Отчёт по практике

**«Разработка приложений для мобильных устройств»**

Задание 4

Реализация игрового процесса с использованием кастомной графики, жестов и фоновой музыки

Выполнил:

Студент группы 8B21 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Ю.Д. Алексейчик

# Задание

Создайте ядро игры "Кошки-мышки".

1. Реализуйте GameFragment с кастомной View (GameView), на которой будет отрисовываться игровой процесс.

* Мышь управляется с помощью жестов (касание и перемещение пальца). Мышь должна следовать за движением пальца.
* На поле в случайных местах и в случайное время появляются "сыры" (цели). При "съедании" мышь должна увеличивать счет.
* Реализуйте анимацию движения мыши и исчезновения сыра.

1. Добавьте фоновую музыку и звуковые эффекты (поедание сыра), которые управляются настройками из Задачи 2. Используйте MediaPlayer или SoundPool.
2. Реализуйте логику "кошек" как движущихся препятствий. При столкновении мыши с кошкой игра заканчивается, открывается диалоговое окно (DialogFragment) с предложением сохранить рекорд (используя логику из Задачи 3).
3. Бонус: Реализуйте простой Foreground Service, который показывает уведомление с текущим счетом, когда игра активна.

# ****Исходные данные:****

1. Проектирование UI: Кастомная View (GameView), переопределяющая методы onDraw() и onTouchEvent(). DialogFragment для окончания игры.
2. Программирование на Kotlin: Классы для игровых объектов (Mouse, Cheese, Cat). Игровой цикл (возможно, с использованием ValueAnimator или корутин с задержкой). Логика коллизий.
3. Работа с Android Components: Работа с SensorManager (опционально, как альтернатива жестам), MediaPlayer, Service, Notification.
4. Тестирование: Написать сложные инструментальные тесты (Espresso) для проверки основных сценариев игры (начало, поедание сыра, столкновение).

# Ход работы

1. Реализуйте GameFragment с кастомной View (GameView), на которой будет отрисовываться игровой процесс.

* Мышь управляется с помощью жестов (касание и перемещение пальца). Мышь должна следовать за движением пальца.
* На поле в случайных местах и в случайное время появляются "сыры" (цели). При "съедании" мышь должна увеличивать счет.
* Реализуйте анимацию движения мыши и исчезновения сыра.

1. Добавьте фоновую музыку и звуковые эффекты (поедание сыра), которые управляются настройками из Задачи 2. Используйте MediaPlayer или SoundPool.
2. Реализуйте логику "кошек" как движущихся препятствий. При столкновении мыши с кошкой игра заканчивается, открывается диалоговое окно (DialogFragment) с предложением сохранить рекорд (используя логику из Задачи 3).
3. Бонус: Реализуйте простой Foreground Service, который показывает уведомление с текущим счетом, когда игра активна.

GameView.kt

// создание игрового поля  
class GameView @JvmOverloads constructor( // @JvmOverloads создание view из xml  
 context: Context, attrs: AttributeSet? = null  
) : View(context, attrs) {  
  
 // слушатель для событий игры (изменение счёта, окончине игры)  
 interface GameListener {  
 fun onScoreChanged(score: Int)  
 fun onGameOver(finalScore: Int)  
 }  
  
 var listener: GameListener? = null  
  
 // ресурсы графики  
 private val mouseBitmapOrig = BitmapFactory.decodeResource(*resources*, R.drawable.*mouse*)  
 private val cheeseBitmapOrig = BitmapFactory.decodeResource(*resources*, R.drawable.*cheese*)  
 private val catBitmapOrig = BitmapFactory.decodeResource(*resources*, R.drawable.*cat*)  
  
 // размеры спрайтов в пикселях  
 private val mouseSizePx = dpToPx(70)  
 private val cheeseSizePx = dpToPx(64)  
 private val catSizePx = dpToPx(140)  
  
 // для сохранения масштаба изображения  
 private fun scaleBitmapPreserveRatio(src: Bitmap, targetSize: Int): Bitmap {  
 val ratio = src.*width*.toFloat() / src.*height*.toFloat()  
 val newWidth: Int  
 val newHeight: Int  
  
 if (ratio > 1f) { // горизонтальное изображение  
 newWidth = targetSize  
 newHeight = (targetSize / ratio).toInt()  
 } else { // вертикальное или квадратное  
 newHeight = targetSize  
 newWidth = (targetSize \* ratio).toInt()  
 }  
  
 return Bitmap.createScaledBitmap(src, newWidth, newHeight, true)  
 }  
 private val mouseBitmap = scaleBitmapPreserveRatio(mouseBitmapOrig, mouseSizePx)  
 private val cheeseBitmap = scaleBitmapPreserveRatio(cheeseBitmapOrig, cheeseSizePx)  
 private val catBitmap = scaleBitmapPreserveRatio(catBitmapOrig, catSizePx)  
  
