**Министерство науки и высшего образования Российской Федерации**

Федеральное государственное автономное образовательное

учреждение высшего образования

**Национальный исследовательский**

**Томский политехнический университет**

Инженерная школа информационных технологий и робототехники

Отделение информационных технологий

Отчет по лабораторной работе №22 по дисциплине

**«Разработка приложений для мобильных устройств»**

Геолокация

Выполнил:

Студент группы 8В21 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Алексейчик Ю. Д.

Проверил:

Ст. преп. ОИТ ИШИТР \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Дорофеев В. А.

# Задание

Разработайте программу-игру «Найди точку!» Программа должна использовать средства геолокации и должна выглядеть примерно так:

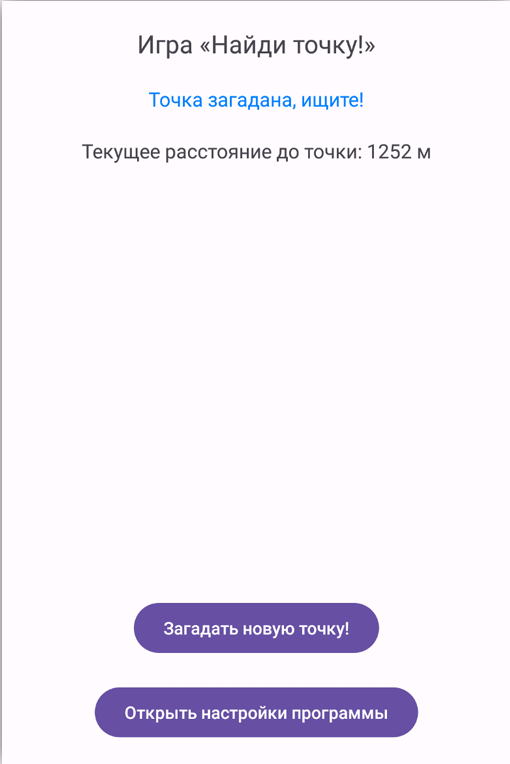


Рисунок 1. Примерный вид приложения

При запуске программы, а также при нажатии кнопки «Загадать новую точку!» программа генерирует случайную точку в радиусе примерно 2.5 км от текущего местоположения пользователя и меняет статус на «Точка загадана, ищите!», цвет надписи синий. Когда местоположение пользователя меняется (реально или с помощью виртуального перемещения по карте в эмуляторе Android Studio) программа определяет расстояние от пользователя до загаданной точки, и выводит это расстояние на экран. Если местоположение пользователя оказывается в пределах 100 метров от загаданной точки, то статус меняется на «Ура, точка найдена!», цвет надписи зелёный. Кнопка «Открыть настройки программы» позволяет открыть окно информации о программе, где можно посмотреть/изменить разрешения, в т. ч. доступ к геолокации.

Чтобы сгенерировать точку в требуемом радиусе от исходной, можно очень условно считать, что расстояние в 0.04 градуса соответствует 5 километрам – такое приближение работает в районе г. Томска, в других местах может быть другое значение. Точное попадание генерируемой точки в указанный радиус потребовало бы сложных вычислений, поэтому в задании это не требуется.

# Ход работы

Для начала необходимо запросить у пользователя разрешения на точное местоположение GPS (ACCESS\_FINE\_LOCATION) и разрешение на приблизительное местоположение (ACCESS\_COARSE\_LOCATION).

В activity\_main.xml задан основой интерфейс игры.

