МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение

высшего образования «Уральский федеральный университет

имени первого Президента России Б.Н. Ельцина»

ИТОГОВАЯ АТТЕСТАЦИОННАЯ РАБОТА

на тему

«Разработка системы мониторинга и управления элементами «Умного дома» на базе МК ESP-32»

по дополнительной профессиональной программе –

программе профессиональной переподготовки

«Программирование современных логических устройств   
(на базе языка программирования С++)»»

(наименование программы)

Группа в составе:

1. Слушатель \_\_\_\_\_\_\_\_\_\_\_\_ Ветров М.А.

2. Слушатель \_\_\_\_\_\_\_\_\_\_\_\_ Митрофанов И.О.

(подпись, дата)

Руководитель Байтимиров Д.Р  
(к.ф.-м.н., доцент) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись, дата)

Допустить к защите

«\_\_\_» \_\_\_\_\_\_\_\_\_ 2025 г. Руководитель программы Моисейкин Е.В.

«\_\_\_» \_\_\_\_\_\_\_\_\_ 2025 г. Секретарь Цижмовска Н.Л.

Екатеринбург 2025

Оглавление

[1. ПАСПОРТ ПРОЕКТА 4](#_Toc199535017)

[2. ОПИСАНИЕ ПРОДУКТА ИЛИ УСЛУГИ 6](#_Toc199535018)

[3. АНАЛИТИКА РЫНКА 11](#_Toc199535019)

[4. ОСНОВНЫЕ ЭТАПЫ/ИТЕРАЦИИ РЕАЛИЗАЦИИ ПРОЕКТА 14](#_Toc199535020)

[5. ОРГАНИЗАЦИЯ РАБОТ В ПРОЕКТЕ 16](#_Toc199535021)

[6. РАЗДЕЛ ПО СПЕЦИФИКЕ ПРОГРАММЫ 17](#_Toc199535022)

[Заключение 30](#_Toc199535023)

[Библиографический список 31](#_Toc199535024)

[Приложение A – Программный код микроконтроллера на Arduino C++ 32](#_Toc199535025)

[Приложение Б – Программный код приложения для анализа данных 53](#_Toc199535026)

**Министерство науки и высшего образования Российской Федерации**

**ФГАОУ ВО «УрФУ имени первого Президента России Б.Н. Ельцина»**

|  |  |
| --- | --- |
| СОГЛАСОВАНО  УрФУ ФТИ  « » \_\_\_\_\_\_\_\_\_\_\_\_ 2025 | СОГЛАСОВАНО  УрФУ  « » \_\_\_\_\_\_\_\_\_\_ 2025 |
| Ответственный за стажировку/практику на предприятии \_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Подпись расшифровка подписи | Руководитель ДПП ПП Моисейкин Евгений Витальевич  \_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Подпись расшифровка подписи |

Подразделение ДПО «Цифровая кафедра» ФТИ Группа ЦК-24-101-001

Наименование ДПП ПП «Программирование современных логических устройств (на базе языка программирования С++)»

**ИНДИВИДУАЛЬНОЕ/ГРУППОВОЕ ЗАДАНИЕ НА СТАЖИРОВКУ/ПРАКТИКУ**

**(оставить нужное)**

*Ветров Максим Алексеевич, Митрофанов Иван Олегович*

(фамилия, имя, отчество)

1. Тема задания на Стажировку/практику: *Разработка системы мониторинга и управления элементами «Умного дома» на базе МК ESP-32*

2. Срок стажировки/практики с 05.05.2025 г. по 25.05.2025 г.

Срок сдачи студентом отчета 26.05 – 31.05.2025г.

3. Место прохождения стажировки/практики: \_\_УрФУ ФТИ

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

4. Содержание отчета: *Паспорт проекта, описание продукта, аналитика рынка, основные этапы реализации проекта, организация работ в проекте, раздел по специфике программы.*

**Рабочий график (план)проведения практики**

|  |  |  |  |
| --- | --- | --- | --- |
| Этапы | Наименование работ студента | Срок | Примечание |
| 1 | Согласование индивидуального задания с  руководителями, оформление документации. | 05.05.2025 –  07.05.2025 |  |
| 2 | Выполнение индивидуального задания, ежедневная работа по месту стажировки, мероприятия по сбору и анализу информации. Обсуждение проблем и поиск решений на  семинарах. | 08.05.2025 –  20.05.2025 |  |
| 3 | Подведение итогов и составление отчета: систематизация, анализ, обработка собранного в ходе стажировки  материала, предоставление отчета. | 21.05.2025 –  25.05.2025 |  |

Содержание стажировки/практики и планируемые результаты согласованы с руководителем от профильной организации

Руководитель от УрФУ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Моисейкин Е.В.

Руководитель от предприятия \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Байтимиров Д.Р.

Задание принял к исполнению(студент) \_\_\_\_\_\_\_\_\_\_\_\_\_ Ветров М.А.

Подпись расшифровка подписи

\_\_\_\_\_\_\_\_\_\_\_\_\_ *Митрофанов И.О.*

Подпись расшифровка подписи

# **1. ПАСПОРТ ПРОЕКТА**

Паспорт проекта «Разработка системы мониторинга и управления элементами «Умного дома» на базе МК ESP-32», реализуемого в рамках модуля практической подготовки по программе ДПП ПП «Программирование современных логических устройств (на базе языка программирования С++)»

|  |
| --- |
| Информация о заказчике |
| Предприятие: УрФУ ФТИ |
| Предприятие: УрФУ ФТИ |
| Направление подготовки 09.03.03 Прикладная информатика |
| Программирование современных логических устройств (на базе языка программирования С++) |
| Период реализации проекта с 01.09.2025 по 31.05.2025 |
|  |
| О проекте |
| Название проекта  *Разработка системы мониторинга и управления элементами «Умного дома» на базе МК ESP-32* |
| Цель проекта  *Разработать прототип устройства, для этого необходимо разработать устройство и программное обеспечение для определения положения человека в комнате с помощью датчика HLK-LD2450. Программное обеспечение микроконтроллера должно опрашивать датчик, выводить положение человека на дисплей и сохранять координаты на карту памяти. Актуальное время для координат необходимо получать с использованием Wi-Fi. С помощью кнопки на микроконтроллере ESP32 необходимо организовать создание и удаление зон с действиями. Разработать программное обеспечение для анализа координат перемещения человека и координат зон при подключении карты памяти к компьютеру.* |
| Результат проекта  *Команда проекта должна разработать программное обеспечение и продемонстрировать его функционирование с использованием микроконтроллерного стенда, включающего дисплей, SD card Adapter и датчик отслеживания движения цели HLK-LD2450. Разработать программное обеспечение для анализа данных, сохранённых на SD-карту.* |
| Критерии приемки  *Презентация проекта.*  *Оценка корректности функционирования устройства в соответствии с техническим заданием.*  *Раздел по специфике программы должен включать следующие составляющие:*   * *структурная схема установки с описанием ее работы;* * *блок-схема и описание работы программы;* * *исходные тексты разработанных функций с подробными комментариями.* |
| Описание проекта  *Предоставляемая аппаратная часть системы включает в себя:*   * *микроконтроллерный стенд на основе esp32 с установленным SD card Adapter и дисплеем;* * *датчик отслеживания движения цели HLK-LD2450.*   *С помощью разработанного программного обеспечения в начале работы микроконтроллер должен выполнить инициализацию дисплея и SD card Adapter. Считывать сохранённые координаты зон с карты памяти и выводить на дисплей. В основном цикле должен опрашивать датчик, осуществлять запись координат человека на карту памяти. В дополнительном цикле должен быть реализован механизм создания и удаления зон с действиями.*  *ПО на компьютере должно анализировать сохранённые координаты, человека и зон и создавать отчет.* |
| Тип проекта |
| Тип проекта  *Прикладной* |
| Количество команд и участников (студентов) |
| Количество участников  *2 студента в одной команде* |
| Требования к функциональным ролям студентов в проекте |
| *Роль 1: Руководитель проекта, Программист микроконтроллеров*  *Роль 2: Разработчик ПО* |
| Требования к роли 1   * *Знать основные функции языка С++ и уметь применять их;* * *Уметь пользоваться интегрированной средой разработки (Arduino IDE);* * *Уметь разрабатывать и отлаживать программы для микроконтроллеров на базе ESP32;* * *Иметь опыт работы с периферийными устройствами и датчиками;* * *Уметь настраивать и использовать протоколы связи (Wi-Fi);*   Требования к роли 2   * *Знать основные функции языка Python и уметь применять их;* * *Уметь разрабатывать и отлаживать программы на Python с графическим интерфейсом с использованием библиотеки (tkinter);* * *Уметь визуализировать данные с использованием библиотеки (matplotlib);* * *Уметь пользоваться системами контроля версий (Git);* |
| Список формируемых компетенций |
| Список формируемых компетенций  ПК 1. Разрабатывать программные алгоритмы для обработки сигналов и данных для управления периферическими электронными и механическими устройствами с широким применением микропроцессорной техники,  ПК 2. Разрабатывать прикладное программное обеспечение систем автоматизации и управления научно-исследовательского и промышленного назначения,  ПК 3. Проводить инсталляцию и настройку системного, прикладного и инструментального программного обеспечения систем автоматизации и управления научно-исследовательского и промышленного назначения,  ПК 4. Применяет языки программирования,  ПК 5. Разрабатывает аппаратные решения в области микропроцессорной и цифровой техники,  ПК 6. Разрабатывает и настраивает автоматические системы управления. |
| Требуемые от заказчика ресурсы |
| Требуемые от профильной организации ресурсы  Для разработки программного обеспечения системы мониторинга и управления элементами «Умного дома»:   * *Микроконтроллер Esp32 c платой расширения ESP32 DEVKIT V1;* * *Датчика отслеживания движения цели hlk-ld2450, SD card Adapter, дисплей ST7789V2 240x320;* * *ПК с установленной интегрированной средой разработки Arduino IDE;* |

# **2. ОПИСАНИЕ ПРОДУКТА ИЛИ УСЛУГИ**

**2.1 Компоненты и схема подключения**

Разрабатываемое устройство представляет собой систему на базе микроконтроллера ESP32, предназначенную для определения положения человека в помещении с использованием датчика HLK-LD2450. Система обеспечивает сбор, хранение и первичную визуализацию координат пользователя, а также предоставляет возможность задания пользовательских зон наблюдения.

В состав аппаратной части входят следующие основные элементы: ESP32, Плата расширения ESP32 30P DEVKIT V1, датчик HLK-LD2450, SD Card Adapter, дисплей ST7789V2 240x320.

ESP32 — это мощный микроконтроллер с двумя ядрами Xtensa LX6, работающими на частоте до 240 МГц. Он поддерживает Wi-Fi и Bluetooth, что делает его идеальным выбором для встраиваемых систем и устройств Интернета вещей. (см. рисунок 1).



Рисунок 1 – микроконтроллер ESP32

Плата расширения ESP32 30P DEVKIT V1 предназначена для упрощения разработки и тестирования проектов на базе микроконтроллера ESP32. Она предоставляет удобные разъемы для подключения различных периферийных устройств и модулей, а также имеет встроенные схемы питания и защиты, что обеспечивает стабильную работу устройства. Плата поддерживает все основные интерфейсы ESP32 и совместима с популярными средами разработки, такими как Arduino IDE и PlatformIO (см. рисунок 2).

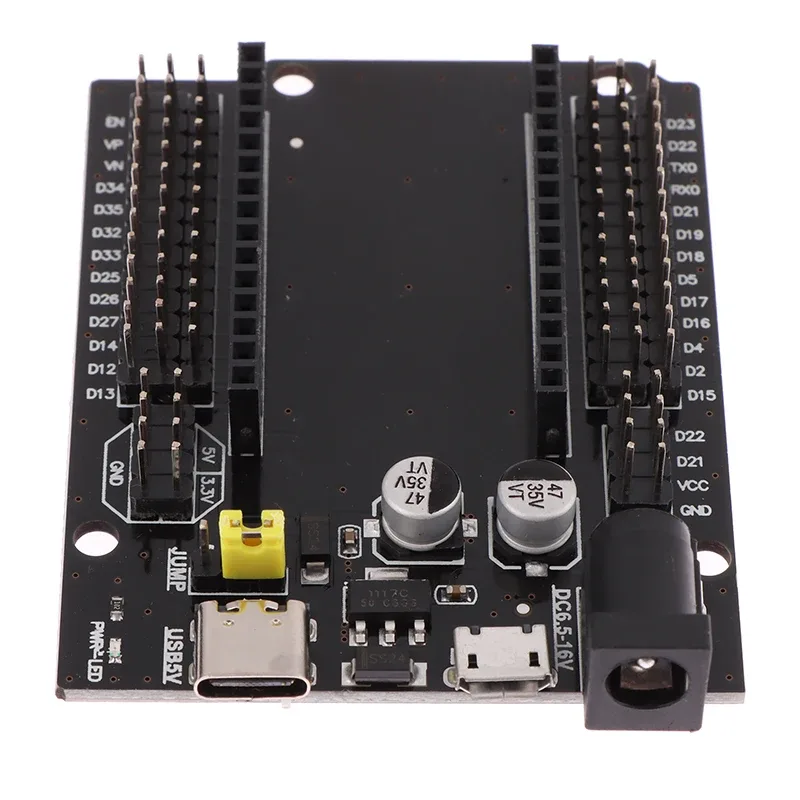


Рисунок 2 – Плата расширения ESP32 30P DEVKIT V1

HLK-LD2450 — это датчик миллиметрового диапазона, работающий на частоте 24 ГГц, который способен определять координаты человека в двумерном пространстве с высокой точностью. Датчик имеет диапазон обнаружения до 6 метров и может определять скорость движения объектов. Он потребляет мало энергии и имеет компактные размеры, что делает его удобным для встраивания в различные устройства (см. рисунок 3).

