Министерство науки и высшего образования Российской Федерации

ФГБОУ ВО «Алтайский государственный университет»

Школа развития цифровых компетенций «Digital Up» (цифровая кафедра)

**ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ**

**на итоговый проект «****Разработка платформера Sylphi»**

**по ДПП ПП «Основы Gamedev и VR-разработки»**

|  |  |  |  |
| --- | --- | --- | --- |
| п/п  № | Задание | Исполнитель | Рабочий график (план) выполнения |
| 1 | Изучить принципы разработки 2D-платформеров, включая: механику движения персонажа, систему взаимодействия с окружением | Бортникова Е.В. | 02.05.2024 – 13.05.2024 |
| 2 | Реализовать игровой прототип с базовыми элементами геймплея | 14.05.2024 – 18.05.2024 |
| 3 | Сделать выводы исходя из результатов работы | 19.05.2024 – 26.05.2024 |
| 4 | Подготовка отчета и видео-презентации | 26.05.2024 – 30.05.2024 |

Руководитель проекта   
ст. пр. Каратаев А.А.

\_\_\_\_\_\_\_\_\_\_\_\_\_ /\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / «\_\_\_ » \_\_\_\_\_\_\_\_\_ 2025 г.

(подпись)

СОГЛАСОВАНО:

Руководитель ДПП ПП

канд. физ.-мат. наук, доцент Козлов Д.Ю.

\_\_\_\_\_\_\_\_\_\_\_\_\_ /\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_2025 г.

(подпись)

Министерство науки и высшего образования Российской Федерации

ФГБОУ ВО «Алтайский государственный университет»

Школа развития цифровых компетенций «Digital Up» (цифровая кафедра)

Отчет о выполнении группового итогового проекта по ДПП ПП

«Основы Gamedev и VR-разработки»

**«Разработка платформера Sylphi»**

Исполнители:

Бортникова Е.В.

«\_\_\_\_» \_\_\_\_\_\_\_\_ 2024 г.

Руководитель проекта

ст. пр. Каратаев А.А.

«\_\_\_\_» \_\_\_\_\_\_\_\_ 2024 г.

г. Барнаул, 2025

# Цель проекта

Цель данного проекта — создание 2D-платформера с элементами экшена и исследовательского геймплея, где игрок управляет персонажем и сражается с врагами. Проект направлен на демонстрацию возможностей игрового движка Godot и разработки игрового процесса.

# Задачи проекта и исполнители

Для достижения поставленной цели требовалось решить следующие задачи:

1. Изучение механик платформеров
   * Анализ успешных игр жанра (*Celeste, Hollow Knight, Ori and the Blind Forest*)
   * Определение ключевых особенностей геймплея (движение, боевая система, дизайн уровней)
2. Разработка игрового прототипа
   * Создание базового движения персонажа (прыжки, бег, атака)
   * Реализация системы столкновений и физики
3. Создание визуального стиля
   * Разработка пиксель-арт графики (персонаж, враги, окружение)
   * Анимация движений и эффектов
4. Программирование искусственного интеллекта врагов
   * Реализация преследования игрока, атакующих паттернов
5. Тестирование и балансировка
   * Проверка игрового баланса (сложность, управление)
   * Исправление багов и доработка механик

# Актуальность и востребованность проекта

Актуальность проекта обусловлена несколькими ключевыми факторами. Прежде всего, платформеры продолжают удерживать лидирующие позиции среди самых популярных жанров в сфере инди-игр. Такие культовые проекты, как Hollow Knight, Celeste и Ori and the Blind Forest, доказали, что даже в эпоху технологически сложных AAA-игр качественно сделанный платформер с продуманным геймплеем и стильным визуалом способен найти своего зрителя и добиться коммерческого успеха.

Данный проект представляет значительную образовательную ценность. Работа над игрой позволяет на практике освоить ключевые принципы геймдизайна, включая проектирование уровней, балансировку сложности и создание игровых механик. Для программистов это отличная возможность углубить знания движка Godot, а для художников - развить навыки в создании пиксель-арт графики и анимации.

Востребованность проекта на современном игровом рынке не вызывает сомнений. Основная целевая аудитория Sylphi - это игроки, которые активно интересуются инди-играми и ценят в них оригинальность, стиль и увлекательный геймплей. Именно эта категория пользователей формирует устойчивый спрос на качественные платформеры в цифровых магазинах.

Особого внимания заслуживает потенциал дальнейшего развития проекта. В перспективе игра может быть дополнена новыми локациями, врагами, а также добавлена сюжетная составляющая. Создание встроенного редактора уровней позволит сообществу игроков участвовать в развитии контента, формируя вокруг игры активное комьюнити. А портирование на мобильные платформы сделает игру доступным для ещё более широкой аудитории, учитывая растущую популярность мобильного гейминга. Все эти направления развития не только повышают коммерческие перспективы проекта, но и значительно увеличивают его жизненный цикл.