 // игровой объект мышь (используеты класс Mouse)  
 private val mouse = Mouse(name = "Mouser", speed = 100)  
 // координаты на экране  
 private var mouseX = 0f  
 private var mouseY = 0f  
  
 // список сыров и котов  
 private val cheeses = *mutableListOf*<Cheese>()  
 private val cats = *mutableListOf*<Cat>()  
  
 // игровые параметры  
 private var score = 0 // счёт  
 var currentScore: Int = 0  
 private set  
  
 private var running = false // флаг активности игры  
  
 // корутина для асинхронных задач  
 private val scope = *CoroutineScope*(Dispatchers.Main + *SupervisorJob*())  
  
 // SoundPool для эффектов  
 private var soundPool: SoundPool? = null // для быстрого воспроизведения звуковых эффктов  
 private var soundEatId: Int = 0 // ID звука поедания сыра  
  
 init {  
 // инициализация звуков (для игр, звуковые эффекты)  
 val attrs = AudioAttributes.Builder().setUsage(AudioAttributes.*USAGE\_GAME*).setContentType(AudioAttributes.*CONTENT\_TYPE\_SONIFICATION*).build()  
 soundPool = SoundPool.Builder().setAudioAttributes(attrs).setMaxStreams(3).build() // максимум три звука одновременно  
 soundEatId = soundPool!!.load(context, R.raw.*eat\_cheese*, 1) // загрузка звукового эффекта в soundpool  
  
 // задание значений стартовой позиции  
 mouseX = 300f  
 mouseY = 300f  
 }  
  
 // запуск игры  
 fun startGame() {  
 if (running) return  
  
 // ждём, пока View будет измерен  
 if (*width* == 0 || *height* == 0) {  
 post **{** startGame() **}** return  
 }  
  
 running = true  
 score = 0  
 currentScore = 0  
 cheeses.clear()  
 cats.clear()  
  
 // инициализация котов в зависимости от уровня сложности  
 // получаем ViewModel чтобы посмотреть выбранный уровень  
 val sharedLevel = (*context* as? androidx.fragment.app.FragmentActivity)?.*let* **{** act **->** ViewModelProvider(act, ViewModelProvider.AndroidViewModelFactory.getInstance(act.*application*)).get(SharedViewModel::class.*java*)  
 **}** // Меняем парамерт уровня сложности в зависимости от текстового значения  
 val level = sharedLevel?.selectedLevel?.*value* val difficulty = when (level?.difficulty) {  
 "Легкий" -> 1  
 "Средний" -> 2  
 "Сложный" -> 3  
 else -> 1  
 }  
 // количество котов и их скорость (зависит от сложности)  
 val catsCount = when (difficulty) {  
 1 -> 2  
 2 -> 3  
 3 -> 4  
 else -> 1  
 }  
 val catBaseSpeed = 5 + difficulty \* 2// 3..5  
  
 // генерация котов  
 *repeat*(catsCount) **{** var x: Float  
 var y: Float  
 do {  
 x = Random.nextInt(0, *width* - catSizePx).toFloat()  
 y = Random.nextInt(0, *height* - catSizePx).toFloat()  
 } while (distance(x, y, mouseX, mouseY) < 500f) // чтобы не рядом с мышью  
  
 // направление и скорость, разнообразие движения  
 val angle = Random.nextDouble(0.0, 2 \* Math.*PI*)  
 val speed = catBaseSpeed + Random.nextDouble(0.0, 3.0)  
 val vx = (Math.cos(angle) \* speed).toFloat()  
 val vy = (Math.sin(angle) \* speed).toFloat()  
  
 cats.add(Cat(x, y, vx, vy, catSizePx))  
 **}** // корутина игрового цикла (параллельные задачи генерации сыра и игрового цикла)  
 scope.*launch* **{** spawnCheesesLoop()  
 **}** scope.*launch* **{** gameLoop()  
 **}** }  
  
 // чтобы мышь была по центру  
 override fun onSizeChanged(w: Int, h: Int, oldw: Int, oldh: Int) {  
 super.onSizeChanged(w, h, oldw, oldh)  
 if (mouseX == 0f && mouseY == 0f) {  
 mouseX = w / 2f - mouseSizePx / 2f  
 mouseY = h / 2f - mouseSizePx / 2f  
 }  
 }  
  
 // функция конца игры  
 fun stopGame() {  
 running = false  
 scope.coroutineContext.*cancelChildren*()  
 scope.*cancel*()  
 soundPool?.release()  
 soundPool = null  
 }  
  
 // функция генерации сыра  
 private suspend fun spawnCheesesLoop() {  
 while (running) {  
 delay(Random.nextLong(800, 2500))  
 if (!running) break  
 val nx = Random.nextInt(50, (*width* - 50).*coerceAtLeast*(50)).toFloat()  
 val ny = Random.nextInt(50, (*height* - 50).*coerceAtLeast*(50)).toFloat()  
 val ch = Cheese(nx, ny, cheeseSizePx)  
 cheeses.add(ch)  
 // исчезновение через некоторое время (если не собран)  
 scope.*launch* **{** delay(7000)  
 cheeses.remove(ch)  
 **}** }  
 }  
  