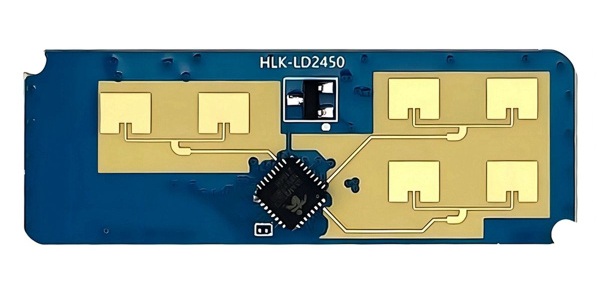


Рисунок 3 – Датчик HLK-LD2450

SD Card Adapter — это модуль, предназначенный для чтения и записи данных на карты памяти формата SD, SDHC и SDXC. Он обеспечивает совместимость с различными микроконтроллерами через интерфейс SPI, что позволяет легко интегрировать его в различные электронные устройства. Модуль поддерживает скорость передачи данных до 25 МБ/с, что обеспечивает быстрый доступ к хранимой информации и эффективное управление данными (см. рисунок 4).

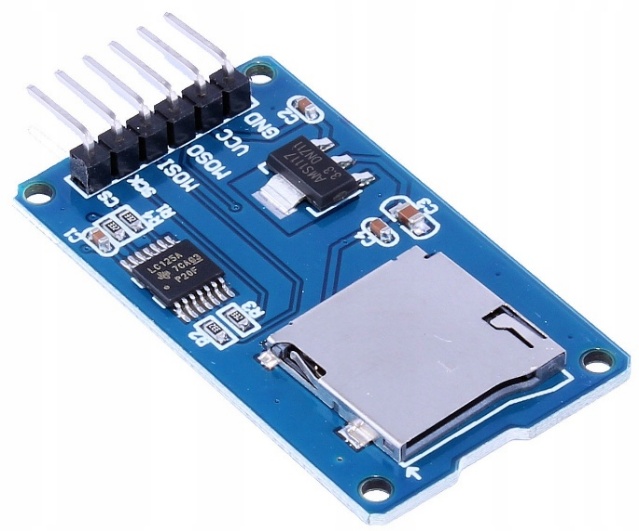


Рисунок 4 – SD Card Adapter

Дисплей ST7789V2 с разрешением 240x320 пикселей основан на технологии TFT и использует интерфейс SPI для связи с микроконтроллером. Он имеет яркость до 200 кд/м² и поддерживает отображение до 262 тысяч цветов. Дисплей обладает низким энергопотреблением и высокой контрастностью, что делает его идеальным для использования в портативных устройствах (см. рисунок 5).

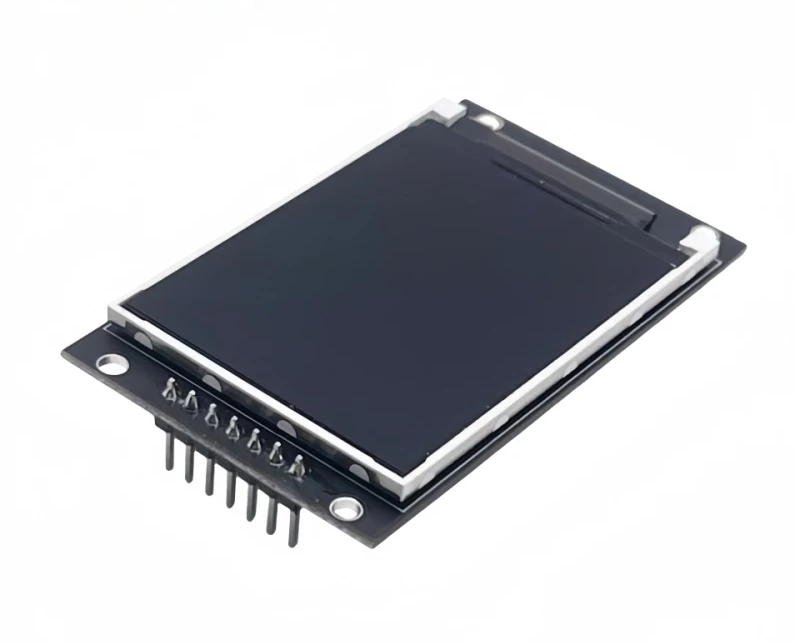


Рисунок 5 – Дисплей ST7789V2

**2.2 Функциональные возможности устройства**

Устройство непрерывно отслеживает положение человека в помещении и сохраняет координаты на SD-карту в текстовом формате. Актуальное время для координат микроконтроллер получает с помощью Wi-Fi. Был реализован механизм задания пользовательских зон с помощью кнопки boot на микроконтроллере. Данные о зонах сохраняются в отдельный файл и автоматически загружаются на дисплей при включении устройства. Текущее положение, а также границы зон отображаются на экране устройства в режиме реального времени.

**2.3 Программное обеспечение анализа данных**

Отдельное приложение, написанное на языке Python, предназначено для обработки и анализа данных, полученных от устройства. Программа выполняет чтение координат и зон из файлов, предоставляя пользователю возможность назначить каждой зоне имя и логическую роль. Конфигурация зон сохраняется в отдельный конфигурационный файл, что обеспечивает удобство и гибкость в настройке. Приложение проводит анализ перемещений пользователя, генерируя отчеты по временным интервалам и зонам, и формирует итоговый текстовый отчет, содержащий интерпретацию активности, что позволяет пользователю получать детальную и понятную информацию о своих перемещениях и активностях.

**2.4 Область применения**

Разработанная система может применяться в следующих направлениях:

* Исследование поведенческих моделей в жилых помещениях;
* Прототипирование решений для систем «умный дом»;
* Анализ повседневной активности.

# **3. АНАЛИТИКА РЫНКА**

**3.1. Значение цифры в отрасли (на которую ориентирована программа ДПП) и конкретной решаемой задачи**

Современные цифровые технологии играют ключевую роль в развитии систем мониторинга и автоматизации повседневной жизни, особенно в таких областях, как «умный дом», здравоохранение, уход за пожилыми людьми, обеспечение безопасности и персональная аналитика поведения.

В частности, рост интереса к технологиям позиционирования и трекинга в помещениях обусловлен потребностью в:

* Повышении уровня комфорта и автоматизации в жилых и офисных пространствах;
* Мониторинге активности и благополучия пожилых людей и пациентов в медицинских учреждениях;
* Сборе поведенческой информации в исследовательских и коммерческих целях;
* Повышении энергоэффективности и безопасности в помещении.

По данным аналитических агентств, объем мирового рынка систем «умного дома» в году превысил 140 млрд долларов США, при этом ожидается рост до 225-250 млрд долларов к 2030 году. Существенную долю в этом сегменте занимают именно технологии отслеживания активности и нахождения человека внутри помещения, включая **motion tracking** (слежка за движением)**, presence detection** (обнаружение присутствия)и **behavior analytics** (аналитика поведения).

Разрабатываемое решение направлено на решение узкой, но актуальной задачи – фиксация координат человека в помещении с последующей интерпретацией активности на основе зонального анализа. Это особенно актуально в контексте задач, связанных с:

* Анализом образа жизни (сон, работа, отдых);
* Обеспечением ухода и мониторинга без вмешательства в личное пространство;
* Формированием отчетов о повседневной активности на основе данных с датчиков.

Таким образом, разработка данного продукта укладывается в глобальный тренд цифровизации быта и медицины, при этом затрагивает аспекты персональной аналитики и автоматизированного поведенческого мониторинга, являющиеся перспективными и востребованными направлениями.

**3.2. Исследование существующих решений на рынке, являющихся аналогами разрабатываемого продукта/услуги**

На сегодняшний день на рынке представлены различные решения для мониторинга присутствия и активности в помещении. Ниже приведены наиболее близкие по функциональности технологии и устройства:

1. **Устройства на базе PIR-датчиков (датчики движения)**

Пример: Aqara Motion Sensor, Philips Hue Motion Sensor.

* Плюсы: простота, низкая стоимость, энергоэффективность.
* Минусы: фиксируют только факт движения, без точных координат. Нет возможности зонального анализа.

1. **Системы на базе видеонаблюдения с ИИ-анализом**

Пример: Google Nest Cam, Amazon Ring, Arlo.

* Плюсы: высокая точность анализа поведения, наличие видеофиксации.
* Минусы: высокая стоимость, проблемы с конфиденциальностью, зависимость от интернета.

1. **Коммерческие системы Indoor Positioning на базе BLE/UWB**

Пример: Pozyx, Ubisense, Sewio.

* Плюсы: высокая точность позиционирования, масштабируемость.
* Минусы: сложность настройки, высокая стоимость оборудования, необходимость носимых меток (в большинстве случаев).

1. **Системы на базе радарных датчиков**

Пример: mmWave sensors от Texas Instruments, HLK-LD2410 и LD2450 DIY-сборки

* Плюсы: бесконтактное определение положения, устойчивость к освещению, высокая точность.
* Минусы: редкая интеграция с системами логического анализа и пользовательским интерфейсом.

# **4. ОСНОВНЫЕ ЭТАПЫ/ИТЕРАЦИИ РЕАЛИЗАЦИИ ПРОЕКТА**

**4.1. Блок описания решенных задач проекта по каждому студенту**

Программист микроконтроллеров (Ветров М.А.): разборка аппаратной части и прошивки микроконтроллера. Обязанности и выполненные задачи:

* Разработка схемы устройства на базе ESP32;
* Подключение и конфигурация компонентов: датчик HLK-LD2450, дисплей, SD Card Adapter, управляющая кнопка;
* Реализация логики прошивки на языке C++ (Arduino IDE):
  + Чтение координат с датчика и сохранение на SD-карту;
  + Визуализация текущего положения на экране;
  + Реализация механизма создания и удаления зон;
  + Сохранение информации о зонах и загрузка при старте.
* Отладка, тестирование и оптимизация работы устройства.

Разработчик ПО (Митрофанов И.О.): разработка программного обеспечения для анализа данных:

* Проектирование и реализация аналитической программы на языке Python;
* Разработка модуля чтения файлов координат и конфигураций зон, сохраненных на SD-карте;
* Разработка интерактивного интерфейса (GUI) для:
  + Просмотра и именования пользовательских зон;
  + Назначения ролей зонам (сон, работа, отдых).
* Реализация логики анализа перемещений пользователя по зонам и временным интервалам;
* Генерация текстового отчета с выводами об активности: анализ сна, работы.
* Проведение тестирования программы на примерах реальных данных.
  1. **Результаты проекта с указанием вклада каждого студента**

Итоговые результаты проекта:

* Разработано полноценное устройство на базе ESP32, способное в реальном времени отслеживать координаты человека в помещении и сохранять их на SD-карту;
* Реализован пользовательский механизм управления зонами через дисплей и кнопку;
* Разработано настольное программное обеспечение на Python, позволяющее:
  + Интерпретировать данные с устройства;
  + Настраивать зоны и роли;
  + Анализировать поведение и активность пользователя;
  + Формировать структурированный отчет.

# **5. ОРГАНИЗАЦИЯ РАБОТ В ПРОЕКТЕ**

Для реализации проекта были получены требуемые от профильной организации ресурсы: микроконтроллер ESP32 с платой расширения ESP32 DEVKIT1, датчик отслеживания HLK-LD2450, SD Card Adapter, дисплей ST7789V2 240x320.

Были распределены роли: 1 программист микроконтроллера, 1 программист ПО для ПК. Для реализации использовалась итерационная модель разработки со следующими этапами:

1. Постановка задачи и определение функционала системы;
2. Разработка аппаратной части и прошивки;
3. Разработка программного обеспечения для анализа данных;
4. Интеграция и тестирование;
5. Подготовка документации и отчета.

Взаимодействие между участниками происходило на регулярной основе. Использовались система контроля версий Git и мессенджеры для согласования изменений. Финальная интеграция обеспечивалась посредством обмена тестовыми данными и сборки релизной версии ПО.

# **РАЗДЕЛ ПО СПЕЦИФИКЕ ПРОГРАММЫ**

**6.1. Структурная схема установки**

Построим схему подключения компонентов к микроконтроллеру ESP32, используя плату разработки ESP32 DEVKIT V1. Начнем с подключения LCD-дисплея к микроконтроллеру через интерфейсы CS, SCK, MOSI и MISO, что обеспечит передачу данных и управление дисплеем. Далее подключим микросхему HLK-LD2450, используя контакты GND для заземления, RX для приема данных и 5V для питания. Затем подключим дисплей ST7789V2, используя контакты BK, MO, SCK, RS, RST, CS и GND, что позволит управлять дисплеем и передавать на него данные. Также необходимо подключить аналого-цифровые преобразователи (ADC) и GPIO контакты микроконтроллера к соответствующим компонентам системы для обеспечения их взаимодействия. В результате получим схему, где микроконтроллер ESP32 будет централизованно управлять всеми подключенными компонентами, обеспечивая их корректную работу и взаимодействие (см. рисунок 6).

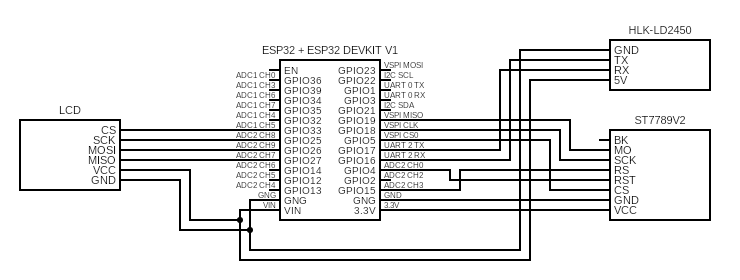


Рисунок 6 – Схема подключения компонентов к микроконтроллеру ESP32

**6.1 Алгоритм работы программ**

Были разработаны специальные алгоритмы для решения поставленных задач, связанных с настройкой и управлением зонами с использованием микроконтроллера, SD карты и датчика HLK-LD2450. Эти алгоритмы обеспечивают корректную инициализацию периферийных устройств микроконтроллера. После успешной инициализации программа проверяет подключение SD карты, что позволяет сохранить и загрузить необходимые данные. В случае отсутствия SD карты на экран выводится соответствующее сообщение, и процесс останавливается, что предотвращает возможные ошибки в работе системы (см. рисунок 7).