# Общие сведения о проделанной работе

Разработка платформера Sylphi велась поэтапно с использованием современных инструментов и технологий игровой индустрии. На первом этапе была проведена аналитическая работа по изучению современных тенденций в жанре платформеров, включая анализ успешных проектов.. Это позволило выделить ключевые элементы успешного геймдизайна - баланс между сложностью и доступностью, продуманную систему прогрессии и боевую механику.

Для реализации проекта был выбран игровой движок Godot, который идеально подходит для создания 2D-платформеров благодаря своей гибкости, открытому исходному коду и низкому порогу вхождения. Основной код игры написан на языке GDScript, специально разработанном для Godot и сочетающем простоту Python с производительностью, достаточной для игровых проектов. Визуальная часть создавалась с использованием пиксель-арт графики, для работы с которой применялся специализированный редактор Aseprite, позволяющий создавать качественные спрайты и анимации.

Разработка началась с создания базового прототипа, включающего основные механики:

* Систему передвижения персонажа (прыжки, дэши)
* Боевую систему
* Основные элементы окружения (платформы, препятствия)

В процессе разработки команда столкнулась с рядом вызовов, наиболее значимым из которых стала реализация плавного и отзывчивого управления персонажем. Решение было найдено через комбинацию методов:

* Интерполяции движений
* Кадровой независимости физики
* Тонкой настройки параметров инерции

Полученный опыт демонстрирует, что даже в рамках относительно небольшого проекта требуется тщательный баланс между креативным замыслом и техническими ограничениями. Дальнейшее развитие Sylphi предполагает:

* Расширение системы способностей персонажа
* Добавление новых врагов и боссов, а также мирных мобов
* Реализацию систему дропа предметов, улучшающих характеристики персонажа
* Реализация квестовой составляющей, с сюжетом и заданиями раскрывающими лор игры

Проект подтвердил жизнеспособность выбранного подхода к разработке и показал хороший потенциал для превращения в коммерческий продукт. Накопленные наработки и решения могут быть использованы в будущих проектах

# Результаты проекта

На текущем этапе разработки удалось создать работоспособное ядро платформера Sylphi, демонстрирующее все ключевые игровые механики в действии. Основным достижением стало создание целостного игрового цикла, включающего один полноценный уровень продолжительностью 2-3 минуты геймплея.

Главный герой игры получил отзывчивую систему управления, включающую плавные прыжки , а также динамичную дэш-механику. Боевая система на текущем этапе представлена базовой атакой ближнего боя, системой получения урона и выразительной анимацией смерти. Также была добавлена камера, следующая за игроком и имеющая ограничение обзора. Это добавляет элемент неожиданности, так как при первом прохождении игрок не знает, что его ждет дальше.

