МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Институт Информационных технологий

Кафедра информационных технологий в атомной энергетике

ОТЧЕТ ПО ПРОЕКТУ

по дисциплине «Разработка приложений на языке Котлин»

Студент группы ИКБО-51-23 Епифанов М.О.

_____

(подпись студента)

Руководитель проектной работы Золотухин С.А.

_____

(подпись руководителя)

Работа представлена «____»_____ 2025 г.

Допущен к работе «___» _____ 2025 г.

Москва 2025

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ 2
1. ПЛАНИРОВАНИЕ И ПОДГОТОВКА 5
1.1 Техническое задание 5
1.1.1 Общие сведения 5
1.1.2 Цели и назначение автоматизированной системы 5
1.1.3 Характеристика объектов автоматизации 5
1.1.4 Требования к системе 6
1.1.5 Состав и содержание работ 6
1.1.6 Порядок разработки и приемки 6
1.1.7 Требования к подготовке объекта к вводу системы в действие 7
1.1.8 Требования к документированию 7
1.1.9 Источники разработки 7
1.2 Диаграмма вариантов использования 7
1.2.1 Обоснование сценариев использования 7
1.2.2 Схематическое представление (Use Case Diagram) 8
1.3 Создание репозитория 9
2. ПРОЕКТИРОВАНИЕ ИНТЕРФЕЙСА 11
2.1 Макеты экранов 11
3. РАЗРАБОТКА ПРИЛОЖЕНИЯ 15
3.1 Архитектура приложения 15
3.2 Реализация ключевых компонентов 17
3.2.1 Frontend 17
3.2.2 Business Logic (Domain Layer) 24
3.2.3 Data Layer 25
3.2.4 Архитектурные особенности 27
3.3 Тестирование приложения
3.4 Документирование кода 30
ЗАКЛЮЧЕНИЕ 35
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 37

ВВЕДЕНИЕ

Fantasy Quest Clicker - это мобильная ролевая игра в жанре кликер, разрабатываемая на Kotlin для Android. Проект сочетает простоту кликера с глубиной классических RPG, предлагая игрокам увлекательный геймплей в фэнтезийном мире.

Проблематика

Рынок мобильных игр переполнен однотипными проектами с упрощенным геймплеем без стратегической глубины. Существует потребность в играх, которые сохраняют доступность жанра, но предлагают сложную и разнообразную механику.

Актуальность

Разработка на Kotlin для Android - перспективное направление. Язык обеспечивает безопасность кода, производительность и удобство поддержки. RPG-кликеры популярны благодаря удовлетворению от постепенного прогресса, что соответствует потребностям мобильных пользователей.

Цели и задачи

Цель: Создать увлекательную мобильную игру, сочетающую простоту кликера с глубиной RPG.

Задачи:

- Разработать интуитивный интерфейс для мобильных устройств
- Реализовать сбалансированную систему прокачки персонажа
- Создать разнообразных противников с уникальными тактиками
- Внедрить систему достижений для поддержания интереса
- Обеспечить стабильную работу на различных устройствах
- Реализовать надежное сохранение прогресса

Состав команды

Проект разрабатывается индивидуально. Все этапы разработки, включая проектирование, программирование и тестирование, выполняются самостоятельно.

1. ПЛАНИРОВАНИЕ И ПОДГОТОВКА

1.1 Техническое задание

1.1.1 Общие сведения

Название проекта: Fantasy Quest Clicker

Тип приложения: Мобильная игра для Android

Жанр: RPG-кликер с элементами фэнтези

Целевая платформа: Android 8.0+ (API 26+)

Технологический стек:

- Kotlin + Android Studio
- Jetpack Compose для интерфейса
- Минимальные требования: Android 8.0+

1.1.2 Цели и назначение автоматизированной системы

Разработка мобильной RPG-игры в жанре кликер, которая предоставляет пользователям увлекательный игровой процесс с элементами прокачки персонажа, сражениями с монстрами и выполнением квестов в фэнтезийном мире.

1.1.3 Характеристика объектов автоматизации

- Игровой персонаж - обладает характеристиками (атака, время), навыками
- Система боя - тактические сражения с различными типами противников
- Квесты - улучшение навыков
- Система квестов - награды за игровой прогресс

1.1.4 Требования к системе

Функциональные требования:

- Интерактивная боевая система с тапами по врагам
- Дерево навыков для прокачки способностей
- Система квестов и достижений
- Локации с различными врагами и боссами
- Сохранение прогресса игры

Нефункциональные требования:

- Адаптивный интерфейс для различных размеров экранов
- Поддержка портретной ориентации
- Оптимизация производительности для мобильных устройств
- Минимальное потребление батареи

1.1.5 Состав и содержание работ

1. Проектирование архитектуры мобильного приложения
2. Разработка пользовательского интерфейса с Jetpack Compose
3. Создание игровой логики и механик
4. Реализация системы сохранения данных
5. Интеграция звуковых эффектов и визуальной обратной связи
6. Тестирование на различных устройствах

1.1.6 Порядок разработки и приемки

Этапы разработки:

- Неделя 1-2: Базовая архитектура и главный экран
- Неделя 3-4: Игровая механика и боевая система
- Неделя 5-6: Система прокачки
- Неделя 7-8: Сохранение данных и полировка

Критерии приемки:

- Стабильная работа на целевых устройствах
- Отсутствие утечек памяти
- Плавная анимация (60 FPS)
- Корректное сохранение прогресса

1.1.7 Требования к подготовке объекта к вводу системы в действие

- Android устройство версии 8.0 или выше
- 100 МБ свободного места в памяти
- Поддержка сенсорного ввода
- Рекомендуется 2 ГБ оперативной памяти

1.1.8 Требования к документированию

- KDoc документация для всех классов и методов
- README с инструкцией по сборке и запуску
- Комментарии для сложной бизнес-логики
- Документация архитектуры приложения

1.1.9 Источники разработки

1. Официальная документация Android Developer
2. Руководства по Material Design
3. Документация Jetpack Compose
4. Примеры игровых приложений на Kotlin

1.2 Диаграмма вариантов использования

1.2.1 Обоснование сценариев использования

Основные сценарии взаимодействия пользователя с приложением:

- Участие в боях с противниками через тапы
- Изучение и прокачка навыков в древе талантов

- Выполнение сюжетных квестов
- Сохранение и загрузка игрового прогресса

1.2.2 Схематическое представление (Use Case Diagram)

Актер "Игрок" взаимодействует с системой через следующие прецеденты:

- Сохранять прогресс
- Сражаться с врагами
- Прокачивать навыки
- Выполнять квесты

Результат создания диаграммы прецедентов представлен на Рисунке 1.