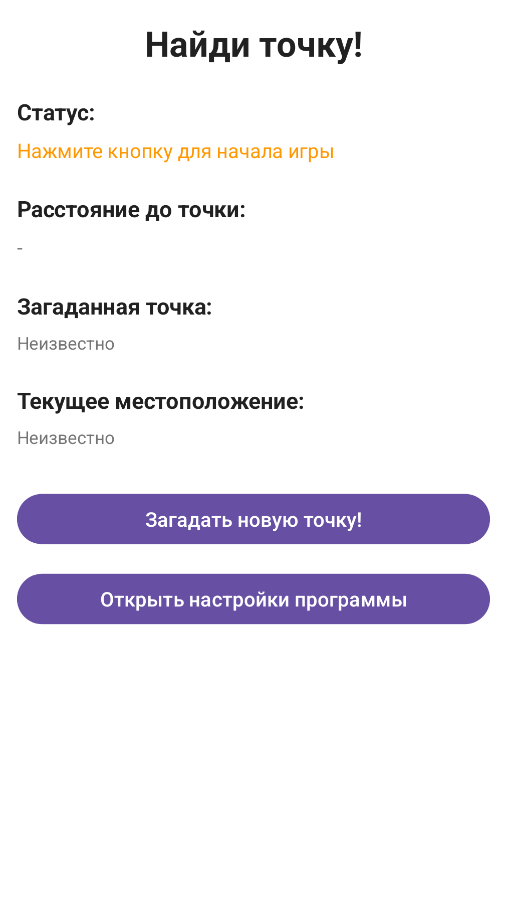


Рисунок 2. Интерфейс игры "Найди точку!"

Кнопка "Сгенерировать точку" отвечает за создание новой точки для поиска: при нажатии проверяет наличие разрешений на доступ к геолокации, определяет текущее местоположение пользователя и генерирует случайную точку в радиусе 2.5 км, после чего активирует игровой режим с отслеживанием расстояния до цели. Кнопка "Настройки" предоставляет доступ к управлению разрешениями приложения: при нажатии открывает диалоговое окно с объяснением необходимости доступа к геолокации и возможностью перехода в системные настройки для предоставления соответствующих разрешений.

В файле MainActivity.kt описана основная логика приложения.

**MainActivity.kt**

class MainActivity : AppCompatActivity(), LocationListener {  
  
 // объявление UI элементов  
 private lateinit var locationManager: LocationManager  
 private lateinit var statusTextView: TextView  
 private lateinit var distanceTextView: TextView  
 private lateinit var targetPointTextView: TextView  
 private lateinit var currentLocationTextView: TextView  
 private lateinit var generatePointButton: Button  
 private lateinit var settingsButton: Button  
  
 // переменные для хранения локаций и состояния игры  
 private var currentLocation: Location? = null  
 private var targetLocation: Location? = null  
 private var isGameActive = false  
 private var isWaitingForLocation = false  
  
 // константы для расчетов расстояний  
 private companion object {  
 const val EARTH\_RADIUS = 6371000.0  
 const val TARGET\_RADIUS\_METERS = 2500.0  
 const val SUCCESS\_DISTANCE\_METERS = 100.0  
 const val DEGREES\_PER\_5\_KM = 0.04  
 }  
  
 // запрос разрешения на локицию  
 private val locationPermissionRequest = registerForActivityResult(  
 ActivityResultContracts.RequestMultiplePermissions()  
 ) **{** permissions **->** when {  
 permissions.getOrDefault(Manifest.permission.*ACCESS\_FINE\_LOCATION*, false) ||  
 permissions.getOrDefault(Manifest.permission.*ACCESS\_COARSE\_LOCATION*, false) -> {  
 // разрешения предоставлены - запускается отслеживание локации  
 startLocationUpdates()  
 Toast.makeText(this, "Доступ к местоположению предоставлен", Toast.*LENGTH\_SHORT*).show()  
 }  
 else -> {  
 // разрешения не предоставлены - показывается диалог  
 showPermissionRequiredDialog()  
 }  
 }  
 **}** override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 setContentView(R.layout.*activity\_main*)  
  
 // инициализация всех компонентов приложения  
 initializeViews()  
 setupClickListeners()  
 initializeLocationManager()  
 checkLocationPermissions()  
 }  
  
 // инициализация всех View элементов из layout  
 private fun initializeViews() {  
 statusTextView = findViewById(R.id.*statusTextView*)  
 distanceTextView = findViewById(R.id.*distanceTextView*)  
 targetPointTextView = findViewById(R.id.*targetPointTextView*)  
 currentLocationTextView = findViewById(R.id.*currentLocationTextView*)  
 generatePointButton = findViewById(R.id.*generatePointButton*)  
 settingsButton = findViewById(R.id.*settingsButton*)  
  
 // показывается сообщение о ожидании местоположения  
 currentLocationTextView.*text* = "Определение местоположения"  
 }  
  
