return false

# Очищаем область под поселением

clear\_area(pos, village\_width, village\_height)

settlement\_instance.position = Vector2(pos.x \* 32, pos.y \* 32)

add\_child(settlement\_instance)

# Сохраняем позиции при создании поселений

if scene == village\_scene:

small\_village\_position = Vector2(pos.x \* 32, pos.y \* 32)

elif scene == village\_middle\_scene:

middle\_village\_position = Vector2(pos.x \* 32, pos.y \* 32)

elif scene == walls\_scene:

walls\_position = Vector2(pos.x \* 32, pos.y \* 32)

# Отправляем сигнал с обновленными позициями

emit\_signal("locations\_updated", small\_village\_position, middle\_village\_position, walls\_position)

var settlement = Settlement.new(type, pos, settlement\_instance)

settlements.append(settlement)

return true

func check\_settlement\_overlap(pos: Vector2i, width: int, height: int) -> bool:

const MINIMUM\_GAP = 5 # Минимальный промежуток между деревнями

for settlement in settlements:

if settlement.type != SettlementType.VILLAGE:

continue

# Проверяем пересечение прямоугольников с учетом промежутка

var other\_pos = settlement.position

var other\_width = SMALL\_VILLAGE\_WIDTH

var other\_height = SMALL\_VILLAGE\_HEIGHT

if settlement.scene.scene\_file\_path == village\_middle\_scene.resource\_path:

other\_width = MIDDLE\_VILLAGE\_WIDTH

other\_height = MIDDLE\_VILLAGE\_HEIGHT

if (pos.x - MINIMUM\_GAP < other\_pos.x + other\_width + MINIMUM\_GAP and

pos.x + width + MINIMUM\_GAP > other\_pos.x - MINIMUM\_GAP and

pos.y - MINIMUM\_GAP < other\_pos.y + other\_height + MINIMUM\_GAP and

pos.y + height + MINIMUM\_GAP > other\_pos.y - MINIMUM\_GAP):

return true

return false

func clear\_settlement\_area(pos: Vector2i, village\_width: int, village\_height: int) -> bool:

for x in range(pos.x - TREE\_SAFE\_DISTANCE, pos.x + village\_width + TREE\_SAFE\_DISTANCE):

for y in range(pos.y - TREE\_SAFE\_DISTANCE, pos.y + village\_height + TREE\_SAFE\_DISTANCE):

if x < 0 or x >= width or y < 0 or y >= height:

continue

if $plants.get\_cell\_source\_id(Vector2i(x,y)) == 1:

$plants.erase\_cell(Vector2i(x,y))

if x >= pos.x and x < pos.x + village\_width and y >= pos.y and y < pos.y + village\_height:

if cell\_map[x][y] == Biome.OCEAN:

return false

$terrain.erase\_cell(Vector2i(x,y))

$grass.erase\_cell(Vector2i(x,y))

$water.erase\_cell(Vector2i(x,y))

return true

func find\_suitable\_locations\_for\_grave() -> Array:

var locations = []

for x in range(VILLAGE\_BORDER, width - VILLAGE\_BORDER - GRAVE\_AREA\_WIDTH):

for y in range(VILLAGE\_BORDER, height - VILLAGE\_BORDER - GRAVE\_AREA\_HEIGHT):

if is\_suitable\_for\_grave(Vector2i(x,y)):

locations.append(Vector2i(x,y))

return locations

func is\_suitable\_for\_grave(pos: Vector2i) -> bool:

# Безопасные расстояния для разных объектов

const VILLAGE\_SAFE\_DISTANCE = 50 # Большое расстояние от деревень

const WATER\_SAFE\_DISTANCE = 5 # Меньшее расстояние от воды

const TREE\_SAFE\_DISTANCE = 3 # Минимальное расстояние от деревьев

# Проверяем расстояние до других деревень

for settlement in settlements:

if settlement.type == SettlementType.VILLAGE: # Проверяем только для деревень

if pos.distance\_to(settlement.position) < VILLAGE\_SAFE\_DISTANCE:

return false

# Проверяем область под надгробие и безопасную зону вокруг

for x in range(pos.x - WATER\_SAFE\_DISTANCE, pos.x + GRAVE\_AREA\_WIDTH + WATER\_SAFE\_DISTANCE):

for y in range(pos.y - WATER\_SAFE\_DISTANCE, pos.y + GRAVE\_AREA\_HEIGHT + WATER\_SAFE\_DISTANCE):

if x < 0 or x >= width or y < 0 or y >= height:

return false

# Проверяем саму область надгробя

if x >= pos.x and x < pos.x + GRAVE\_AREA\_WIDTH and y >= pos.y and y < pos.y + GRAVE\_AREA\_HEIGHT:

# Разрешаем спавн на разных биомах, кроме океана

if cell\_map[x][y] == Biome.OCEAN:

return false

# Проверяем наличие воды в безопасной зоне

if cell\_map[x][y] == Biome.OCEAN:

return false

# Проверяем наличие деревьев в безопасной зоне

if $plants.get\_cell\_source\_id(Vector2i(x,y)) == 1:

var tree\_distance = Vector2i(x,y).distance\_to(pos)

if tree\_distance < TREE\_SAFE\_DISTANCE:

return false

return true

func spawn\_grave(scene: PackedScene, pos: Vector2i) -> bool:

if scene == null:

print("ОШИБКА: scene надгробия не установлена!")

return false

var grave\_instance = scene.instantiate()

if grave\_instance == null:

print("ОШИБКА: Не удалось создать экземпляр надгробия!")

return false

# Очищаем тайлы под надгробием

for x in range(pos.x, pos.x + GRAVE\_AREA\_WIDTH):

for y in range(pos.y, pos.y + GRAVE\_AREA\_HEIGHT):

if x >= 0 and x < width and y >= 0 and y < height:

$plants.erase\_cell(Vector2i(x,y))

$terrain.erase\_cell(Vector2i(x,y))

# Устанавливаем тайл травы

$grass.set\_cell(Vector2i(x,y), 0, grass\_atlas\_arr.pick\_random())

grave\_instance.position = Vector2(pos.x \* 32, pos.y \* 32)

add\_child(grave\_instance)

var settlement = Settlement.new(SettlementType.GRAVE, pos, grave\_instance)

settlements.append(settlement)

return true

func setup\_camera\_and\_player() -> void:

if camera:

camera.limit\_left = 0

camera.limit\_top = 0

camera.limit\_right = width \* 32

camera.limit\_bottom = height \* 32

# Ищем маленькую деревню для спавна игрока

var small\_village = find\_small\_village()

if small\_village != null and player:

var spawn\_offset = Vector2(SMALL\_VILLAGE\_WIDTH/2, SMALL\_VILLAGE\_HEIGHT/2) \* 32

player.position = Vector2(small\_village.position.x \* 32, small\_village.position.y \* 32) + spawn\_offset

player.add\_to\_group("player")

print("Игрок перемещен в маленькую деревню: ", player.position)

else:

push\_error("КРИТИЧЕСКАЯ ОШИБКА: Маленькая деревня не найдена!")