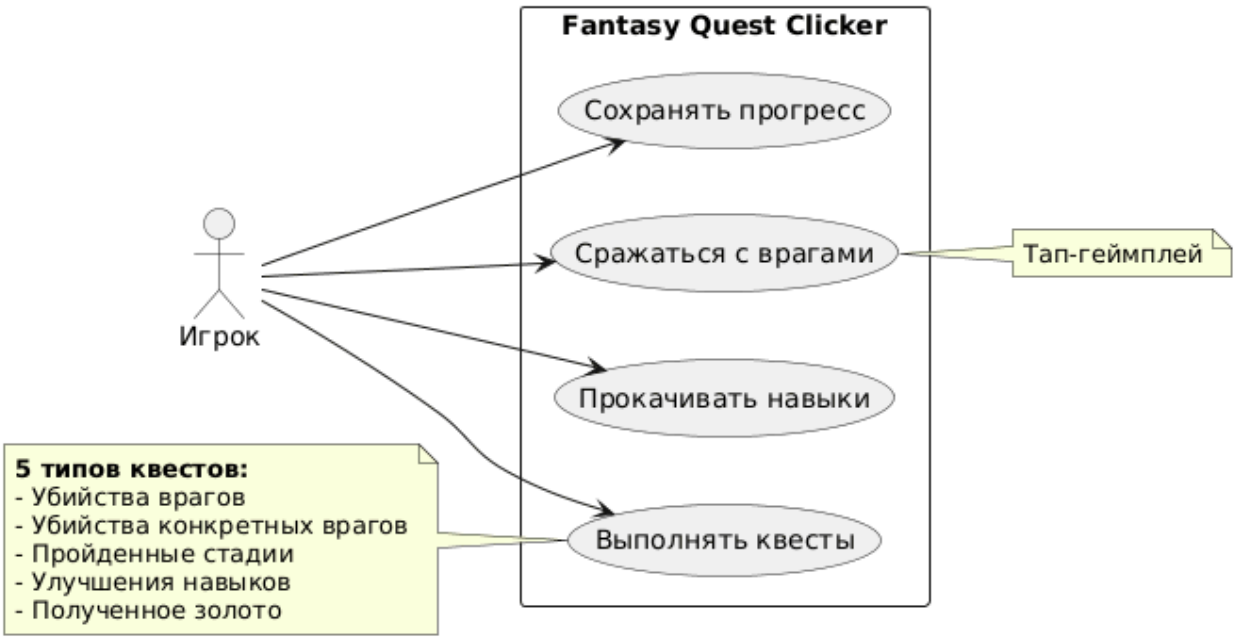
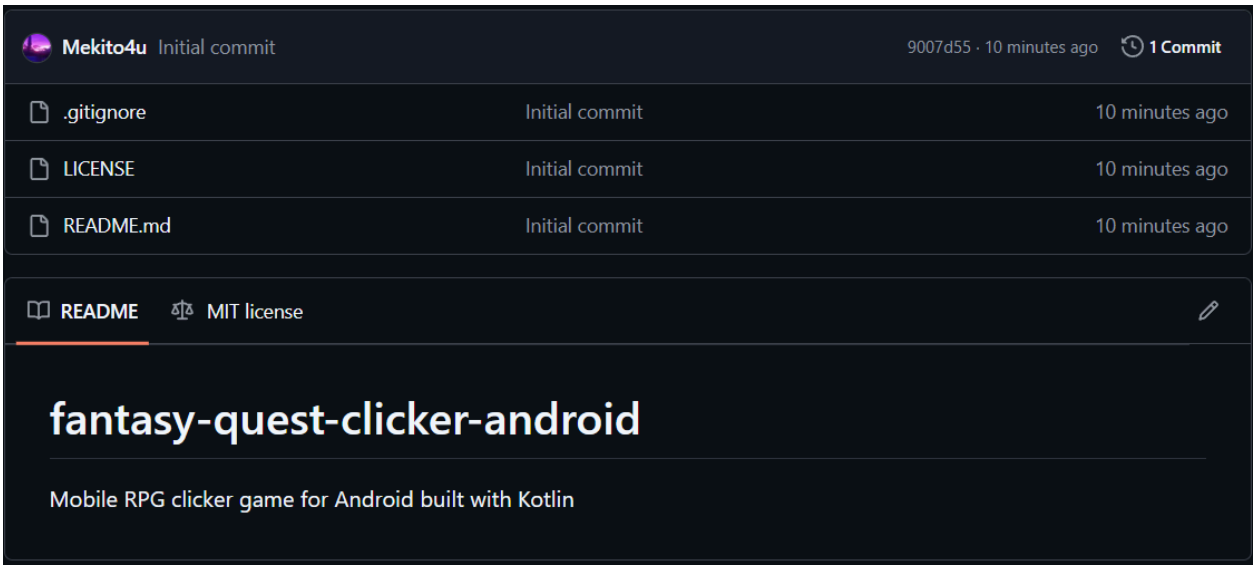


Рисунок 1 - Диаграмма прецедентов мобильной игры Fantasy Quest Clicker

1.3 Создание репозитория

Для контроля версий и управления разработкой создан Git-репозиторий на платформе GitHub. Структура репозитория включает:

- Модуль приложения Android
- Исходный код на Kotlin с использованием современных архитектурных подходов
- Файлы ресурсов (изображения, звуки, строки)
- Конфигурационные файлы Gradle
- Файл README с описанием проекта и инструкциями по сборке



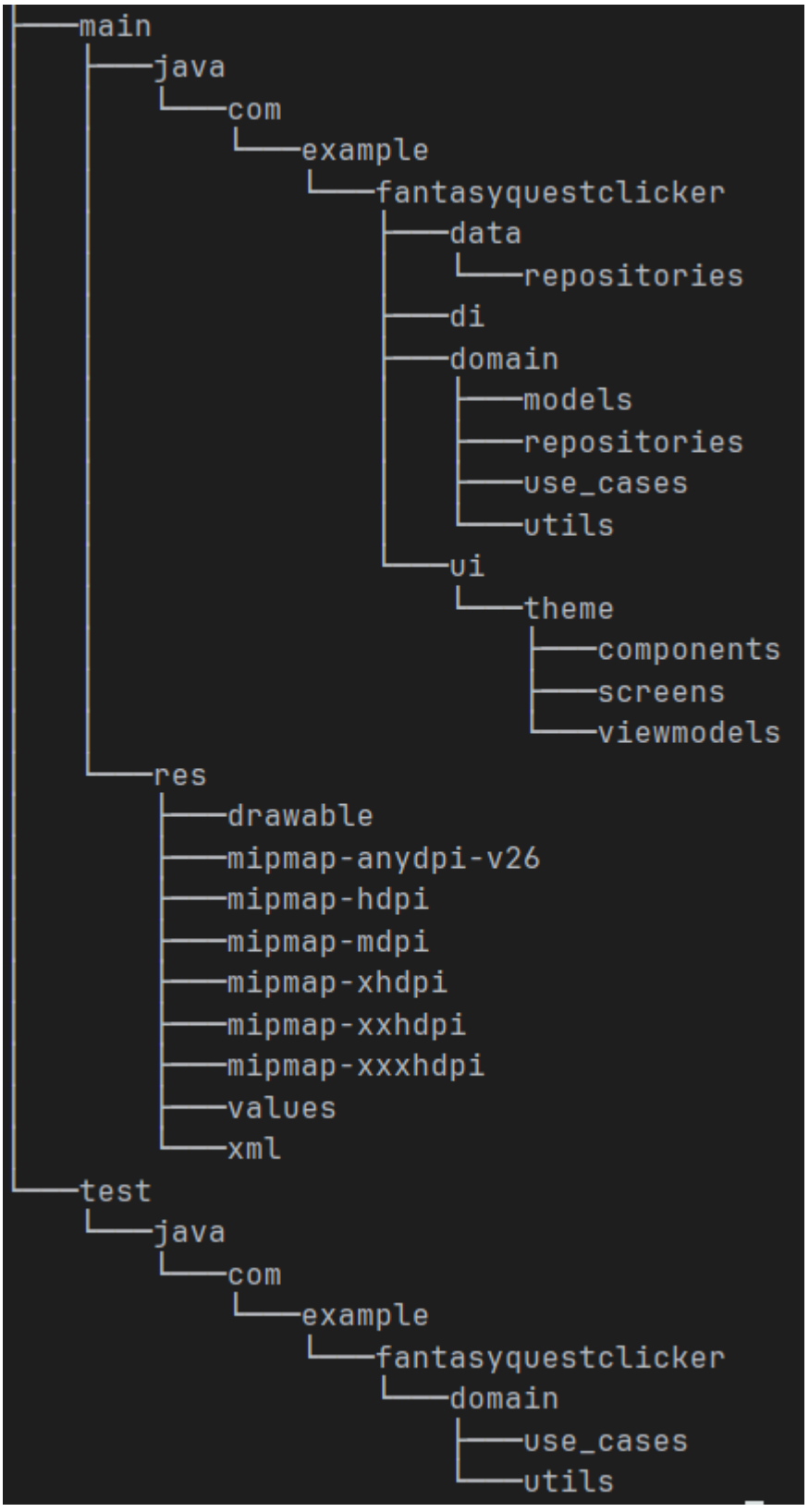


Рисунок 2 - Структура репозитория мобильного приложения

Ссылка на репозиторий: https://github.com/Mekito4u/fantasy-quest-clicker-android

2. ПРОЕКТИРОВАНИЕ ИНТЕРФЕЙСА

2.1 Макеты экранов

Интерфейс спроектирован по принципу минимализма и фокуса на основном геймплее.

