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»

Студенты группы ИКБО-50-23 Петруничев А.А.

_____

(подпись студента)

Руководитель проектной работы Золотухин С.А.

_____

(подпись руководителя)

Работа представлена «____»_____ 2025 г.

Допущен к работе «___» _____ 2025 г.

Москва 2025

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В процессе обучения может возникать необходимость организации интерактивов с группой. Проект «Своя игра» на языке Kotlin направлен на разработку платформы, обеспечивающей функциональность запуска, функционирования, создания и редактирования игр, созданных или скачанных. Основной целью является создание простого в использовании приложения с высокой производительностью, понятным интерфейсом и возможностью быстро создавать «свои игры» быстро и без затруднений.

Необходимо реализовать систему, работающую по раундам, в каждом из которых могут быть свои правила. Также требуется создать обработку правильных/неправильных ответов и действий, следующих за ними. Для отслеживания лидеров игры нужна система подсчёта очков, учитывающая всех условия: раунд, стоимость правильного ответа, тип вопроса.

При выполнении задания на проект необходимо сделать простой редактор игр, позволяющий настраивать количество раундов, тем, вопросов, количество участников, стоимость правильных ответов, создавать и изменять вопросы, указывать правильные ответы, вести подсчет очков. Реализация функции сохранения созданной игры в файл ускорит и упростит создание новой игры.

Проект актуален, как демонстрация владения современным языком Kotlin, позволяющим писать производительные кроссплатформенные приложения. В проекте также будут использованы современные архитектурные паттерны, такие как MVVM или MVI.

Готовое приложение можно адаптировать для использования в учебных заведениях для проведения викторин, в компаниях для корпоративных мероприятий или просто для игры с друзьями.

Для достижения цели необходимо выполнить следующие задачи:

- Задача 1: Разработка интерфейса,
- Задача 2: Разработка игровой логики,

- Задача 3: Разработка редактора игр,
- Задача 4: Создание системы сохранения игр.

Состав команды и распределение ролей:

- Петруничев А.А.: Разработчик, Менеджер проекта
- Лысачева М.М.: Разработчик, Тестировщик

1. ПЛАНИРОВАНИЕ И ПОДГОТОВКА

1.1 Техническое задание

1.1.1 Общие сведения

Полное наименование: Система для создания «Своей игры».

Краткое наименование: Система.

1.1.2 Цели и назначение автоматизированной системы

Назначение: Система предназначена для обеспечения простого способа реализации интерактива в формате «Своей игры».

Цель: создать Систему, упрощающую и ускоряющую процесс создания и использования «Своей игры».

1.1.3 Характеристика объектов автоматизации

Объектом автоматизации Системы является процесс «Реализация Своей игры». Данный процесс подразумевает:

- Разработку банка вопросов на заданное количество различных тем;
- Подготовку ответов на составленные вопросы;
- Распределение вопросов по стоимости правильного ответа в соответствии со сложностью вопроса;
- Визуальную реализацию в виде таблицы;
- Реализация интерактивного интерфейса: открытие выбранного вопроса, раскрытие правильного ответа, возврат к главной странице, отображение пройденных и не пройденных вопросов;
- Подсчет очков.

1.1.4 Требования к системе

Система должна обеспечивать следующий функционал:

- Создание игры:
  - Выбор количества игроков
  - Выбор количества раундов
  - Выбор количества тем
  - Выбор количества вопросов
  - Создание банка вопросов
  - Настройка правильных ответов
  - Настройка системы подсчета очков
- Использование игры:
  - Настройка количества экранов
  - Запуск игры с отображением правил
  - Объявление раунда и его тем
  - Табличное представление тем и количества вопросов с указанием стоимости правильного ответа
  - Переход на выбранный вопрос с отображением его формулировки
  - Демонстрация правильного ответа
  - Увеличение количества очков ответившим правильно
  - Возврат к списку тем и вопросов
  - Различное отображение пройденных и не пройденных вопросов
  - Переход к следующему раунду
  - Демонстрация результатов

1.1.5 Состав и содержание работ

Разработка Системы осуществляется поэтапно в течение семестра в соответствии с заданием, выданным преподавателем, в установленные им сроки.

1.1.6 Порядок разработки и приемки

Разработка Системы выполняется в соответствии с описанием процесса «Создания Своей игры» в сроки, установленные преподавателем для каждого этапа.

По окончании разработки предоставляется следующая документация:

– Техническое задание на разработку;

– Исходный код;

– Руководство пользователя.

Прием проекта осуществляется преподавателем.

1.1.7 Требования к подготовке объекта к вводу системы в действие

Ввод информации в Систему осуществляется в электронном виде.

1.1.8 Требования к документированию

В состав отчетной технической документации входят:

– Техническое задание на разработку;

– Исходный код;

– Руководство пользователя.

Документация к проекту составляется на русском языке.

1.1.9 Источники разработки

Для разработки проекта используются официальная документация JetBrains Compose Multiplatform и Android Jetpack Compose (описание UI-компонентов, систем расположения, обработка ввода и управление состоянием), а также примеры из репозиториев GitHub по реализации игровых интерфейсов и табло счёта. В качестве справочного материала применяются руководства по Kotlin и Gradle из официальных источников JetBrains и Gradle, а также статьи и обсуждения на Stack Overflow, посвящённые построению адаптивных интерфейсов. Все заимствованные идеи и решения адаптируются

под структуру проекта SvoyaIgra и стилистику кода, используемую разработчиком.

Помимо вышеперечисленного, производилась периодическая консультация с ChatGPT 5.1 для уточнения деталей реализации конкретного функционала.

1.2 Диаграмма вариантов использования

1.2.1 Обоснование сценариев использования

Пользователь создал свою игру.

Пользователь запустил игру.

Пользователь проводит игру в качестве ведущего.

Пользователь участвует в игре как игрок.

Пользователь решил изменить созданную им до этого игру.

Пользователь хочет передать созданную им игру другу или переместить на другое устройство.

1.2.2 Схематическое представление (Use Case Diagram)

Включает в себя актеров (Пользователь (игрок/ведущий), Система) и сценарии (сценарии перечислены выше, в п. 1.2.1)

Результат создания диаграммы прецедентов представлен на Рисунке 1.