 // настройка обработчиков кликов для кнопок  
 private fun setupClickListeners() {  
 generatePointButton.setOnClickListener **{** // проверка разрешений перед генерацией точки  
 if (!hasLocationPermissions()) {  
 showPermissionExplanationDialog()  
 return@setOnClickListener  
 }  
  
 if (currentLocation == null) {  
 // ожидание определения местоположения, если оно не определено  
 isWaitingForLocation = true  
 statusTextView.*text* = "Ожидание определения местоположения"  
 statusTextView.setTextColor(ContextCompat.getColor(this, R.color.*color\_waiting*))  
 Toast.makeText(this, "Ожидайте определения местоположения", Toast.*LENGTH\_SHORT*).show()  
  
 // попытка получить последнее известное местоположение  
 tryGetLastKnownLocation()  
 } else {  
 // есть локация - генерация новой точки  
 generateNewTargetPoint()  
 }  
 **}** settingsButton.setOnClickListener **{** // показывается диалог настроек  
 showSettingsDialog()  
 **}** }  
  
 // получение сервиса менеджера локации  
 private fun initializeLocationManager() {  
 locationManager = getSystemService(*LOCATION\_SERVICE*) as LocationManager  
 }  
  
 // проверка наличия разрешений на локацию  
 private fun hasLocationPermissions(): Boolean {  
 val fineLocationGranted = ContextCompat.checkSelfPermission(  
 this,  
 Manifest.permission.*ACCESS\_FINE\_LOCATION* ) == PackageManager.*PERMISSION\_GRANTED* val coarseLocationGranted = ContextCompat.checkSelfPermission(  
 this,  
 Manifest.permission.*ACCESS\_COARSE\_LOCATION* ) == PackageManager.*PERMISSION\_GRANTED* return fineLocationGranted || coarseLocationGranted  
 }  
  
 // проверка и запрос разрешений  
 private fun checkLocationPermissions() {  
 if (hasLocationPermissions()) {  
 startLocationUpdates()  
 }  
 }  
  
 // запрос разрешений у пользователя  
 private fun requestLocationPermissions() {  
 locationPermissionRequest.launch(  
 *arrayOf*(  
 Manifest.permission.*ACCESS\_FINE\_LOCATION*,  
 Manifest.permission.*ACCESS\_COARSE\_LOCATION* )  
 )  
 }  
  
 // запуск получения обновлений локации  
 private fun startLocationUpdates() {  
 if (!hasLocationPermissions()) return  
  
 try {  
 // проверка доступности провайдеров GPS и сети  
 val isGpsEnabled = locationManager.isProviderEnabled(LocationManager.*GPS\_PROVIDER*)  
 val isNetworkEnabled = locationManager.isProviderEnabled(LocationManager.*NETWORK\_PROVIDER*)  
  
 if (!isGpsEnabled && !isNetworkEnabled) {  
 showLocationServicesDialog()  
 return  
 }  
  
 // запрос частых обновлений от GPS провайдера  
 if (isGpsEnabled) {  
 locationManager.requestLocationUpdates(  
 LocationManager.*GPS\_PROVIDER*,  
 500L, // Интервал 500 мс  
 0f, // Без минимального расстояния  
 this  
 )  
 }  
  
 // запрос обновлений от сетевого провайдера  
 if (isNetworkEnabled) {  
 locationManager.requestLocationUpdates(  
 LocationManager.*NETWORK\_PROVIDER*,  
 1000L, // интервал 1 секунда  
 0f, // без минимального расстояния  
 this  
 )  
 }  
  
 // попытка получения последнего местоположения  
 tryGetLastKnownLocation()  
  
 } catch (e: SecurityException) {  
 Toast.makeText(this, "Ошибка безопасности при доступе к местоположению", Toast.*LENGTH\_SHORT*).show()  
 } catch (e: Exception) {  
 Toast.makeText(this, "Ошибка при запуске службы местоположения", Toast.*LENGTH\_SHORT*).show()  
 }  
 }  
  
 // попытка получить последнюю известную локацию  
 private fun tryGetLastKnownLocation() {  
 try {  
 var lastLocation: Location? = null  
  
 // получение точной GPS локации  
 if (ContextCompat.checkSelfPermission(this, Manifest.permission.*ACCESS\_FINE\_LOCATION*) == PackageManager.*PERMISSION\_GRANTED*) {  
 lastLocation = locationManager.getLastKnownLocation(LocationManager.*GPS\_PROVIDER*)  
 }  
  