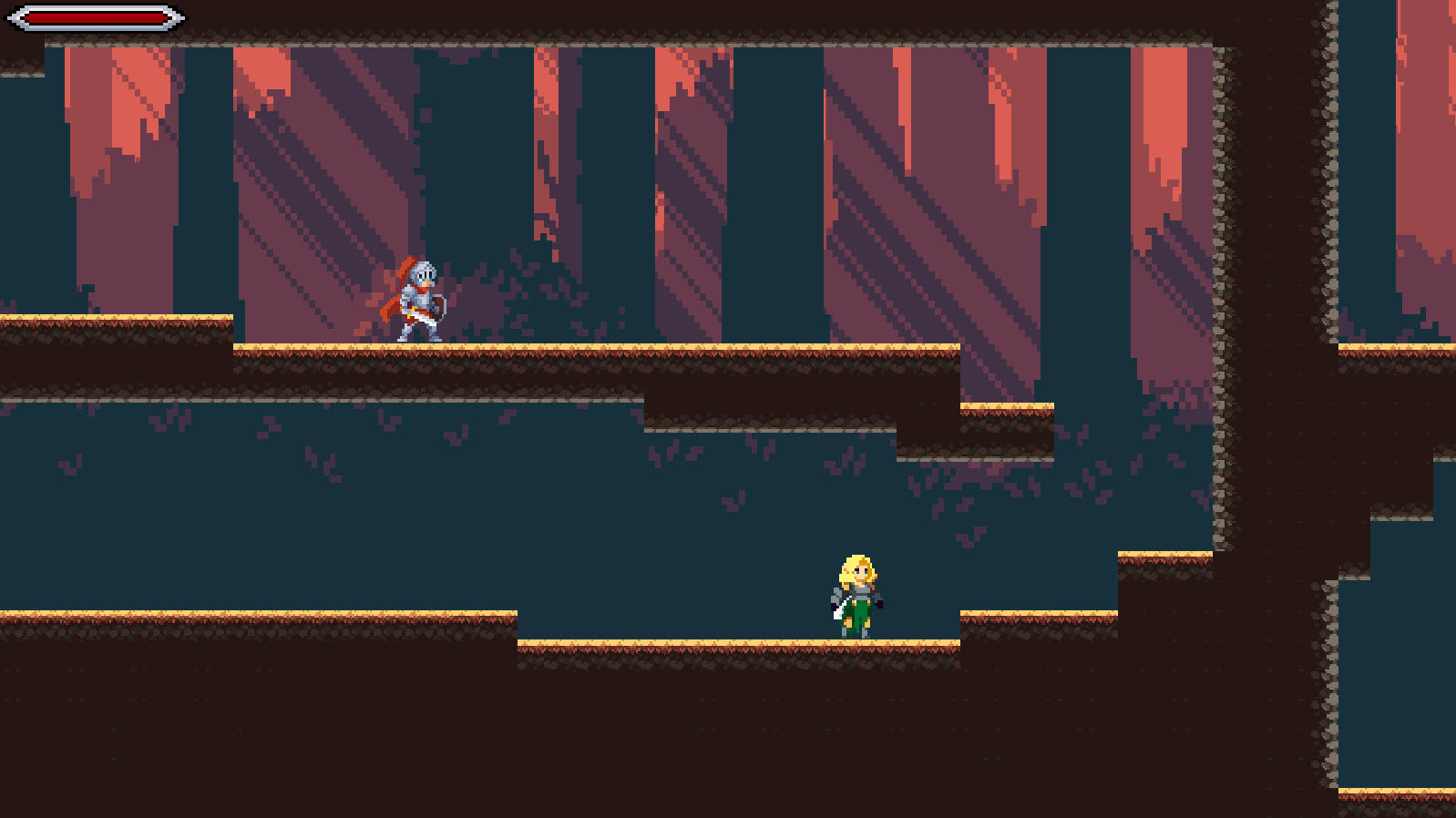


Рисунок 1

Вражеская составляющая игры представлена двумя типами противников. Стандартный враг патрулирует отведенные зоны, наносит контактный урон и обладает 50 единицами здоровья. Особый интерес представляет мини-босс - более мощный противник с увеличенным запасом здоровья (90 HP). Эти противники создают разнообразные игровые ситуации, проверяя навыки игрока.



Рисунок 2

Важным элементом игрового процесса стал экран смерти, который появляется при гибели персонажа. Он включает плавную анимацию появления и две функциональные кнопки - для рестарта уровня и выхода из игры. Техническая реализация экрана учитывает необходимость корректной работы даже при вызове из любого игрового состояния.

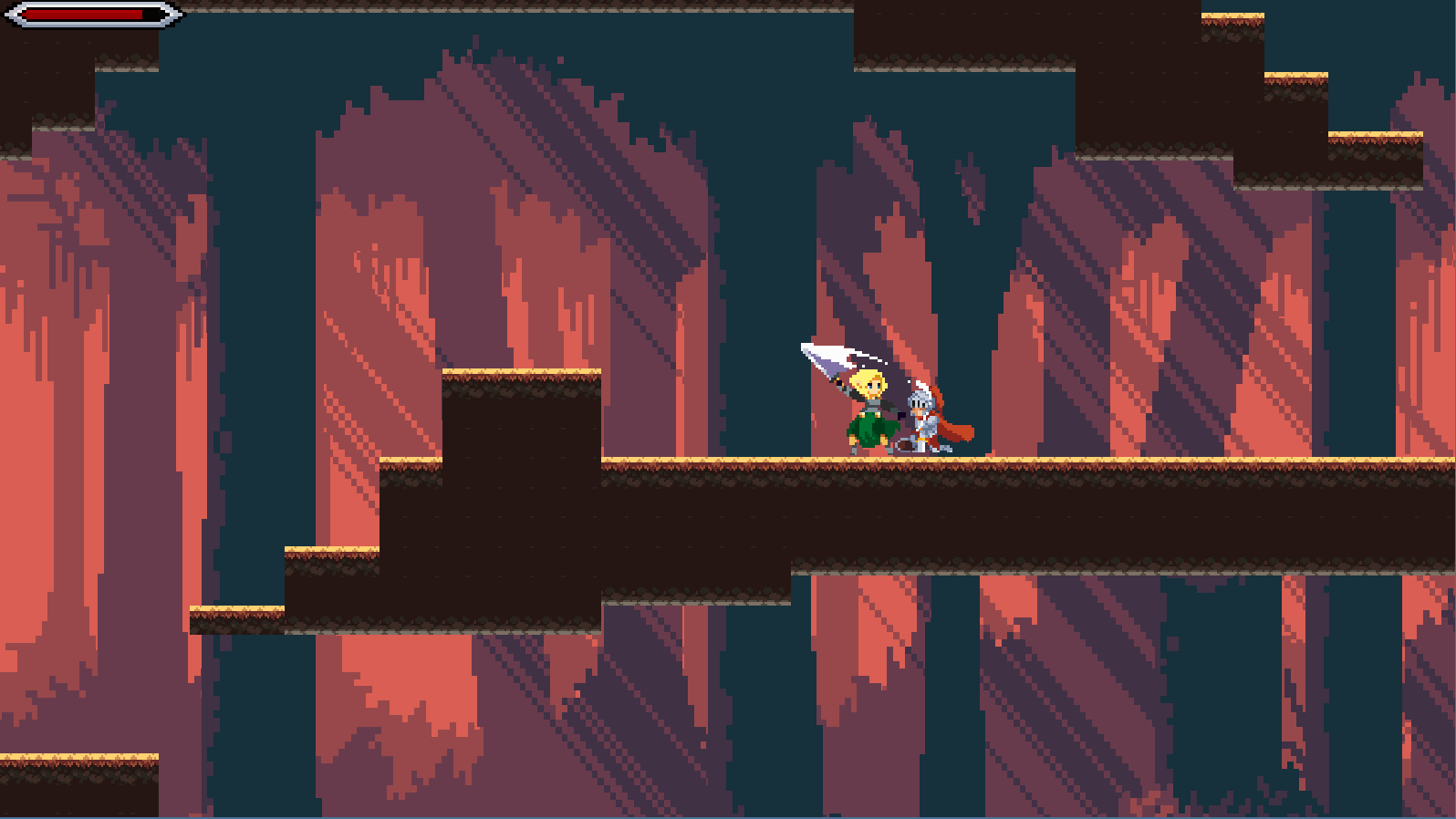
В процессе разработки успешно решили несколько технических проблем. Первоначально мини-босс оказался слишком сложным - после анализа тестовых прохождений его здоровье было снижено со 120 до 90 единиц. Другая проблема проявлялась в редких случаях "зависания" персонажа в текстурах при определенных прыжках - она была устранена тщательной настройкой коллайдеров.