Архитектура экранов:

- Экран боя - центральный игровой экран с врагом и кнопкой атаки
- Экран прокачки - система навыков с визуальным прогрессом
- Экран квестов - список заданий с отслеживанием выполнения

Принципы навигации:

- Быстрый переход между основными разделами
- Экран боя как центральный хаб игры
- Минимальное количество промежуточных экранов
- Интуитивная система возврата к предыдущим экранам

Визуальная иерархия:

- Ключевые действия выделены размером и контрастом
- Важная информация расположена в верхней части экрана
- Основные элементы управления - в нижней зоне доступа

Макет окон игры представлен на Рисунках 3-6.

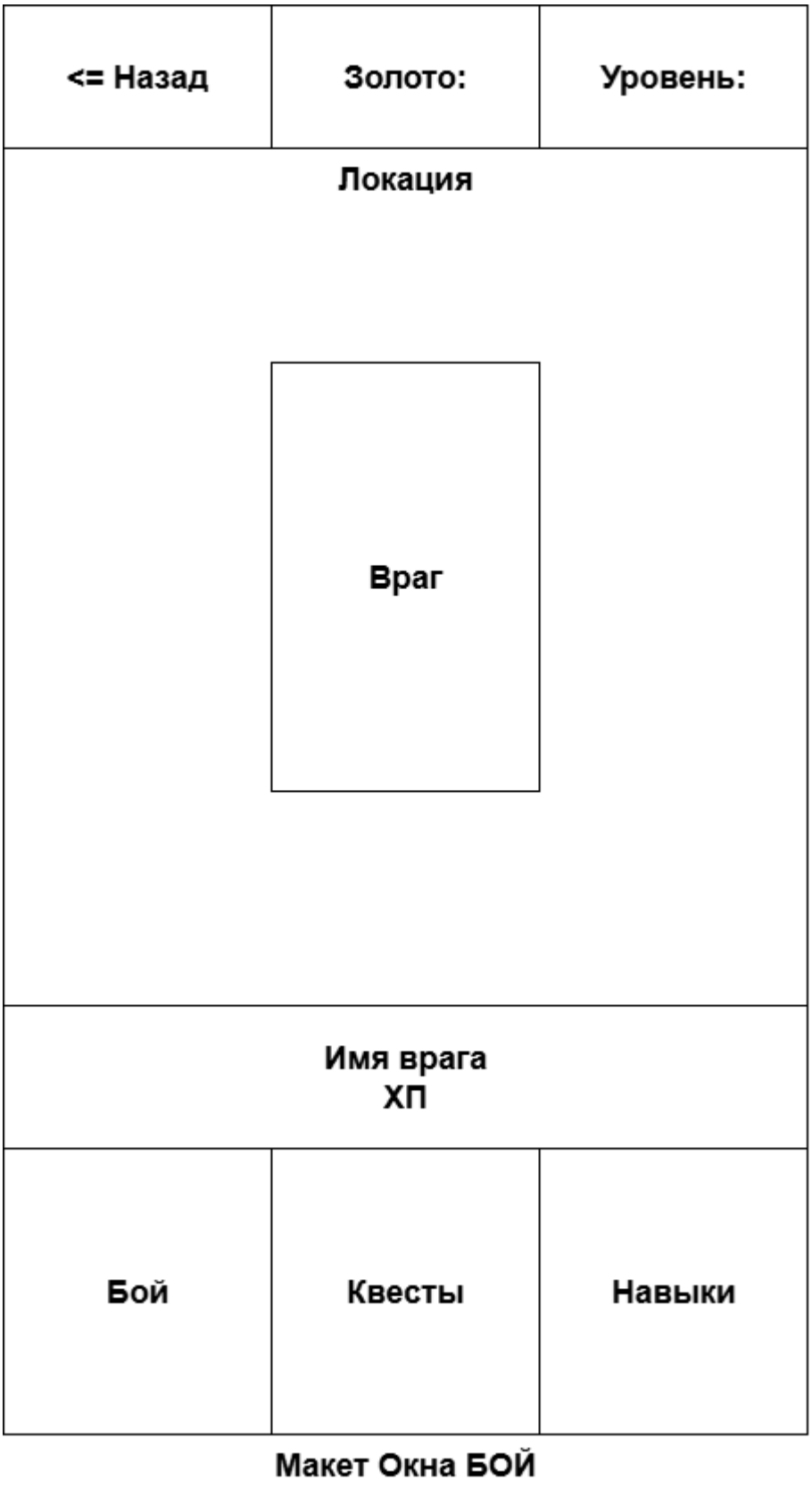


Рисунок 3 — Результат создания макета окна Бой

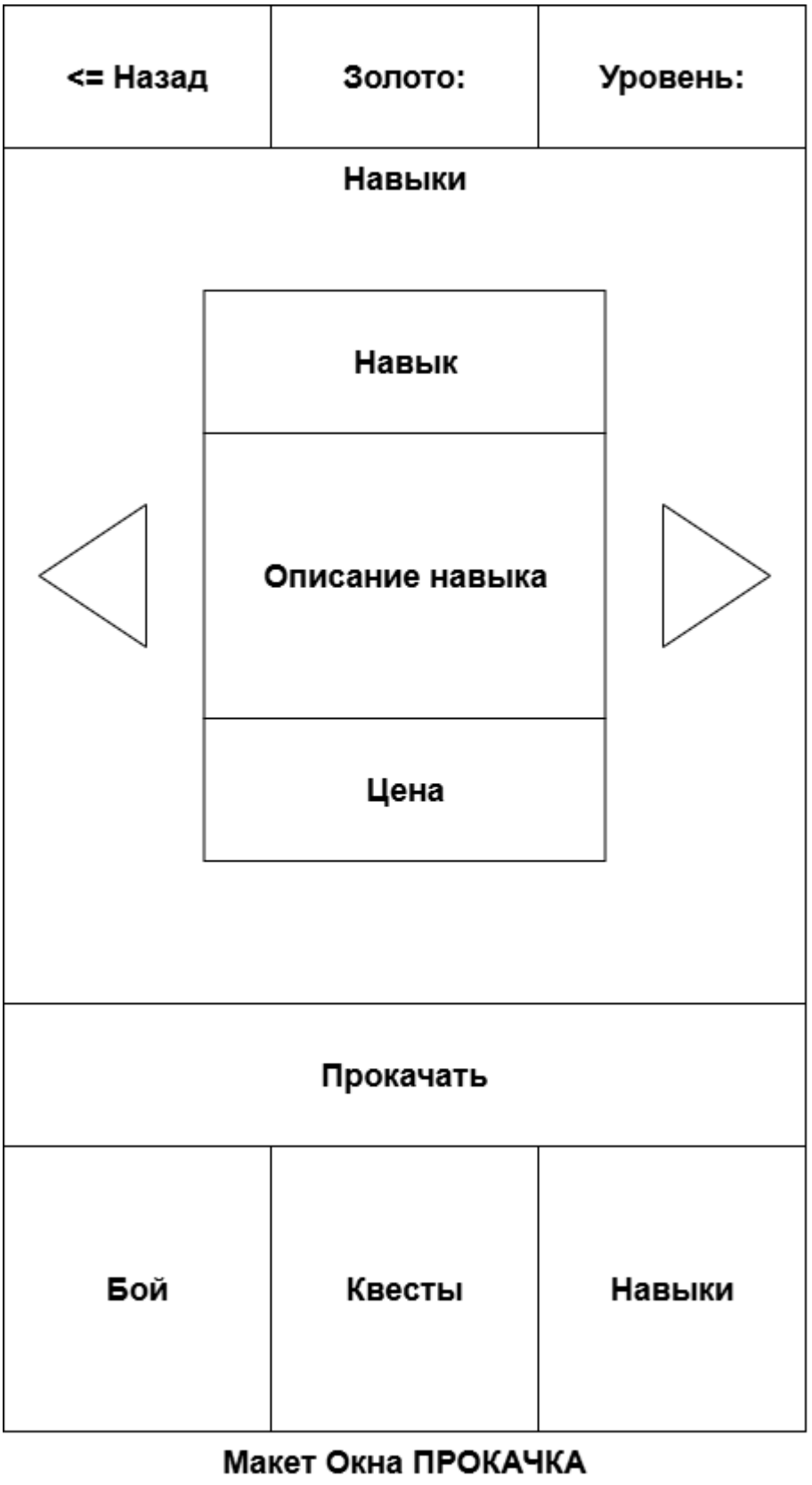


Рисунок 4 — Результат создания макета окна Прокачка

Рисунок 5 — Результат создания макета окна Квесты

3. РАЗРАБОТКА ПРИЛОЖЕНИЯ

3.1 Архитектура приложения

Система сборки и зависимости

Для управления сборкой и зависимостями используется Gradle с Kotlin DSL.