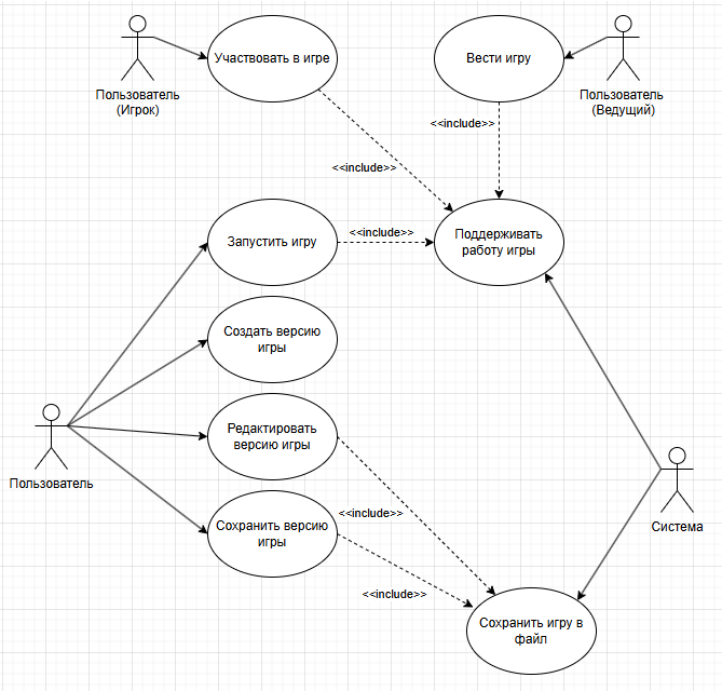

Рисунок 1 — Диаграмма вариантов использования разрабатываемого проекта

1.3 Создание репозитория

На платформе GitHub был создан открытый репозиторий. В участников были добавлены весь состав группы разработки. Результат создания репозитория разрабатываемого проекта представлен на Рисунке 2.

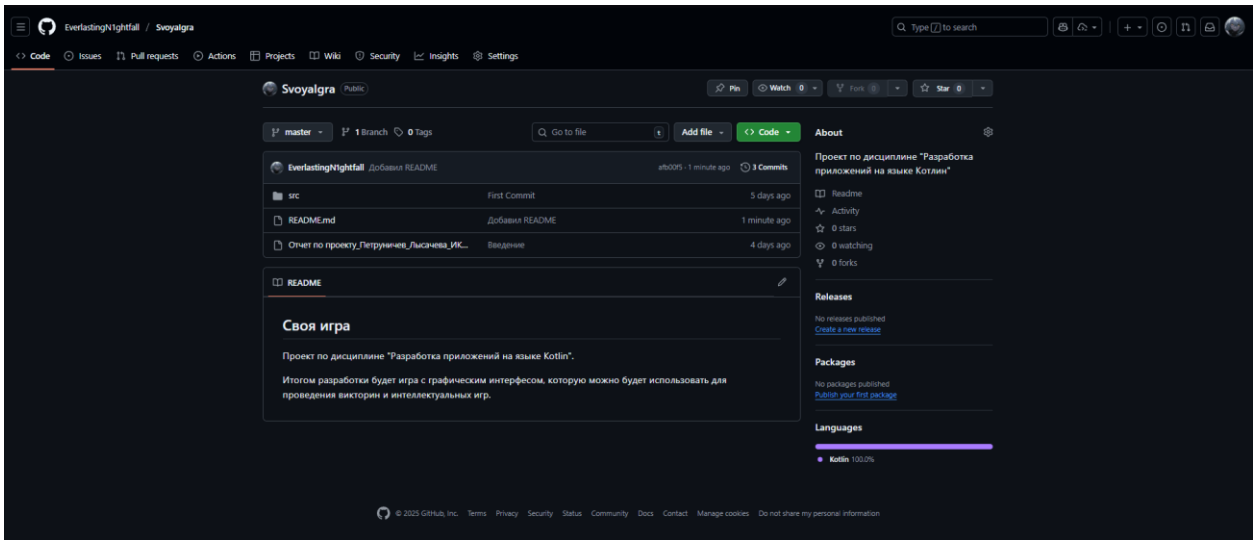

Рисунок 2 — Результат создания репозитория проекта

Ссылка на репозиторий: https://github.com/EverlastingN1ghtfall/SvoyaIgra

2. ПРОЕКТИРОВАНИЕ ИНТЕРФЕЙСА

2.1 Макеты экранов

Большая часть интерфейсов будет очень схожа с экранами в самом шоу «Своя игра».

Макет экрана выбора вопросов показан ниже (рис. 3). Помимо него, также был разработан макет экрана отображения выбранного вопроса (рис. 4).

Рисунок 3 — Результат создания макета экрана страницы «Выбор вопроса»

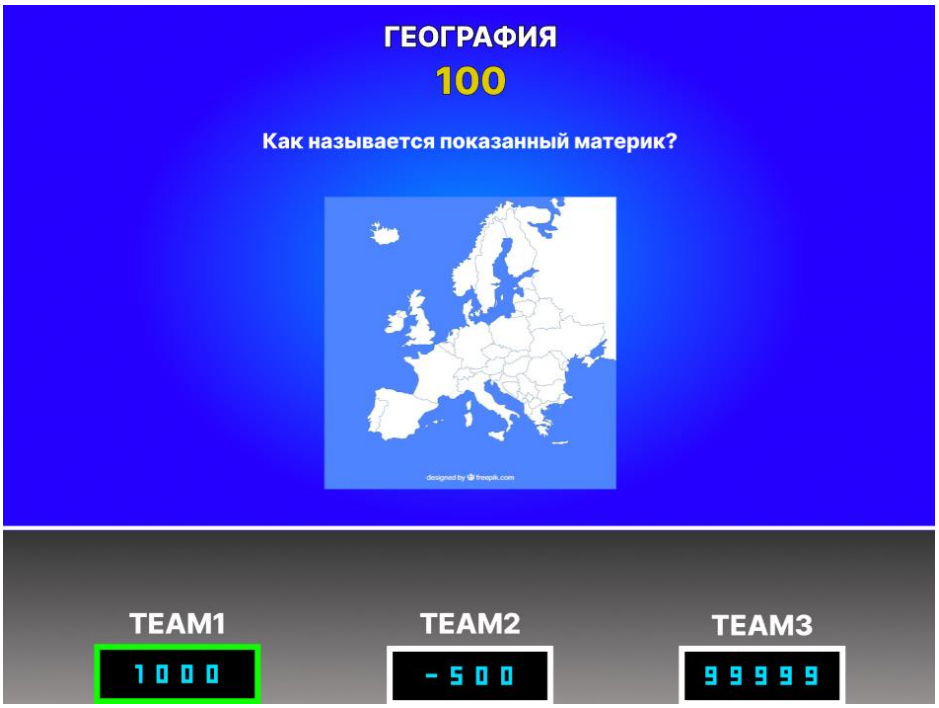

Рисунок 4 – Результат создания макета «Выбранный вопрос»