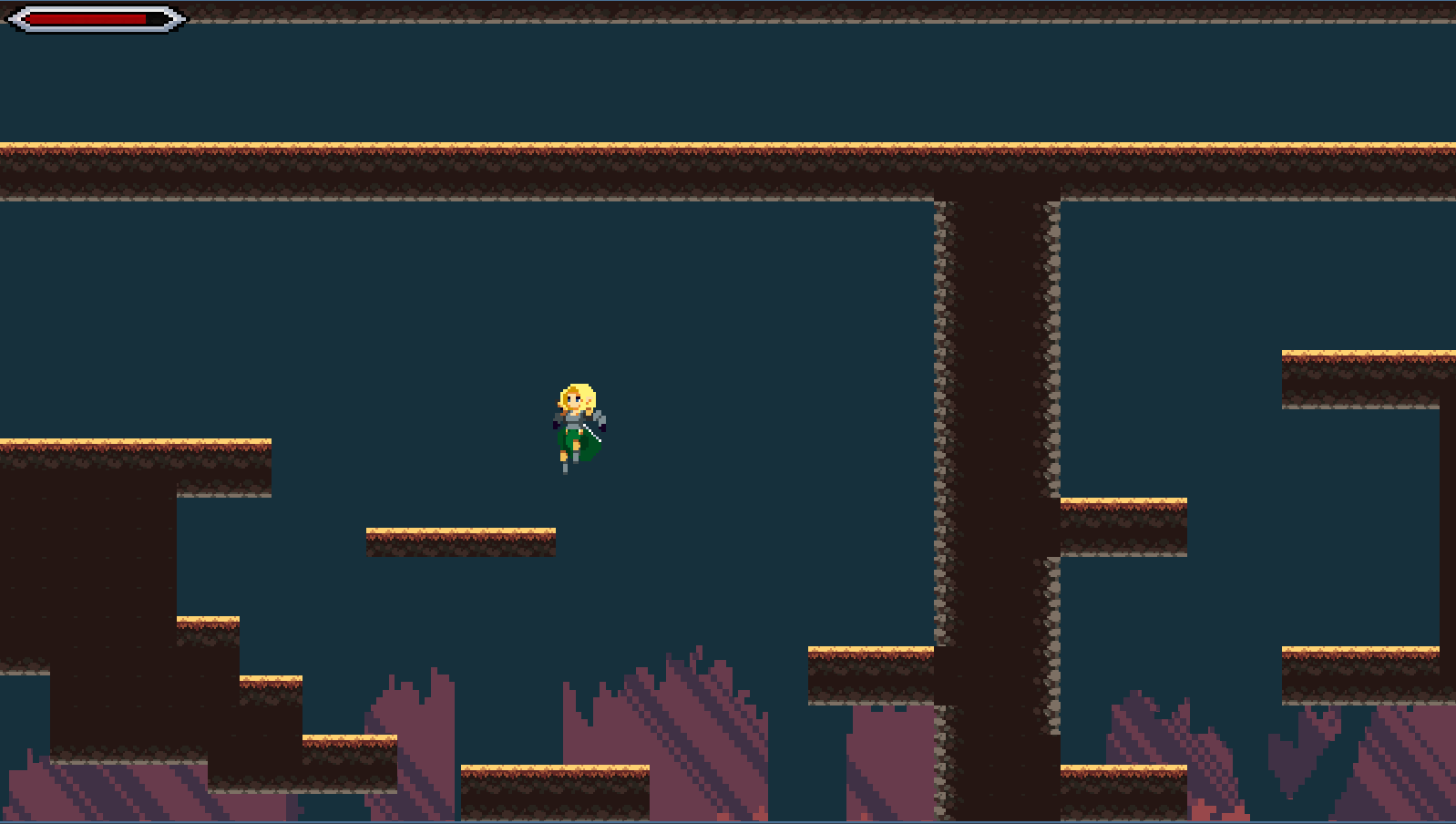
Перспективы развития Sylphi включают как ближайшие задачи, так и долгосрочные планы. В ближайшей перспективе мы планируем добавить 2-3 новых уровня, систему прокачки персонажа и 3-4 новый тип врагов. В долгосрочной перспективе рассматривается реализация босс-файналов для каждого уровня.