# Перезапускаем генерацию мира

clear\_previous\_generation()

var new\_seed = randi() # Генерируем новый сид

setup\_noise(new\_seed) # Передаем сид в функцию

generate\_world\_data()

generate\_water\_borders()

if !await place\_settlements():

push\_error("Не удалось сгенерировать мир с маленькой деревней!")

func find\_small\_village() -> Settlement:

# Ищем именно маленькую деревню (проверяем и тип, и сцену)

for settlement in settlements:

if settlement.type == SettlementType.VILLAGE and settlement.scene.scene\_file\_path == village\_scene.resource\_path:

return settlement

return null

func place\_graves() -> void:

var graves\_placed = 0

var chunk\_size = 64 # Размер чанка для оптимизации поиска

var chunks = [] # Массив подходящих чанков

# Разбиваем карту на чанки и находим подходящие

for x in range(BORDER\_SAFE\_DISTANCE, width - BORDER\_SAFE\_DISTANCE, chunk\_size):

for y in range(BORDER\_SAFE\_DISTANCE, height - BORDER\_SAFE\_DISTANCE, chunk\_size):

var chunk\_pos = Vector2i(x, y)

if is\_chunk\_suitable\_for\_graves(chunk\_pos, chunk\_size):

chunks.append(chunk\_pos)

# Перемешиваем чанки

chunks.shuffle()

# Размещаем надгробия в подходящих чанках

for chunk\_pos in chunks:

if graves\_placed >= GRAVES\_PER\_AREA:

break

# Пытаемся разместить несколько надгробий в одном чанке

var graves\_in\_chunk = 0

var max\_graves\_per\_chunk = 12 # Увеличиваем максимум надгробий в чанке

var attempts = 0

while graves\_in\_chunk < max\_graves\_per\_chunk and graves\_placed < GRAVES\_PER\_AREA and attempts < 20:

var x = chunk\_pos.x + randi() % (chunk\_size - GRAVE\_AREA\_WIDTH)

var y = chunk\_pos.y + randi() % (chunk\_size - GRAVE\_AREA\_HEIGHT)

var grave\_pos = Vector2i(x, y)

if is\_suitable\_for\_grave(grave\_pos):

if spawn\_grave(grave\_scene, grave\_pos):

graves\_placed += 1

graves\_in\_chunk += 1

attempts += 1

print("Всего размещено надгробий: ", graves\_placed)

func is\_chunk\_suitable\_for\_graves(chunk\_pos: Vector2i, chunk\_size: int) -> bool:

# Быстрая проверка чанка

var check\_points = [

chunk\_pos,

chunk\_pos + Vector2i(chunk\_size, 0),

chunk\_pos + Vector2i(0, chunk\_size),

chunk\_pos + Vector2i(chunk\_size, chunk\_size),

chunk\_pos + Vector2i(chunk\_size/2, chunk\_size/2)

]

# Проверяем расстояние до поселений

for settlement in settlements:

if settlement.type != SettlementType.GRAVE:

var dist = chunk\_pos.distance\_to(settlement.position)

if dist < VILLAGE\_SAFE\_DISTANCE \* 2: # Увеличенная зона проверки

return false

# Проверяем наличие воды в ключевых точках

for point in check\_points:

if point.x >= width or point.y >= height:

return false

if cell\_map[point.x][point.y] == Biome.OCEAN:

return false

return true

func place\_walls() -> bool:

var attempts = 0

var walls\_placed = 0

# Сначала пробуем разместить в центре

var center\_x = width / 2 - WALLS\_WIDTH / 2

var center\_y = height / 2 - WALLS\_HEIGHT / 2

var center\_pos = Vector2i(center\_x, center\_y)

if is\_position\_suitable\_for\_walls(center\_pos):

if spawn\_walls(walls\_scene, center\_pos):

print("Размещена большая локация в центре карты")

return true

# Если центр не подходит, ищем по сетке с большим шагом

var grid\_step = 100 # Увеличенный шаг сетки

for x in range(BORDER\_SAFE\_DISTANCE, width - BORDER\_SAFE\_DISTANCE - WALLS\_WIDTH, grid\_step):

for y in range(BORDER\_SAFE\_DISTANCE, height - BORDER\_SAFE\_DISTANCE - WALLS\_HEIGHT, grid\_step):

var pos = Vector2i(x, y)

if is\_position\_suitable\_for\_walls(pos):

if spawn\_walls(walls\_scene, pos):

print("Размещена большая локация в позиции: ", pos)

return true

print("Не удалось найти подходящее место для большой локации")

return false

func is\_position\_suitable\_for\_walls(pos: Vector2i) -> bool:

# Проверяем границы карты

if pos.x < BORDER\_SAFE\_DISTANCE or pos.y < BORDER\_SAFE\_DISTANCE or \

pos.x + WALLS\_WIDTH > width - BORDER\_SAFE\_DISTANCE or \

pos.y + WALLS\_HEIGHT > height - BORDER\_SAFE\_DISTANCE:

return false

# Проверяем расстояние до других поселений

for settlement in settlements:

var dist = pos.distance\_to(settlement.position)

if settlement.type == SettlementType.VILLAGE:

if dist < WALLS\_MIN\_DISTANCE: # Используем увеличенное расстояние для деревень

return false

elif settlement.type == SettlementType.WALLS:

if dist < WALLS\_MIN\_DISTANCE:

return false

return true

func spawn\_walls(scene: PackedScene, pos: Vector2i) -> bool:

if scene == null:

push\_error("ОШИБКА: scene большой локации не установлена!")

return false

var walls\_instance = scene.instantiate()

if walls\_instance == null:

push\_error("ОШИБКА: Не удалось создать экземпляр большой локации!")

return false

# Очищаем всю область под стенами

clear\_area(pos, WALLS\_WIDTH, WALLS\_HEIGHT)

walls\_instance.position = Vector2(pos.x \* 32, pos.y \* 32)

add\_child(walls\_instance)

var settlement = Settlement.new(SettlementType.WALLS, pos, walls\_instance)

settlements.append(settlement)

return true

func clear\_area(pos: Vector2i, width: int, height: int) -> void:

for x in range(pos.x, pos.x + width):

for y in range(pos.y, pos.y + height):

if x >= 0 and x < self.width and y >= 0 and y < self.height:

var cell\_pos = Vector2i(x, y)