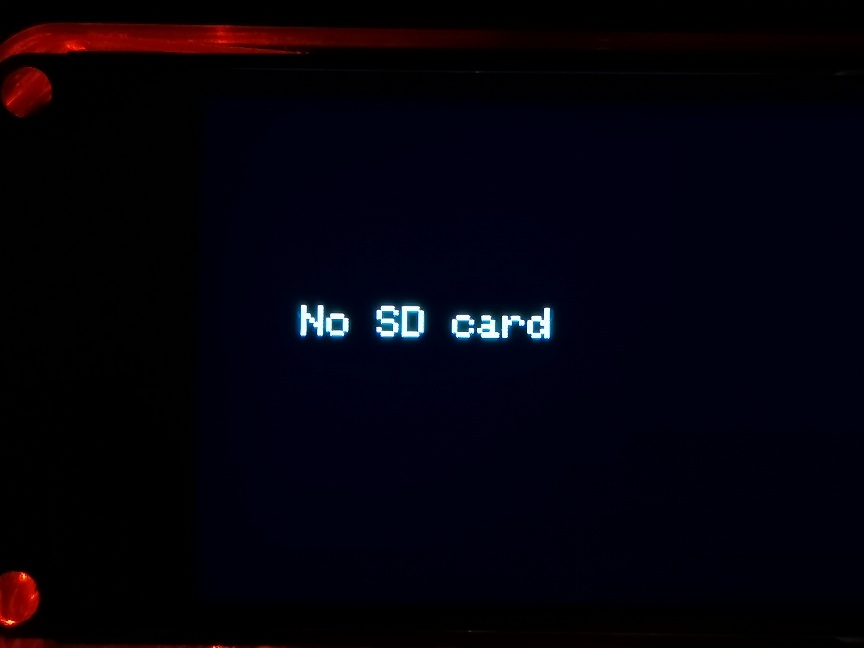


Рисунок 7 – Предупреждение нет SD карты

После проверки подключения SD-карты устройство подключается к сети Wi-Fi для получения актуального времени. На дисплей выводятся зоны, сохраненные на SD карте. (см. рисунок 8).

Изображение выглядит как снимок экрана, свет

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 8 – Загрузка зон с карты памяти и отображение текущего положения человека

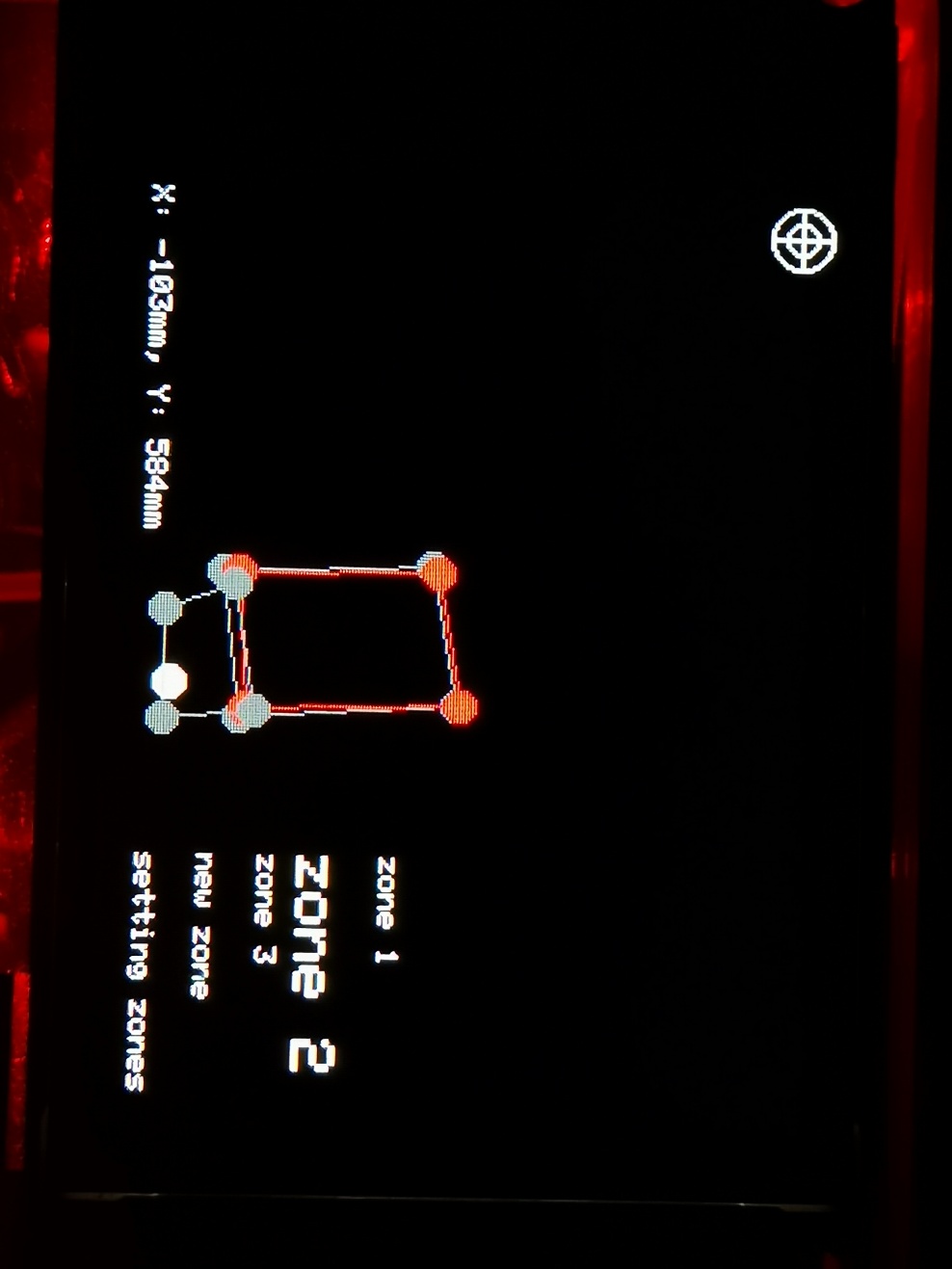
Далее происходит конфигурация датчика HLK-LD2450, что включает в себя настройку его параметров для корректной работы. Программа проверяет, было ли долгое нажатие, и если такое нажатие было, процесс переходит к настройке зон. В противном случае, датчик начинает передавать координаты человека, которые затем выводятся на дисплей и сохраняются на SD карту. Это обеспечивает постоянный мониторинг и запись данных, что является важным для анализа и управления зонами. При переходе к настройке зон на дисплей выводится сообщение "setting zones"(см. рисунок 9).

Изображение выглядит как текст, снимок экрана, черный

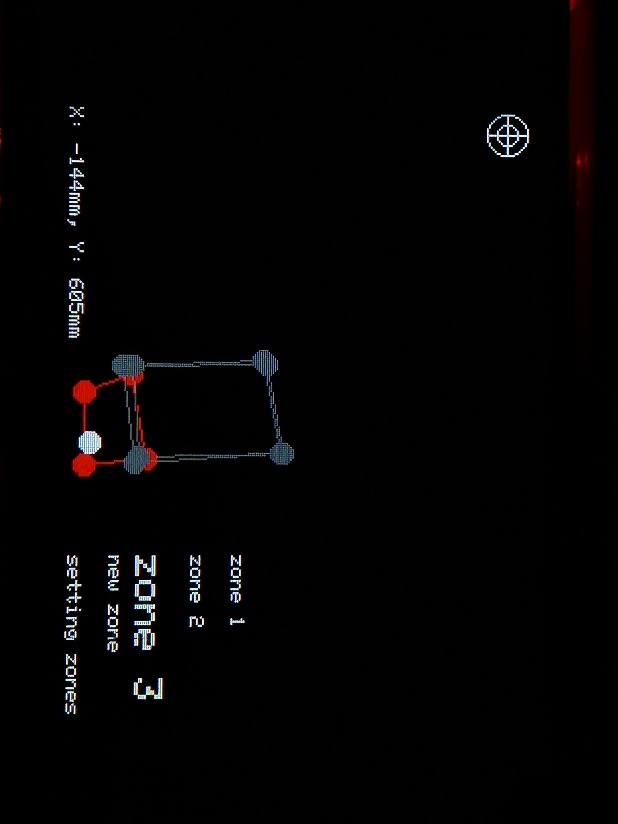
Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 9 – Вход в меню настроек

Пользователь может взаимодействовать с меню. Если было одиночное нажатие в меню, то выбирается следующий пункт. Выбранные зоны подсвечиваются красным (см. рисунок 10 а, б).



а)



б)

Рисунок 10 а, б– Переключение между пунктами меню

После двойного нажатия на New zone программа предлагает указать количество точек для зоны. С помощью одинарного нажатия количество точек можно увеличивать (см. рисунок 11).

Изображение выглядит как текст, снимок экрана, Шрифт, дизайн

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 11 – Выбор количества точек у новой зоны

Двойным нажатием запускается механизм создания зоны. На экране пишется: номер точки dot и сколько времени осталось до создания точки (см. рисунок 12).

Изображение выглядит как текст, снимок экрана, Шрифт, черный

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 12 – Создание точки для зоны

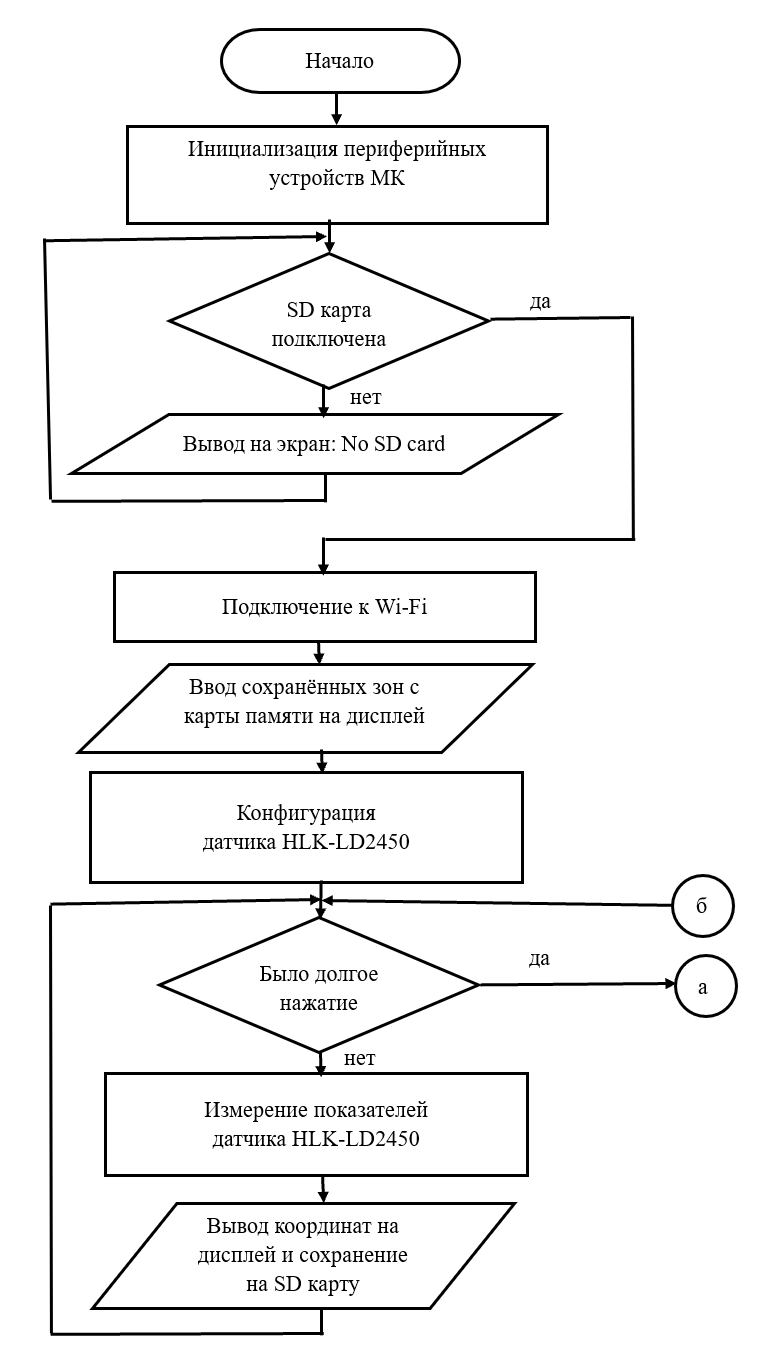
Если было двойное нажатие на существующую зону, она удаляется с дисплея и SD карты (см. рисунок 13)