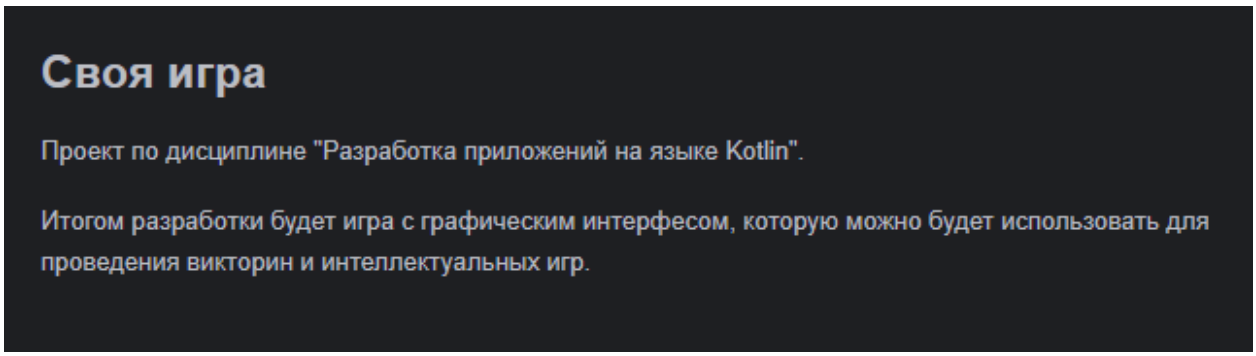

Рисунок 5 – Файл README.md

```
plugins {
    // this is necessary to avoid the plugins to be loaded multiple times
    // in each subproject's classloader
    alias(libs.plugins.composeHotReload) apply false
    alias(libs.plugins.composeMultiplatform) apply false
    alias(libs.plugins.composeCompiler) apply false
    alias(libs.plugins.kotlinMultiplatform) apply false
}
```

Рисунок 6 – Файл build.gradle.kts

3. РАЗРАБОТКА ПРИЛОЖЕНИЯ

3.1 Архитектура приложения

Архитектура приложения может быть построена на основе паттерна MVVM: данные и бизнес-логика хранятся во ViewModel, а UI реализован через Composable-функции Jetpack Compose. Состояния экрана управляются через remember и mutableStateOf, что обеспечивает реактивность интерфейса. Для навигации между экранами можно использовать собственные флаги состояния или интегрировать Compose Navigation. Данные (например, вопросы и темы) могут храниться в репозитории, который взаимодействует с ViewModel.

3.2 Реализация ключевых компонентов

3.2.1 Frontend

Для реализации клиентской части проекта был выбран Kotlin Multiplatform Compose.

Ниже показаны этапы разработки основного экрана.