 // иначе получение сетевуой локации  
 if (lastLocation == null && ContextCompat.checkSelfPermission(this, Manifest.permission.*ACCESS\_COARSE\_LOCATION*) == PackageManager.*PERMISSION\_GRANTED*) {  
 lastLocation = locationManager.getLastKnownLocation(LocationManager.*NETWORK\_PROVIDER*)  
 }  
  
 // обновление интерфейса, если локация найдена  
 lastLocation?.*let* **{** location **->** onLocationChanged(location)  
 **}** } catch (e: SecurityException) {  
  
 }  
 }  
  
 // обработчик изменения локации  
 override fun onLocationChanged(location: Location) {  
 // проверка видимости местоположения  
 if (location.*latitude* == 0.0 && location.*longitude* == 0.0) {  
 return  
 }  
  
 currentLocation = location  
 updateCurrentLocationDisplay()  
  
 // генерации точки, если было ожидание  
 if (isWaitingForLocation) {  
 isWaitingForLocation = false  
 generateNewTargetPoint()  
 }  
  
 // обновление расстояние до цели при активной игре  
 if (isGameActive && targetLocation != null) {  
 calculateAndUpdateDistance()  
 }  
 }  
  
 // обновление отображения текущей локации  
 private fun updateCurrentLocationDisplay() {  
 currentLocation?.*let* **{** location **->** val locationText = String.*format*(  
 "Ш: %.6f\nД: %.6f\nТочность: %.1fм",  
 location.*latitude*,  
 location.*longitude*,  
 location.*accuracy* )  
 currentLocationTextView.*text* = locationText  
  
 // обнавление статуса, если игра не активна  
 if (!isGameActive) {  
 statusTextView.*text* = getString(R.string.*status\_waiting*)  
 statusTextView.setTextColor(ContextCompat.getColor(this, R.color.*color\_waiting*))  
 }  
 **}** }  
  
 // генерация новой точки в радиусе 2.5 км  
 private fun generateNewTargetPoint() {  
 currentLocation?.*let* **{** currentLoc **->** // генерация случайного расстояния и угла  
 val randomDistance = Math.random() \* TARGET\_RADIUS\_METERS  
 val randomAngle = Math.random() \* 2 \* Math.*PI* // преобразование расстояния в градусы  
 val degreesOffset = (randomDistance / 5000) \* DEGREES\_PER\_5\_KM  
  
 // расчет координат целевой точки  
 val targetLat = currentLoc.*latitude* + degreesOffset \* *cos*(randomAngle)  
 val targetLon = currentLoc.*longitude* + degreesOffset \* *sin*(randomAngle)  
  
 targetLocation = Location("target").*apply* **{** *latitude* = targetLat  
 *longitude* = targetLon  
 **}** updateTargetPointDisplay()  
 startGame()  
  
 Toast.makeText(this, "Точка загадана! Начинайте поиск!", Toast.*LENGTH\_SHORT*).show()  
  
 **}** ?: *run* **{** Toast.makeText(this, "Не удалось определить текущее местоположение", Toast.*LENGTH\_SHORT*).show()  
 **}** }  
  
 // обновление отображения целевой точки  
 private fun updateTargetPointDisplay() {  
 targetLocation?.*let* **{** location **->** val targetText = String.*format*(  
 "Ш: %.6f\nД: %.6f",  
 location.*latitude*,  
 location.*longitude* )  
 targetPointTextView.*text* = targetText  
 **}** }  
  
 // запуск игры - активация отслеживания расстояния  
 private fun startGame() {  
 isGameActive = true  
 statusTextView.*text* = getString(R.string.*status\_point\_generated*)  
 statusTextView.setTextColor(ContextCompat.getColor(this, R.color.*color\_searching*))  
  
 // расчёт начального расстояния  
 currentLocation?.*let* **{** calculateAndUpdateDistance()  
 **}** }  
  