Изображение выглядит как текст, снимок экрана

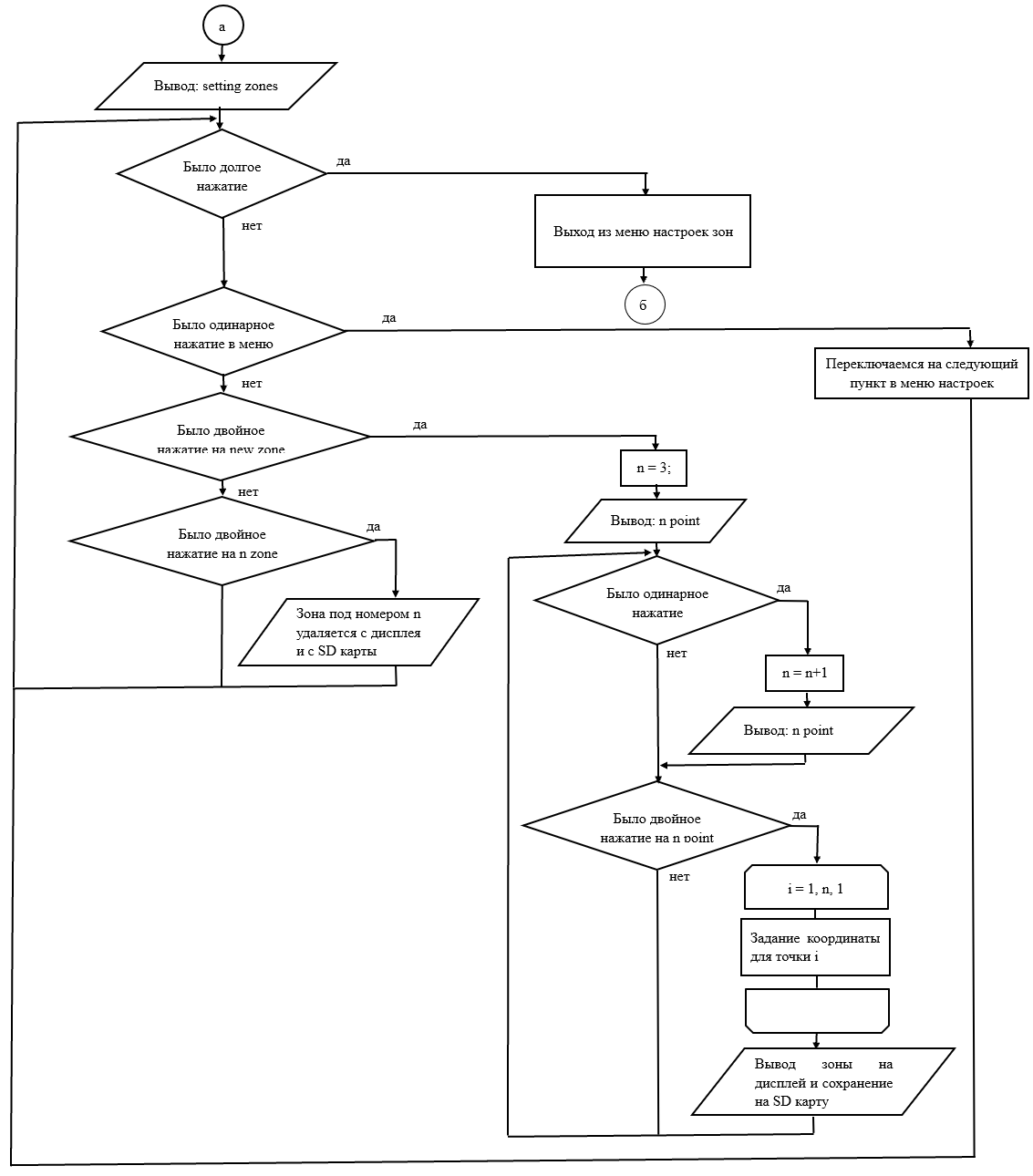
Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 13 – Удаление зоны

Таким образом, разработанные алгоритмы обеспечивают полный цикл настройки и управления зонами (см. приложение А), предоставляя пользователю удобный интерфейс для взаимодействия и настройки системы (см. рисунок 14 а, б).



а)



б)

Рисунок 14 а, б – Блок-схема алгоритма программы на микроконтроллере

Был разработан алгоритм программы для анализа сохраненных данных на карту памяти, который позволяет эффективно обрабатывать и визуализировать информацию о зонах и ролях в графическом интерфейсе пользователя (GUI). Этот алгоритм начинается с запуска приложения и инициализации GUI интерфейса, что подготавливает главное окно программы к работе. После успешной инициализации пользователю дается выбор из пяти функций: «Ввод зон, ролей и имен», «Анализ по координатам», «Визуализация зон и движения», «Найденные зоны» и «Очистить консоль».

При первом запуске пользователю необходимо запустить функцию «Найденные зоны» для проверки корректности отображения найденных зон. Зонам выдаются временные метки, которые будут указывать на ту или иную зону при выборе имен и ролей для зон (см. рисунок 15, 16).

Изображение выглядит как текст, снимок экрана, дисплей, программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 15 – Основное окно приложения

Изображение выглядит как текст, диаграмма, линия, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 16 – Функция «Найденные зоны»

Далее за ввод имен и ролей для зон и сохранение их конфигураций отвечает функция «Ввод зон, ролей и имен». При запуске программа будет запрашивать имена и роли зонам, которые обозначаются временными метками с предыдущей функции. После ввода информации по всем зонам, информация сохраняется в конфигурационный файл «roles.txt», который будет необходим для следующих функций (см. рисунок 17, 18).

Изображение выглядит как текст, электроника, снимок экрана, дисплей

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 17 – Функция «Ввод зон, ролей и имен»

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 18 – Пример roles.txt

Для визуализации зон и движений программа проверяет наличие файлов coordinates.txt и roles.txt. Если эти файлы существуют, запускается алгоритм визуализации, и выводятся окна с траекториями движений и зонами. Это позволяет пользователю наглядно увидеть и проанализировать данные, что значительно упрощает процесс принятия решений на основе полученной информации (см. рисунок 19).

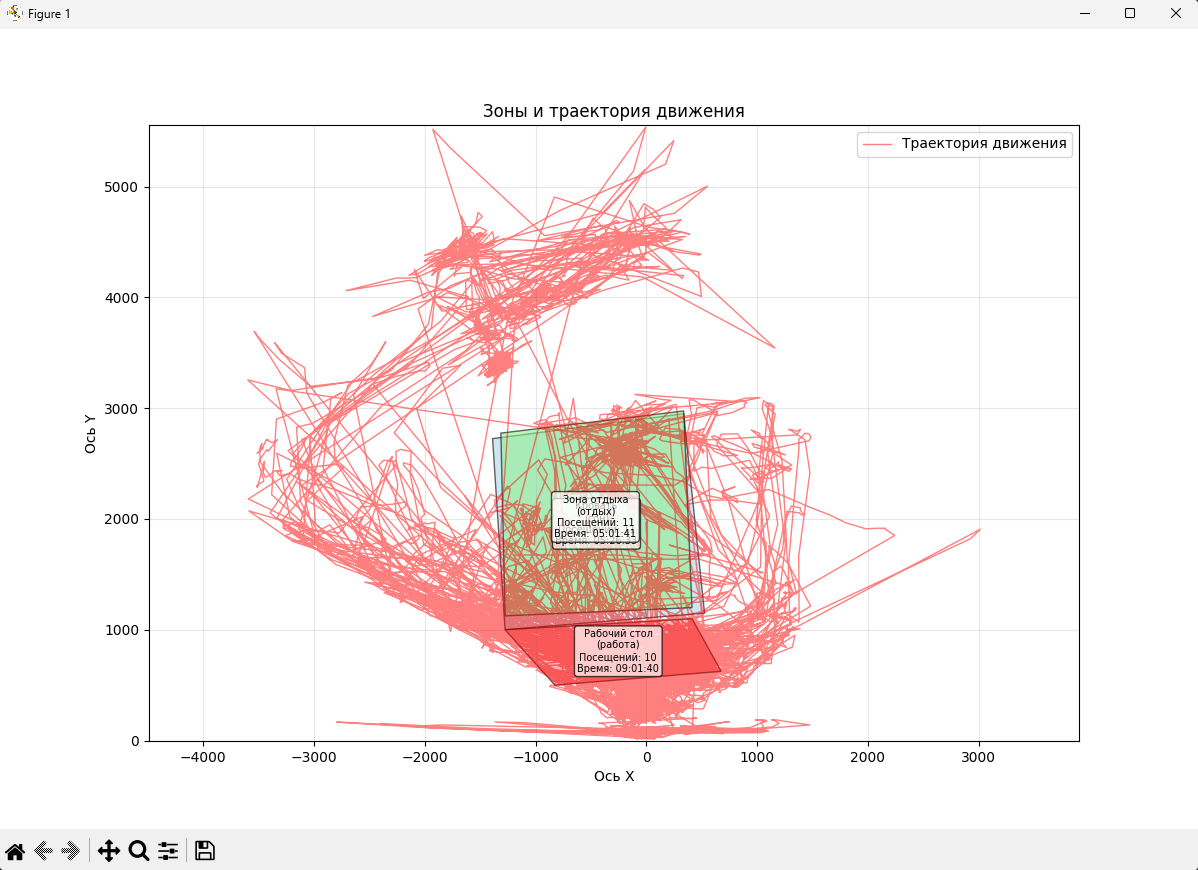


Рисунок 19 – Функция «Визуализация зон и движения»

Для получения текстового отчета по активности пользователя существует функция «Анализ по координатам». Данная функция анализирует информацию из coordinates.txt и roles.txt и составляет информацию по нахождению пользователя в помещении и его активности в зонах (см. рисунок 20).

Изображение выглядит как текст, снимок экрана, Шрифт, дизайн

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 20 – Пример текстового отчета

Таким образом, разработанный алгоритм обеспечивает комплексный подход к анализу и визуализации данных о зонах и ролях, предоставляя пользователю широкие возможности для обработки информации и генерации отчетов (см. приложение Б). Это делает программу мощным инструментом для работы с данными, сохраненными на карту памяти (см. рисунок 21).

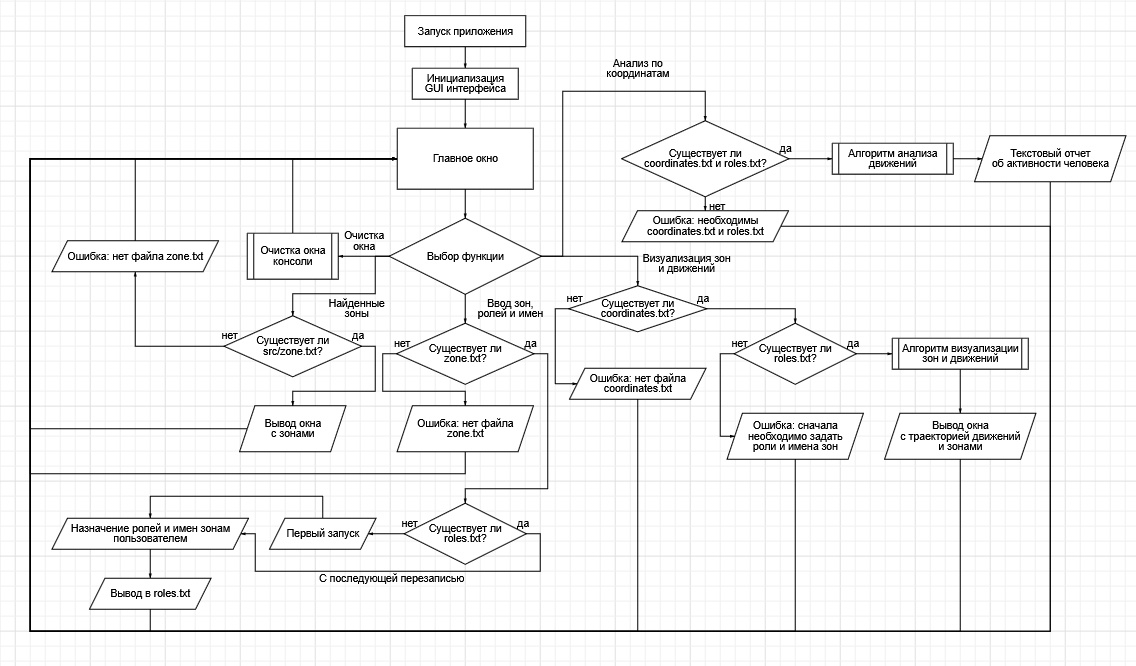


Рисунок 21 – Блок-схема алгоритма программы для анализа сохраненных данных

# **Заключение**

В ходе выполнения курсового проекта была успешно реализована система, предназначенная для определения положения человека в помещении и анализа его активности с использованием данных, получаемых от радиолокационного датчика HLK-LD2450. Проект включал как разработку аппаратной части, так и программной аналитической части.

В результате была создана работоспособная система, демонстрирующая практическое применение микроконтроллеров, сенсорных технологий и инструментов анализа данных. Решение отличается конфиденциальностью (отсутствие видеонаблюдения), точностью и простотой реализации.

Проект подтвердил важность командной работы, итерационного подхода к разработке и комплексного применения знаний в области схемотехники, программирования микроконтроллеров и анализа данных. Полученные результаты могут быть использованы как основа для дальнейших исследований или доработки в полноценную систему мониторинга поведения для бытового или исследовательского применения.

# **Библиографический список**

1. Arduino IDE: Руководство пользователя [Электронный ресурс] // Официальный сайт Arduino. – URL: https://www.arduino.cc/en/software (дата обращения: 30.05.2025).
2. Espressif Systems. ESP32 Technical Reference Manual [Электронный ресурс]. – URL: https://www.espressif.com/sites/default/files/documentation/esp32\_technical\_reference\_manual\_en.pdf (дата обращения: 30.05.2025).
3. HLK-LD2450 3D Radar Module: Data Sheet and Integration Guide [Электронный ресурс] // Hi-Link. – URL: https://www.hlktech.net/product\_detail/HLK-LD2450 (дата обращения: 30.05.2025).
4. Arduino Language Reference (C/C++) [Электронный ресурс] // Arduino.cc. – URL: https://www.arduino.cc/reference/en/ (дата обращения: 30.05.2025).
5. Python Software Foundation. Python 3.12 Documentation [Электронный ресурс]. – URL: https://docs.python.org/3/ (дата обращения: 30.05.2025).
6. Hunter, J. D. Matplotlib: A 2D Graphics Environment // Computing in Science & Engineering. – 2007. – Vol. 9, No. 3. – P. 90–95.
7. Tkinter – Python Interface to Tcl/Tk GUI Toolkit [Электронный ресурс] // Python Docs. – URL: https://docs.python.org/3/library/tkinter.html (дата обращения: 30.05.2025).
8. Statista. Smart Home Market – Global Revenue Forecast [Электронный ресурс]. – URL: https://www.statista.com/outlook/tmo/smart-home/worldwide (дата обращения: 30.05.2025).
9. WHO. Physical activity and adult health recommendations [Электронный ресурс] // World Health Organization. – URL: https://www.who.int/news-room/fact-sheets/detail/physical-activity (дата обращения: 30.05.2025).
10. McKinsey & Company. The Future of Work After COVID-19 [Электронный ресурс] // McKinsey Global Institute. – URL: https://www.mckinsey.com/featured-insights/future-of-work/the-future-of-work-after-covid-19 (дата обращения: 30.05.2025).