Основные конфигурации:

- Версия Android Gradle Plugin: 8.2.0+
- Минимальная SDK:** 26 (Android 8.0)
- Целевая SDK:** 34 (Android 14)

Ключевые зависимости:

- Jetpack Compose (UI)
- Kotlin Coroutines (асинхронность)
- Lifecycle ViewModel (архитектура)
- DataStore (сохранение прогресса)

```
dependencies {
    implementation(libs.androidx.core.ktx)
    implementation(libs.androidx.lifecycle.runtime.ktx)
    implementation(libs.androidx.activity.compose)
    implementation(platform( dependencyProvider = libs.androidx.compose.bom))
    implementation(libs.androidx.compose.ui)
    implementation(libs.androidx.compose.ui.graphics)
    implementation(libs.androidx.compose.ui.tooling.preview)
    implementation(libs.androidx.compose.material3)
    testImplementation(libs.junit)
    androidTestImplementation(libs.androidx.junit)
    androidTestImplementation(libs.androidx.espresso.core)
    androidTestImplementation(platform( dependencyProvider = libs.androidx.compose.bom))
    androidTestImplementation(libs.androidx.compose.ui.test.junit4)
    debugImplementation(libs.androidx.compose.ui.tooling)
    debugImplementation(libs.androidx.compose.ui.test.manifest)
    implementation("androidx.lifecycle:lifecycle-viewmodel-compose:2.9.4")
    implementation("androidx.compose.runtime:runtime-livedata:1.9.4")
    implementation("androidx.datastore:datastore-preferences:1.1.7")
    testImplementation(libs.junit)
    testImplementation("org.jetbrains.kotlinx:kotlinx-coroutines-test:1.10.2")
    testImplementation("androidx.test:core:1.7.0")
}
```

Рисунок 7 - Конфигурация зависимостей в build.gradle.kts

Схематичное изображение архитектуры разрабатываемого проекта представлено на Рисунке 8.

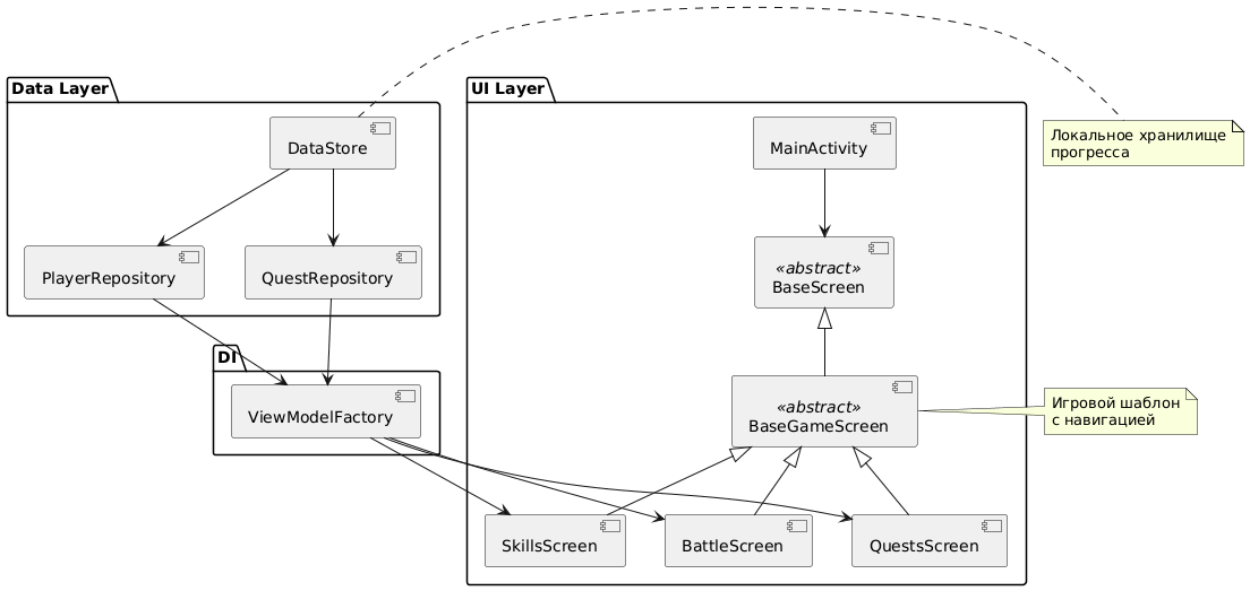


Рисунок 8 — Результат создания схемы архитектуры проекта

3.2 Реализация ключевых компонентов

3.2.1 Frontend

Для реализации клиентской части проекта выбраны технологии Jetpack Compose для UI и ViewModel для управления состоянием. Создана иерархия экранов с базовыми шаблонами для единообразия интерфейса.