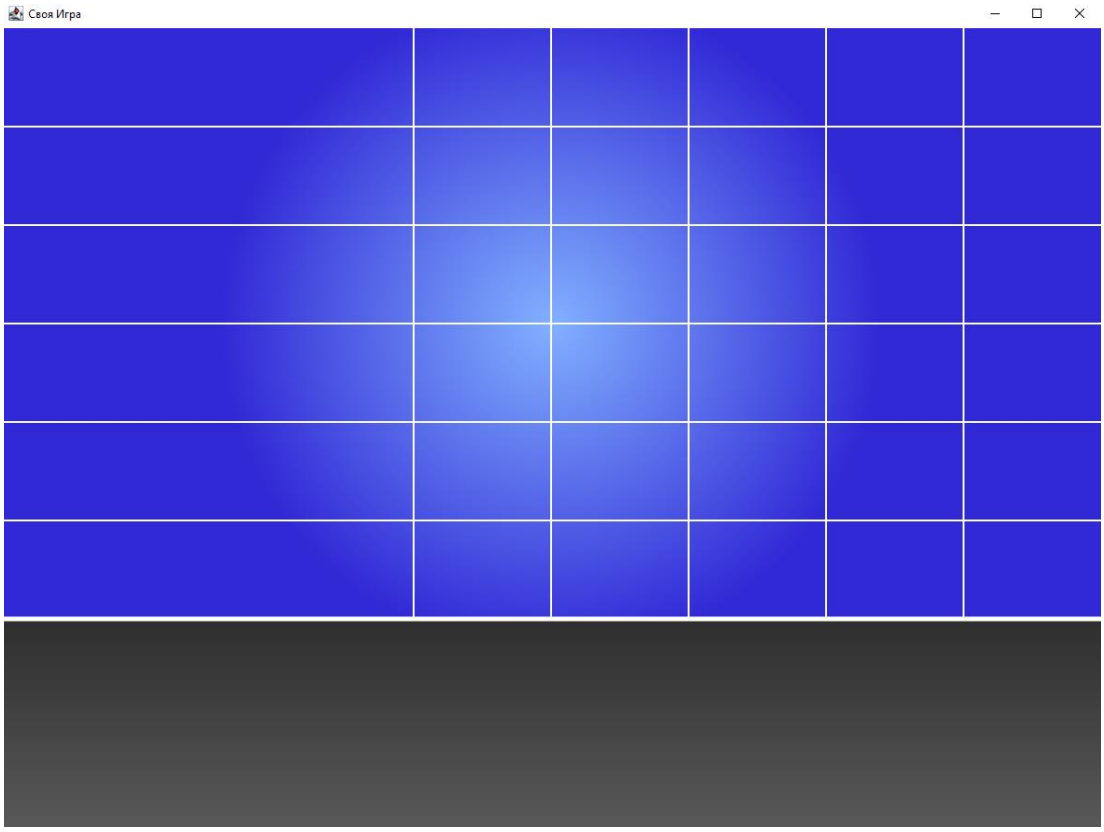

Рисунок 7 – Фон интерфейса и разлиновка для клеток с вопросами

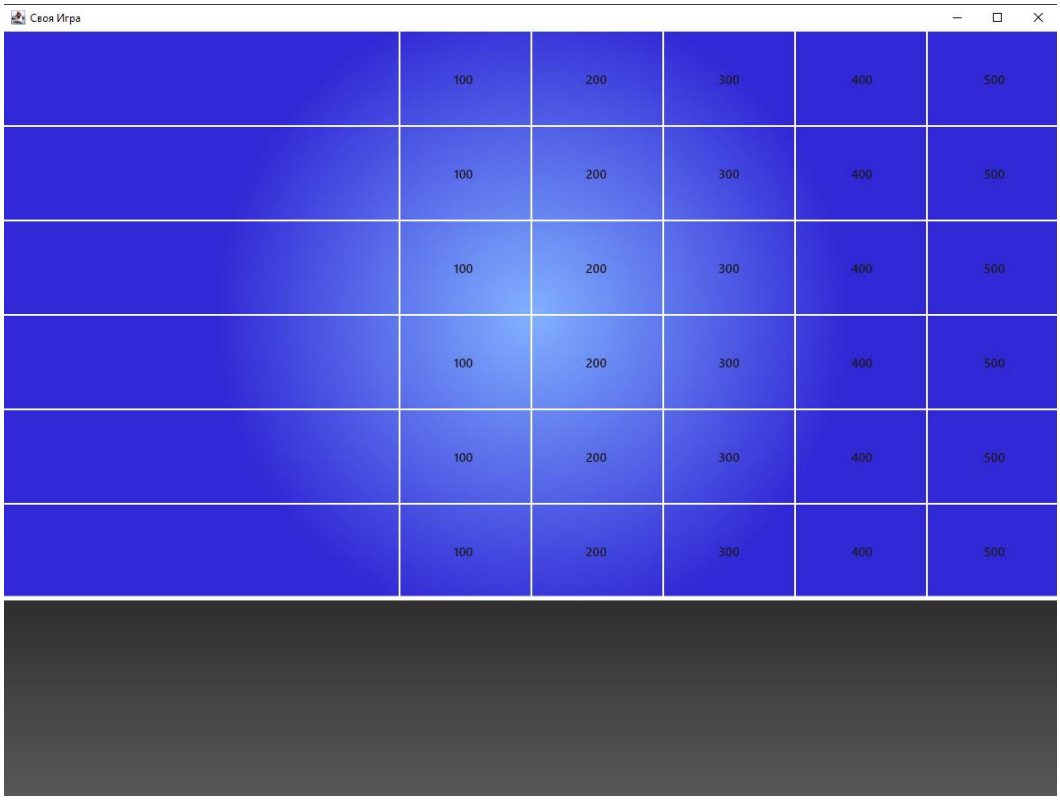

Рисунок 8 – Сетка кнопок с вопросами

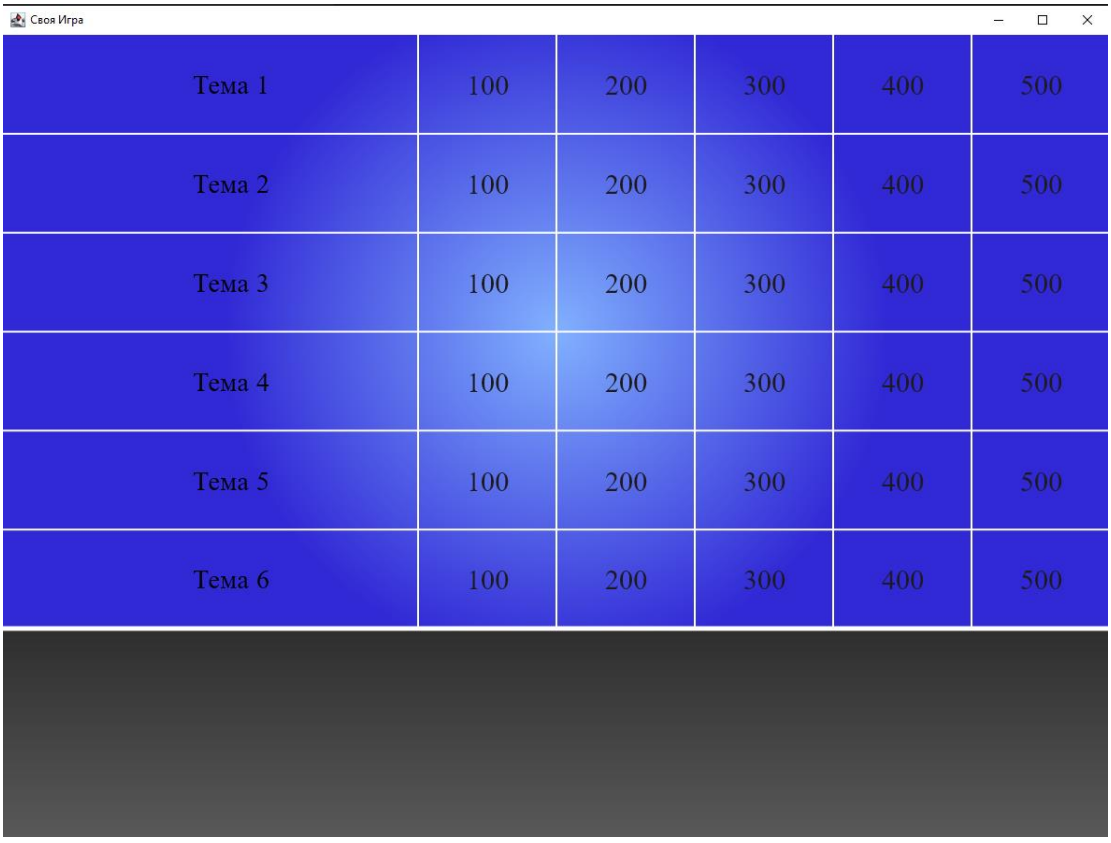

Рисунок 9 – Прототип интерфейса выбора вопроса

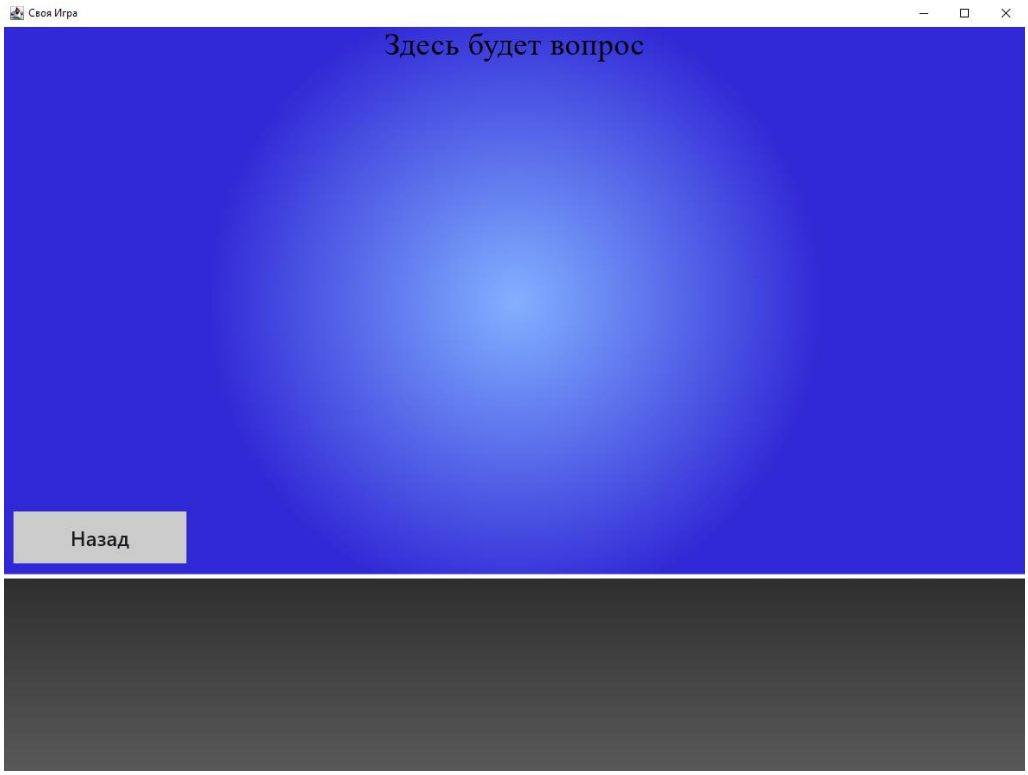

Рисунок 10 – Примерный макет экрана выбранного вопроса