 // основной цикл  
 private suspend fun gameLoop() {  
 val frameDelay = 16L // количество кадров 60 fps  
 while (running) {  
 updatePhysics() // обновление состояния  
 invalidate() // перерисовка экрана  
 delay(frameDelay)  
 }  
 }  
  
 // физика и коллизии  
 private fun updatePhysics() {  
 // движение котов  
 for (cat in cats) {  
 cat.x += cat.vx  
 cat.y += cat.vy  
  
 // изменение направления котов раз в секунду  
 if (Random.nextInt(0, 20) == 0) { // 1 из 60 кадров (~раз в секунду)  
 val angleChange = Random.nextDouble(-1.5, 1.5) // в радианах  
 val speed = *hypot*(cat.vx.toDouble(), cat.vy.toDouble())  
 val angle = Math.atan2(cat.vy.toDouble(), cat.vx.toDouble()) + angleChange  
 cat.vx = (Math.cos(angle) \* speed).toFloat()  
 cat.vy = (Math.sin(angle) \* speed).toFloat()  
 }  
  
 // Левая и правая границы  
 if (cat.x < 0) {  
 cat.x = 0f  
 cat.vx = *abs*(cat.vx)  
 } else if (cat.x + cat.size > *width*) {  
 cat.x = (*width* - cat.size).toFloat()  
 cat.vx = -*abs*(cat.vx)  
 }  
  
 // Верхняя и нижняя границы  
 if (cat.y < 0) {  
 cat.y = 0f  
 cat.vy = *abs*(cat.vy)  
 } else if (cat.y + cat.size > *height*) {  
 cat.y = (*height* - cat.size).toFloat()  
 cat.vy = -*abs*(cat.vy)  
 }  
  
 }  
  
 // проверка столкновений c сыром (круг)  
 val it = cheeses.*toList*()  
 for (ch in it) {  
 if (distance(mouseX + mouseSizePx/2f, mouseY + mouseSizePx/2f, ch.x + ch.size/2f, ch.y + ch.size/2f) < (mouseSizePx/2f + ch.size/2f)) {  
 // съедено  
 cheeses.remove(ch)  
 score += 1  
 currentScore = score  
 listener?.onScoreChanged(score)  
 soundPool?.play(soundEatId, 1f, 1f, 1, 0, 1f)  
 }  
 }  
  
 // проверка столкновений c сыром  
 fun isCollidingRect(  
 x1: Float, y1: Float, size1: Int,  
 x2: Float, y2: Float, size2: Int  
 ): Boolean {  
 return !(x1 + size1 < x2 || x1 > x2 + size2 ||  
 y1 + size1 < y2 || y1 > y2 + size2)  
 }  
  
 // проверка столкновений с котом (прямоугольник)  
 for (cat in cats) {  
 val catHitboxSize = cat.size / 2 // хитбокс в 2 раза меньше  
 val catHitboxX = cat.x + cat.size / 4 // центрируем хитбокс  
 val catHitboxY = cat.y + cat.size / 4  
  
 if (isCollidingRect(mouseX, mouseY, mouseSizePx, catHitboxX, catHitboxY, catHitboxSize)) {  
 running = false  
 listener?.onGameOver(score)  
 break  
 }  
 }  
 }  
  
 // отрисовка игрового экрана  
 override fun onDraw(canvas: Canvas) {  
 super.onDraw(canvas)  
  
 // рисует сыры  
 for (c in cheeses) {  
 canvas.drawBitmap(cheeseBitmap, c.x, c.y, null)  
 }  
  
 // рисует котов  
 for (cat in cats) {  
 canvas.drawBitmap(catBitmap, cat.x, cat.y, null)  
 }  
  
 // рисует мышь  
 canvas.drawBitmap(mouseBitmap, mouseX, mouseY, null)  
 }  
  
 // управление касанием пальца  
 override fun onTouchEvent(event: MotionEvent?): Boolean {  
 if (event == null) return false  
 when (event.*actionMasked*) {  
 MotionEvent.*ACTION\_DOWN*, MotionEvent.*ACTION\_MOVE* -> {  
 // мышь следует за касанием, плавно интерполируем  
 val tx = event.*x* - mouseSizePx/2f  
 val ty = event.*y* - mouseSizePx/2f  
 // перемещаем с ограничением скорости (используется mouse.speed)  
 val dx = tx - mouseX  
 val dy = ty - mouseY  
 val dist = *hypot*(dx.toDouble(), dy.toDouble()).toFloat()  
 if (dist > 0) {  
 val maxStep = mouse.speed.toFloat()  
 val step = if (dist > maxStep) maxStep else dist  
 mouseX += dx / dist \* step  
 mouseY += dy / dist \* step  
 }  
 return true  
 }  
 MotionEvent.*ACTION\_UP*, MotionEvent.*ACTION\_CANCEL* -> return true  
 }  
 return super.onTouchEvent(event)  
 }  
  