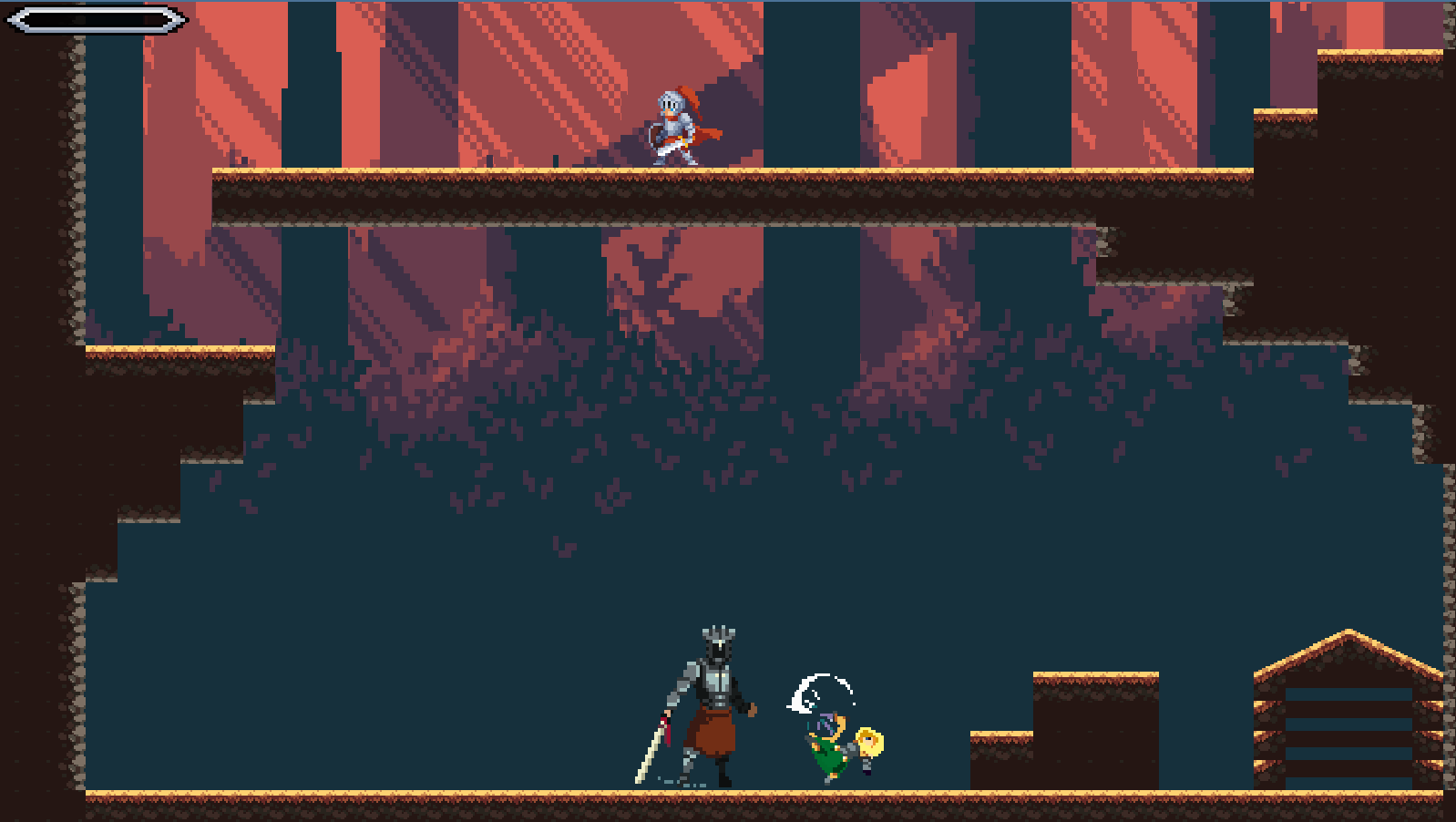
Текущая версия Sylphi представляет собой законченный игровой цикл "рождение-смерть-рестарт", подтверждающий жизнеспособность выбранного подхода. Все базовые системы работают стабильно, а архитектура проекта позволяет относительно легко добавлять новый контент. Этот фундамент дает нам уверенность в успешном развитии проекта в будущем.