Далее был реализован интерфейс подсчёта очков, который показан на рисунке ниже. В нём предусмотрено динамическое управление количеством очков у каждой команды.

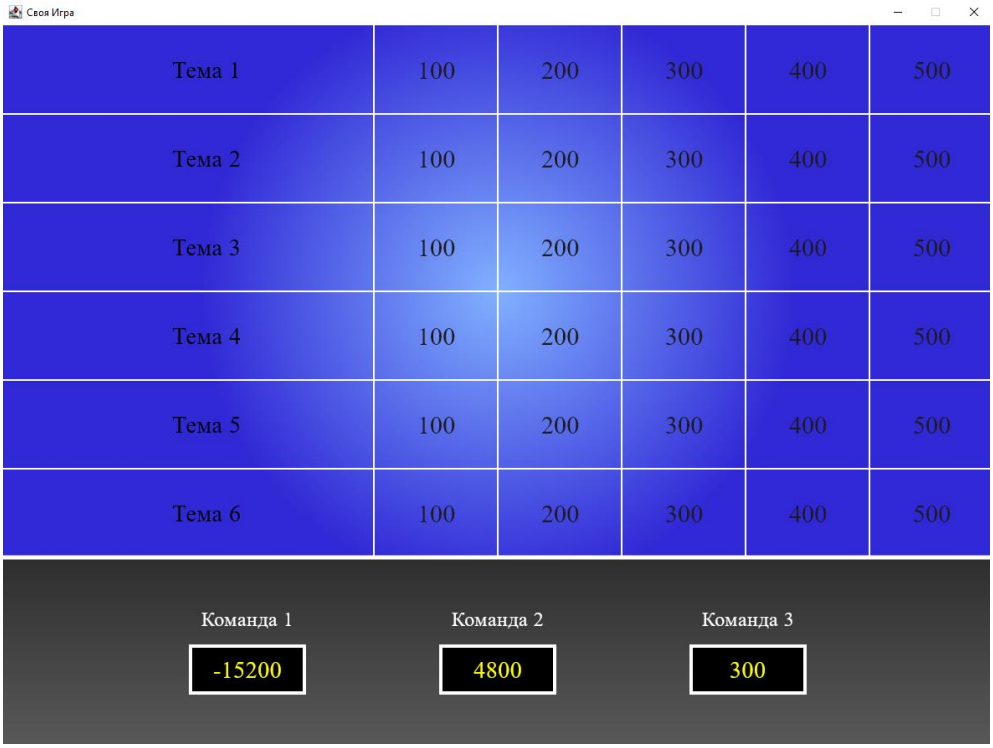

Рисунок 11 – Реализованный подсчёт очков

Далее был реализован механизм, замены кнопок на спейсеры при их выборе, чтобы один и тот же вопрос нельзя было выбрать заново.