# Приложение A – Программный код микроконтроллера на Arduino C++

#include <Arduino.h>

#include <LD2450.h>

#include <Adafruit\_GFX.h>

#include <Adafruit\_ST7789.h>

#include <SPI.h>

#include <vector>

#include "FS.h"

#include "SD.h"

#include <TimeLib.h>

#include <WiFi.h>

#include <NTPClient.h>

#include <WiFiUdp.h>

// Определение серого цвета

#define ST77XX\_GRAY 0x8C51 // Пример определения серого цвета

// Определение пинов для подключения SD-карты

#define SD\_CS 33

#define SD\_SCLK 25

#define SD\_MOSI 26

#define SD\_MISO 27

// Объявление пинов для дисплея ST7789V2

#define TFT\_CS 5 // CS

#define TFT\_RST 4 // RST

#define TFT\_DC 15 // DC (RS)

#define TFT\_MOSI 19 // MOSI

#define TFT\_SCLK 18 // SCLK

// Пин для кнопки Boot

#define BOOT\_BUTTON\_PIN 0

// Пин для светодиода

#define LED\_PIN 2

// Создаем объект для работы с дисплеем

Adafruit\_ST7789 tft = Adafruit\_ST7789(TFT\_CS, TFT\_DC, TFT\_MOSI, TFT\_SCLK, TFT\_RST);

// Создаем объект для работы с лидаром

LD2450 ld2450;

// Флаг для режима настройки

bool setupMode = false;

// Флаг для режима удаления

bool deleteMode = false;

// Переменные для отслеживания состояния кнопки

volatile unsigned long buttonPressStartTime = 0;

volatile bool buttonPressed = false;

volatile bool buttonHandled = false;

volatile unsigned long lastPressTime = 0;

volatile bool flagRaised = false;

volatile unsigned long flagRaiseTime = 0;

const unsigned long debounceDelay = 200; // Задержка для устранения дребезга

const unsigned long doublePressDelay = 500; // Задержка для определения двойного нажатия

const unsigned long longPressDelay = 2000; // Задержка для определения длинного нажатия (2 секунды)

// Переменные для отслеживания состояния отображения и количества точек

bool displayNewZone = true;

int pointCount = 3; // Изменено значение по умолчанию на 3

bool zoneCreationMode = false;

unsigned long zoneCreationStartTime = 0;

int countdown = 5; // Начальное значение отсчета

bool showCheckmark = false;

unsigned long checkmarkStartTime = 0;

// Переменные для хранения предыдущих координат и состояния отображения

int prevX[3] = {-1, -1, -1}, prevY[3] = {-1, -1, -1};

bool prevDisplayNewZone = true;

int prevPointCount = 3; // Изменено значение по умолчанию на 3

int prevTargetX = 0, prevTargetY = 0;

bool prevZoneCreationMode = false;

int prevCountdown = 5;

bool prevShowCheckmark = false;

// Динамические массивы для хранения координат красных точек

std::vector<int> redPointsX;

std::vector<int> redPointsY;

// Флаг для отслеживания сохранения зоны

bool zoneSaved = false;

// Вектор для хранения зон

std::vector<std::vector<std::pair<int, int>>> zones;

// Флаг для отслеживания наличия SD-карты

bool sdCardPresent = true;

// Количество зон на карте памяти

int ZoneCount = 0;

// Флаг для отслеживания необходимости обновления меню

bool menuNeedsUpdate = true;

// Переменная для отслеживания текущей выбранной зоны

int selectedZone = -1;

// Флаг для отслеживания текущей жирной надписи

bool isNewZoneBold = false;

bool isZoneBold = false;

int boldZoneNumber = -1;

// Флаг для отслеживания состояния, когда "delete" написано жирным, а "new zone" не жирным

bool deleteBoldAndNewZoneNotBold = false;

// Флаг для отслеживания состояния, когда "new zone" написано жирным

bool newZoneBold = false;

// Переменная для отслеживания номера зоны, которую нужно удалить

int zoneToDelete = -1;

// WiFi и NTP настройки

String ssid = "";

String password = "";

int utcOffset = 0;

WiFiUDP ntpUDP;

NTPClient timeClient(ntpUDP, "pool.ntp.org");

// Флаг для отслеживания состояния подключения к WiFi

bool wifiConnected = false;

// Флаг для отслеживания отображения текста "point"

bool showPointText = true;

void IRAM\_ATTR handleButtonPress() {

if (digitalRead(BOOT\_BUTTON\_PIN) == LOW) {

buttonPressStartTime = millis();

buttonPressed = true;

buttonHandled = false;

} else {

if (buttonPressed && !buttonHandled && (millis() - lastPressTime > debounceDelay)) {

if (millis() - lastPressTime < doublePressDelay) {

Serial.println("Double press");

if (setupMode) {

if (zoneCreationMode) {

zoneCreationMode = false;

} else {

if (isNewZoneBold) {

// Если жирная надпись "new zone", меняем на "1 point"

displayNewZone = false;

isNewZoneBold = false;

isZoneBold = false;

boldZoneNumber = -1;

newZoneBold = true; // Устанавливаем флаг

} else if (isZoneBold) {

// Если жирная надпись "zone X", меняем на "delete"

deleteMode = true;

zoneToDelete = boldZoneNumber; // Запоминаем номер зоны для удаления

tft.fillRect(tft.width() - 100, tft.height() - 40 - (ZoneCount - boldZoneNumber + 1) \* 20, 100, 20, ST77XX\_BLACK);

tft.setCursor(tft.width() - 100, tft.height() - 40 - (ZoneCount - boldZoneNumber + 1) \* 20);

tft.setTextColor(ST77XX\_WHITE);

tft.setTextSize(2); // Увеличиваем размер шрифта для жирного отображения

tft.print("delete");

isZoneBold = false;

boldZoneNumber = -1;

deleteBoldAndNewZoneNotBold = true; // Устанавливаем флаг

} else {

zoneCreationMode = true;

zoneCreationStartTime = millis();

countdown = 5; // Сбрасываем отсчет

showPointText = false; // Сбрасываем флаг отображения текста "point"

// Очищаем предыдущие точки при входе в режим создания зоны

redPointsX.clear();

redPointsY.clear();

}

}

}

flagRaised = false; // Сбрасываем флаг, чтобы не выводить "Short press"

} else {

if (!flagRaised) {

flagRaised = true;

flagRaiseTime = millis();

}

}

buttonHandled = true;

lastPressTime = millis();

}

buttonPressed = false; // Сбрасываем флаг нажатия, если кнопка отпущена

}

}

uint16\_t getRandomZoneColor(int zoneIndex) {

// Массив предопределенных ярких цветов (можно добавить больше)

const uint16\_t zoneColors[] = {

ST77XX\_RED, ST77XX\_GREEN, ST77XX\_BLUE, ST77XX\_YELLOW,

ST77XX\_MAGENTA, ST77XX\_CYAN, 0x07E0, 0xF81F, 0xFFE0, 0x07FF

};

// Используем индекс зоны для выбора цвета (по модулю количества цветов)

return zoneColors[zoneIndex % (sizeof(zoneColors) / sizeof(zoneColors[0]))];

}

void saveAllZonesToSD() {

if (!sdCardPresent) return;

// Удаляем старый файл

SD.remove("/src/zone.txt");

// Создаем новый файл и записываем все зоны

File file = SD.open("/src/zone.txt", FILE\_WRITE);

if (!file) {

Serial.println("Failed to open file for writing");

return;

}

for (const auto& zone : zones) {

for (size\_t i = 0; i < zone.size(); i++) {

if (i > 0) file.print(";");

file.print(zone[i].first);

file.print(",");

file.print(zone[i].second);

}

file.println();

}

file.close();

Serial.println("All zones saved to SD card");

}

void saveZoneNewToSD(int zoneNumber) {

if (!sdCardPresent) return;

// Открываем исходный файл для чтения

File srcFile = SD.open("/src/zone.txt");

if (!srcFile) {

Serial.println("Failed to open source file for reading");

return;

}

// Создаем новый файл zone\_new.txt для записи

File destFile = SD.open("/src/zone\_new.txt", FILE\_WRITE);

if (!destFile) {

Serial.println("Failed to open destination file for writing");

srcFile.close();

return;

}

int currentLine = 1;

while (srcFile.available()) {

String line = srcFile.readStringUntil('\n');

// Пропускаем строку с номером zoneNumber

if (currentLine != zoneNumber) {

destFile.print(line);

// Добавляем перенос строки

destFile.print("\n");

}

currentLine++;

}

srcFile.close();

destFile.close();

Serial.println("Zone file copied to zone\_new.txt with specified line omitted");

// Удаляем старый файл zone.txt

if (SD.remove("/src/zone.txt")) {

Serial.println("Old zone.txt file removed");

} else {

Serial.println("Failed to remove old zone.txt file");

}

// Переименовываем zone\_new.txt в zone.txt

if (SD.rename("/src/zone\_new.txt", "/src/zone.txt")) {

Serial.println("zone\_new.txt renamed to zone.txt");

} else {

Serial.println("Failed to rename zone\_new.txt to zone.txt");

}

// Очищаем экран и перерисовываем все как при включении

tft.fillScreen(ST77XX\_BLACK);

// Перечитываем зоны с SD-карты

zones.clear();

readZonesFromSD();

ZoneCount = zones.size();

// Сбрасываем все флаги и состояния

setupMode = false;

displayNewZone = true;

pointCount = 3; // Изменено значение по умолчанию на 3

zoneCreationMode = false;

selectedZone = -1;

deleteMode = false;

isNewZoneBold = false;

isZoneBold = false;

boldZoneNumber = -1;

deleteBoldAndNewZoneNotBold = false;

newZoneBold = false;

showPointText = true;

redPointsX.clear();

redPointsY.clear();

// Отрисовываем WiFi иконку

drawWiFiIcon(wifiConnected);

// Если SD-карта не обнаружена, выводим сообщение

if (!sdCardPresent) {

tft.setCursor(50, 100);

tft.setTextColor(ST77XX\_WHITE);

tft.setTextSize(2);

tft.print("No SD card");

}

}

void saveCoordinatesToSD(int x, int y) {

if (!sdCardPresent) {

return;

}

// Открываем файл для записи координат

File file = SD.open("/src/coordinates.txt", FILE\_APPEND);

if (!file) {

Serial.println("Failed to open file for writing");

return;

}

// Получаем текущее время

String timeStr = "";

if (wifiConnected) {

time\_t now = timeClient.getEpochTime();

struct tm \* timeinfo = localtime(&now);

char timeStrBuf[20];

sprintf(timeStrBuf, "%04d:%02d:%02d:%02d:%02d:%02d", timeinfo->tm\_year + 1900, timeinfo->tm\_mon + 1, timeinfo->tm\_mday, timeinfo->tm\_hour, timeinfo->tm\_min, timeinfo->tm\_sec);

timeStr = String(timeStrBuf);

}

// Записываем координаты и время в файл

file.print(x);

file.print(",");

file.print(y);

file.print(",");

file.print(timeStr);

file.println();

file.close();

Serial.println("Coordinates saved to SD card");

}

void readWiFiFromSD() {

if (!sdCardPresent) {

return;

}

File file = SD.open("/src/Wi-Fi.txt");

if (!file) {

Serial.println("Failed to open Wi-Fi file for reading");

return;

}

while (file.available()) {

String line = file.readStringUntil('\n');

if (line.startsWith("name:")) {

ssid = line.substring(5);

ssid.trim(); // Удаляем лишние пробелы и символы новой строки

Serial.print("SSID: ");

Serial.println(ssid);

} else if (line.startsWith("password:")) {

password = line.substring(9);

password.trim(); // Удаляем лишние пробелы и символы новой строки

Serial.print("Password: ");

Serial.println(password);

} else if (line.startsWith("UTC:")) {

utcOffset = line.substring(4).toInt();

Serial.print("UTC Offset: ");

Serial.println(utcOffset);

}

}

file.close();

Serial.println("Wi-Fi credentials read from SD card");

}

void drawWiFiIcon(bool connected) {

int x = 10;

int y = 10;

int size = 20;

// Стираем предыдущий значок WiFi

tft.fillRect(x - size, y - size, size \* 2, size \* 2, ST77XX\_BLACK);

// Рисуем значок WiFi

if (connected) {

// Рисуем подключенный значок WiFi

tft.drawCircle(x, y, size / 2, ST77XX\_WHITE);

tft.drawCircle(x, y, size / 4, ST77XX\_WHITE);

tft.drawLine(x, y - size / 2, x, y + size / 2, ST77XX\_WHITE);

tft.drawLine(x - size / 2, y, x + size / 2, y, ST77XX\_WHITE);

} else {

// Рисуем зачеркнутый значок WiFi

tft.drawCircle(x, y, size / 2, ST77XX\_WHITE);

tft.drawCircle(x, y, size / 4, ST77XX\_WHITE);

tft.drawLine(x, y - size / 2, x, y + size / 2, ST77XX\_WHITE);

tft.drawLine(x - size / 2, y, x + size / 2, y, ST77XX\_WHITE);

tft.drawLine(x - size / 2, y - size / 2, x + size / 2, y + size / 2, ST77XX\_WHITE);

}

}

void setup() {

// Инициализация последовательного порта для отладки

Serial.begin(115200);

// Настраиваем пин кнопки Boot как вход с подтяжкой вверх

pinMode(BOOT\_BUTTON\_PIN, INPUT\_PULLUP);

// Настройка прерывания для кнопки Boot

attachInterrupt(digitalPinToInterrupt(BOOT\_BUTTON\_PIN), handleButtonPress, CHANGE);

// Инициализация последовательного порта для отладки

Serial1.begin(256000, SERIAL\_8N1, 16, 17);

while (!Serial) {

Serial.println("wait for serial port to connect");

}

while (!Serial1) {

Serial.println("wait for serial1 port to connect");

}

// Инициализация дисплея

tft.init(240, 320); // Инициализация с размером 240x320

tft.setRotation(1); // Установите ориентацию дисплея по необходимости

tft.fillScreen(ST77XX\_BLACK);

// Инициализация светодиода

pinMode(LED\_PIN, OUTPUT);

digitalWrite(LED\_PIN, LOW);

// Инициализация SD-карты

SPI.begin(SD\_SCLK, SD\_MISO, SD\_MOSI, SD\_CS);

if (!SD.begin(SD\_CS, SPI, 4000000)) {

Serial.println("Card Mount Failed");

sdCardPresent = false;

tft.fillScreen(ST77XX\_BLACK);

tft.setCursor(50, 100);

tft.setTextColor(ST77XX\_WHITE);

tft.setTextSize(2);

tft.print("No SD card");

return;