Листинг 1 – Базовый игровой экран (BaseGameScreen.kt)

```
@Composable
fun BaseGameScreen(
    player: Player,
    showNavigationArrows: Boolean = false,
    onLeftArrowClick: () -> Unit = {},
    onRightArrowClick: () -> Unit = {},

    currentScreen: String = "battle",
    onScreenChange: (String) -> Unit = {},

    onCenterClick: () -> Unit = {},

    centerAdditionalContentTop: @Composable () -> Unit = {},
    centerMainContent: @Composable () -> Unit = {},
    centerAdditionalContentBottom: @Composable () -> Unit = {},

    backgroundColor: Color = Color(0xFF1E1E1E)
) {
    val backgroundRes = remember(player.currentStage) {
        getBackgroundForStage(player.currentStage)
    }

    BaseScreen(
        backgroundColor = backgroundColor,
        topContent = {
            // ВЕРХНЯЯ ПАНЕЛЬ
            Row(
                modifier = Modifier
                    .fillMaxWidth()
                    .padding(8.dp),
                horizontalArrangement = Arrangement.Center,
                verticalAlignment = Alignment.CenterVertically
            ) {
                Text(
                    text = "(${player.enemiesDefeated}/10)",
                    color = Color.White,
                    fontSize = 22.sp,
                    textAlign = TextAlign.Start,
                    modifier = Modifier.weight(0.5f)
                )

                Text(
                    text = when ((player.currentStage - 1) / 5 + 1) {
                        1 -> "Светлая Долина"
                        2 -> "Заброшенная Деревня"
                        3 -> "Врата Руин"
                        4 -> "Мрачный Лес"
                        5 -> "Болота Смерти"
```

```
                        else -> "Гнездо Дракона"
                    } + " ${player.currentStage}",
                    color = Color.White,
                    fontSize = 20.sp,
                    textAlign = TextAlign.Center,
                    modifier = Modifier.weight(1f)
                )

                Text(
                    text = "\uD83E\uDE99: ${formatNumber(player.gold)}",
                    color = Color(0xFFFFD700),
                    fontSize = 22.sp,
                    textAlign = TextAlign.End,
                    modifier = Modifier.weight(0.5f)
                )
            }
        },
        centerContent = {
            // ЦЕНТРАЛЬНЫЙ КОНТЕНТ
            Image(
                painter = painterResource(backgroundRes),
                contentDescription = "Background",
                modifier = Modifier.fillMaxSize(),
                contentScale = ContentScale.Crop
            )
            Column(
                modifier = Modifier
                    .fillMaxSize()
                    .clickable(
                                                interactionSource = remember {
MutableInteractionSource() },
                        indication = null,
                        onClick = onCenterClick
                    )
            ) {
                // ВЕРХНИЙ ДОП. КОНТЕНТ (6.25%)
                Box(
                    modifier = Modifier
                        .fillMaxWidth()
                        .weight(0.0625f),
                    contentAlignment = Alignment.Center
                ) {
                    centerAdditionalContentTop()
                }

                // ОСНОВНОЙ КОНТЕНТ (81.25%)
                Box(
                    modifier = Modifier
                        .fillMaxWidth()
                        .weight(0.8125f),
                    contentAlignment = Alignment.Center
                ) {
                    if (showNavigationArrows) {
                        Row(
                            modifier = Modifier.fillMaxSize(),
                            horizontalArrangement = Arrangement.SpaceBetween,
                            verticalAlignment = Alignment.CenterVertically
                        ) {
                            // ЛЕВАЯ СТРЕЛКА
                            TriangleButton(
                                onClick = onLeftArrowClick,
                                direction = TriangleDirection.LEFT,
                                modifier = Modifier
```

```
                                    .size(80.dp)
                                    .weight(0.2f)
                            )

                            Spacer(modifier = Modifier.weight(0.015f))

                            // ОСНОВНОЙ КОНТЕНТ
                            Box(
                                modifier = Modifier
                                    .fillMaxWidth(0.5f)
                                    .aspectRatio(225f / 294f)
                                                .background(Color.DarkGray,
RoundedCornerShape(16.dp)),
                                contentAlignment = Alignment.Center
                            ) {
                                 centerMainContent()
                            }

                             Spacer(modifier = Modifier.weight(0.015f))

                            // ПРАВАЯ СТРЕЛКА
                            TriangleButton(
                                onClick = onRightArrowClick,
                                direction = TriangleDirection.RIGHT,
                                modifier = Modifier
                                    .size(80.dp)
                                    .weight(0.2f)
                            )
                        }
                    } else {
                        Box(
                            modifier = Modifier
                                .fillMaxWidth(0.5f)
                                .aspectRatio(225f / 294f),
                                                //.background(Color.DarkGray,
RoundedCornerShape(16.dp)),
                            contentAlignment = Alignment.Center
                        ) {
                             centerMainContent()
                        }
                    }
                }

                // НИЖНИЙ ДОП. КОНТЕНТ (12.5%)
                Box(
                    modifier = Modifier
                        .fillMaxWidth()
                        .weight(0.125f),
                    contentAlignment = Alignment.Center
                ) {
                     centerAdditionalContentBottom()
                }
            }
        },
        bottomContent = {
            // НИЖНИЕ КНОПКИ
            Row(
                modifier = Modifier
                    .fillMaxWidth()
                    .padding(8.dp),
                horizontalArrangement = Arrangement.SpaceBetween,
                verticalAlignment = Alignment.CenterVertically
            ) {
```

```
                Box(modifier = Modifier.weight(1f)) {
                    Button(
                        onClick = { onScreenChange("battle") },
                        modifier = Modifier.aspectRatio(1f),
                        shape = RoundedCornerShape(12.dp),
                        colors = ButtonDefaults.buttonColors(
                                containerColor = if (currentScreen == "battle")
Color(0xFF363636) else Color.DarkGray,
                            disabledContainerColor = Color(0xFF363636),
                            disabledContentColor = Color(0xCCCCCCCC)
                        ),
                        elevation = ButtonDefaults.buttonElevation(
                            defaultElevation = 0.dp,
                            pressedElevation = 0.dp
                        ),
                        enabled = currentScreen != "battle"
                    ) {
                                Text("Бой", fontSize = 16.sp, fontWeight =
FontWeight.Bold)
                    }
                }

                Spacer(modifier = Modifier.weight(0.1f))

                Box(modifier = Modifier.weight(1f)) {
                    Button(
                        onClick = { onScreenChange("quests") },
                        modifier = Modifier.aspectRatio(1f),
                        shape = RoundedCornerShape(12.dp),
                        colors = ButtonDefaults.buttonColors(
                                containerColor = if (currentScreen == "quests")
Color(0xFF363636) else Color.DarkGray,
                            disabledContainerColor = Color(0xFF363636),
                            disabledContentColor = Color(0xCCCCCCCC)
                        ),
                        elevation = ButtonDefaults.buttonElevation(
                            defaultElevation = 0.dp,
                            pressedElevation = 0.dp
                        ),
                        enabled = currentScreen != "quests"
                    ) {
                                Text("Квесты", fontSize = 16.sp, fontWeight =
FontWeight.Bold)
                    }
                }

                Spacer(modifier = Modifier.weight(0.1f))

                Box(modifier = Modifier.weight(1f)) {
                    Button(
                        onClick = { onScreenChange("skills") },
                        modifier = Modifier.aspectRatio(1f),
                        shape = RoundedCornerShape(12.dp),
                        colors = ButtonDefaults.buttonColors(
                                containerColor = if (currentScreen == "skills")
Color(0xFF363636) else Color.DarkGray,
                            disabledContainerColor = Color(0xFF363636),
                            disabledContentColor = Color(0xCCCCCCCC)
                        ),
                        elevation = ButtonDefaults.buttonElevation(
                            defaultElevation = 0.dp,
                            pressedElevation = 0.dp
                        ),
```

```
                        enabled = currentScreen != "skills"
                    ) {
                                Text("Навыки", fontSize = 16.sp, fontWeight =
FontWeight.Bold)
                    }
                }
            }
        }
    )
}

enum class TriangleDirection {
    LEFT, RIGHT
}

// КНОПКА С ТРЕУГОЛЬНИКАМИ
@Composable
private fun TriangleButton(
    onClick: () -> Unit,
    direction: TriangleDirection,
    modifier: Modifier = Modifier
) {
    Box(
        modifier = modifier
            .fillMaxSize()
            .clip(TriangleShape(direction))
            .background(Color.DarkGray)
            .clickable(onClick = onClick),
        contentAlignment = Alignment.Center
    ) {
        Box(
            modifier = Modifier
                .fillMaxWidth(0.85f)
                .fillMaxHeight(0.55f)
                .clip(TriangleShape(direction))
                .background(Color.Gray)
        )
    }
}

// ФОРМА ТРЕУГОЛЬНИКА
private class TriangleShape(private val direction: TriangleDirection) :
androidx.compose.ui.graphics.Shape {
    override fun createOutline(
        size: androidx.compose.ui.geometry.Size,
        layoutDirection: androidx.compose.ui.unit.LayoutDirection,
        density: androidx.compose.ui.unit.Density
    ): androidx.compose.ui.graphics.Outline {
        val path = androidx.compose.ui.graphics.Path().apply {
            when (direction) {
                TriangleDirection.LEFT -> {
                    moveTo(size.width, 0f)
                    lineTo(0f, size.height / 2)
                    lineTo(size.width, size.height)
                    close()
                }
                TriangleDirection.RIGHT -> {
                    moveTo(0f, 0f)
                    lineTo(size.width, size.height / 2)
                    lineTo(0f, size.height)
                    close()
                }
            }
```

```
        }
        return androidx.compose.ui.graphics.Outline.Generic(path)
    }
}
```

Листинг 2 – Кастомный компонент HealthBar

```
@Composable
fun HealthBar(
    currentHealth: Int,
    maxHealth: Int,
    modifier: Modifier = Modifier
) {
    val progress = currentHealth.toFloat() / maxHealth.toFloat()
    val healthColor = when {
        progress > 0.6f -> Color.Green
        progress > 0.3f -> Color.Yellow
        else -> Color.Red
    }

    Column(modifier = modifier) {
        Text(
            text = "HP: $currentHealth/$maxHealth",
            color = Color.LightGray,
            modifier = Modifier.align(Alignment.CenterHorizontally)
        )

        Spacer(modifier = Modifier.height(4.dp))

        // ОКРУГЛЫЙ PROGRESS BAR
        Box(
            modifier = Modifier
                .fillMaxWidth()
                .height(20.dp)
                .clip(RoundedCornerShape(10.dp))
                .background(Color(0xFF444444))
        ) {
            // Заливка здоровья
            Box(
                modifier = Modifier
                    .fillMaxWidth(progress)
                    .fillMaxHeight()
                    .clip(RoundedCornerShape(10.dp))
                    .background(healthColor)
            )
        }
    }
}
```