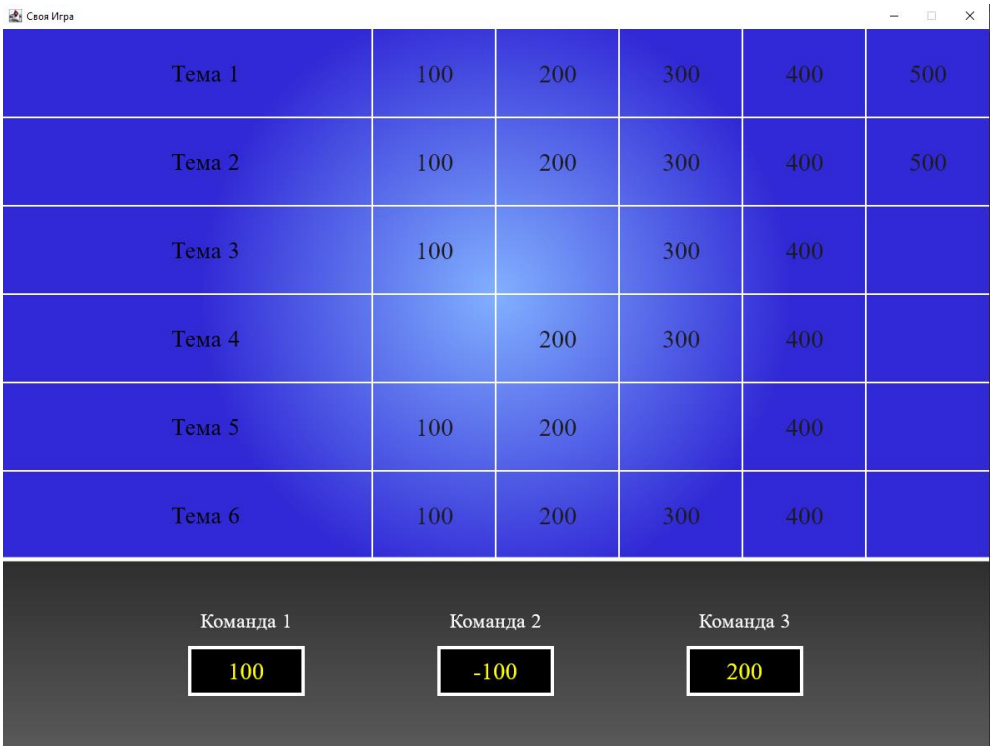

Рисунок 12 – Исчезающие вопросы после выбора

Реализованный интерфейс основного экрана выбора вопросов представлен в Листинге 1.