 // расчет и обновление расстояния до цели  
 private fun calculateAndUpdateDistance() {  
 if (currentLocation == null || targetLocation == null) return  
  
 val distance = calculateDistance(  
 currentLocation!!.*latitude*,  
 currentLocation!!.*longitude*,  
 targetLocation!!.*latitude*,  
 targetLocation!!.*longitude* )  
  
 updateDistanceDisplay(distance)  
 checkForSuccess(distance)  
 }  
  
 // расчет расстояния между двумя точками по формуле гаверсинусов  
 private fun calculateDistance(lat1: Double, lon1: Double, lat2: Double, lon2: Double): Double {  
 val lat1Rad = Math.toRadians(lat1)  
 val lat2Rad = Math.toRadians(lat2)  
 val deltaLonRad = Math.toRadians(lon2 - lon1)  
  
 val centralAngle = *acos*(  
 *sin*(lat1Rad) \* *sin*(lat2Rad) +  
 *cos*(lat1Rad) \* *cos*(lat2Rad) \* *cos*(deltaLonRad)  
 )  
  
 return EARTH\_RADIUS \* centralAngle  
 }  
  
 // обновление отображения расстояния (в метрах или км)  
 private fun updateDistanceDisplay(distance: Double) {  
 val distanceText = when {  
 distance >= 1000 -> String.*format*("%.2f км", distance / 1000)  
 else -> String.*format*("%.0f метров", distance)  
 }  
 distanceTextView.*text* = distanceText  
 }  
  
 // проверка достижения цели (в радиусе 100 метров)  
 private fun checkForSuccess(distance: Double) {  
 if (distance <= SUCCESS\_DISTANCE\_METERS) {  
 isGameActive = false  
 statusTextView.*text* = getString(R.string.*status\_point\_found*)  
 statusTextView.setTextColor(ContextCompat.getColor(this, R.color.*color\_found*))  
 Toast.makeText(this, "Поздравляем! Вы нашли точку!", Toast.*LENGTH\_LONG*).show()  
 }  
 }  
  
 // диалог объяснения необходимости разрешений  
 private fun showPermissionExplanationDialog() {  
 AlertDialog.Builder(this)  
 .setTitle(getString(R.string.*permission\_required*))  
 .setMessage(getString(R.string.*location\_permission\_description*))  
 .setPositiveButton(getString(R.string.*grant\_permission*)) **{** \_, \_ **->** requestLocationPermissions()  
 **}** .setNegativeButton(getString(R.string.*close*), null)  
 .show()  
 }  
  
 // диалог когда разрешения не даны  
 private fun showPermissionRequiredDialog() {  
 AlertDialog.Builder(this)  
 .setTitle("Разрешения не предоставлены")  
 .setMessage("Для работы игры необходимо предоставить доступ к местоположению.")  
 .setPositiveButton("Предоставить") **{** \_, \_ **->** requestLocationPermissions()  
 **}** .setNegativeButton("Закрыть", null)  
 .show()  
 }  
  
 // диалог когда службы локации выключены  
 private fun showLocationServicesDialog() {  
 AlertDialog.Builder(this)  
 .setTitle("Службы местоположения отключены")  
 .setMessage("Для работы приложения необходимо включить службы местоположения (GPS или сеть).")  
 .setPositiveButton("Включить") **{** \_, \_ **->** openLocationSettings()  
 **}** .setNegativeButton("Отмена", null)  
 .show()  
 }  
  
 // диалог настроек приложения  
 private fun showSettingsDialog() {  
 AlertDialog.Builder(this)  
 .setTitle(getString(R.string.*settings\_title*))  
 .setMessage(getString(R.string.*location\_permission\_description*))  
 .setPositiveButton(getString(R.string.*open\_settings*)) **{** \_, \_ **->** openAppSettings()  
 **}** .setNegativeButton(getString(R.string.*close*), null)  
 .show()  
 }  
  
 // открытие системных настроек локации  
 private fun openLocationSettings() {  
 val intent = Intent(Settings.*ACTION\_LOCATION\_SOURCE\_SETTINGS*)  
 startActivity(intent)  
 }  
  
 // открытие настроек приложения  
 private fun openAppSettings() {  
 val intent = Intent(Settings.*ACTION\_APPLICATION\_DETAILS\_SETTINGS*).*apply* **{** *data* = Uri.fromParts("package", *packageName*, null)  
 **}** startActivity(intent)  
 }  
  