 // конвертация единиц времени  
 private fun dpToPx(dp: Int): Int {  
 val density = *resources*.*displayMetrics*.density  
 return (dp \* density).toInt()  
 }  
  
 // расчёт расстояния между точками  
 private fun distance(x1: Float, y1: Float, x2: Float, y2: Float): Float {  
 return *hypot*((x1 - x2).toDouble(), (y1 - y2).toDouble()).toFloat()  
 }  
}

Класс GameView является пользовательским компонентом, отвечающим за визуализацию и логику игрового процесса.Он используется для отрисовки всех игровых объектов (мыши, котов, сыров) и обработки взаимодействия с пользователем.

Основные функции:

Отрисовка объектов: в методе onDraw() рисуются спрайты мыши, сыров и котов на холсте Canvas.

Управление касаниями: метод onTouchEvent() обрабатывает движение пальца по экрану, перемещая мышь с заданной скоростью.

Игровой цикл: реализован через корутины gameLoop() и spawnCheesesLoop(), которые работают параллельно.

gameLoop() отвечает за обновление физики и перерисовку, а spawnCheesesLoop() – за периодическое появление сыров.

Обработка столкновений: метод updatePhysics() проверяет пересечения мыши с сырами и котами, начисляет очки и вызывает событие окончания игры при столкновении с котом.

Масштабирование спрайтов: функция scaleBitmapPreserveRatio() масштабирует изображения без искажения пропорций.

Работа со звуком: используется SoundPool для воспроизведения коротких звуковых эффектов (звук поедания сыра).

Интерфейс слушателя: через интерфейс GameListener (onScoreChanged, onGameOver) уведомляет фрагмент об изменении счёта или завершении игры.

Компонент связан с:

GameFragment – получает события о счёте и окончании игры.

SharedViewModel – определяет уровень сложности и количество котов.

MainActivity – используется для контекста и доступа к ресурсам.

GameFragment.kt

class GameFragment : Fragment(), GameView.GameListener {  
  
 private lateinit var gameView: GameView  
 private lateinit var tvScoreTop: TextView // текст счёта  
 private lateinit var btnExit: Button // кнопка выхода  
 private var mediaPlayer: MediaPlayer? = null // проигрыватель музыки  
  
 private lateinit var leaderboardViewModel: LeaderboardViewModel // для рекордов  
  
 override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?): View? {  
 // создания контейнера для занятия всего экрана  
 val root = inflater.inflate(R.layout.*fragment\_game*, container, false)  
 return root  
 }  
  
 override fun onViewCreated(view: View, savedInstanceState: Bundle?) {  
 super.onViewCreated(view, savedInstanceState)  
  
 // привязка к активити, чтобы данные сохранялись между фрагментами  
 leaderboardViewModel = ViewModelProvider(requireActivity(), ViewModelProvider.AndroidViewModelFactory.getInstance(requireActivity().*application*)).get(LeaderboardViewModel::class.*java*)  
  
 // нахождение view элементов по id  
 tvScoreTop = view.findViewById(R.id.*tvGameScore*)  
 btnExit = view.findViewById(R.id.*btnExitGame*)  
  
 // програмное создание GameView для установки размера на весь экран  
 val frame: FrameLayout = view.findViewById(R.id.*gameHost*)  
 gameView = GameView(requireContext())  
 gameView.*layoutParams* = FrameLayout.LayoutParams(FrameLayout.LayoutParams.*MATCH\_PARENT*, FrameLayout.LayoutParams.*MATCH\_PARENT*)  
 gameView.listener = this  
 frame.addView(gameView)  
  
 // старт/стоп фоновой музыки согласно настройке (считывание из SharedPreferences)  
 val prefs = requireActivity().getSharedPreferences("game\_prefs", Context.*MODE\_PRIVATE*)  
 val musicOn = prefs.getBoolean("pref\_music", true)  
 if (musicOn) { // если настроки включены, создаётся MediaPlayer, который зацикливает музыку  
 mediaPlayer = MediaPlayer.create(requireContext(), R.raw.*bg\_music*)  
 mediaPlayer?.*isLooping* = true  
 mediaPlayer?.start()  
 }  
  
 // слушатель выхода  
 btnExit.setOnClickListener **{** endGameAndSave()  
 **}** // скрытие всех кнопок из меню  
 (*activity* as? MainActivity)?.*let* **{** act **->** act.findViewById<View>(R.id.*tvTitle*)?.*visibility* = View.*GONE* act.findViewById<View>(R.id.*btnLevels*)?.*visibility* = View.*GONE* act.findViewById<View>(R.id.*btnSettings*)?.*visibility* = View.*GONE* act.findViewById<View>(R.id.*btnLeaderboard*)?.*visibility* = View.*GONE* act.findViewById<View>(R.id.*btnStart*)?.*visibility* = View.*GONE* **}** // запуск игры, счётчик очков = 0  
 updateTopScore(0)  
 gameView.startGame()  
 }  
  