# Очищаем все слои

$plants.erase\_cell(cell\_pos)

$terrain.erase\_cell(cell\_pos)

$water.erase\_cell(cell\_pos) # Удаляем воду

# Ставим траву

$grass.set\_cell(cell\_pos, 0, grass\_atlas\_arr.pick\_random())

# Обновляем cell\_map

if cell\_map[x][y] == Biome.OCEAN:

cell\_map[x][y] = Biome.PLAINS

func \_process(\_delta:float)->void:

pass  
enemy.gd:

extends CharacterBody2D

var SPEED = 100.0 # базовая скорость

var DODGE\_SPEED = 300.0 # скорость уклонения

var BASE\_HEALTH = 200 # базовое здоровье

var BASE\_ATTACK = 15 # базовая атака

var BASE\_DEFENSE = 10 # базовая защита

var health = BASE\_HEALTH

var max\_health = BASE\_HEALTH

var attack\_power = BASE\_ATTACK

var defense = BASE\_DEFENSE

var target = null

var is\_aggro = false

var min\_distance = 20.0

var knockback\_strength = 5.0

var knockback\_duration = 0.2

var knockback\_timer = 0.0

var attack\_cooldown = 1.0

var current\_cooldown = 0.0

var is\_dodging = false

var dodge\_cooldown = 2.0

var current\_dodge\_cooldown = 0.0

var dodge\_duration = 0.5

var current\_dodge\_duration = 0.0

var dodge\_direction = Vector2.ZERO

var dodge\_damage\_reduction = 0.25

# Характеристики врага

var enemy\_max\_health = 100

var enemy\_attack\_damage = 10

var enemy\_defense = 5

var enemy\_speed = 100

# Базовые значения

var base\_max\_health = BASE\_HEALTH

var base\_attack\_damage = BASE\_ATTACK

var base\_defense = BASE\_DEFENSE

var base\_speed = SPEED

@onready var anim = $AnimatedSprite2D

@onready var aggro\_area = $AggroArea

func \_ready():

add\_to\_group("enemies")

target = get\_tree().get\_nodes\_in\_group("player")[0]

aggro\_area.connect("body\_entered", \_on\_aggro\_area\_body\_entered)

aggro\_area.connect("body\_exited", \_on\_aggro\_area\_body\_exited)

# Масштабируем характеристики в зависимости от уровня игрока

scale\_stats\_to\_player\_level()

base\_max\_health = enemy\_max\_health

base\_attack\_damage = enemy\_attack\_damage

base\_defense = enemy\_defense

base\_speed = enemy\_speed

health = enemy\_max\_health

func scale\_stats\_to\_player\_level():

if target:

var player\_level = target.level

var scaling\_factor = 1.0 + (player\_level - 1) \* 0.1 # Увеличение на 10% за уровень

# Масштабируем характеристики

max\_health = int(BASE\_HEALTH \* scaling\_factor)

health = max\_health

attack\_power = int(BASE\_ATTACK \* scaling\_factor)

defense = int(BASE\_DEFENSE \* scaling\_factor)

func adapt\_to\_player\_level(player\_level):

# Значительно усиленные множители

var health\_multiplier = 2.0 \* player\_level # Увеличен множитель здоровья

var damage\_multiplier = 1.8 \* player\_level # Увеличен множитель урона

var defense\_multiplier = 1.5 \* player\_level # Увеличен множитель защиты

var speed\_multiplier = 1.3 \* player\_level # Увеличен множитель скорости

# Применяем усиление

max\_health = base\_max\_health \* health\_multiplier

health = max\_health # Обновляем текущее здоровье

attack\_power = base\_attack\_damage \* damage\_multiplier

defense = base\_defense \* defense\_multiplier

SPEED = base\_speed \* speed\_multiplier

# Увеличен лимит максимальной скорости

SPEED = min(SPEED, base\_speed \* 5) # Теперь может быть в 5 раз быстрее базовой

print("Враг адаптирован под уровень игрока ", player\_level)

print("Здоровье: ", max\_health)

print("Урон: ", attack\_power)

print("Защита: ", defense)

print("Скорость: ", SPEED)

func \_physics\_process(delta):

if knockback\_timer > 0:

knockback\_timer -= delta

anim.play("hurt")

move\_and\_slide()

elif is\_aggro and target:

# Проверяем, не мертв ли игрок

if target.is\_dead:

is\_aggro = false

velocity = Vector2.ZERO

return

# Обработка движения

var direction = (target.global\_position - global\_position).normalized()

var distance = global\_position.distance\_to(target.global\_position)

if distance > min\_distance:

velocity = direction \* SPEED

anim.play("run")

# Поворачиваем спрайт в зависимости от направления движения

anim.flip\_h = direction.x < 0

else:

velocity = Vector2.ZERO

if current\_cooldown <= 0:

attack()

current\_cooldown = attack\_cooldown

current\_cooldown -= delta

move\_and\_slide()

else:

# Если не преследуем цель, останавливаемся

velocity = Vector2.ZERO

anim.stop() # Останавливаем анимацию

func should\_dodge() -> bool:

if not target:

return false

# Проверяем, атакует ли игрок и близко ли он

var distance = global\_position.distance\_to(target.global\_position)

var is\_player\_attacking = target.get("is\_attacking") if target.has\_method("get") else false

return distance < 40.0 and is\_player\_attacking

func start\_dodge():

is\_dodging = true

current\_dodge\_duration = dodge\_duration

current\_dodge\_cooldown = dodge\_cooldown

# Выбираем случайное направление для уворота

var to\_player = (target.global\_position - global\_position).normalized()

var perpendicular = Vector2(-to\_player.y, to\_player.x)

dodge\_direction = (perpendicular if randf() > 0.5 else -perpendicular) + to\_player \* 0.5

dodge\_direction = dodge\_direction.normalized()

func \_on\_aggro\_area\_body\_entered(body):

if body == target:

is\_aggro = true

print("Враг заметил игрока!")

func \_on\_aggro\_area\_body\_exited(body):

if body == target:

is\_aggro = false

print("Враг потерял игрока из виду.")

func attack():

print("Враг атакует! Сила атаки:", attack\_power)

# Ускоряем анимацию атаки

anim.speed\_scale = 2.0

anim.play("attack")

# Ждем середины анимации для нанесения урона

# При speed = 10 fps и speed\_scale = 2.0, один кадр = 0.05 секунды

# У нас 10 кадров, значит до 5-го кадра нужно ждать 0.1 секунды

await get\_tree().create\_timer(0.1).timeout

# Наносим урон

if target and target.has\_method("take\_damage"):

target.take\_damage(attack\_power)