Листинг 3 – Игровой экран боя (BattleScreen.kt)

```
@Composable
fun BattleScreen(
    currentScreen: String = "battle",
    onScreenChange: (String) -> Unit = { _ -> },
) {
            val    viewModel:    BattleViewModel    =    viewModel(factory    =
ViewModelFactory(LocalContext.current))
    val enemy by viewModel.currentEnemy.collectAsState()
    val player by viewModel.player.collectAsState()

    LaunchedEffect(Unit) {
        viewModel.loadPlayerProgress()
    }

    BaseGameScreen(
        player = player,
        showNavigationArrows = false,
        currentScreen = currentScreen,
        onScreenChange = onScreenChange,
        onCenterClick = { viewModel.attackEnemy() },

        // ЦЕНТРАЛЬНАЯ ЧАСТЬ - PLAYER BAR
        centerAdditionalContentTop = {
            PlayerTimer(
                currentTime = player.currentTime,
            )
        },

        // ЦЕНТРАЛЬНАЯ ЧАСТЬ - ВРАГ
        centerMainContent = {
            Box(
                modifier = Modifier
                    .fillMaxWidth(1f)
                    .aspectRatio(225f / 294f),
                contentAlignment = Alignment.Center
            ) {
                Text(enemy.imageRes, fontSize = 150.sp, color = Color.White)
            }

        },

        // ЦЕНТРАЛЬНАЯ ЧАСТЬ - HEALTH BAR
        centerAdditionalContentBottom = {
            Column(
                horizontalAlignment = Alignment.CenterHorizontally,
                verticalArrangement = Arrangement.Center
            ) {
                Text(
                    enemy.name,
                    color = Color.White,
                    fontSize = 20.sp,
                    fontWeight = FontWeight.Bold
                )
                Spacer(modifier = Modifier.height(8.dp))
                HealthBar(
                    currentHealth = enemy.currentHealth,
                    maxHealth = enemy.maxHealth,
                    modifier = Modifier
                        .fillMaxWidth(0.9f)
                        .fillMaxHeight(0.8f)
```

```
                )
            }
        },
    )
}
```

3.2.2 Business Logic (Domain Layer)

Для реализации бизнес-логики проекта выбраны технологии Kotlin Coroutines, Flow и Clean Architecture. Доменный слой содержит Use Cases для игровой логики и модели данных.

Листинг 4 – Модель игрока (Player.kt)

```
data class Player(
    val gold: Int = 0,
    val baseAttack: Int = 5,
    val maxTime: Int = 10,
    val currentTime: Int = 10,
    val currentStage: Int = 1,
    val totalKillsEnemy: Int = 0,
    val totalKills: Int = 0,
    val enemiesDefeated: Int = 0,
    val criticalChance: Double = 0.0,
    val upgradeSkills: Int = 0,
    val totalGoldEarned: Int = 0
) {
    fun calculateDamage(): Int = if (Math.random() < criticalChance) baseAttack
* 2 else baseAttack

    val isDefeated: Boolean get() = currentTime <= 0
    val isBossFight: Boolean get() = enemiesDefeated >= 10
}
```

Листинг 5 – Use Case атаки врага (AttackEnemyUseCase.kt)

```
class AttackEnemyUseCase {
    operator fun invoke(player: Player, enemy: Enemy): AttackResult {
        val damage = player.calculateDamage()
        val updatedEnemy = enemy.takeDamage(damage)
        val isEnemyDefeated = updatedEnemy.isDefeated

        val updatedPlayer = if (isEnemyDefeated) {
            player.copy(gold = player.gold + enemy.baseReward,
                totalGoldEarned = player.totalGoldEarned + enemy.baseReward)
        } else {
            player
        }

        return AttackResult(
            damageDealt = damage,
            isEnemyDefeated = isEnemyDefeated,
            goldReward = if (isEnemyDefeated) enemy.baseReward else 0,
            updatedPlayer = updatedPlayer,
```

```
            updatedEnemy = updatedEnemy
        )
    }
}
```

Листинг 6 – Модель врага (Enemy.kt)

```
data class Enemy(
    val id: Int,
    val name: String,
    val currentHealth: Int,
    val maxHealth: Int,
    val baseReward: Int,
    val imageRes: String,
) {
    val isDefeated: Boolean get() = currentHealth <= 0

    fun takeDamage(damage: Int): Enemy = copy(
        currentHealth = (currentHealth - damage).coerceAtLeast(0)
    )
}
```

Листинг 7 – Система квестов (Quest.kt)

```
data class Quest(
    val type: QuestType,
    val targetValue: Int,
    val targetName: String = "",
    val isCompleted: Boolean = false,
    val reward: Int
)
```

3.2.3 Data Layer

Для работы с данными проекта выбрана технология Android DataStore (Preferences) для локального хранения прогресса игрока. DataStore обеспечивает потокобезопасность и реактивные обновления.

Листинг 8 – Репозиторий игрока (PlayerRepository.kt)

```
private val Context.playerDataStore: DataStore<Preferences> by
preferencesDataStore(name = "player_data")

class PlayerRepository(private val context: Context) {

    companion object {
        private val GOLD_KEY = intPreferencesKey("gold")
        private val BASE_ATTACK_KEY = intPreferencesKey("base_attack")
        private val MAX_TIME_KEY = intPreferencesKey("max_time")
                                    private val CRITICAL_CHANCE_KEY =
floatPreferencesKey("critical_chance")
        private val CURRENT_STAGE_KEY = intPreferencesKey("current_stage")
        private val TOTAL_KILLS_KEY = intPreferencesKey("total_kills")
```

```
                                private     val     TOTAL_KILLS_ENEMY_KEY     =
intPreferencesKey("total_kills_enemy")
        private val UPGRADE_SKILLS_KEY = intPreferencesKey("upgrade_skills")
                                private     val     TOTAL_GOLD_EARNED_KEY     =
intPreferencesKey("total_gold_earned")
    }

    suspend fun savePlayer(player: Player) {
        context.playerDataStore.edit { preferences ->
            preferences[GOLD_KEY] = player.gold
            preferences[BASE_ATTACK_KEY] = player.baseAttack
            preferences[MAX_TIME_KEY] = player.maxTime
            preferences[CRITICAL_CHANCE_KEY] = player.criticalChance.toFloat()
            preferences[CURRENT_STAGE_KEY] = player.currentStage
            preferences[TOTAL_KILLS_KEY] = player.totalKills
            preferences[TOTAL_KILLS_ENEMY_KEY] = player.totalKillsEnemy
            preferences[UPGRADE_SKILLS_KEY] = player.upgradeSkills
            preferences[TOTAL_GOLD_EARNED_KEY] = player.totalGoldEarned
        }
    }

    suspend fun loadPlayer(): Player {
        val preferences = context.playerDataStore.data.first()
        return Player(
            gold = preferences[GOLD_KEY] ?: 0,
            baseAttack = preferences[BASE_ATTACK_KEY] ?: 5,
            maxTime = preferences[MAX_TIME_KEY] ?: 10,
               criticalChance = preferences[CRITICAL_CHANCE_KEY]?.toDouble() ?:
0.0,
            currentStage = preferences[CURRENT_STAGE_KEY] ?: 1,
            totalKills = preferences[TOTAL_KILLS_KEY] ?: 0,
            totalKillsEnemy = preferences[TOTAL_KILLS_ENEMY_KEY] ?: 0,
            upgradeSkills = preferences[UPGRADE_SKILLS_KEY] ?: 0,
            totalGoldEarned = preferences[TOTAL_GOLD_EARNED_KEY] ?: 0
        )
    }
}
```

Листинг 9 – Репозиторий квестов (QuestRepository.kt)

```
private     val     Context.questDataStore:     DataStore<Preferences>     by
preferencesDataStore(name = "quest_data")

class QuestRepository(private val context: Context) {

    suspend fun saveQuests(quests: List<Quest>) {
        context.questDataStore.edit { preferences ->
            quests.forEachIndexed { index, quest ->
                    preferences[intPreferencesKey("quest_${index}_type")] =
quest.type.ordinal
                preferences[intPreferencesKey("quest_${index}_target")] =
quest.targetValue
                preferences[intPreferencesKey("quest_${index}_reward")] =
quest.reward
               preferences[stringPreferencesKey("quest_${index}_name")] =
quest.targetName
            }
            preferences[intPreferencesKey("quests_count")] = quests.size
        }
    }

    suspend fun loadQuests(): List<Quest> {
```

```
        val preferences = context.questDataStore.data.first()
        val count = preferences[intPreferencesKey("quests_count")] ?: 0

        return (0 until count).mapNotNull { index ->
                                                    val     typeOrdinal     =
preferences[intPreferencesKey("quest_${index}_type")]   ?:   return@mapNotNull
null
                    val  type  =  QuestType.entries.getOrNull(typeOrdinal)  ?:
return@mapNotNull null

            Quest(
                type = type,
                                                                targetValue    =
preferences[intPreferencesKey("quest_${index}_target")] ?: 0,
                                                                    reward     =
preferences[intPreferencesKey("quest_${index}_reward")] ?: 0,
                                                                targetName     =
preferences[stringPreferencesKey("quest_${index}_name")] ?: ""
            )
        }
    }
}
```

3.2.4 Архитектурные особенности

Ключевой особенностью является реактивная архитектура на основе StateFlow и комбинирования потоков для синхронизации состояния.