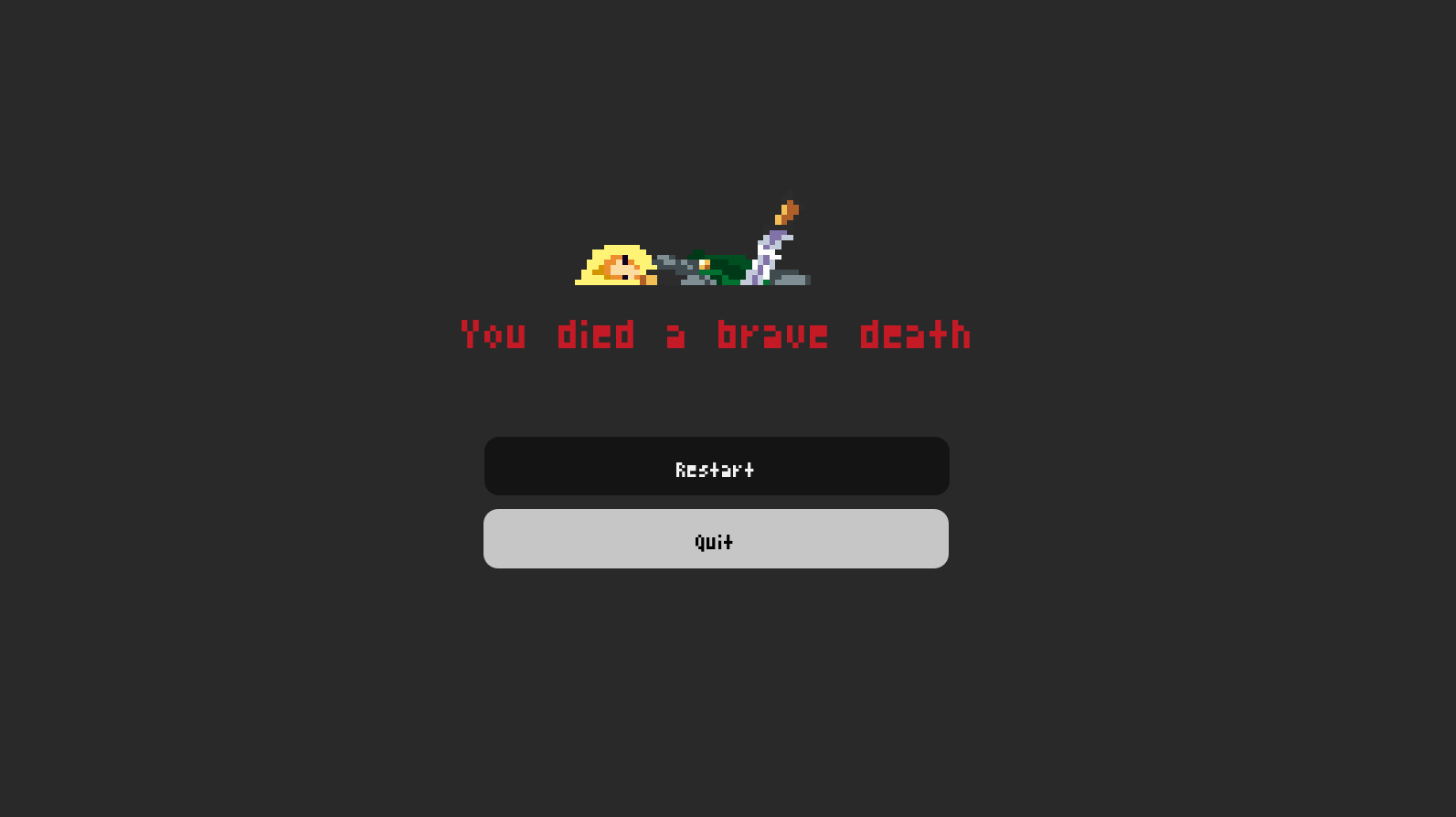
Приложение

**Приложение 1.  
**









**Приложение 2.**

Программный код

**Menu.gd**

еxtends Node2D

func \_on\_start\_pressed():

get\_tree().change\_scene\_to\_file("res://level.tscn")

func \_on\_exit\_pressed():

get\_tree().quit()

**Level.gd**  
extends Node2D

@onready var health\_bar = $CanvasLayer/HealthBar

@onready var player = $Player/Player

# Called when the node enters the scene tree for the first time.

func \_ready():

health\_bar.max\_value = player.health

health\_bar.value = health\_bar.max\_value

print("Уровень инициализирован")

# Called every frame. 'delta' is the elapsed time since the previous frame.

func \_process(delta):

pass

func \_on\_player\_health\_changed(new\_health):

health\_bar.value = new\_health

**Player.gd**

class\_name Player

extends CharacterBody2D

@onready var death\_screen = preload("res://death\_screen.tscn")

signal health\_changed (new\_health)

const SPEED = 200.0

const DASH\_SPEED = 400.0

const JUMP\_VELOCITY = -400.0

const GRAVITY = 900.0

@onready var sprite = $AnimatedSprite2D

@onready var attack\_area\_right = $AttackAreaRight

@onready var attack\_area\_left = $AttackAreaLeft

var is\_dashing = false

var dash\_time = 0.2

var dash\_timer = 0.0

var dash\_direction = 0

var health = 120

var attack\_damage = 25

var is\_dead = false

var active\_attack\_area: Area2D = null

func \_ready():