Листинг 1 – Листинг кода экрана «Своей игры»

```
package org.example.svoyaigra

import androidx.compose.foundation.border
import androidx.compose.foundation.layout.*
import androidx.compose.material3.Button
import androidx.compose.material3.ButtonDefaults
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Text
import androidx.compose.runtime.*
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.drawBehind
import androidx.compose.ui.geometry.Offset
import androidx.compose.ui.graphics.Brush
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.unit.Dp
import androidx.compose.ui.unit.dp
import org.jetbrains.compose.ui.tooling.preview.Preview
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.ui.input.pointer.isPrimaryPressed
import androidx.compose.ui.input.pointer.isSecondaryPressed
import androidx.compose.ui.input.pointer.pointerInput
```

```
import androidx.compose.ui.unit.sp
import androidx.compose.ui.text.font.FontFamily

import kotlin.math.max


@Composable
fun SimpleBackgroundScreen() {
    val darkBlue = Color(0xFF3129D6)
    val lightBlue = Color(0xFF82B1FF)
    val darkGray = Color(0xFF2E2E2E)
    val lightGray = Color(0xFF595959)

    Box(
        modifier = Modifier
            .fillMaxSize()
            .drawBehind {
                val center = Offset(size.width / 2f, (size.height -
225f) / 2f)
                val radius = max(size.width, size.height) * 0.3f
                drawRect(
                    brush = Brush.radialGradient(
                        colors = listOf(lightBlue, darkBlue),
                        center = center,
                        radius = radius
                    )
                )
                val rectHeight = 225f
                val top = size.height - rectHeight
                drawRect(
                    brush = Brush.verticalGradient(
                        colors = listOf(darkGray, lightGray),
                        startY = top,
                        endY = size.height
                    ),
                    topLeft = Offset(0f, top),
                    size =
androidx.compose.ui.geometry.Size(size.width, rectHeight)
                )
                drawLine(
                    color = Color.White,
                    strokeWidth = 5f,
                    start = Offset(0f, top),
                    end = Offset(size.width, top)
                )
            }
            .safeContentPadding()
    )
}


@Composable
fun QuestionLineGrid() {
    Box(modifier = Modifier.fillMaxSize().drawBehind {
        val rectHeight = 225f
        val verticalMargin = (size.height - rectHeight) / 6f
        drawLine(
```

```
            color = Color.White,
            strokeWidth = 2f,
            start = Offset(0f, verticalMargin * 1f),
            end = Offset(size.width, verticalMargin * 1f)
        )
        drawLine(
            color = Color.White,
            strokeWidth = 2f,
            start = Offset(0f, verticalMargin * 2f),
            end = Offset(size.width, verticalMargin * 2f)
        )
        drawLine(
            color = Color.White,
            strokeWidth = 2f,
            start = Offset(0f, verticalMargin * 3f),
            end = Offset(size.width, verticalMargin * 3f)
        )
        drawLine(
            color = Color.White,
            strokeWidth = 2f,
            start = Offset(0f, verticalMargin * 4f),
            end = Offset(size.width, verticalMargin * 4f)
        )
        drawLine(
            color = Color.White,
            strokeWidth = 2f,
            start = Offset(0f, verticalMargin * 5f),
            end = Offset(size.width, verticalMargin * 5f)
        )

        val horizontalMargin = size.width / 8f
        drawLine(
            color = Color.White,
            strokeWidth = 2f,
            start = Offset(horizontalMargin * 3f, 0f),
            end = Offset(horizontalMargin * 3f, size.height -
rectHeight)
        )
        drawLine(
            color = Color.White,
            strokeWidth = 2f,
            start = Offset(horizontalMargin * 4f, 0f),
            end = Offset(horizontalMargin * 4f, size.height -
rectHeight)
        )
        drawLine(
            color = Color.White,
            strokeWidth = 2f,
            start = Offset(horizontalMargin * 5f, 0f),
            end = Offset(horizontalMargin * 5f, size.height -
rectHeight)
        )
        drawLine(
            color = Color.White,
            strokeWidth = 2f,
            start = Offset(horizontalMargin * 6f, 0f),
```

```
            end = Offset(horizontalMargin * 6f, size.height -
rectHeight)
        )
        drawLine(
            color = Color.White,
            strokeWidth = 2f,
            start = Offset(horizontalMargin * 7f, 0f),
            end = Offset(horizontalMargin * 7f, size.height -
rectHeight)
        )
    })
}


@Composable
fun ButtonGridWithLabels(
    topics: List<String>,
    rows: Int = 6,
    cols: Int = 5,
    buttonWidth: Dp = 148.dp,
    buttonHeight: Dp = 106.dp,
    labelWidth: Dp = 445.dp,
    answered: List<Boolean>,
    onClick: (row: Int, col: Int, index: Int) -> Unit = { _, _, _ ->
}
) {
    Column(
        modifier = Modifier
            .fillMaxSize()
            .padding(bottom = 225.dp)
    ) {
        for (r in 0 until rows) {
            Row(
                modifier = Modifier
                    .fillMaxWidth()
                    .height(buttonHeight),
                verticalAlignment = Alignment.CenterVertically
            ) {
                Text(
                    text = topics.getOrNull(r) ?: "",
                    modifier = Modifier
                        .width(labelWidth)
                        .padding(start = (labelWidth / 2) - 20.dp),
                    fontSize = 28.sp,
                    fontFamily = FontFamily.Serif,
                    color = Color.Black
                )
                for (c in 0 until cols) {
                    val index = r * cols + c
                    if (answered.getOrNull(index) == true) {
                        // уже отвеченная клетка: делаем её невидимой
и некликабельной
                        Spacer(
                            modifier = Modifier
                                .width(buttonWidth)
                                .height(buttonHeight)
                        )
```

```
                    } else {
                        Button(
                            onClick = { onClick(r, c, index) },
                            modifier = Modifier
                                .width(buttonWidth)
                                .height(buttonHeight),
                            shape = RoundedCornerShape(0.dp),
                            colors = ButtonDefaults.buttonColors(
                                containerColor = Color.Transparent,
                                contentColor =
MaterialTheme.colorScheme.onBackground
                            ),
                            contentPadding = PaddingValues(0.dp)
                        ) {
                            Text(
                                "${100 * (c + 1)}",
                                fontSize = 30.sp,
                                fontFamily = FontFamily.Serif
                            )
                        }
                    }
                }
            }
        }
    }
}


@Composable
fun RenderQuestion() {
    Box(modifier = Modifier.fillMaxSize(), ) {
        Text(
            text = "Здесь будет вопрос", // PLACEHOLDER
            fontSize = 36.sp,
            fontFamily = FontFamily.Serif,
            color = Color.Black,
            modifier = Modifier.align(Alignment.TopCenter)
        )
    }
}


@Composable
fun TeamScoreBar(
    modifier: Modifier = Modifier,
    teamNames: List<String> = listOf("Команда 1", "Команда 2",
"Команда 3"),
    teamScores: List<Int> = listOf(0, 0, 0),
    onScoreIncrease: (teamIndex: Int) -> Unit = {},
    onScoreDecrease: (teamIndex: Int) -> Unit = {}
) {
    Box(
        modifier = modifier
            .fillMaxWidth()
            .height(225.dp),
        contentAlignment = Alignment.Center
    ) {
```

```
        Row(
            modifier = Modifier
                .fillMaxWidth()
                .padding(horizontal = 64.dp),
            horizontalArrangement = Arrangement.SpaceEvenly,
            verticalAlignment = Alignment.CenterVertically
        ) {
            for (i in 0 until 3) {
                Column(
                    horizontalAlignment =
Alignment.CenterHorizontally
                ) {
                    Text(
                        text = teamNames.getOrNull(i) ?: "",
                        fontSize = 24.sp,
                        fontFamily = FontFamily.Serif,
                        color = Color.White
                    )

                    Spacer(modifier = Modifier.height(16.dp))

                    Box(
                        modifier = Modifier
                            .width(140.dp)
                            .height(60.dp)
                            .border(
                                width = 4.dp,
                                color = Color.White,
                                shape = RoundedCornerShape(0.dp)
                            )
                            .padding(4.dp)
                            .drawBehind { drawRect(Color.Black) }
                            .pointerInput(Unit) {
                                awaitPointerEventScope {
                                    while (true) {
                                        val event =
awaitPointerEvent()
                                        val button = event.buttons
                                        if (button.isPrimaryPressed)
{
                                            onScoreIncrease(i)
                                        } else if
(button.isSecondaryPressed) {
                                            onScoreDecrease(i)
                                        }
                                    }
                                }
                            },
                        contentAlignment = Alignment.Center
                    ) {
                        Text(
                            text = "${teamScores.getOrNull(i) ?: 0}",
                            fontSize = 28.sp,
                            fontFamily = FontFamily.Serif,
                            color = Color.Yellow
                        )
                    }
```

```
                }
            }
        }
    }
}


@Composable
@Preview
fun QuestionSelectionScreen() {
    MaterialTheme {
        var showQuestion by remember { mutableStateOf(false) }
        var teamScores by remember { mutableStateOf(listOf(0, 0, 0))
}
        val rows = 6
        val cols = 5
        var answered by remember { mutableStateOf(List(rows * cols) {
false }) }

        Box(modifier = Modifier.fillMaxSize()){
            SimpleBackgroundScreen()
            if (showQuestion) {
                Box(
                    modifier = Modifier.fillMaxSize(),
                    contentAlignment = Alignment.BottomCenter
                ) {
                    RenderQuestion()
                    Button(
                        onClick = { showQuestion = false },
                        modifier = Modifier
                            .absoluteOffset(x = (-480).dp, y = (-
240).dp)
                            .width(200.dp)
                            .height(60.dp)
                            .border(width = 2.dp, color =
Color.Black, shape = RoundedCornerShape(0.dp)),

                        colors = ButtonDefaults.buttonColors(
                            containerColor = Color.LightGray,
                            contentColor =
MaterialTheme.colorScheme.onBackground
                        ),
                        shape = RoundedCornerShape(0.dp)
                    ) {
                        Text("Назад", fontSize = 24.sp)
                    }
                }
            } else {
                QuestionLineGrid()
                val topics = listOf("Тема 1", "Тема 2", "Тема 3",
"Тема 4", "Тема 5", "Тема 6")
                ButtonGridWithLabels(
                    topics = topics,
                    rows = rows,
                    cols = cols,
                    answered = answered,
                    onClick = { row, col, index ->
```

```
                        // показываем вопрос
                        showQuestion = true
                        // помечаем клетку как отвеченную
                        answered = answered.toMutableList().also {
list ->
                            list[index] = true
                        }
                        // здесь же можно запомнить, за сколько очков
был вопрос и кому их дать
                    }
                )
            }
            TeamScoreBar(
                modifier = Modifier.align(Alignment.BottomCenter),
                teamScores = teamScores,
                onScoreIncrease = { index ->
                    teamScores = teamScores.toMutableList().also {
list ->
                        list[index] = list[index] + 100
                    }
                },
                onScoreDecrease = { index ->
                    teamScores = teamScores.toMutableList().also {
list ->
                        list[index] = list[index] - 100
                    }
                }
            )
        }
    }
}
```

3.2.2 Особенности реализации отдельных компонентов (опционально)

Для того, чтобы сетка кнопок с вопросами генерировалась сама, была сделана функция, которая генерирует сетку размера 6*5, состоящую из кликабельнных кнопок.