}

uint8\_t cardType = SD.cardType();

if (cardType == CARD\_NONE) {

Serial.println("No SD card attached");

sdCardPresent = false;

tft.fillScreen(ST77XX\_BLACK);

tft.setCursor(50, 100);

tft.setTextColor(ST77XX\_WHITE);

tft.setTextSize(2);

tft.print("No SD card");

return;

}

// Чтение данных Wi-Fi из файла на SD-карте

readWiFiFromSD();

// Подключение к WiFi

WiFi.begin(ssid.c\_str(), password.c\_str());

Serial.print("Connecting to WiFi: ");

Serial.println(ssid);

Serial.print("With password: ");

Serial.println(password);

unsigned long startTime = millis();

while (WiFi.status() != WL\_CONNECTED && millis() - startTime < 10000) { // Пытаемся подключиться в течение 10 секунд

delay(500);

Serial.print(".");

drawWiFiIcon(false); // Рисуем зачеркнутый значок WiFi

}

if (WiFi.status() == WL\_CONNECTED) {

Serial.println("WiFi connected");

wifiConnected = true;

drawWiFiIcon(true); // Рисуем значок WiFi

// Инициализация NTP клиента

timeClient.begin();

timeClient.setTimeOffset(3600 \* utcOffset); // Устанавливаем смещение времени

while (!timeClient.update()) {

timeClient.forceUpdate();

}

} else {

Serial.println("WiFi not connected");

wifiConnected = false;

drawWiFiIcon(false); // Рисуем зачеркнутый значок WiFi

}

// Чтение зон из файла на SD-карте

readZonesFromSD();

// Настройка лидара

byte enConfig[LD2450\_SERIAL\_BUFFER] = "";

enConfig[0] = 0xFD;

enConfig[1] = 0xFC;

enConfig[2] = 0xFB;

enConfig[3] = 0xFA;

enConfig[4] = 0x04;

enConfig[5] = 0x00;

enConfig[6] = 0xFF;

enConfig[7] = 0x00;

enConfig[8] = 0x01;

enConfig[9] = 0x00;

enConfig[10] = 0x04;

enConfig[11] = 0x03;

enConfig[12] = 0x02;

enConfig[13] = 0x01;

Serial1.write(enConfig, sizeof(enConfig));

while (Serial1.available() <= 0) {

Serial.print('.'); // send a capital A

delay(1);

}

while (Serial1.available()) {

char inChar = (char)Serial1.read();

Serial.print(inChar);

}

byte endConfig[LD2450\_SERIAL\_BUFFER] = "";

endConfig[0] = 0xFD;

endConfig[1] = 0xFC;

endConfig[2] = 0xFB;

endConfig[3] = 0xFA;

endConfig[4] = 0x02;

endConfig[5] = 0x00;

endConfig[6] = 0xFE;

endConfig[7] = 0x00;

endConfig[8] = 0x04;

endConfig[9] = 0x03;

endConfig[10] = 0x02;

endConfig[11] = 0x01;

Serial1.write(endConfig, sizeof(endConfig));

while (Serial1.available() <= 0) {

Serial.print('.'); // send a capital A

delay(1);

}

while (Serial1.available()) {

char inChar = (char)Serial1.read();

Serial.print(inChar);

}

ld2450.begin(Serial1, true);

if (!ld2450.waitForSensorMessage()) {

Serial.println("SENSOR CONNECTION SEEMS OK");

} else {

Serial.println("SENSOR TEST: GOT NO VALID SENSORDATA - PLEASE CHECK CONNECTION!");

}

}

void drawCheckmark() {

int centerX = tft.width() / 2;

int centerY = tft.height() / 2;

int size = 50; // Размер галочки

// Включаем светодиод

digitalWrite(LED\_PIN, HIGH);

// Рисование жирной галочки

for (int i = -2; i <= 2; i++) {

tft.drawLine(centerX - size / 2, centerY + i, centerX, centerY + size / 2 + i, ST77XX\_GREEN);

tft.drawLine(centerX, centerY + size / 2 + i, centerX + size, centerY - size + i, ST77XX\_GREEN);

}

}

void eraseCheckmark() {

int centerX = tft.width() / 2;

int centerY = tft.height() / 2;

int size = 150; // Увеличенный размер области стирания

// Стирание галочки

tft.fillRect(centerX - size / 2, centerY - size / 2, size, size, ST77XX\_BLACK);

// Выключаем светодиод

digitalWrite(LED\_PIN, LOW);

}

void saveZoneToSD() {

if (!sdCardPresent) {

return;

}

// Создаем новую зону и добавляем в нее текущие красные точки

std::vector<std::pair<int, int>> newZone;

for (size\_t i = 0; i < redPointsX.size(); i++) {

// Преобразуем координаты из пикселей обратно в миллиметры

int x\_mm = map(redPointsX[i], 0, tft.width(), -6000, 6000);

int y\_mm = map(redPointsY[i], 0, tft.height(), 6000, 0);

newZone.push\_back(std::make\_pair(x\_mm, y\_mm));

}

// Добавляем новую зону в вектор зон

zones.push\_back(newZone);

File file = SD.open("/src/zone.txt", FILE\_APPEND);

if (!file) {

Serial.println("Failed to open file for writing");

return;

}

// Записываем координаты точек зоны в файл

for (size\_t i = 0; i < newZone.size(); i++) {

if (i > 0) {

file.print(";");

}

file.print(newZone[i].first);

file.print(",");

file.print(newZone[i].second);

}

file.println(); // Переход на новую строку для следующей зоны

file.close();

Serial.println("Zone coordinates saved to SD card");

ZoneCount++; // Увеличиваем количество зон после сохранения

}

void readZonesFromSD() {

if (!sdCardPresent) {

return;

}

File file = SD.open("/src/zone.txt");

if (!file) {

Serial.println("Failed to open file for reading");

return;

}

while (file.available()) {

String line = file.readStringUntil('\n');

if (line.length() > 0) {

std::vector<std::pair<int, int>> zone;

int start = 0;

int end = line.indexOf(';');

while (end != -1) {

String point = line.substring(start, end);

int comma = point.indexOf(',');

if (comma != -1) {

int x = point.substring(0, comma).toInt();

int y = point.substring(comma + 1).toInt();

zone.push\_back(std::make\_pair(x, y));

}

start = end + 1;

end = line.indexOf(';', start);

}

// Обработка последней точки в строке

String point = line.substring(start);

int comma = point.indexOf(',');

if (comma != -1) {

int x = point.substring(0, comma).toInt();

int y = point.substring(comma + 1).toInt();

zone.push\_back(std::make\_pair(x, y));

}

zones.push\_back(zone);

}

}

file.close();

Serial.println("Zones read from SD card");

ZoneCount = zones.size(); // Устанавливаем количество зон

}

void drawZones(int selectedZone = -1) {

for (size\_t i = 0; i < zones.size(); i++) {

uint16\_t color;

if (setupMode) {

// В режиме настройки - серый или красный для выбранной зоны

color = (i + 1 == selectedZone) ? ST77XX\_RED : ST77XX\_GRAY;

} else {

// Вне режима настройки - случайный цвет для каждой зоны

color = getRandomZoneColor(i);

}

for (size\_t j = 0; j < zones[i].size(); j++) {

// Преобразуем координаты с противоположным знаком X для отображения

int x = map(-zones[i][j].first, -6000, 6000, tft.width(), 0); // Отрицательный X для отображения

int y = map(zones[i][j].second, 0, 6000, tft.height(), 0);

tft.fillCircle(x, y, 5, color);

if (j > 0) {

int prevX = map(-zones[i][j - 1].first, -6000, 6000, tft.width(), 0); // Отрицательный X для отображения

int prevY = map(zones[i][j - 1].second, 0, 6000, tft.height(), 0);

tft.drawLine(prevX, prevY, x, y, color);

}

}

// Соединение последней и первой точки зоны

if (zones[i].size() > 1) {

int firstX = map(-zones[i][0].first, -6000, 6000, tft.width(), 0); // Отрицательный X для отображения

int firstY = map(zones[i][0].second, 0, 6000, tft.height(), 0);

int lastX = map(-zones[i][zones[i].size() - 1].first, -6000, 6000, tft.width(), 0); // Отрицательный X для отображения

int lastY = map(zones[i][zones[i].size() - 1].second, 0, 6000, tft.height(), 0);

tft.drawLine(lastX, lastY, firstX, firstY, color);

}

}

}

void displayZonesInMenu() {

tft.fillRect(tft.width() - 100, tft.height() - 40 - (ZoneCount + 1) \* 20, 100, (ZoneCount + 1) \* 20, ST77XX\_BLACK);

tft.setTextColor(ST77XX\_WHITE);

// Вывод зон в столбик

for (int i = ZoneCount; i > 0; i--) {

tft.setCursor(tft.width() - 100, tft.height() - 40 - (ZoneCount - i + 1) \* 20);

if (selectedZone == i) {

tft.setTextSize(2); // Увеличиваем размер шрифта для жирного отображения

if (deleteMode && selectedZone == i) {

tft.print("delete");

} else {

tft.print("zone ");

tft.print(i);

}

isZoneBold = true;

boldZoneNumber = i;

} else {

tft.setTextSize(1); // Уменьшаем размер шрифта для обычного отображения

tft.print("zone ");

tft.print(i);

}

}

// Очищаем область перед выводом "new zone"

tft.fillRect(tft.width() - 100, tft.height() - 40, 100, 20, ST77XX\_BLACK);

// Вывод "new zone" внизу

tft.setCursor(tft.width() - 100, tft.height() - 40);

if (selectedZone == -1) {

tft.setTextSize(2); // Увеличиваем размер шрифта для жирного отображения

isNewZoneBold = true;

isZoneBold = false;

boldZoneNumber = -1;

} else {

tft.setTextSize(1); // Уменьшаем размер шрифта для обычного отображения

isNewZoneBold = false;

}

tft.print("new zone");

}

void clearMenu() {

tft.fillRect(tft.width() - 100, tft.height() - 40 - (ZoneCount + 1) \* 20, 100, (ZoneCount + 1) \* 20 + 50, ST77XX\_BLACK);

}

void loop() {

// Обновляем время

if (wifiConnected) {

timeClient.update();

}

// Проверяем, не прошла ли секунда с момента поднятия флага

if (flagRaised && (millis() - flagRaiseTime >= doublePressDelay)) {

flagRaised = false;

Serial.println("Short press");

if (setupMode) {

if (displayNewZone && selectedZone == -1) {

// Если "new zone" выбрана и нажата, делаем "new zone" не жирным и выбираем zone 1

tft.fillRect(tft.width() - 100, tft.height() - 40, 100, 20, ST77XX\_BLACK); // Стираем жирную надпись "new zone"

selectedZone = 1;

menuNeedsUpdate = true;

} else if (!displayNewZone) {

pointCount++;

Serial.print("Point count: ");

Serial.println(pointCount);

} else {

// Переключаемся на следующую зону

if (selectedZone == ZoneCount) {

// Если текущая зона последняя, выбираем "new zone"

selectedZone = -1;

deleteMode = false; // Сбрасываем режим удаления

} else {

selectedZone = (selectedZone % ZoneCount) + 1;

deleteMode = false; // Сбрасываем режим удаления

}

menuNeedsUpdate = true;

}

}

}

// Если кнопка удерживается более 2 секунд

if (buttonPressed && (millis() - buttonPressStartTime >= longPressDelay)) {

setupMode = !setupMode; // Переключаем режим настройки

if (setupMode) {

Serial.println("Entering setup mode");

menuNeedsUpdate = true; // Устанавливаем флаг для обновления меню

} else {

Serial.println("Exiting setup mode");

displayNewZone = true; // Сбрасываем состояние отображения

pointCount = 3; // Изменено значение по умолчанию на 3

zoneCreationMode = false; // Сбрасываем режим создания зоны

clearMenu(); // Очищаем меню

selectedZone = -1; // Сбрасываем выбранную зону

deleteMode = false; // Сбрасываем режим удаления

isNewZoneBold = false;

isZoneBold = false;

boldZoneNumber = -1;

deleteBoldAndNewZoneNotBold = false; // Сбрасываем флаг

newZoneBold = false; // Сбрасываем флаг

showPointText = true; // Восстанавливаем флаг отображения текста "point"

}

buttonPressed = false; // Сбрасываем флаг нажатия

}

const int sensor\_got\_valid\_targets = ld2450.read();

if (sensor\_got\_valid\_targets > 0) {

for (int i = 0; i < ld2450.getSensorSupportedTargetCount(); i++) {

const LD2450::RadarTarget result\_target = ld2450.getTarget(i);

if (result\_target.valid) {

// Преобразование координат из миллиметров в пиксели с изменением знака X

int x = map(result\_target.x, -6000, 6000, tft.width(), 0);

int y = map(result\_target.y, 0, 6000, tft.height(), 0);

// Стираем предыдущий круг, если координаты изменились и это не красная точка

bool isRedPoint = false;

for (size\_t j = 0; j < redPointsX.size(); j++) {

if (redPointsX[j] == prevX[i] && redPointsY[j] == prevY[i]) {

isRedPoint = true;

break;

}

}

if (prevX[i] != -1 && prevY[i] != -1 && !isRedPoint) {

tft.fillCircle(prevX[i], prevY[i], 5, ST77XX\_BLACK);

}

// Рисование круга на дисплее

tft.fillCircle(x, y, 5, ST77XX\_WHITE); // Размер круга - 5 пикселей

// Обновляем предыдущие координаты

prevX[i] = x;

prevY[i] = y;

// Вывод координат в нижнем левом углу только для первой цели

if (i == 0) {

// Стираем предыдущие координаты

tft.setCursor(0, tft.height() - 20);

tft.setTextColor(ST77XX\_BLACK);

tft.setTextSize(1);

tft.print("X: ");

tft.print(prevTargetX);

tft.print("mm, Y: ");

tft.print(prevTargetY);

tft.print("mm");