 // возобновление отслеживания локации при возврате в приложение  
 override fun onResume() {  
 super.onResume()  
 if (hasLocationPermissions()) {  
 startLocationUpdates()  
 }  
 }  
  
 // остановка отслеживания локации при уходе из приложения  
 override fun onPause() {  
 super.onPause()  
 stopLocationUpdates()  
 }  
  
 // остановка получения обновлений локации  
 private fun stopLocationUpdates() {  
 try {  
 locationManager.removeUpdates(this)  
 } catch (e: SecurityException) {  
  
 }  
 }  
  
 // обработчик включения провайдера локации  
 override fun onProviderEnabled(provider: String) {  
 Toast.makeText(this, "Провайдер $provider включен", Toast.*LENGTH\_SHORT*).show()  
 }  
  
 // обработчик отключения провайдера локации  
 override fun onProviderDisabled(provider: String) {  
 Toast.makeText(this, "Провайдер $provider отключен", Toast.*LENGTH\_SHORT*).show()  
 }  
   
 override fun onStatusChanged(provider: String?, status: Int, extras: Bundle?) {}  
}

Данный код представляет собой основную активность приложения для поиска точек по координатам, которая управляет всей логикой игры: обрабатывает запросы разрешений на доступ к геолокации, получает и отслеживает текущее местоположение пользователя через LocationManager, генерирует случайные целевые точки в радиусе 2.5 км, вычисляет расстояние между текущей и целевой позицией с использованием формулы гаверсинусов, обновляет интерфейс с отображением координат и расстояния, а также определяет победу при приближении к цели менее чем на 100 метров.