Управление очками каждой команды осуществляется следующим образом: при нажатии на левую кнопку мыши при наведении на соответствующую команду, количество очков увеличивается на 100. Аналогично, при нажатии на правую кнопку мыши, количество очков уменьшается на 100.

3.3 Тестирование приложения

Проводилось ручное тестирование, в результате которого была выявлена следующая некритическая аномалия: иногда при увеличении количества очков у команды, нажатие регистрируется несколько раз, что вынуждает ведущего внимательнее следить за количеством очков у команд.

Результат тестирования показан на рисунке ниже.

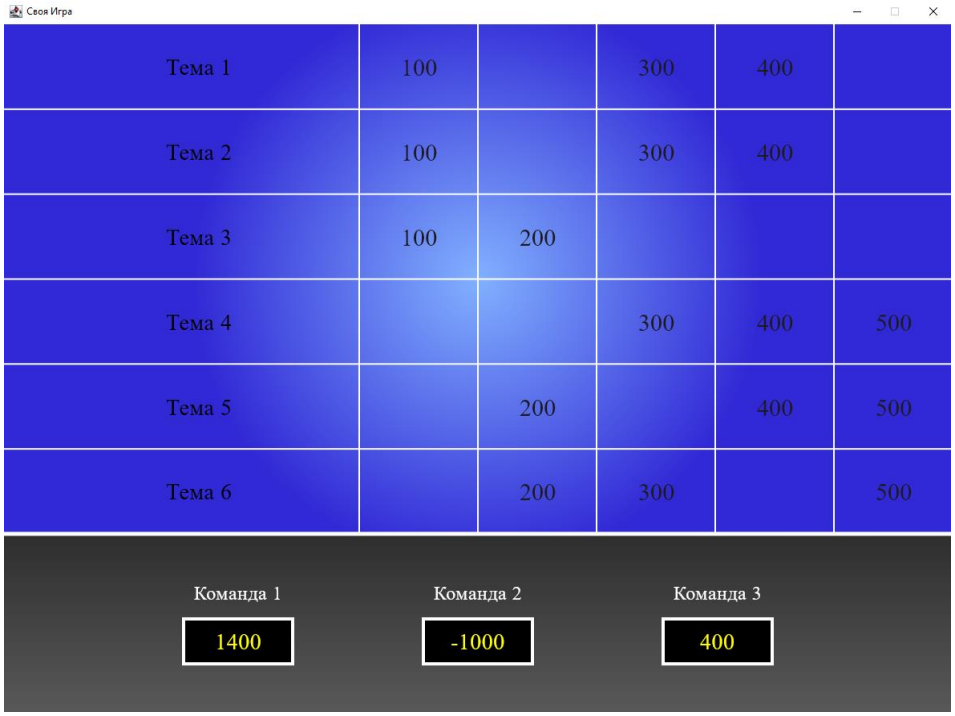

Рисунок 13 – Пример результата тестирования

3.4 Документирование кода

Документирование кода выполнялось с помощью KDoc. Сгенерированная документация показана ниже.

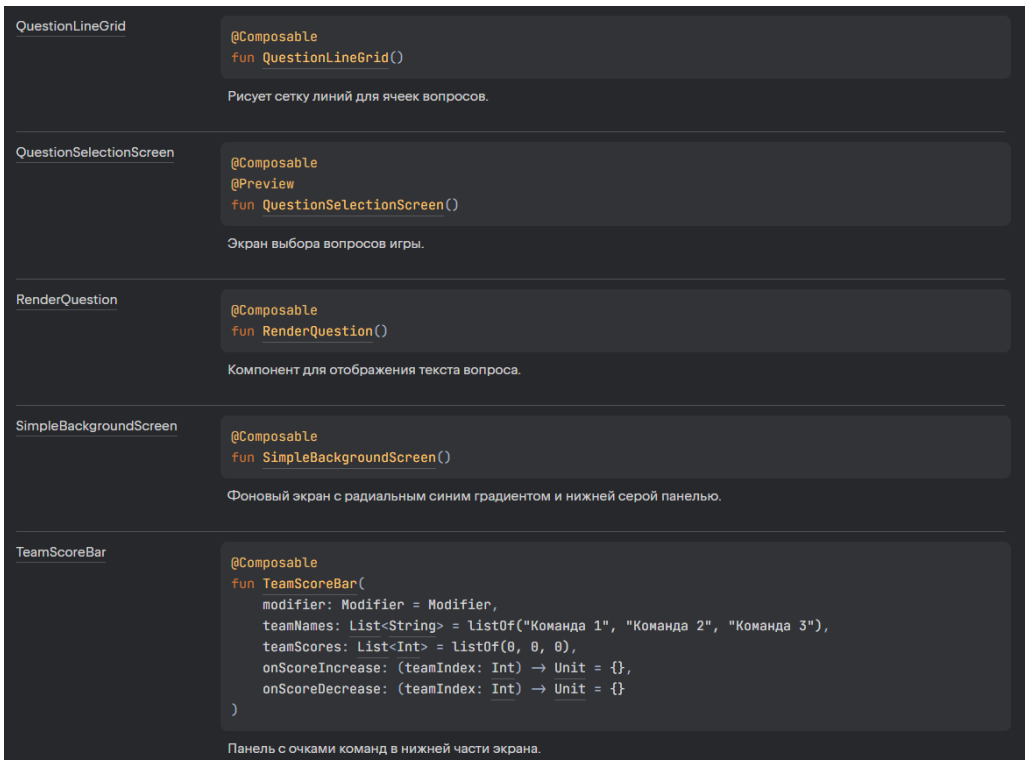

Рисунок 14 – Сгенерированная документация проекта

ЗАКЛЮЧЕНИЕ

В ходе разработки проекта SvoyaIgra был создан прототип интерактивного игрового интерфейса, реализованный на Kotlin с использованием JetBrains Compose Multiplatform. Были проработаны основные экраны: поле выбора вопросов, отображение текста вопроса и панель счёта команд с интерактивным изменением очков. Архитектура приложения заложена таким образом, чтобы упростить дальнейшее расширение функциональности: подключение файлов с пакетами вопросов, добавление настроек игры и улучшение визуального оформления. Полученный результат демонстрирует работоспособность выбранных технологий и служит основой для последующей доработки проекта до полноценного игрового приложения.

Ссылка на GitHub-репозиторий разработанного проекта: https://github.com/EverlastingN1ghtfall/SvoyaIgra

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).