# Ждем окончания анимации и возвращаем нормальную скорость

await anim.animation\_finished

anim.speed\_scale = 1.0

func take\_damage(amount: int):

# Если враг уже мертв, не обрабатываем урон

if health <= 0:

return

var damage\_multiplier = dodge\_damage\_reduction if is\_dodging else 1.0

var actual\_damage = max(int((amount - defense) \* damage\_multiplier), 0)

health -= actual\_damage

# Прерываем текущую анимацию

anim.stop()

# Проигрываем анимацию получения урона с увеличенной скоростью

anim.speed\_scale = 2.0

anim.play("hurt")

print("Враг получил", actual\_damage, "урона. Осталось здоровья:", health)

if target:

var knockback\_direction = (position - target.position).normalized()

velocity = knockback\_direction \* knockback\_strength \* 1

knockback\_timer = knockback\_duration \* 1.5

if health <= 0:

die()

else:

# Ждем окончания анимации получения урона

await anim.animation\_finished

# Возвращаем нормальную скорость и сбрасываем анимацию

anim.speed\_scale = 1.0

# Если все еще есть цель, возвращаемся к анимации движения

if target and is\_aggro:

var direction = (target.global\_position - global\_position).normalized()

anim.play("run")

anim.flip\_h = direction.x < 0

func die():

print("Враг умер")

# Отключаем коллизии сразу

self.collision\_layer = 0

self.collision\_mask = 0

# Останавливаем все текущие анимации и проигрываем смерть

anim.stop()

anim.speed\_scale = 1.0

anim.play("death")

# Отключаем физику и движение

velocity = Vector2.ZERO

set\_physics\_process(false)

# Обновляем квесты и выдаем награду

QuestManager.update\_quest\_progress("kill")

QuestManager.update\_quest\_progress("kill\_weak", 1, "kill\_weak")

QuestManager.update\_quest\_progress("clear\_first\_hall", 1, "clear\_first\_hall")

var player = get\_tree().get\_nodes\_in\_group("player")[0]

if player and player.has\_method("gain\_experience"):

player.gain\_experience(10)

print("Награда получена: +10 опыта")

# Добавляем выпадение предметов

drop\_loot()

anim.sprite\_frames.set\_animation\_loop("death", false)

# Ждем завершения анимации смерти перед удалением

await anim.animation\_finished

queue\_free()

func drop\_loot():

var dropped\_item\_scene = preload("res://scenes/dropped\_item.tscn")

var item\_database = get\_node("/root/ItemDatabase")

# Шанс выпадения предметов

if randf() < 0.3: # 30% шанс

var possible\_items = ["Зелье здоровья"]

var item\_name = possible\_items[randi() % possible\_items.size()]

var item\_resource = item\_database.get\_item(item\_name)

if item\_resource:

var dropped\_item = dropped\_item\_scene.instantiate()

dropped\_item.item\_name = item\_name

dropped\_item.item\_texture = item\_resource.icon

# Добавляем случайное смещение при выпадении

var offset = Vector2(randf\_range(-10, 10), randf\_range(-10, 10))

dropped\_item.position = position + offset

get\_parent().add\_child(dropped\_item)

enemy2.gd:

extends CharacterBody2D

var SPEED = 100.0

var DODGE\_SPEED = 300.0

var BASE\_HEALTH = 200

var BASE\_ATTACK = 15

var BASE\_DEFENSE = 10

var health = BASE\_HEALTH

var max\_health = BASE\_HEALTH

var attack\_power = BASE\_ATTACK

var defense = BASE\_DEFENSE

var target = null

var is\_aggro = false

var min\_distance = 20.0

# Базовые значения

var base\_max\_health = BASE\_HEALTH

var base\_attack\_damage = BASE\_ATTACK

var base\_defense = BASE\_DEFENSE

var base\_speed = SPEED

# Добавляем недостающие переменные

@onready var anim = $AnimatedSprite2D

var knockback\_timer = 0.0

var knockback\_duration = 0.2

var knockback\_strength = 200.0

var current\_cooldown = 0.0

var attack\_cooldown = 0.8

var is\_dodging = false

var dodge\_duration = 0.3

var current\_dodge\_duration = 0.0

var dodge\_cooldown = 1.5

var current\_dodge\_cooldown = 0.0

var dodge\_direction = Vector2.ZERO

var dodge\_damage\_reduction = 0.7

func adapt\_to\_player\_level(player\_level):

# Значительно усиленные множители

var health\_multiplier = 2.0 \* player\_level

var damage\_multiplier = 1.8 \* player\_level

var defense\_multiplier = 1.5 \* player\_level

var speed\_multiplier = 1.3 \* player\_level

# Применяем усиление

max\_health = base\_max\_health \* health\_multiplier

health = max\_health # Обновляем текущее здоровье

attack\_power = base\_attack\_damage \* damage\_multiplier

defense = base\_defense \* defense\_multiplier

SPEED = base\_speed \* speed\_multiplier

# Увеличен лимит максимальной скорости

SPEED = min(SPEED, base\_speed \* 5)

print("Враг2 адаптирован под уровень игрока ", player\_level)

print("Здоровье: ", max\_health)

print("Урон: ", attack\_power)

print("Защита: ", defense)

print("Скорость: ", SPEED)

func \_ready():

add\_to\_group("enemies")

base\_max\_health = max\_health

base\_attack\_damage = attack\_power

base\_defense = defense

base\_speed = SPEED

# Настраиваем область агро как у обычного enemy

var aggro\_area = $AggroArea

var aggro\_shape = aggro\_area.get\_node("CollisionShape2D")

if aggro\_shape and aggro\_shape.shape is CircleShape2D:

aggro\_shape.shape.radius = 200.0 # Устанавливаем такой же радиус как у обычного enemy

$AggroArea.connect("body\_entered", \_on\_aggro\_area\_body\_entered)

$AggroArea.connect("body\_exited", \_on\_aggro\_area\_body\_exited)

func \_physics\_process(delta):

if knockback\_timer > 0:

knockback\_timer -= delta

anim.play("hurt")

move\_and\_slide()

elif is\_aggro and target: # Возвращаем проверку is\_aggro

# Проверяем, не мертв ли игрок

if target.is\_dead:

is\_aggro = false

velocity = Vector2.ZERO

return

# Обработка движения

var direction = (target.global\_position - global\_position).normalized()

var distance = global\_position.distance\_to(target.global\_position)

if distance > min\_distance:

velocity = direction \* SPEED

if direction.x < 0:

anim.play("run\_left")

else:

anim.play("run\_right")

else:

velocity = Vector2.ZERO

if current\_cooldown <= 0:

attack(direction.x < 0)

current\_cooldown = attack\_cooldown

current\_cooldown -= delta

move\_and\_slide()

else:

# Если не в агро, просто стоим

velocity = Vector2.ZERO

anim.play("run\_right")

func should\_dodge() -> bool:

if not target:

return false

# Проверяем, атакует ли игрок и близко ли он

var distance = global\_position.distance\_to(target.global\_position)

var is\_player\_attacking = target.get("is\_attacking") if target.has\_method("get") else false

return distance < 40.0 and is\_player\_attacking

func start\_dodge():

is\_dodging = true

current\_dodge\_duration = dodge\_duration

current\_dodge\_cooldown = dodge\_cooldown

# Выбираем случайное направление для уворота

var to\_player = (target.global\_position - global\_position).normalized()

var perpendicular = Vector2(-to\_player.y, to\_player.x)

dodge\_direction = (perpendicular if randf() > 0.5 else -perpendicular) + to\_player \* 0.5

dodge\_direction = dodge\_direction.normalized()

func \_on\_aggro\_area\_body\_entered(body):

# Проверяем, является ли body игроком

if body.is\_in\_group("player"):

target = body

is\_aggro = true

print("Враг заметил игрока!")

func \_on\_aggro\_area\_body\_exited(body):

# Проверяем, является ли body игроком

if body.is\_in\_group("player"):

is\_aggro = false

print("Враг потерял игрока из виду.")

func attack(attack\_left: bool = false):

print("Враг атакует! Сила атаки:", attack\_power)

# Увеличиваем скорость анимации атаки еще больше

anim.speed\_scale = 3.0 # Увеличиваем с 2.0 до 3.0

if attack\_left:

anim.play("attack\_left")

else:

anim.play("attack\_right")

# Корректируем время до нанесения урона с учетом новой скорости

await get\_tree().create\_timer(0.07).timeout

# Наносим урон

if target and target.has\_method("take\_damage"):

target.take\_damage(attack\_power)

# Ждем окончания анимации и возвращаем нормальную скорость

await anim.animation\_finished

anim.speed\_scale = 1.0

func take\_damage(amount: int):

print("Попытка нанести урон врагу: ", amount)

# Если враг уже мертв, не обрабатываем урон

if health <= 0:

return

var damage\_multiplier = dodge\_damage\_reduction if is\_dodging else 1.0

var actual\_damage = max(int((amount - defense) \* damage\_multiplier), 0)

health -= actual\_damage

print("Враг получил урон. Текущее здоровье: ", health)

# Прерываем текущую анимацию

anim.stop()

# Проигрываем анимацию получения урона с увеличенной скоростью

anim.speed\_scale = 2.0

anim.play("hurt")

if target:

var knockback\_direction = (position - target.position).normalized()

velocity = knockback\_direction \* knockback\_strength \* 1

knockback\_timer = knockback\_duration \* 1.5

if health <= 0:

die()

else:

# Ждем окончания анимации получения урона

await anim.animation\_finished

# Возвращаем нормальную скорость и сбрасываем анимацию

anim.speed\_scale = 1.0

# Если все еще есть цель, возвращаемся к анимации движения

if target:

var direction = (target.global\_position - global\_position).normalized()

if direction.x < 0:

anim.play("run\_left")

else:

anim.play("run\_right")

func die():

print("Враг умер")

# Отключаем коллизии сразу

self.collision\_layer = 0

self.collision\_mask = 0

# Останавливаем все текущие анимации и проигрываем смерть

anim.stop()

anim.speed\_scale = 1.0

anim.play("death")

# Отключаем физику и движение

velocity = Vector2.ZERO

set\_physics\_process(false)

# Обновляем квесты и выдаем награду

QuestManager.update\_quest\_progress("kill")

QuestManager.update\_quest\_progress("kill\_weak", 1, "kill\_weak")

QuestManager.update\_quest\_progress("clear\_first\_hall", 1, "clear\_first\_hall")

var player = get\_tree().get\_nodes\_in\_group("player")[0]

if player and player.has\_method("gain\_experience"):

player.gain\_experience(10)

print("Награда получена: +10 опыта")

# Добавляем выпадение предметов

drop\_loot()

anim.sprite\_frames.set\_animation\_loop("death", false)

# Ждем завершения анимации смерти перед удалением

await anim.animation\_finished

queue\_free()

func drop\_loot():

var dropped\_item\_scene = preload("res://scenes/dropped\_item.tscn")

var item\_database = get\_node("/root/ItemDatabase")

# Шанс ыпадения предметов

if randf() < 0.3: # 30% шанс

var possible\_items = ["Зелье здоровья"]

var item\_name = possible\_items[randi() % possible\_items.size()]

var item\_resource = item\_database.get\_item(item\_name)

if item\_resource:

var dropped\_item = dropped\_item\_scene.instantiate()

dropped\_item.item\_name = item\_name

dropped\_item.item\_texture = item\_resource.icon

# Добавляем случайное смещение при выпадении

var offset = Vector2(randf\_range(-10, 10), randf\_range(-10, 10))

dropped\_item.position = position + offset

get\_parent().add\_child(dropped\_item)

# boss.gd:

extends "res://scripts/enemy.gd"

# Константы для атаки

const ATTACK\_COOLDOWN = 2.0 # Увеличиваем время между атаками

const ATTACK\_DISTANCE = 100.0

const ATTACK\_DURATION = 1.2 # Увеличиваем длительность анимации атаки

# Фазы босса

enum BossPhase {PHASE\_1, PHASE\_2, PHASE\_3}

var current\_phase = BossPhase.PHASE\_1

# Характеристики босса

var boss\_speed = 60.0

const PHASE\_THRESHOLD = 0.66 # 66% и 33% здоровья для смены фаз

# Способности

var dash\_cooldown = 4.0

var current\_dash\_cooldown = 0.0

var is\_dashing = false

var dash\_speed = 300.0

var summon\_cooldown = 8.0

var current\_summon\_cooldown = 0.0

var minion\_scene = preload("res://scenes/enemy.tscn")

var aoe\_attack\_cooldown = 6.0

var current\_aoe\_cooldown = 0.0

var aoe\_radius = 100.0

var shield\_cooldown = 15.0

var current\_shield\_cooldown = 0.0

var shield\_duration = 3.0

var has\_shield = false

var knockback\_distance = 150.0 # Расстояние отталкивания от игрока

# Новые переменные для способностей

var teleport\_cooldown = 10.0

var current\_teleport\_cooldown = 0.0

var can\_teleport = true

var flame\_trail\_cooldown = 7.0

var current\_flame\_trail\_cooldown = 0.0

var flame\_damage = 5

var rage\_mode = false

var rage\_duration = 8.0

var rage\_cooldown = 20.0

var current\_rage\_cooldown = 0.0

# Добавляем новые переменные для эффектов

var effect\_particles: CPUParticles2D

var aoe\_circle: Node2D

# В начале файла добавим новые переменные

var minion\_spawn\_particles: CPUParticles2D

var aoe\_particles: CPUParticles2D

var flame\_particles: CPUParticles2D

var is\_attacking = false

var current\_attack\_cooldown = 0.0

var facing\_direction = Vector2.RIGHT

func \_ready():