// Вывод координат в нижнем левом углу

tft.setCursor(0, tft.height() - 20);

tft.setTextColor(ST77XX\_WHITE);

tft.setTextSize(1);

tft.print("X: ");

tft.print(result\_target.x);

tft.print("mm, Y: ");

tft.print(result\_target.y);

tft.print("mm");

// Обновляем предыдущие координаты

prevTargetX = result\_target.x;

prevTargetY = result\_target.y;

// Сохраняем координаты на SD-карту

saveCoordinatesToSD(result\_target.x, result\_target.y);

}

// Вывод надписи в правом нижнем углу

tft.setCursor(tft.width() - 100, tft.height() - 20);

tft.setTextColor(ST77XX\_WHITE);

tft.setTextSize(1);

if (setupMode) {

tft.print("setting zones");

// Вывод надписи "new zone" или "n dot: Lc" жирным шрифтом

if (menuNeedsUpdate) {

displayZonesInMenu();

menuNeedsUpdate = false;

}

tft.setCursor(tft.width() - 100, tft.height() - 40);

tft.setTextColor(ST77XX\_WHITE);

if (displayNewZone) {

tft.setTextSize(1); // Уменьшаем размер шрифта для обычного отображения

} else {

tft.setTextSize(2); // Увеличиваем размер шрифта для жирного отображения

}

if (displayNewZone) {

// Надпись "new zone" уже выведена в displayZonesInMenu

} else {

// Стираем надпись "new zone"

tft.fillRect(tft.width() - 100, tft.height() - 40, 100, 20, ST77XX\_BLACK);

if (zoneCreationMode) {

tft.print(pointCount);

tft.print("dot: ");

tft.print(countdown);

tft.print("c");

} else if (selectedZone != -1 && showPointText) {

tft.print(pointCount);

tft.print(" point");

}

}

// Обновляем предыдущие состояния

prevDisplayNewZone = displayNewZone;

prevPointCount = pointCount;

prevZoneCreationMode = zoneCreationMode;

prevCountdown = countdown;

prevShowCheckmark = showCheckmark;

} else {

clearMenu(); // Очищаем меню

tft.print(" ");

}

}

}

}

// Проверяем, не прошло ли 1 секунда с момента последнего обновления отсчета

static unsigned long lastCountdownUpdate = 0;

if (zoneCreationMode && newZoneBold && (millis() - lastCountdownUpdate >= 1000)) {

lastCountdownUpdate = millis();

countdown--;

if (countdown <= 0) {

countdown = 5;

pointCount--;

if (pointCount <= 0) {

pointCount = 3; // Изменено значение по умолчанию на 3

zoneCreationMode = false;

displayNewZone = true;

menuNeedsUpdate = true; // Устанавливаем флаг для обновления меню

showPointText = true; // Восстанавливаем флаг отображения текста "point"

}

}

if (countdown == 1) {

showCheckmark = true;

checkmarkStartTime = millis();

// Добавляем новые координаты в динамический массив

redPointsX.push\_back(prevX[0]);

redPointsY.push\_back(prevY[0]);

}

}

// Проверяем, не прошла ли 1 секунда с момента отображения галочки

if (showCheckmark && (millis() - checkmarkStartTime >= 1000)) {

showCheckmark = false;

eraseCheckmark();

}

// Если нужно отобразить галочку

if (showCheckmark) {

drawCheckmark();

}

// Рисование всех красных точек

for (size\_t i = 0; i < redPointsX.size(); i++) {

if (redPointsX[i] != -1 && redPointsY[i] != -1) {

tft.fillCircle(redPointsX[i], redPointsY[i], 5, ST77XX\_RED);

}

}

// Соединение красных точек линиями (если есть хотя бы 2 точки)

if (redPointsX.size() >= 2) {

for (size\_t i = 0; i < redPointsX.size(); i++) {

size\_t next\_i = (i + 1) % redPointsX.size(); // Следующая точка (для последней точки - первая)

// Проверяем, что обе точки валидны

if (redPointsX[i] != -1 && redPointsY[i] != -1 &&

redPointsX[next\_i] != -1 && redPointsY[next\_i] != -1) {

// Рисуем линию между текущей и следующей точкой

tft.drawLine(redPointsX[i], redPointsY[i],

redPointsX[next\_i], redPointsY[next\_i], ST77XX\_RED);

}

}

}

// Сохранение зоны на SD-карту, если pointCount = 1 и countdown = 1

if (pointCount == 1 && countdown == 1 && !zoneSaved && newZoneBold) {

saveZoneToSD();

zoneSaved = true;

menuNeedsUpdate = true; // Устанавливаем флаг для обновления меню

// Сбрасываем все флаги после создания зоны

displayNewZone = true;

zoneCreationMode = false;

showPointText = true;

isNewZoneBold = false;

isZoneBold = false;

deleteBoldAndNewZoneNotBold = false;

newZoneBold = false;

deleteMode = false;

// Очищаем динамические массивы для хранения координат красных точек

redPointsX.clear();

redPointsY.clear();

// Сбрасываем количество точек

pointCount = 3; // Изменено значение по умолчанию на 3

// Сбрасываем отсчет

countdown = 5;

// Сбрасываем выбранную зону

selectedZone = -1;

}

// Сбрасываем флаг сохранения зоны, если режим создания зоны завершен

if (!zoneCreationMode) {

zoneSaved = false;

}

// Отрисовка зон на переднем плане

drawZones(selectedZone);

// Проверяем, не нужно ли создать файл zone\_new.txt

if (deleteMode && zoneToDelete != -1) {

saveZoneNewToSD(zoneToDelete);

zoneToDelete = -1; // Сбрасываем номер зоны для удаления

}

// Обновляем отображение количества точек

if (!zoneCreationMode && !displayNewZone && selectedZone == -1) {

tft.fillRect(tft.width() - 100, tft.height() - 40, 100, 20, ST77XX\_BLACK);

tft.setCursor(tft.width() - 100, tft.height() - 40);

tft.setTextColor(ST77XX\_WHITE);

tft.setTextSize(2); // Увеличиваем размер шрифта для жирного отображения

tft.print(pointCount);

tft.print(" point");

}

}

# Приложение Б – Программный код приложения для анализа данных

import tkinter as tk

from tkinter import ttk, scrolledtext, messagebox

import threading

import queue

import os

import sys

import time

import ast

import math

from datetime import datetime, timedelta

import matplotlib.pyplot as plt

import matplotlib.patches as patches

ZONE\_FILE\_PATH = "src/zone.txt"

ROLES\_FILE\_PATH = "roles.txt"

ROLE\_OPTIONS = {

    "1": "сон",

    "2": "работа",

    "3": "отдых"

}

def read\_zones\_from\_file(filepath):

    zones = []

    try:

        with open(filepath, 'r', encoding='utf-8') as f:

            for line\_num, line in enumerate(f, 1):

                line = line.strip()

                if not line:

                    continue

                coord\_pairs\_str = line.split(';')

                if len(coord\_pairs\_str) < 3:

                    continue

                current\_zone\_points = []

                valid\_zone = True

                for i, pair\_str in enumerate(coord\_pairs\_str):

                    try:

                        x\_str, y\_str = pair\_str.split(',')

                        x = int(x\_str.strip())

                        y = int(y\_str.strip())

                        current\_zone\_points.append((x, y))

                    except ValueError:

                        valid\_zone = False

                        break

                if valid\_zone:

                    zones.append(current\_zone\_points)

    except FileNotFoundError:

        print(f"Ошибка: Файл '{filepath}' не найден.")

    except Exception as e:

        print(f"Произошла ошибка при чтении файла '{filepath}': {e}")

    return zones

def display\_zones(zones\_data, zone\_details=None):

    if not zones\_data:

        return

    fig, ax = plt.subplots()

    ax.set\_aspect('equal', adjustable='datalim')

    all\_x\_coords = []

    all\_y\_coords = []

    for i, zone\_points in enumerate(zones\_data):

        polygon = patches.Polygon(zone\_points, closed=True, edgecolor='blue', facecolor='lightblue', alpha=0.7)

        ax.add\_patch(polygon)

        for x, y in zone\_points:

            all\_x\_coords.append(x)

            all\_y\_coords.append(y)

        label\_text = f"Зона {i+1}"

        if zone\_points:

            cx = sum(p[0] for p in zone\_points) / len(zone\_points)

            cy = sum(p[1] for p in zone\_points) / len(zone\_points)

            ax.text(cx, cy, label\_text, ha='center', va='center', color='black', fontsize=8)

    if not all\_x\_coords or not all\_y\_coords:

        print("Не удалось определить границы для графика.")

        return

    padding = 10

    ax.set\_xlim(min(all\_x\_coords) - padding, max(all\_x\_coords) + padding)

    ax.set\_ylim(min(all\_y\_coords) - padding, max(all\_y\_coords) + padding)

    ax.set\_xlabel("Ось X")

    ax.set\_ylabel("Ось Y")

    ax.set\_title("Определённые зоны")

    ax.grid(True)

    print("Окно с зонами будет отображено. Закройте его, чтобы продолжить.")

    plt.show()

def get\_zone\_details\_from\_user(zones\_data, gui\_input=None, gui\_output=None):

    """gui\_input -- функция для получения строки от пользователя через GUI,

    gui\_output -- функция для вывода текста в консоль GUI."""

    if not zones\_data:

        return []

    assigned\_details\_data = []

    if gui\_output:

        gui\_output("\n--- Ввод данных для зон (имя и роль) ---\n")

    for i, zone\_points in enumerate(zones\_data):

        zone\_id = i + 1

        if gui\_output:

            gui\_output(f"\nОбработка Зоны {zone\_id} (координаты: {zone\_points})\n")

        zone\_name = ""

        while True:

            if gui\_input:

                gui\_output(f"Введите имя для Зоны {zone\_id}: ")

                zone\_name = gui\_input()

            else:

                zone\_name = input(f"Введите имя для Зоны {zone\_id}: \n").strip()

            if zone\_name:

                break

            else:

                if gui\_output:

                    gui\_output("Имя зоны не может быть пустым. Пожалуйста, введите имя.\n")

                else:

                    print("Имя зоны не может быть пустым. Пожалуйста, введите имя.")

        prompt\_message = f"Выберите роль для Зоны {zone\_id} (Имя: '{zone\_name}'):\n"

        for key, value in ROLE\_OPTIONS.items():

            prompt\_message += f"  {key}: {value}\n"

        prompt\_message += "Ваш выбор (1/2/3): "

        while True:

            if gui\_input:

                gui\_output(prompt\_message)

                choice = gui\_input()

            else:

                choice = input(prompt\_message).strip()

            if choice in ROLE\_OPTIONS:

                role\_name = ROLE\_OPTIONS[choice]

                assigned\_details\_data.append({

                    "id": zone\_id,

                    "coordinates": zone\_points,

                    "name": zone\_name,

                    "role": role\_name

                })

                if gui\_output:

                    gui\_output(f"Для Зоны {zone\_id} (Имя: '{zone\_name}') назначена роль: '{role\_name}'.\n")

                else:

                    print(f"Для Зоны {zone\_id} (Имя: '{zone\_name}') назначена роль: '{role\_name}'.")

                break

            else:

                if gui\_output:

                    gui\_output("Неверный ввод. Пожалуйста, выберите 1, 2 или 3.\n")

                else:

                    print("Неверный ввод. Пожалуйста, выберите 1, 2 или 3.")

    return assigned\_details\_data

def save\_details\_to\_file(assigned\_details, filepath):

    try:

        with open(filepath, 'w', encoding='utf-8') as f:

            if not assigned\_details:

                f.write("")

                print(f"Файл '{filepath}' очищен/оставлен пустым.")

                return

            for entry in assigned\_details:

                f.write(f"Zone ID: {entry['id']}\n")

                f.write(f"Coordinates: {entry['coordinates']}\n")

                f.write(f"Name: {entry['name']}\n")

                f.write(f"Role: {entry['role']}\n")

                f.write("---\n")

        print(f"\nДанные о зонах успешно сохранены в файл '{filepath}'.")

    except Exception as e:

        print(f"Ошибка при сохранении файла '{filepath}': {e}")

def read\_previous\_zone\_details(filepath):

    previous\_details = []

    try:

        with open(filepath, 'r', encoding='utf-8') as f:

            content = f.read().split('---\n')

            for block in content:

                block = block.strip()

                if not block:

                    continue

                detail\_data = {}

                lines = block.split('\n')

                if len(lines) >= 4:

                    try:

                        if lines[0].startswith("Zone ID: "):

                            detail\_data['id'] = int(lines[0].replace("Zone ID: ", "").strip())

                        if lines[1].startswith("Coordinates: "):

                            coords\_str = lines[1].replace("Coordinates: ", "").strip()

                            detail\_data['coordinates'] = ast.literal\_eval(coords\_str)

                        if lines[2].startswith("Name: "):

                            detail\_data['name'] = lines[2].replace("Name: ", "").strip()

                        if lines[3].startswith("Role: "):

                            detail\_data['role'] = lines[3].replace("Role: ", "").strip()

                        if 'id' in detail\_data and 'coordinates' in detail\_data and \

                            'name' in detail\_data and 'role' in detail\_data:

                            previous\_details.append(detail\_data)

                        else:

                            print(f"Предупреждение: Не удалось полностью разобрать блок в '{filepath}':\n{block}")

                    except (ValueError, SyntaxError) as e:

                        print(f"Ошибка парсинга блока в '{filepath}': {e}\nБлок:\n{block}")

    except FileNotFoundError:

        print(f"Инфо: Файл '{filepath}' не найден (вероятно, первый запуск).")

    except Exception as e:

        print(f"Ошибка при чтении файла '{filepath}': {e}")

    previous\_details.sort(key=lambda x: x.get('id', float('inf')))

    return previous\_details

COORDINATES\_FILE\_PATH = "src/coordinates.txt"