Листинг 10 – ViewModel с комбинированием потоков (BattleViewModel.kt)

```
class BattleViewModel(
    gameRepository: GameRepository
) : BaseGameViewModel(gameRepository) {
    private var stageTimerJob: Job? = null
                            private         val         _currentEnemy         =
MutableStateFlow(EnemyGenerator.generateEnemy(stage = 1))
    val currentEnemy: StateFlow<Enemy> = _currentEnemy.asStateFlow()

    override fun onPlayerLoaded(player: Player) {
        spawnNewEnemy()
        startStageTimer()
    }

    // Use Cases
    private val attackEnemyUseCase = AttackEnemyUseCase()
    private val stageProgressUseCase = StageProgressUseCase()

    private fun startStageTimer() {
        stageTimerJob?.cancel()
        stageTimerJob = viewModelScope.launch {
            while (isActive) {
                delay(1000)
                decreaseStageTime()
            }
```

```
        }
    }

    private fun decreaseStageTime() {
        val currentPlayer = _playerState.value
        if (currentPlayer.currentTime > 0) {
            val updatedPlayer = currentPlayer.copy(
                currentTime = currentPlayer.currentTime - 1
            )
            _playerState.value = updatedPlayer

            if (updatedPlayer.isDefeated) {
                handlePlayerDefeat()
            }
        }
    }

    private fun spawnNewEnemy() {
        val currentPlayer = _playerState.value
        val isBoss = currentPlayer.isBossFight

        val newEnemy = EnemyGenerator.generateEnemy(
            stage = currentPlayer.currentStage,
            isBoss = isBoss
        )

        _currentEnemy.value = newEnemy
    }

    fun attackEnemy() {
                    val   result   =   attackEnemyUseCase(_playerState.value,
_currentEnemy.value)
        val currentEnemy = _currentEnemy.value

        val updatedPlayer = if (result.isEnemyDefeated) {
                                                val    isKillQuestEnemy    =
QuestGenerator.getQuest(QuestType.KILL_COUNT)?.targetName                    ==
currentEnemy.name

            result.updatedPlayer.copy(
                totalKills = result.updatedPlayer.totalKills + 1,
                totalKillsEnemy = if (isKillQuestEnemy) {
                    result.updatedPlayer.totalKillsEnemy + 1
                } else {
                    result.updatedPlayer.totalKillsEnemy
                }
            )
        } else {
            result.updatedPlayer
        }

        _playerState.value = updatedPlayer
        _currentEnemy.value = result.updatedEnemy

        if (result.isEnemyDefeated) {
            handleEnemyDefeat()
        }

        savePlayerProgress()
    }

    private fun handleEnemyDefeat() {
        val playerAfterProgress = stageProgressUseCase(_playerState.value)
```

```
        _playerState.value = playerAfterProgress

        savePlayerProgress()
        spawnNewEnemy()
    }

    private fun handlePlayerDefeat() {
        val resetPlayer = _playerState.value.copy(
            enemiesDefeated = 0,
            currentTime = _playerState.value.maxTime
        )
        _playerState.value = resetPlayer
        spawnNewEnemy()
    }
}
```

Ключевые архитектурные особенности:

- Реактивность: автоматическое обновление UI при изменении данных
- Разделение ответственности: четкие границы между слоями
- Тестируемость: Use Cases и репозитории легко покрываются тестами
- Масштабируемость: простая добавка новой функциональности

3.3 Тестирование приложения

Для обеспечения надежности ключевой игровой логики разработан комплекс модульных тестов на JUnit. Покрыты основные Use Cases, генераторы контента и системы расчета.

Листинг 11 – Тестирование Use Case атаки (AttackEnemyUseCaseTest.kt)

```
class AttackEnemyUseCaseTest {
    // Создание тестового противника
    private fun createTestEnemy(
        currentHealth: Int = 50,
        maxHealth: Int = 50,
        baseReward: Int = 25
    ): Enemy {
        return Enemy(
            currentHealth = currentHealth,
            maxHealth = maxHealth,
            id = 1,
            name = "Test Enemy",
            baseReward = baseReward,
            imageRes = ""
        )
    }

    // Тест для атаки противника
    @Test
    fun `attack enemy reduces health`() {
        // Given - подготовка данных
        val useCase = AttackEnemyUseCase()
        val player = Player(baseAttack = 10)
        val enemy = createTestEnemy(currentHealth = 50, maxHealth = 50)

        // When - выполнение действия
        val result = useCase(player, enemy)

        // Then - проверка результатов
                                assertTrue("Враг должен получать урон",
result.updatedEnemy.currentHealth < enemy.currentHealth)
            assertEquals("Здоровье должно уменьшаться на атаку игрока", 40,
result.updatedEnemy.currentHealth)
    }

    // Тест для критической атаки
    @Test
    fun `critical attack deals double damage`() {
        val useCase = AttackEnemyUseCase()
        val player = Player(baseAttack = 10, criticalChance = 1.0) // 100% шанс
критического удара
        val enemy = createTestEnemy(currentHealth = 50, maxHealth = 50)

        val result = useCase(player, enemy)

           assertTrue("Должен нанести критический урон", result.damageDealt >
player.baseAttack)
                            assertEquals("Урон должен быть удвоен", 30,
result.updatedEnemy.currentHealth)
```

```
    }

    // Тест что здоровье не может быть ниже нуля
    @Test
    fun `enemy health never goes below zero`() {
        val useCase = AttackEnemyUseCase()
        val player = Player(baseAttack = 100) // Большой урон
        val enemy = createTestEnemy(currentHealth = 10, maxHealth = 50) // Мало
здоровья

        val result = useCase(player, enemy)

                    assertEquals("Здоровье не должно быть ниже нуля", 0,
result.updatedEnemy.currentHealth)
    }

    // Тест на награду за убийство
    @Test
    fun `gold reward when enemy defeated`() {
        val useCase = AttackEnemyUseCase()
        val player = Player(baseAttack = 60)
            val enemy = createTestEnemy(currentHealth = 50, maxHealth = 50,
baseReward = 25)

        val result = useCase(player, enemy)

        assertTrue("Враг должен быть убит", result.isEnemyDefeated)
             assertEquals("Игрок должен получить награду за убийство", 25,
result.goldReward)
    }
}
```

Листинг 12 — Тестирование генераторов контента (EnemyGeneratorTest.kt)

```
class EnemyGeneratorTest {

    @Test
    fun `generate normal enemy for stage`() {
        val enemy = EnemyGenerator.generateEnemy(stage = 1)

        assertTrue("Враг должен иметь здоровье", enemy.maxHealth > 0)
        assertTrue("Враг должен иметь награду", enemy.baseReward > 0)
    }

    @Test
    fun `boss has increased stats`() {
        val normal = EnemyGenerator.generateEnemy(stage = 3)
        val boss = EnemyGenerator.generateEnemy(stage = 3, isBoss = true)

            assertTrue("Босс должен иметь больше здоровья", boss.maxHealth >
normal.maxHealth)
        assertTrue("Босс должен иметь имя", boss.name.contains("Босс"))
    }
}
```