# Вызываем родительский \_ready()

super.\_ready()

# Устанавливаем новые значения для босса

health = 2000

max\_health = 2000

attack\_power = 100

defense = 15

# Обновляем базовые значения для масштабирования

base\_max\_health = 2000

base\_attack\_damage = 100

base\_defense = 15

base\_speed = 200

# Устанавливаем скорости

velocity = Vector2.ZERO

boss\_speed = 200.0

# Инициализация базовых характеристик

min\_distance = ATTACK\_DISTANCE # Обновляем дистанцию атаки

# Увеличиваем размер босса и его области агро

scale = Vector2(2, 2)

$AggroArea/CollisionShape2D.scale = Vector2(3, 3) # Увеличиваем радиус агро

# Сразу находим и устанавливаем цель (игрока)

target = get\_tree().get\_nodes\_in\_group("player")[0]

is\_aggro = true # Сразу активируем агро

# Добавляем босса в группу

add\_to\_group("boss")

add\_to\_group("enemies")

# Создаём частицы для эффектов

setup\_particles()

setup\_aoe\_circle()

setup\_rage\_particles()

setup\_all\_particles()

# Сразу проверяем призыв миньонов

check\_minion\_spawn()

# Начинаем с анимации простоя

$AnimatedSprite2D2.animation = "idle"

$AnimatedSprite2D2.play()

# Теперь спрайт изначально смотрит влево

$AnimatedSprite2D2.flip\_h = true

# Проверяем доступные анимации

print("Available animations: ", $AnimatedSprite2D2.sprite\_frames.get\_animation\_names())

# Проверяем настройки анимации атаки

print("Attack animation frames: ", $AnimatedSprite2D2.sprite\_frames.get\_frame\_count("attack"))

# Отключаем зацикливание анимации смерти

$AnimatedSprite2D2.sprite\_frames.set\_animation\_loop("death", false)

func setup\_particles():

effect\_particles = CPUParticles2D.new()

effect\_particles.emitting = false

effect\_particles.amount = 20

effect\_particles.lifetime = 0.5

effect\_particles.explosiveness = 0.5

effect\_particles.direction = Vector2(0, -1)

effect\_particles.spread = 180

effect\_particles.gravity = Vector2(0, 0)

effect\_particles.initial\_velocity\_min = 50

effect\_particles.initial\_velocity\_max = 100

add\_child(effect\_particles)

func setup\_aoe\_circle():

aoe\_circle = Node2D.new()

add\_child(aoe\_circle)

aoe\_circle.visible = false

func setup\_rage\_particles():

var rage\_particles = CPUParticles2D.new()

rage\_particles.name = "RageParticles"

rage\_particles.emitting = false

rage\_particles.amount = 30

rage\_particles.lifetime = 0.5

rage\_particles.explosiveness = 0.2

rage\_particles.emission\_shape = CPUParticles2D.EMISSION\_SHAPE\_SPHERE

rage\_particles.emission\_sphere\_radius = 30.0

rage\_particles.direction = Vector2(0, -1)

rage\_particles.spread = 180

rage\_particles.gravity = Vector2(0, 0)

rage\_particles.initial\_velocity\_min = 50

rage\_particles.initial\_velocity\_max = 100

rage\_particles.color = Color(1, 0, 0, 0.5) # Красные частицы для ярости

add\_child(rage\_particles)

func setup\_all\_particles():

# Частицы для призыва миньонов (фиолетовые)

minion\_spawn\_particles = CPUParticles2D.new()

minion\_spawn\_particles.amount = 30

minion\_spawn\_particles.lifetime = 0.5

minion\_spawn\_particles.explosiveness = 0.8

minion\_spawn\_particles.emission\_shape = CPUParticles2D.EMISSION\_SHAPE\_SPHERE

minion\_spawn\_particles.emission\_sphere\_radius = 50.0

minion\_spawn\_particles.color = Color(0.7, 0, 1, 1)

add\_child(minion\_spawn\_particles)

# Частицы для АОЕ атаки (красные)

aoe\_particles = CPUParticles2D.new()

aoe\_particles.amount = 40

aoe\_particles.lifetime = 0.3

aoe\_particles.explosiveness = 1.0

aoe\_particles.spread = 180

aoe\_particles.color = Color(1, 0, 0, 1)

add\_child(aoe\_particles)

# Частицы для огненного следа (оранжевые)

flame\_particles = CPUParticles2D.new()

flame\_particles.amount = 20

flame\_particles.lifetime = 0.8

flame\_particles.color = Color(1, 0.5, 0, 1)

add\_child(flame\_particles)

func \_physics\_process(delta):

if knockback\_timer > 0:

knockback\_timer -= delta

move\_and\_slide()

return

if not is\_aggro or not target or target.is\_dead:

velocity = Vector2.ZERO

play\_animation("idle")

return

if current\_attack\_cooldown > 0:

current\_attack\_cooldown -= delta

var direction = (target.global\_position - global\_position).normalized()

var distance = global\_position.distance\_to(target.global\_position)

if distance > min\_distance and !is\_attacking:

velocity = direction \* boss\_speed

# Инвертируем логику направления

if direction.x > 0: # Движение вправо

$AnimatedSprite2D2.flip\_h = true # Было false

play\_animation("run")

else: # Движение влево

$AnimatedSprite2D2.flip\_h = false # Было true

play\_animation("run")

facing\_direction = direction

else:

velocity = Vector2.ZERO

if current\_attack\_cooldown <= 0 and !is\_attacking:

await perform\_attack()

elif !is\_attacking:

play\_animation("idle")

move\_and\_slide()

func perform\_attack():

if not target or is\_attacking:

return

is\_attacking = true

current\_attack\_cooldown = ATTACK\_COOLDOWN

# Инвертируем направлене атаки

var direction = (target.global\_position - global\_position).normalized()

$AnimatedSprite2D2.flip\_h = direction.x > 0 # Инвертировали условие

play\_animation("attack")

await get\_tree().create\_timer(0.3).timeout

if target and global\_position.distance\_to(target.global\_position) <= ATTACK\_DISTANCE + 50:

target.take\_damage(attack\_power)

await get\_tree().create\_timer(ATTACK\_DURATION).timeout

is\_attacking = false

play\_animation("idle")

func play\_animation(anim\_name: String):

if $AnimatedSprite2D2.animation != anim\_name:

$AnimatedSprite2D2.play(anim\_name)

if anim\_name == "attack":

$AnimatedSprite2D2.speed\_scale = 0.8

elif anim\_name == "death":

$AnimatedSprite2D2.speed\_scale = 1.0

elif anim\_name == "run":

$AnimatedSprite2D2.speed\_scale = 1.0

else:

$AnimatedSprite2D2.speed\_scale = 1.0

func \_on\_animated\_sprite\_2d\_2\_animation\_finished():

if $AnimatedSprite2D2.animation == "attack":

is\_attacking = false

play\_animation("idle")

func take\_damage(amount: int):

if health <= 0:

return

var actual\_damage = max(amount - defense, 0)

if has\_shield:

actual\_damage = int(actual\_damage \* 0.5)

health -= actual\_damage

# Прерываем текущую анимацию и проигрываем получение урона

play\_animation("hurt\_left")

$AnimatedSprite2D2.flip\_h = target and target.global\_position.x < global\_position.x

if target:

var knockback\_direction = (global\_position - target.global\_position).normalized()

velocity = knockback\_direction \* knockback\_strength

knockback\_timer = knockback\_duration

if health <= 0:

die()

else:

await get\_tree().create\_timer(0.3).timeout # Длительность анимции получения урона

play\_animation("idle")

func die():

print("Босс побежден!")

# Отключаем физику и коллизии

collision\_layer = 0

collision\_mask = 0

set\_physics\_process(false)

is\_attacking = false

# Удаляем миньонов

for minion in get\_tree().get\_nodes\_in\_group("minions"):

minion.queue\_free()

# Выдаем награду

var player = get\_tree().get\_nodes\_in\_group("player")[0]

if player and player.has\_method("gain\_experience"):

player.gain\_experience(100)

# Проигрываем анимацию смерти

$AnimatedSprite2D2.sprite\_frames.set\_animation\_loop("death", false)

$AnimatedSprite2D2.speed\_scale = 1.0

$AnimatedSprite2D2.play("death")

# Ждем некоторое время и удаляем босса

await get\_tree().create\_timer(1.5).timeout

queue\_free()

func update\_cooldowns(delta):

if current\_cooldown > 0:

current\_cooldown -= delta

if current\_dash\_cooldown > 0:

current\_dash\_cooldown -= delta

if current\_summon\_cooldown > 0:

current\_summon\_cooldown -= delta

if current\_aoe\_cooldown > 0:

current\_aoe\_cooldown -= delta

if current\_shield\_cooldown > 0:

current\_shield\_cooldown -= delta

if current\_teleport\_cooldown > 0:

current\_teleport\_cooldown -= delta

if current\_flame\_trail\_cooldown > 0:

current\_flame\_trail\_cooldown -= delta

if current\_rage\_cooldown > 0:

current\_rage\_cooldown -= delta

func check\_phase():

var health\_percent = float(health) / max\_health

if health\_percent <= 0.33 and current\_phase != BossPhase.PHASE\_3:

enter\_phase\_3()

elif health\_percent <= 0.66 and current\_phase == BossPhase.PHASE\_1:

enter\_phase\_2()

func perform\_dash():

if not target:

return

is\_dashing = true

current\_dash\_cooldown = dash\_cooldown

# Направление рывка к игроку

var dash\_direction = (target.global\_position - global\_position).normalized()

velocity = dash\_direction \* dash\_speed

# Создаём урон во время рывка

var dash\_damage = attack\_power \* 1.5 # Увеличиваем урон от рывка

# Эффект рывка

effect\_particles.emitting = true

# Таймер для окончания рывка

await get\_tree().create\_timer(0.3).timeout

is\_dashing = false

effect\_particles.emitting = false

# Наносим урон, если попали по игроку

if target and global\_position.distance\_to(target.global\_position) < 50:

target.take\_damage(dash\_damage)

func summon\_minions():

current\_summon\_cooldown = summon\_cooldown

# Эффект призыва

minion\_spawn\_particles.emitting = true

# Создаём 4 миньона вокруг босса

for i in range(4):

var minion = minion\_scene.instantiate()

var angle = TAU \* i / 4 # Равномерно распределяем по кругу

var spawn\_pos = global\_position + Vector2(cos(angle), sin(angle)) \* 100

# Настройка миньона

minion.position = spawn\_pos

minion.health = 50

minion.max\_health = 50

minion.attack\_power = 8

minion.defense = 2

minion.min\_distance = 30.0

minion.knockback\_strength = 8.0

minion.attack\_cooldown = 0.8

minion.is\_aggro = true

minion.target = target

minion.add\_to\_group("minions")

# Добавляем миньона на сцену

get\_parent().call\_deferred("add\_child", minion)

# Эффект появления

var spawn\_effect = CPUParticles2D.new()

spawn\_effect.position = spawn\_pos

spawn\_effect.emitting = true

spawn\_effect.one\_shot = true

spawn\_effect.amount = 15

spawn\_effect.lifetime = 0.5

spawn\_effect.explosiveness = 1.0

spawn\_effect.color = Color(0.5, 0, 1)

get\_parent().call\_deferred("add\_child", spawn\_effect)

await get\_tree().create\_timer(0.2).timeout

await get\_tree().create\_timer(0.5).timeout

minion\_spawn\_particles.emitting = false

func perform\_aoe\_attack():

current\_aoe\_cooldown = aoe\_attack\_cooldown

# Создаём предупреждающий круг

var warning\_circle = Node2D.new()

add\_child(warning\_circle)

# Рисуем предупреждающий круг

warning\_circle.draw.connect(func():

warning\_circle.draw\_circle(Vector2.ZERO, aoe\_radius, Color(1, 0, 0, 0.3))

warning\_circle.draw\_arc(Vector2.ZERO, aoe\_radius, 0, TAU, 32, Color(1, 0, 0, 0.7), 2.0)

)

# Ждём перед атакой

await get\_tree().create\_timer(0.5).timeout

# Наносим урон

if target and target.global\_position.distance\_to(global\_position) < aoe\_radius:

target.take\_damage(attack\_power \* 2) # Увеличенный урон

# Создаём эффект взрыва

var explosion = CPUParticles2D.new()

explosion.emitting = true

explosion.one\_shot = true

explosion.explosiveness = 1.0

explosion.amount = 30

explosion.lifetime = 0.5

explosion.spread = 180

explosion.initial\_velocity\_min = 100

explosion.initial\_velocity\_max = 200

explosion.color = Color(1, 0, 0)

add\_child(explosion)

await get\_tree().create\_timer(0.5).timeout

warning\_circle.queue\_free()

explosion.queue\_free()

func create\_flame\_trail():

current\_flame\_trail\_cooldown = flame\_trail\_cooldown

for i in range(5): # Создаём 5 точек огня

var flame = Node2D.new()

var pos = global\_position - velocity.normalized() \* (i \* 30) # Расстояние между тчками

flame.position = pos

get\_parent().add\_child(flame)