 // отчистка ресурсов, возвращение к начальному меню  
 override fun onDestroyView() {  
 super.onDestroyView() // удаляем фоновую музыку  
 mediaPlayer?.stop()  
 mediaPlayer?.release()  
 mediaPlayer = null  
 gameView.stopGame()  
  
 // возвращение кнопок из стартового меню  
 (*activity* as? MainActivity)?.*let* **{** act **->** act.findViewById<View>(R.id.*tvTitle*)?.*visibility* = View.*VISIBLE* act.findViewById<View>(R.id.*btnLevels*)?.*visibility* = View.*VISIBLE* act.findViewById<View>(R.id.*btnSettings*)?.*visibility* = View.*VISIBLE* act.findViewById<View>(R.id.*btnLeaderboard*)?.*visibility* = View.*VISIBLE* act.findViewById<View>(R.id.*btnStart*)?.*visibility* = View.*VISIBLE* **}** }  
  
 // GameListener  
 override fun onScoreChanged(score: Int) {  
 updateTopScore(score)  
 // обновление счёта  
 val it = android.content.Intent(requireContext(), GameForegroundService::class.*java*)  
 it.putExtra("score", score)  
 it.*action* = GameForegroundService.ACTION\_UPDATE  
 requireContext().startService(it)  
 }  
  
 // окончание игры  
 override fun onGameOver(finalScore: Int) {  
 // музыка останавливается  
 mediaPlayer?.pause()  
 // показывается диалог об окончании игры  
 val dlg = GameOverDialogFragment.newInstance(finalScore)  
 dlg.show(*parentFragmentManager*, "game\_over")  
 }  
  
 // обновление счёта на экране  
 private fun updateTopScore(score: Int) {  
 tvScoreTop.*text* = "Счет: $score"  
 }  
  
 // завершение игры с сохранением  
 private fun endGameAndSave() { // получение имя игрока из настроек  
 val playerName = requireActivity().getSharedPreferences("game\_prefs", Context.*MODE\_PRIVATE*)  
 .getString("pref\_player\_name", "Игрок") ?: "Игрок"  
 val finalScore = gameView.currentScore  
 leaderboardViewModel.addSampleRecord(playerName, finalScore)  
 // вернуться назад на начальный экран  
 *parentFragmentManager*.popBackStack()  
 }  
}

Файл GameFragment.kt отвечает за отображение игрового экрана и управление состоянием игры. Он связывает интерфейс пользователя, игровое поле и музыкальное сопровождение.

Основные функции:

Создание интерфейса: загружает макет fragment\_game.xml, находит элементы TextView и Button, и программно добавляет GameView в контейнер FrameLayout.

Инициализация ViewModel: подключает LeaderboardViewModel, который используется для сохранения рекордов после завершения игры.

Музыкальное сопровождение: при старте игры запускает фоновую музыку через MediaPlayer, если она включена в настройках.

Обработка событий игры: реализует интерфейс GameView.GameListener:

onScoreChanged(score) – обновляет счёт на экране и передаёт данные в GameForegroundService для отображения уведомления.

onGameOver(finalScore) – при завершении игры останавливает музыку и вызывает диалоговое окно GameOverDialogFragment.

Кнопка выхода: при нажатии на кнопку "Выход" вызывается метод endGameAndSave(), который сохраняет результат и возвращает пользователя на главный экран.

Скрытие элементов меню: при запуске фрагмента скрывает кнопки из главного меню, а при закрытии – возвращает их обратно.

Связи:

GameView – игровой компонент, откуда поступают события.

LeaderboardViewModel – отвечает за сохранение рекордов.

GameForegroundService – обновляет уведомление с текущим счётом.

MainActivity – используется для управления элементами интерфейса меню.

GameOverDialogFragment.kt

// диалоговое окно окончания игры (диалоговое окно)  
class GameOverDialogFragment : DialogFragment() {  
  
 companion object {  
 private const val ARG\_SCORE = "arg\_score"  
 fun newInstance(score: Int): GameOverDialogFragment {  
 val f = GameOverDialogFragment()  
 val b = Bundle()  
 b.putInt(ARG\_SCORE, score)  
 f.*arguments* = b  
 return f  
 }  
 }  
  