sprite.connect("animation\_finished", Callable(self, "\_on\_AnimatedSprite2D\_animation\_finished"))

attack\_area\_right.monitoring = false

attack\_area\_left.monitoring = false

attack\_area\_right.body\_entered.connect(\_on\_attack\_area\_body\_entered)

attack\_area\_left.body\_entered.connect(\_on\_attack\_area\_body\_entered)

func \_physics\_process(delta):

if is\_dead:

return

# Движение и даш

var direction = 0.0

if Input.is\_action\_just\_pressed("Dash") and sprite.animation != "Attack" and not is\_dashing:

is\_dashing = true

dash\_timer = dash\_time

dash\_direction = Input.get\_action\_strength("Right") - Input.get\_action\_strength("Left")

if dash\_direction == 0:

dash\_direction = 1

sprite.play("Dash")

if is\_dashing:

velocity.x = dash\_direction \* DASH\_SPEED

dash\_timer -= delta

if dash\_timer <= 0:

is\_dashing = false

else:

if sprite.animation != "Attack":

direction = Input.get\_action\_strength("Right") - Input.get\_action\_strength("Left")

velocity.x = direction \* SPEED

# Гравитация и прыжок

if not is\_on\_floor():

velocity.y += GRAVITY \* delta

else:

velocity.y = 0

if Input.is\_action\_just\_pressed("Jump") and is\_on\_floor() and sprite.animation != "Attack" and not is\_dashing:

velocity.y = JUMP\_VELOCITY

# Атака

if Input.is\_action\_just\_pressed("Attack") and sprite.animation != "Attack" and not is\_dashing:

sprite.play("Attack")

if sprite.flip\_h:

active\_attack\_area = attack\_area\_left

else:

active\_attack\_area = attack\_area\_right

print("Атака начата, активна зона: ", active\_attack\_area.name)

# Поворот спрайта

if direction != 0 and not is\_dashing:

sprite.flip\_h = direction < 0

# Активация зоны атаки

if sprite.animation == "Attack" and sprite.frame >= 2:

if not active\_attack\_area.monitoring:

active\_attack\_area.monitoring = true

elif sprite.animation != "Attack":

attack\_area\_right.monitoring = false

attack\_area\_left.monitoring = false

move\_and\_slide()

# Анимации

if sprite.animation != "Attack" and not is\_dashing:

if not is\_on\_floor():

sprite.play("Jump")

elif abs(velocity.x) > 0:

sprite.play("Run")

else:

sprite.play("Idle")

func \_on\_AnimatedSprite2D\_animation\_finished():

if is\_dead:

queue\_free()

return

if sprite.animation == "Attack":

attack\_area\_right.monitoring = false

attack\_area\_left.monitoring = false

if not is\_on\_floor():

sprite.play("Jump")

elif abs(velocity.x) > 0:

sprite.play("Run")

else:

sprite.play("Idle")

elif sprite.animation == "Dash":

if not is\_on\_floor():

sprite.play("Jump")

elif abs(velocity.x) > 0:

sprite.play("Run")

else:

sprite.play("Idle")

elif sprite.animation == "Death":

queue\_free()

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

if body is Enemy or Enemy2 and not body.is\_dead:

print("Попадание по врагу: ", body.name)

body.take\_damage(attack\_damage)

func take\_damage(amount):

if is\_dead:

return

health -= amount

print("Игрок получил урон: ", amount, " | Осталось HP: ", health)

emit\_signal("health\_changed", health)

if health <= 0:

die()

func die():

is\_dead = true

velocity = Vector2.ZERO

sprite.play("Death")

await sprite.animation\_finished

print("Анимация смерти завершена")

# Создаем экран смерти

var death\_screen\_scene = load("res://death\_screen.tscn")

if death\_screen\_scene:

var death\_screen\_instance = death\_screen\_scene.instantiate()

get\_tree().root.add\_child(death\_screen\_instance)

death\_screen\_instance.show\_death\_screen()

print("Экран смерти добавлен и показан")

else:

printerr("Не удалось загрузить сцену смерти!")

# Скрываем healthbar

get\_tree().call\_group("health\_bar", "hide")

# Удаляем игрока после всего

queue\_free()

print("Игрок удален")

**Knight.gd**

class\_name Enemy

extends CharacterBody2D

const SPEED = 100.0

const ATTACK\_DISTANCE = 30.0

const ATTACK\_COOLDOWN = 1.0

@onready var sprite = $AnimatedSprite2D

@onready var detect\_area = $Detector

var target = null

var is\_dead = false

var is\_attacking = false

var health = 50

var damage = 10

func \_ready():

add\_to\_group("enemies")

detect\_area.body\_entered.connect(\_on\_body\_entered)

detect\_area.body\_exited.connect(\_on\_body\_exited)

sprite.animation\_finished.connect(\_on\_animation\_finished)

func \_physics\_process(\_delta):

if is\_dead:

velocity.x = 0

move\_and\_slide()

return

if target:

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

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

if distance > ATTACK\_DISTANCE:

if not is\_attacking:

velocity.x = direction.x \* SPEED

sprite.flip\_h = velocity.x < 0

if sprite.animation != "Walk":

sprite.play("Walk")

else:

velocity.x = 0

if not is\_attacking:

start\_attack()

else:

velocity.x = 0

if sprite.animation != "Idle":

sprite.play("Idle")

move\_and\_slide()

func \_on\_body\_entered(body):

if body is Player:

target = body

print("Обнаружен игрок")

func \_on\_body\_exited(body):

if body is Player and body == target:

target = null

is\_attacking = false

print("Игрок ушёл")

func start\_attack():

is\_attacking = true

sprite.play("Attack")

print("Враг атакует!")

func \_on\_animation\_finished():

if sprite.animation == "Attack":

if target and not target.is\_dead and target.has\_method("take\_damage"):

target.take\_damage(damage)

start\_attack\_cooldown()

elif sprite.animation == "Death":

queue\_free()

func start\_attack\_cooldown():

is\_attacking = true

var timer = Timer.new()

timer.wait\_time = ATTACK\_COOLDOWN

timer.one\_shot = true

add\_child(timer)

timer.start()

timer.timeout.connect(\_on\_attack\_cooldown\_timeout.bind(timer))

func \_on\_attack\_cooldown\_timeout(timer):

is\_attacking = false

timer.queue\_free()

func take\_damage(amount):

if is\_dead:

return

# Визуальная обратная связь

sprite.modulate = Color.RED

var tween = create\_tween()

tween.tween\_property(sprite, "modulate", Color.WHITE, 0.2)

health -= amount

print("Враг получил урон: ", amount, " | Осталось HP: ", health)

if health <= 0:

die()

func die():

is\_dead = true

velocity.x = 0

sprite.play("Death")

set\_collision\_layer\_value(1, false)

**DeathScreen.gd**

extends Node2D

@onready var animation\_player = $DeathScreen/DeathAnimation

@onready var restart\_button = $DeathScreen/Restart

@onready var quit\_button = $DeathScreen/Quit

func \_ready():

# Настройки для корректного отображения

process\_mode = Node.PROCESS\_MODE\_ALWAYS # Работает даже на паузе

hide()

func \_on\_restart\_pressed():

print("Начало перезагрузки уровня")

# 1. Полностью снимаем паузу

get\_tree().paused = false

# 2. Очищаем текущую сцену

get\_tree().current\_scene.queue\_free()

# 3. Альтернативный способ загрузки с проверкой

var level\_scene = load("res://level.tscn") as PackedScene

if level\_scene:

print("Сцена level.tscn успешно загружена")

var new\_level = level\_scene.instantiate()

# 4. Добавляем новую сцену

get\_tree().root.add\_child(new\_level)

get\_tree().current\_scene = new\_level

# 5. Удаляем экран смерти

queue\_free()

print("Уровень успешно перезагружен")

else:

printerr("Ошибка: Не удалось загрузить level.tscn!")

func \_on\_quit\_pressed():

get\_tree().quit()

func show\_death\_screen():

print("Функция show\_death\_screen вызвана")

show()

# Для AnimatedSprite2D в DeathScreen (если он есть)

if has\_node("DeathAnimation"):

var death\_sprite = $DeathScreen

if death\_sprite.sprite\_frames.has\_animation("Death"):

print("Запускаю анимацию смерти на экране")

death\_sprite.play("Death")

else:

printerr("Анимация Death не найдена в AnimatedSprite2D! Доступные анимации:", death\_sprite.sprite\_frames.get\_animation\_names())

get\_tree().paused = true

print("Игра поставлена на паузу")

**pause\_menu (canvas\_layer.gd)**

extends CanvasLayer

func \_ready():

# Проверка наличия кнопок с выводом ошибки в консоль

if not $Continue:

push\_error("Кнопка Continue не найдена! Проверьте имя нода.")

if not $Exit:

push\_error("Кнопка Exit не найдена! Проверьте имя нода.")

# Подключение сигналов

$Continue.pressed.connect(\_on\_continue\_pressed)

$Exit.pressed.connect(\_on\_quit\_pressed)

hide()

get\_tree().paused = false

func \_input(event):

if event.is\_action\_pressed("cancel"):

toggle\_pause\_menu()

func toggle\_pause\_menu():

visible = not visible

get\_tree().paused = visible

func \_on\_continue\_pressed():

toggle\_pause\_menu()

func \_on\_quit\_pressed():

get\_tree().paused = false

# Проверка существования сцены без assert

if ResourceLoader.exists("res://menu.tscn"):

get\_tree().change\_scene\_to\_file("res://menu.tscn")

else:

push\_error("Файл главного меню не найден!")

**Приложение 3.**

Ссылка на репозиторий: <https://github.com/ManG565565/Project.git>  
Ссылка на видео-демонстрацию: <https://drive.google.com/file/d/1H-DRX9jI_uiqzssam7lRGZPCLiacDj85/view?usp=sharing>

К сожалению, из-за ограничений технического характера и отсутствия необходимого оборудования для записи, голосовые комментарии не были добавлены в видео-демонстрацию.