Листинг 13 — Тестирование системы апгрейдов (UpgradeGeneratorTest.kt)

```
class UpgradeGeneratorTest {
    // Тесты на логику генерации апгрейдов
    @Test
    fun `upgrade costs are positive`() {
         val player = Player(baseAttack = 10, criticalChance = 0.1, maxTime =
60)

                  val   attackCost   =   UpgradeGenerator.getUpgradeCost(player,
SkillType.ATTACK)
                   val   critCost   =   UpgradeGenerator.getUpgradeCost(player,
SkillType.CRITICAL)
        val timeCost = UpgradeGenerator.getUpgradeCost(player, SkillType.TIME)

          assertTrue("Все цены должны быть положительными", attackCost > 0 &&
critCost > 0 && timeCost > 0)
    }

    // Тест на логику апгрейдов атаки
    @Test
    fun `upgrade attack logic works`() {
        val player = Player(gold = 1000, baseAttack = 10)
        val cost = UpgradeGenerator.getUpgradeCost(player, SkillType.ATTACK)

        // Проверяем что хватает денег и статы увеличиваются
        assertTrue("Денег должно хватать", cost <= player.gold)
            assertTrue("Цена должна быть в диапазоне от 1 до 1000", cost in
1..1000)
    }
}
```

Листинг 14 — Тестирование квестовой системы (QuestGeneratorTest.kt)

```
class QuestGeneratorTest {

    // Тестируем генерацию квестов для всех типов
    @Test
    fun `generate quests for all types`() {
        val player = Player(currentStage = 5, totalKills = 100, totalGoldEarned
= 1000)

        val quests = QuestGenerator.getQuests(player)

          assertEquals("Должно быть 5 типов квестов", QuestType.entries.size,
quests.size)

        quests.forEach { quest ->
                assertTrue("Квест должен иметь положительное значение цели",
quest.targetValue > 0)
             assertTrue("Квест должен иметь положительное значение награды",
quest.reward > 0)
        }
    }

    // Тестируем расчет прогресса для каждого типа квеста
    @Test
    fun `quest progress calculation`() {
        val player = Player(
```

```
            currentStage = 3,
            totalKillsEnemy = 15,
            gold = 500,
            totalKills = 25,
            upgradeSkills = 2
        )

                                    assertEquals("Прогресс убийств", 15,
QuestGenerator.getQuestProgress(player, QuestType.KILL_COUNT))
                                     assertEquals("Прогресс этапов", 3,
QuestGenerator.getQuestProgress(player, QuestType.STAGE_PROGRESS))
                            assertEquals("Прогресс золота собрано", 500,
QuestGenerator.getQuestProgress(player, QuestType.GOLD_EARN))
                             assertEquals("Прогресс убийств всего", 25,
QuestGenerator.getQuestProgress(player, QuestType.TOTAL_KILLS))
                                   assertEquals("Прогресс улучшений", 2,
QuestGenerator.getQuestProgress(player, QuestType.UPGRADE_SKILLS))
    }
}
```

Результаты тестирования:

- Покрытие ключевой логики: боевая система, генерация врагов, квесты, прокачка
- Проверка граничных условий: здоровье не уходит ниже нуля, корректность расчетов
- Изолированность тестов: каждый тест проверяет одну конкретную функциональность
- 100% успешное выполнение: все тесты проходят, подтверждая корректность игровой механики

3.4 Документирование кода

Код проекта документирован через inline-комментарии, объясняющие назначение классов и сложную бизнес-логику. Комментарии добавлены для ключевых методов генерации контента, расчета игровых параметров и системы квестов.

Листинг 15 — Документация Use Case атаки (AttackEnemyUseCase.kt)

```
// UseCase для атаки врага
class AttackEnemyUseCase {
    // Функция для атаки врага
    operator fun invoke(player: Player, enemy: Enemy): AttackResult {
        // Возвращаем результат атаки
        return AttackResult()
     }
}
```

Листинг 16 — Документация системы квестов (QuestGenerator.kt)

```
// Класс для генерации квестов
object QuestGenerator {
    private var generatedQuests: List<Quest> = emptyList()

    // Возвращает список сгенерированных квестов
    fun getQuests(player: Player): List<Quest> {}

    // Возвращает текущий квест по типу
    fun getQuest(questType: QuestType): Quest? {}

    // Возвращает прогресс выполнения квеста
    fun getQuestProgress(player: Player, questType: QuestType): Int {}

    // Генерирует новый квест по типу
    private fun generateQuest(player: Player, questType: QuestType): Quest {
        // Возвращает сгенерированный квест
        return Quest()}

    // Заменяет текущий квест на новый
    fun replaceQuest(player: Player, questType: QuestType) {}

    // Восстанавливает список квестов
    fun restoreQuests(quests: List<Quest>) {}
}
```

Листинг 17 — Документация архитектурных компонентов (BattleViewModel.kt)

```
// ViewModel для боя
class BattleViewModel(
    gameRepository: GameRepository
) : BaseGameViewModel(gameRepository) {
    private var stageTimerJob: Job? = null
                private              val              _currentEnemy              =
MutableStateFlow(EnemyGenerator.generateEnemy(stage = 1))
    val currentEnemy: StateFlow<Enemy> = _currentEnemy.asStateFlow()

    // Загрузка игрока
    override fun onPlayerLoaded(player: Player) {}

    // Use Cases
    private val attackEnemyUseCase = AttackEnemyUseCase()
    private val stageProgressUseCase = StageProgressUseCase()

    // Таймер для стадии
    private fun startStageTimer() {}

    // Уменьшение времени на стадии
```

```
    private fun decreaseStageTime() {}

    // Спавн нового врага
    private fun spawnNewEnemy() {}

    // Атака врага
    fun attackEnemy() {}

    // Обработка победы врага
    private fun handleEnemyDefeat() {}

    // Обработка поражения игрока
    private fun handlePlayerDefeat() {}
}
```

ЗАКЛЮЧЕНИЕ

В ходе проектной работы была успешно разработана мобильная игра "Fantasy Quest Clicker" для Android. Поставленные цели достигнуты: реализован интуитивный интерфейс на Jetpack Compose, сбалансированная боевая система с тактической глубиной и надежное сохранение прогресса.

Ключевые результаты:

- Чистая архитектура на основе Clean Architecture с разделением на Data, Domain и UI слои
- Реактивная система состояния с использованием StateFlow и Combine для автоматического обновления UI
- Полноценная игровая механика: боевая система с критическими ударами, 5 типов постоянных квестов, прокачка характеристик
- Профессиональная кодовая база: модульные тесты покрывают ключевую логику, код документирован комментариями

Технические достижения:

- Jetpack Compose для современного декларативного UI
- Android DataStore для надежного сохранения прогресса
- Kotlin Coroutines и Flow для асинхронных операций
- Система генерации контента: враги, квесты, апгрейды

Практическая работа над проектом позволила углубить навыки современной Android-разработки, включая реактивное программирование, чистую архитектуру и тестирование.

Ссылка на GitHub-репозиторий разработанного проекта: https://github.com/Mekito4u/fantasy-quest-clicker-android

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. ГОСТ 34.602—2020 ("Техническое задание на создание автоматизированной системы") является стандартом, который определяет общие требования к разработке автоматизированных систем (АС). Он используется в разных отраслях, включая медицину, и применим для разработки автоматизированных информационных систем (АИС).
2. ГОСТ 34.201—2020. Межгосударственный стандарт. Информационные технологии. Комплекс стандартов на автоматизированные системы. Виды, комплектность и обозначение документов при создании автоматизированных систем: Приказом Федерального агентства по техническому регулированию и метрологии № 1521-ст от 19 ноября 2021 г.: дата введения 2022-01-01. – М.: Российский институт стандартизации, 2021. – 10 с.
3. Блоштин А.В., Лаптев Д.В. Современные подходы к проектированию клиент-серверных приложений // Программные системы: теория и приложения. 2021. №2. URL: https://elibrary.ru/item.asp?id=46523678 (дата обращения: 26.11.2025)
4. Android Developers — официальный сайт документации по разработке на Kotlin: https://developer.android.com/kotlin (дата обращения: 26.11.2025).
5. Kotlin Documentation — полный справочник по языку Котлин: https://kotlinlang.org/docs (дата обращения: 26.11.2025).
6. Жемеров С., Исакова С. Kotlin в действии. 2-е издание. М.: Питер, 2022. URL: https://www.piter.com/product/kotlin-v-dejstvii (дата обращения: 26.11.2025).
7. Документация JUnit — для тестирования Kotlin-приложений: https://junit.org/junit5 (дата обращения: 26.11.2025).