 // создание диалога, считываются счёт и имя игрока  
 override fun onCreateDialog(savedInstanceState: Bundle?): Dialog {  
 val score = *arguments*?.getInt(ARG\_SCORE) ?: 0  
 val prefs = requireActivity().getSharedPreferences("game\_prefs", Context.*MODE\_PRIVATE*)  
 val playerName = prefs.getString("pref\_player\_name", "Игрок") ?: "Игрок"  
  
 return AlertDialog.Builder(requireContext())  
 .setTitle("Игра окончена")  
 .setMessage("Ваш счет: $score")  
 .setPositiveButton("Сохранить") **{** \_, \_ **->** // сохраняем рекорд  
 val vm = androidx.lifecycle.ViewModelProvider(requireActivity(), ViewModelProvider.AndroidViewModelFactory.getInstance(requireActivity().*application*)).get(LeaderboardViewModel::class.*java*)  
 vm.addSampleRecord(playerName, score)  
 dismiss()  
 *parentFragmentManager*.popBackStack()  
 **}** .setNegativeButton("Отмена") **{** \_, \_ **->** dismiss()  
 *parentFragmentManager*.popBackStack()  
 **}** .create()  
 }  
}

GameOverDialogFragment – диалоговое окно, которое отображается после завершения игры. Оно сообщает пользователю результат и предлагает сохранить рекорд или выйти без сохранения.

GameForegroundService.kt

// постоянное уведомление для счета  
class GameForegroundService : Service() {  
  
 companion object {  
 const val CHANNEL\_ID = "game\_channel" // ID канала уведомлений  
 const val NOTIF\_ID = 0x1234 // ID уведомления  
 const val ACTION\_UPDATE = "ru.alekseychick.game.ACTION\_UPDATE\_SCORE"  
 }  
  
 // создание канала уведомлений  
 override fun onCreate() {  
 super.onCreate()  
 if (Build.VERSION.*SDK\_INT* >= Build.VERSION\_CODES.*O*) {  
 val chan = NotificationChannel(CHANNEL\_ID, "Game", NotificationManager.*IMPORTANCE\_LOW*)  
 getSystemService(NotificationManager::class.*java*).createNotificationChannel(chan)  
 }  
 }  
  
 // запуск сервера  
 override fun onStartCommand(intent: Intent?, flags: Int, startId: Int): Int {  
 val score = intent?.getIntExtra("score", 0) ?: 0  
 val notif = createNotification(score)  
 startForeground(NOTIF\_ID, notif)  
 return *START\_STICKY* }  
  
 // создание уведомления  
 private fun createNotification(score: Int): Notification {  
 val builder = androidx.core.app.NotificationCompat.Builder(this, CHANNEL\_ID)  
 .setContentTitle("Кошки-Мышки — игра")  
 .setContentText("Текущий счет: $score")  
 .setOngoing(true)  
 return builder.build()  
 }  
  
 override fun onBind(intent: Intent?): IBinder? = null  
}

GameForegroundService – Android-служба, которая работает в режиме foreground и отображает текущее состояние игры в системной шторке уведомлений. Она используется для обеспечения постоянного отображения счёта игрока во время игры.

Для создания сыра и кошек используются классы Cat.kt и Cheese.kt. В MainActivity.kt было добавлено событие при нажатии на кнопку “Начать игру” openFragment(GameFragment()), которое запускает начало игры.

Тестирование:

GameInstrumentTest.kt

@RunWith(AndroidJUnit4::class)  
@LargeTest  
class GameInstrumentedTest {  
  
 @Before  
 fun setup() {  
 // Тестовый режим, отключаем реальный игровой цикл  
 GameView.isTestMode = true  
  
 // Отключаем музыку и сервисы, чтобы не мешали тестам  
 val ctx = InstrumentationRegistry.getInstrumentation().*targetContext* val prefs = ctx.getSharedPreferences("game\_prefs", 0)  
 prefs.edit().putBoolean("pref\_music", false).apply()  
 }  
  
  
 // Проверка: старт игры  
  
 @Test  
 fun testGameStart\_showsGameScreen() {  
 val scenario = *launchFragmentInContainer*<GameFragment>(themeResId = R.style.*Theme\_Game*)  
 scenario.moveToState(Lifecycle.State.*RESUMED*)  
  
 // Проверяем, что отображается счёт и кнопка выхода  
 onView(withId(R.id.*tvGameScore*)).check(matches(isDisplayed()))  
 onView(withId(R.id.*btnExitGame*)).check(matches(isDisplayed()))  
  
 // Проверяем, что счёт изначально равен 0  
 onView(withText("Счет: 0")).check(matches(isDisplayed()))  
 }  
  
 // Проверка: поедание сыра  
 @Test  
 fun testMouseEatsCheese\_increasesScore() {  
 val scenario = *launchFragmentInContainer*<GameFragment>(themeResId = R.style.*Theme\_Game*)  
 scenario.moveToState(Lifecycle.State.*RESUMED*)  
  
 scenario.onFragment **{** fragment **->** val gv = fragment.requireView()  
 .findViewById<FrameLayout>(R.id.*gameHost*)  
 .getChildAt(0) as GameView  
  