# Добавляем частицы огня

var particles = CPUParticles2D.new()

particles.amount = 20

particles.lifetime = 0.8

particles.explosiveness = 0.2

particles.direction = Vector2.UP

particles.spread = 180

particles.initial\_velocity\_min = 50

particles.initial\_velocity\_max = 80

particles.color = Color(1, 0.5, 0)

particles.emitting = true

flame.add\_child(particles)

# Область повреждения

var area = Area2D.new()

var collision = CollisionShape2D.new()

var shape = CircleShape2D.new()

shape.radius = 20

collision.shape = shape

area.add\_child(collision)

flame.add\_child(area)

# Таймер для удаления

await get\_tree().create\_timer(3.0).timeout

flame.queue\_free()

func activate\_shield():

current\_shield\_cooldown = shield\_cooldown

has\_shield = true

defense \*= 3

await get\_tree().create\_timer(shield\_duration).timeout

has\_shield = false

defense /= 3

func enter\_phase\_2():

current\_phase = BossPhase.PHASE\_2

attack\_power \*= 1.1

boss\_speed \*= 1.2

summon\_minions()

perform\_aoe\_attack()

func enter\_phase\_3():

current\_phase = BossPhase.PHASE\_3

attack\_power \*= 1.2

boss\_speed \*= 1.3

activate\_shield()

activate\_rage()

func teleport\_behind\_player():

if not target:

return

current\_teleport\_cooldown = teleport\_cooldown

var behind\_position = target.global\_position + (target.global\_position - global\_position).normalized() \* 50

global\_position = behind\_position

perform\_aoe\_attack() # Сразу делаем АОЕ атаку после телепорта

func activate\_rage():

current\_rage\_cooldown = rage\_cooldown

rage\_mode = true

attack\_power \*= 1.5

boss\_speed \*= 1.3

# Влючаем частицы ярости

$RageParticles.emitting = true

# Призываем миньонов при входе в ярость

if health < max\_health \* 0.5:

summon\_minions()

await get\_tree().create\_timer(rage\_duration).timeout

rage\_mode = false

attack\_power /= 1.5

boss\_speed /= 1.3

$RageParticles.emitting = false

func check\_minion\_spawn():

# Проверяем количество существующих миньонов

var existing\_minions = get\_tree().get\_nodes\_in\_group("minions").size()

if existing\_minions < 3 and current\_summon\_cooldown <= 0:

call\_deferred("summon\_minions") # Используем call\_deferred

func adapt\_to\_player\_level(player\_level):

# Уменьшенные множители роста от уровня

var health\_multiplier = 0.5 \* player\_level # Уменьшен множитель здоровья

var damage\_multiplier = 0.4 \* player\_level # Уменьшен множитель урона

var defense\_multiplier = 0.3 \* player\_level # Уменьшен множитель защиты

var speed\_multiplier = 0.2 \* player\_level # Уменьшен множитель скорости

# Применяем усиление

max\_health = base\_max\_health \* health\_multiplier

health = max\_health # Обновляем текущее здоровье

attack\_power = base\_attack\_damage \* damage\_multiplier

defense = base\_defense \* defense\_multiplier

boss\_speed = base\_speed \* speed\_multiplier

# Увеличен лимит максимальной скорости

boss\_speed = min(boss\_speed, base\_speed \* 6) # Может быть в 6 раз быстрее базовой

**Заключение**

В результате выполнения курсовой работы была разработана игра "World without a king" на движке Godot. В ходе разработки были успешно решены следующие задачи:

1. Создана система процедурной генерации мира:

• Реализация генерации ландшафта с использованием шума Перлина

• Система размещения поселений с учетом безопасных зон

• Динамическое создание игровых объектов

• Оптимизация генерации больших территорий

2. Разработана система противников:

• Базовые противники с различными типами поведения

• Улучшенные противники с направленными атаками

• Система получения урона и анимаций

• Искусственный интеллект для навигации и атак

3. Реализована система босса:

• Многофазовая боевая система

• Уникальные паттерны атак для каждой фазы

• Система призыва и управления миньонами

• Визуальные эффекты и специальные способности

4. Внедрена система тестирования:

• Модульные тесты для основных компонентов

• Тестирование генерации мира

• Проверка боевой системы

• Валидация поведения противников и босса

5. Разработаны вспомогательные системы:

• Оптимизация производительности

• Система сохранения состояния

• Обработка событий

• Визуальные эффекты

Программа демонстрирует практическое применение:

- Процедурной генерации контента

- Искусственного интеллекта

- Объектно-ориентированного программирования

- Работы с игровым движком Godot

- Модульного тестирования

- Оптимизации производительности

Созданная игра представляет собой полноценный проект с процедурной генерацией мира и развитой боевой системой, имеющий потенциал для дальнейшего расширения функционала через добавление новых типов биомов, противников и боссов.

# **Источники, использованные при разработке**

1. Официальная документация Godot Engine на русском языке [Электронный ресурс] // Godot Docs: https://docs.godotengine.org/ru/stable/ – Режим доступа: свободный.
2. Волосатов, В.А. Разработка игр на Godot Engine: от идеи до реализации. – М.: ДМК Пресс, 2023. – 328 с.
3. Хабр, "Godot Engine - Руководство по разработке 2D игр" [Электронный ресурс] // https://habr.com/ru/articles/438868/ – Режим доступа: свободный.
4. В.В. Шишкин, Д.С. Афонин РАЗРАБОТКА ЛОГИЧЕСКИХ КОМПЬЮТЕРНЫХ ИГР С ГРАФИЧЕСКИМ ИНТЕРФЕЙСОМ В СРЕДЕ ПИТОН
5. Петров, И.С. Основы разработки компьютерных игр в среде Godot. – СПб.: БХВ-Петербург, 2022. – 256 с.
6. Комлев, Н.Ю. Разработка пользовательских интерфейсов в Godot Engine. – М.: Солон-Пресс, 2023. – 184 с.
7. Godot Russia Community [Электронный ресурс] // https://godot-engine.ru/ – Режим доступа: свободный.
8. Сидоров, А.В. Программирование игровых механик на GDScript. – М.: Прогресс книга, 2023. – 292 с
9. Годот Форум на русском [Электронный ресурс] // https://godot-ru.org/forum/ – Режим доступа: свободный.

# 

# **ПРИЛОЖЕНИЕ 1**

***MINDMAP***