# Результаты работы

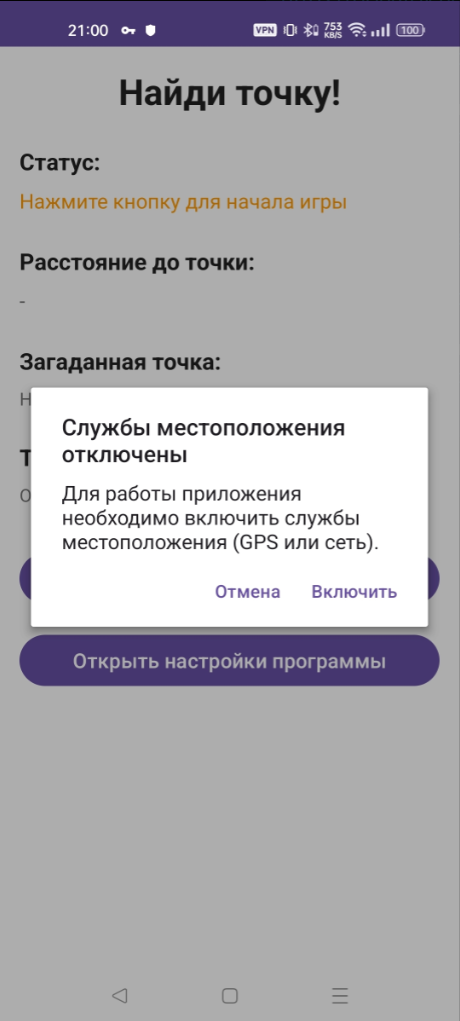


Рисунок 3. Запрос о включении службы местоположения

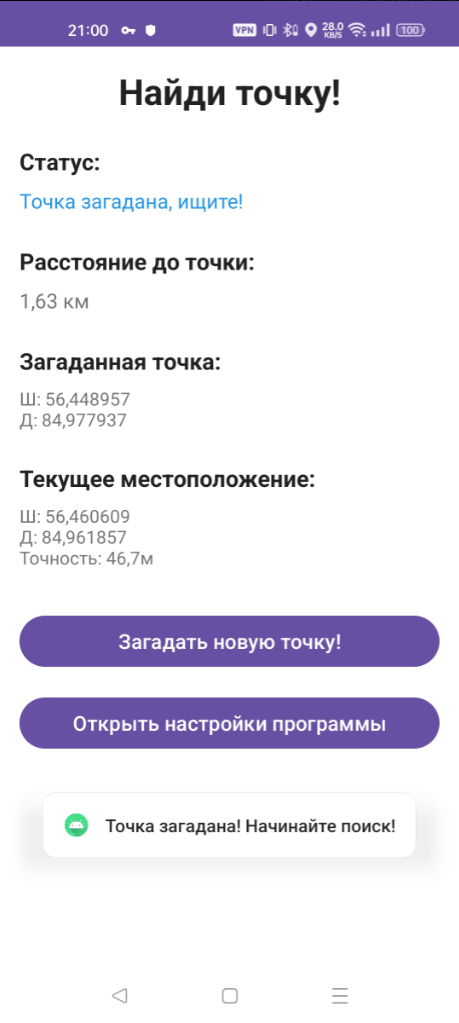


Рисунок 4. Нажатие на кнопку “Загадать новую точку!”

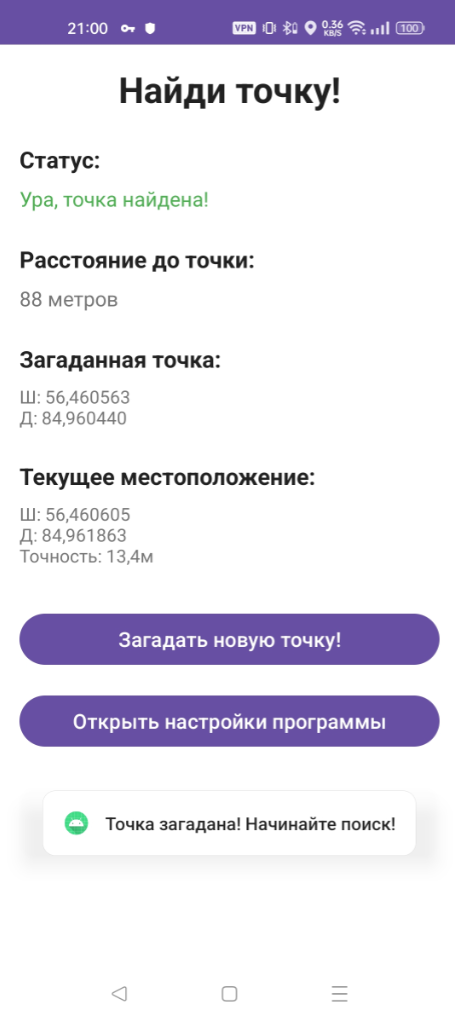


Рисунок 5. Нахождение загаданной точки

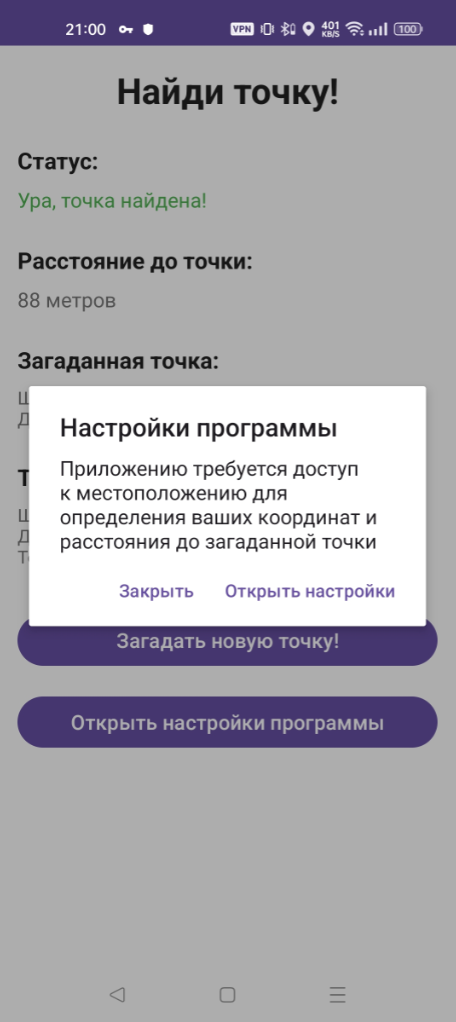


Рисунок 6. Нажатие на кнопку “Открыть настройки программы”

# Выводы

В ходе выполнения лабораторной работы была разработана игра "Найди точку!", которая использует геолокацию в основе игрового процесса. Приложение генерирует случайную точку в радиусе 2.5 км от пользователя и предлагает найти её, отслеживая расстояние до цели в реальном времени.