 // эмулируем событие — как будто мышь съела сыр  
 fragment.requireActivity().runOnUiThread **{** gv.listener?.onScoreChanged(1)  
 **}** // ждём пока UI обновится  
 InstrumentationRegistry.getInstrumentation().waitForIdleSync()  
 **}** // Проверяем, что счёт изменился на 1  
 onView(withId(R.id.*tvGameScore*)).check(matches(withText("Счет: 1")))  
 }  
  
  
 // Проверка: столкновение с котом  
  
 @Test  
 fun testMouseCollidesWithCat\_showsGameOverDialog() {  
 val scenario = *launchFragmentInContainer*<GameFragment>(themeResId = R.style.*Theme\_Game*)  
 scenario.moveToState(Lifecycle.State.*RESUMED*)  
  
 scenario.onFragment **{** fragment **->** val gv = fragment.requireView()  
 .findViewById<FrameLayout>(R.id.*gameHost*)  
 .getChildAt(0) as GameView  
  
 // эмулируем событие столкновения  
 fragment.requireActivity().runOnUiThread **{** gv.listener?.onGameOver(5)  
 **}** InstrumentationRegistry.getInstrumentation().waitForIdleSync()  
 **}** // Проверяем, что показан диалог "Игра окончена"  
 onView(withText("Игра окончена")).check(matches(isDisplayed()))  
 }  
}

Espresso тесты для проверки базовой логики игры: запуск начала игры, поедание сыра, столкновение с котом.

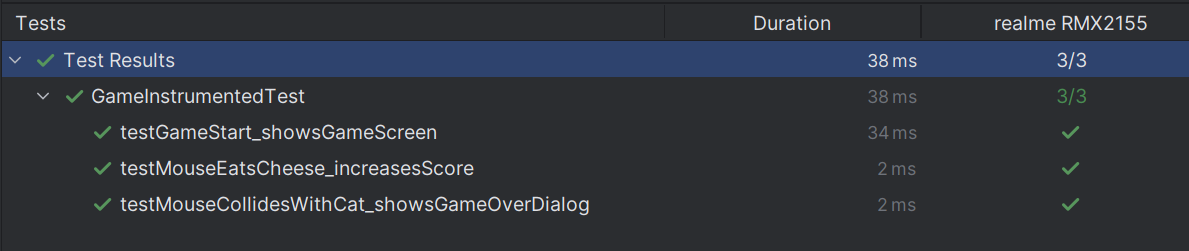


Рисунок 1. Успешно пройденные тесты

# Результат работы

Ссылка на GitHub: <https://github.com/AlekJulia/Alekseychik_MobileApplicationDevelopment.git>

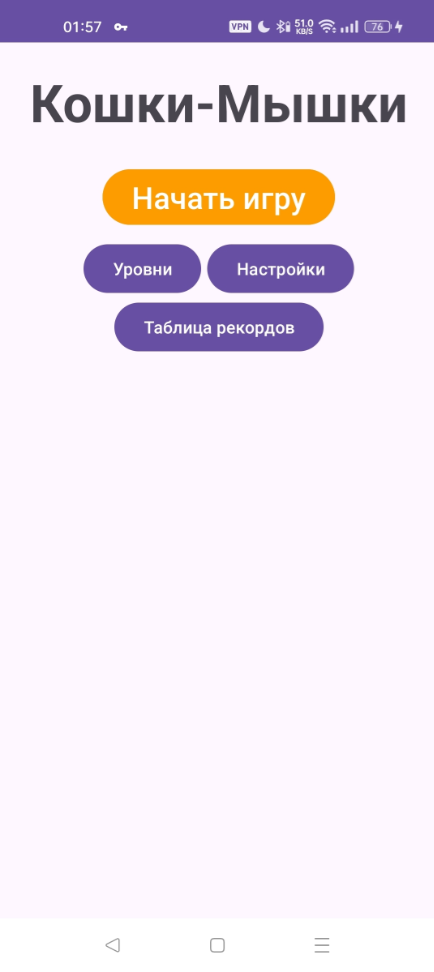


Рисунок 2. Главное меню игры

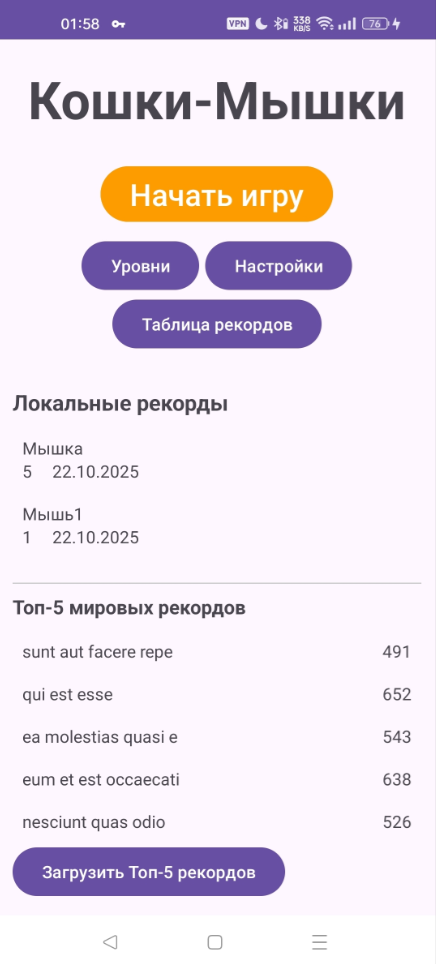


Рисунок 3. Таблица рекордов с локальными рекордами игрока

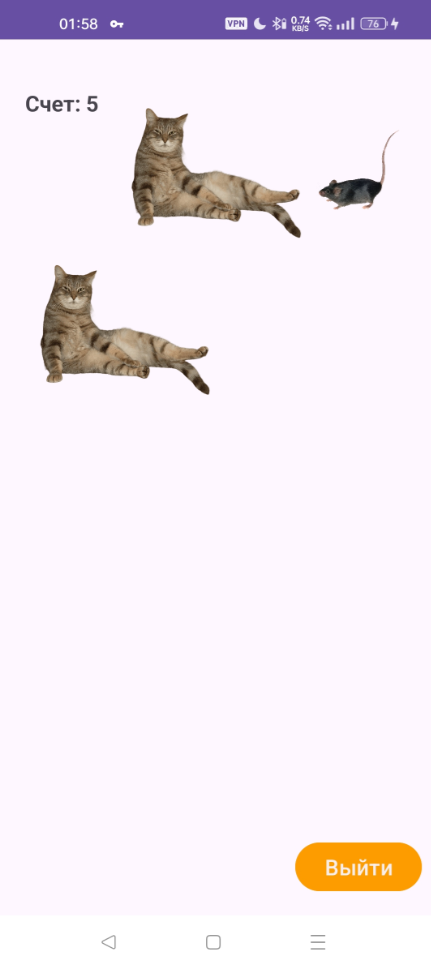


Рисунок 4. Процесс игры с первым уровнем сложности

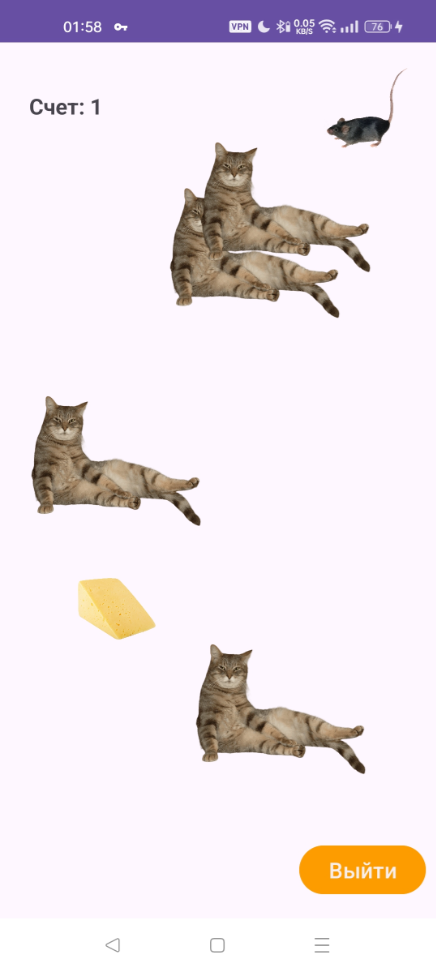


Рисунок 5. Процесс игры с третьим уровнем сложности

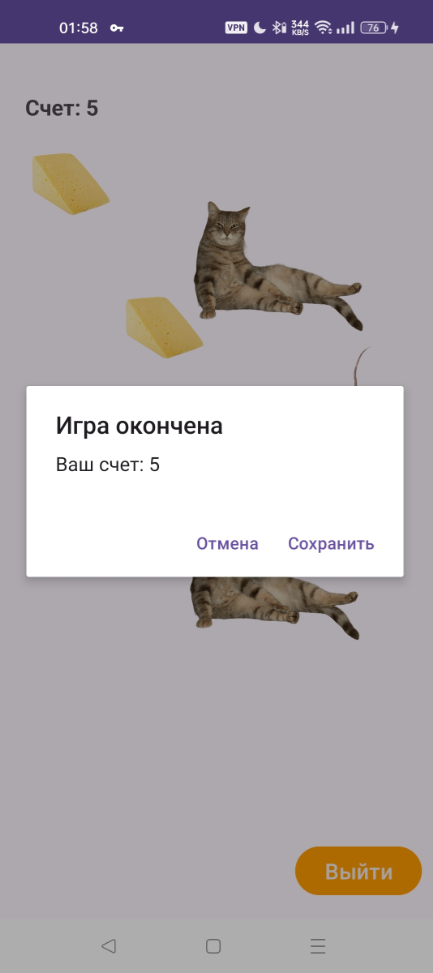


Рисунок 6. Окно о завершении игры