OUTPUT\_FILE\_PATH = "zone\_analysis.txt"

def time\_to\_seconds\_from\_midnight(t):

    return t.hour \* 3600 + t.minute \* 60 + t.second

def seconds\_from\_midnight\_to\_time\_str(s):

    if s is None or math.isnan(s) or s < 0:

        return "N/A"

    s = round(s)

    hours = int(s // 3600) % 24

    minutes = int((s % 3600) // 60)

    seconds = int(s % 60)

    return f"{hours:02d}:{minutes:02d}:{seconds:02d}"

def format\_timedelta\_hhmmss(td):

    if td is None:

        return "N/A"

    total\_seconds = round(td.total\_seconds())

    if total\_seconds < 0:

        return "Invalid timedelta"

    hours, remainder = divmod(total\_seconds, 3600)

    minutes, seconds = divmod(remainder, 60)

    return f"{int(hours):02d}:{int(minutes):02d}:{int(seconds):02d}"

def read\_coordinates\_from\_file(filepath):

    coordinates\_data = []

    try:

        with open(filepath, 'r', encoding='utf-8') as f:

            for line\_num, line in enumerate(f, 1):

                line = line.strip()

                if not line:

                    continue

                try:

                    parts = line.split(',')

                    if len(parts) != 3:

                        print(f"Предупреждение: Строка {line\_num} имеет неверный формат: '{line}'")

                        continue

                    x = int(parts[0].strip())

                    y = int(parts[1].strip())

                    time\_str = parts[2].strip()

                    time\_parts = time\_str.split(':')

                    if len(time\_parts) != 6:

                        print(f"Предупреждение: Неверный формат времени в строке {line\_num}: '{time\_str}'")

                        continue

                    year, month, day, hour, minute, second = map(int, time\_parts)

                    datetime\_obj = datetime(year, month, day, hour, minute, second)

                    coordinates\_data.append({

                        'x': x,

                        'y': y,

                        'timestamp': time\_str,

                        'datetime\_obj': datetime\_obj

                    })

                except ValueError as e:

                    print(f"Ошибка парсинга строки {line\_num}: '{line}' - {e}")

                    continue

    except FileNotFoundError:

        print(f"Ошибка: Файл '{filepath}' не найден.")

    except Exception as e:

        print(f"Ошибка при чтении файла '{filepath}': {e}")

    coordinates\_data.sort(key=lambda x: x['datetime\_obj'])

    return coordinates\_data

def read\_zone\_details(filepath):

    return read\_previous\_zone\_details(filepath)

def point\_in\_polygon(x, y, polygon\_coords):

    n = len(polygon\_coords)

    inside = False

    p1x, p1y = polygon\_coords[0]

    for i in range(1, n + 1):

        p2x, p2y = polygon\_coords[i % n]

        if y > min(p1y, p2y):

            if y <= max(p1y, p2y):

                if x <= max(p1x, p2x):

                    if p1y != p2y:

                        xinters = (y - p1y) \* (p2x - p1x) / (p2y - p1y) + p1x

                    if p1x == p2x or x <= xinters:

                        inside = not inside

        p1x, p1y = p2x, p2y

    return inside

def analyze\_zone\_presence(coordinates\_data, zone\_details):

    zone\_analysis = {}

    for zone in zone\_details:

        zone\_id = zone['id']

        zone\_name = zone['name']

        zone\_role = zone['role']

        zone\_coords = zone['coordinates']

        points\_in\_zone = []

        for coord in coordinates\_data:

            if point\_in\_polygon(coord['x'], coord['y'], zone\_coords):

                points\_in\_zone.append(coord)

        if not points\_in\_zone:

            zone\_analysis[zone\_id] = {

                'name': zone\_name,

                'role': zone\_role,

                'coordinates': zone\_coords,

                'visits': [],

                'total\_time': timedelta(0),

                'visit\_count': 0

            }

            continue

        visits = []

        current\_visit = [points\_in\_zone[0]]

        for i in range(1, len(points\_in\_zone)):

            current\_point = points\_in\_zone[i]

            previous\_point = points\_in\_zone[i-1]

            time\_diff = current\_point['datetime\_obj'] - previous\_point['datetime\_obj']

            if time\_diff > timedelta(minutes=30):

                visits.append(current\_visit)

                current\_visit = [current\_point]

            else:

                current\_visit.append(current\_point)

        visits.append(current\_visit)

        visit\_periods = []

        total\_time\_in\_zone = timedelta(0)

        for visit\_points\_list in visits:

            duration = timedelta(0)

            start\_time = visit\_points\_list[0]['datetime\_obj']

            end\_time = visit\_points\_list[-1]['datetime\_obj']

            if len(visit\_points\_list) == 1:

                duration = timedelta(minutes=5)

                end\_time = start\_time + duration

            else:

                duration = end\_time - start\_time

                if duration <= timedelta(0):

                    duration = timedelta(minutes=1)

            visit\_periods.append({

                'start': start\_time,

                'end': end\_time,

                'duration': duration,

                'points\_count': len(visit\_points\_list),

                'date': start\_time.date()

            })

            total\_time\_in\_zone += duration

        zone\_analysis[zone\_id] = {

            'name': zone\_name,

            'role': zone\_role,

            'coordinates': zone\_coords,

            'visits': visit\_periods,

            'total\_time': total\_time\_in\_zone,

            'visit\_count': len(visit\_periods)

        }

    return zone\_analysis

def calculate\_behavioral\_patterns(zone\_analysis\_data):

    sleep\_periods = []

    work\_periods = []

    for zone\_id, data in zone\_analysis\_data.items():

        role = data.get('role', '').lower()

        for visit in data.get('visits', []):

            if visit['duration'] > timedelta(minutes=5):

                period\_detail = {

                    'zone\_name': data.get('name', 'N/A'),

                    'start\_datetime': visit['start'],

                    'end\_datetime': visit['end'],

                    'duration\_timedelta': visit['duration']

                }

                if role == 'сон':

                    sleep\_periods.append(period\_detail)

                elif role == 'работа':

                    work\_periods.append(period\_detail)

    sleep\_summary = {

        "periods": sorted(sleep\_periods, key=lambda x: x['start\_datetime']),

        "avg\_bedtime": "N/A",

        "avg\_waketime": "N/A",

        "avg\_duration": "N/A",

        "total\_duration": format\_timedelta\_hhmmss(sum((p['duration\_timedelta'] for p in sleep\_periods), timedelta())),

        "count": len(sleep\_periods)

    }

    if sleep\_periods:

        bedtimes\_seconds = [time\_to\_seconds\_from\_midnight(p['start\_datetime'].time()) for p in sleep\_periods]

        waketimes\_seconds = [time\_to\_seconds\_from\_midnight(p['end\_datetime'].time()) for p in sleep\_periods]

        avg\_bedtime\_s = sum(bedtimes\_seconds) / len(bedtimes\_seconds)

        avg\_waketime\_s = sum(waketimes\_seconds) / len(waketimes\_seconds)

        sleep\_summary["avg\_bedtime"] = seconds\_from\_midnight\_to\_time\_str(avg\_bedtime\_s)

        sleep\_summary["avg\_waketime"] = seconds\_from\_midnight\_to\_time\_str(avg\_waketime\_s)

        total\_sleep\_td = sum((p['duration\_timedelta'] for p in sleep\_periods), timedelta())

        if sleep\_periods:

            sleep\_summary["avg\_duration"] = format\_timedelta\_hhmmss(total\_sleep\_td / len(sleep\_periods))

    work\_summary = {

        "periods": sorted(work\_periods, key=lambda x: x['start\_datetime']),

        "avg\_start\_time": "N/A",

        "avg\_end\_time": "N/A",

        "avg\_duration\_per\_session": "N/A",

        "total\_duration": format\_timedelta\_hhmmss(sum((p['duration\_timedelta'] for p in work\_periods), timedelta())),

        "count": len(work\_periods)

    }

    if work\_periods:

        work\_start\_times\_s = [time\_to\_seconds\_from\_midnight(p['start\_datetime'].time()) for p in work\_periods]

        work\_end\_times\_s = [time\_to\_seconds\_from\_midnight(p['end\_datetime'].time()) for p in work\_periods]

        avg\_work\_start\_s = sum(work\_start\_times\_s) / len(work\_start\_times\_s)

        avg\_work\_end\_s = sum(work\_end\_times\_s) / len(work\_end\_times\_s)

        work\_summary["avg\_start\_time"] = seconds\_from\_midnight\_to\_time\_str(avg\_work\_start\_s)

        work\_summary["avg\_end\_time"] = seconds\_from\_midnight\_to\_time\_str(avg\_work\_end\_s)

        total\_work\_td = sum((p['duration\_timedelta'] for p in work\_periods), timedelta())

        if work\_periods:

            work\_summary["avg\_duration\_per\_session"] = format\_timedelta\_hhmmss(total\_work\_td / len(work\_periods))

    return {"sleep": sleep\_summary, "work": work\_summary}

def save\_analysis\_to\_file(zone\_analysis, behavioral\_patterns, filepath):

    try:

        with open(filepath, 'w', encoding='utf-8') as f:

            f.write("=" \* 60 + "\n")

            f.write("АНАЛИЗ ВРЕМЕНИ ПРЕБЫВАНИЯ В ЗОНАХ\n")

            f.write("=" \* 60 + "\n\n")

            if not zone\_analysis:

                f.write("Данные о зонах отсутствуют.\n")

            else:

                total\_zones = len(zone\_analysis)

                zones\_with\_visits = sum(1 for z in zone\_analysis.values() if z['visit\_count'] > 0)

                total\_visits\_all\_zones = sum(z['visit\_count'] for z in zone\_analysis.values())

                f.write(f"ОБЩАЯ СТАТИСТИКА ПО ЗОНАМ:\n")

                f.write(f"- Всего зон: {total\_zones}\n")

                f.write(f"- Зон с посещениями: {zones\_with\_visits}\n")

                f.write(f"- Общее количество посещений (все зоны): {total\_visits\_all\_zones}\n\n")

                for zone\_id in sorted(zone\_analysis.keys()):

                    zone\_data = zone\_analysis[zone\_id]

                    f.write("-" \* 50 + "\n")

                    f.write(f"ЗОНА {zone\_id}: {zone\_data['name']}\n")

                    f.write(f"Роль: {zone\_data['role']}\n")

                    f.write(f"Координаты: {zone\_data['coordinates']}\n")

                    f.write("-" \* 50 + "\n")

                    if zone\_data['visit\_count'] == 0:

                        f.write("Посещений не было.\n\n")

                        continue

                    f.write(f"Количество посещений: {zone\_data['visit\_count']}\n")

                    f.write(f"Общее время пребывания: {format\_timedelta\_hhmmss(zone\_data['total\_time'])}\n\n")

                    f.write("ПЕРИОДЫ ПОСЕЩЕНИЙ:\n")

                    for i, visit in enumerate(zone\_data['visits'], 1):

                        f.write(f"  {i}. {visit['start'].strftime('%Y-%m-%d %H:%M')} - "

                                f"{visit['end'].strftime('%Y-%m-%d %H:%M')} "

                                f"(Длительность: {format\_timedelta\_hhmmss(visit['duration'])}, "

                                f"Точек: {visit['points\_count']})\n")

                    f.write("\n")

            # Анализ сна

            sleep\_summary = behavioral\_patterns.get("sleep", {})

            f.write("\n" + "=" \* 60 + "\n")

            f.write("АНАЛИЗ СНА\n")

            f.write("=" \* 60 + "\n")

            if not sleep\_summary or sleep\_summary.get("count", 0) == 0:

                f.write("Нет данных для анализа сна или периоды сна отсутствуют/слишком короткие.\n")

            else:

                f.write(f"Всего периодов сна (>5 мин): {sleep\_summary['count']}\n")

                f.write(f"Среднее время отхода ко сну: {sleep\_summary['avg\_bedtime']}\n")

                f.write(f"Среднее время пробуждения: {sleep\_summary['avg\_waketime']}\n")

                f.write(f"Средняя продолжительность сна: {sleep\_summary['avg\_duration']}\n")

                f.write(f"Общая продолжительность сна: {sleep\_summary['total\_duration']}\n\n")

                f.write("ДЕТАЛИЗАЦИЯ ПЕРИОДОВ СНА:\n")

                for i, period in enumerate(sleep\_summary.get("periods", []), 1):

                    f.write(f"  {i}. Зона: '{period['zone\_name']}', "

                            f"{period['start\_datetime'].strftime('%Y-%m-%d %H:%M')} - "

                            f"{period['end\_datetime'].strftime('%Y-%m-%d %H:%M')} "

                            f"(Длительность: {format\_timedelta\_hhmmss(period['duration\_timedelta'])})\n")

            f.write("\n")

            # Анализ работы

            work\_summary = behavioral\_patterns.get("work", {})

            f.write("\n" + "=" \* 60 + "\n")

            f.write("АНАЛИЗ РАБОТЫ\n")

            f.write("=" \* 60 + "\n")

            if not work\_summary or work\_summary.get("count", 0) == 0:

                f.write("Нет данных для анализа работы или рабочие сессии отсутствуют/слишком короткие.\n")

            else:

                f.write(f"Всего рабочих сессий (>5 мин): {work\_summary['count']}\n")

                f.write(f"Среднее время начала работы: {work\_summary['avg\_start\_time']}\n")

                f.write(f"Среднее время окончания работы: {work\_summary['avg\_end\_time']}\n")

                f.write(f"Средняя продолжительность рабочей сессии: {work\_summary['avg\_duration\_per\_session']}\n")

                f.write(f"Общая продолжительность работы: {work\_summary['total\_duration']}\n\n")

                f.write("ДЕТАЛИЗАЦИЯ РАБОЧИХ СЕССИЙ:\n")

                for i, period in enumerate(work\_summary.get("periods", []), 1):

                    f.write(f"  {i}. Зона: '{period['zone\_name']}', "

                            f"{period['start\_datetime'].strftime('%Y-%m-%d %H:%M')} - "

                            f"{period['end\_datetime'].strftime('%Y-%m-%d %H:%M')} "

                            f"(Длительность: {format\_timedelta\_hhmmss(period['duration\_timedelta'])})\n")

        print(f"Результаты анализа сохранены в файл '{filepath}'")

    except Exception as e:

        print(f"Ошибка при сохранении файла '{filepath}': {e}")

def visualize\_zones\_and\_movement(zone\_details, coordinates\_data, zone\_analysis):

    if not zone\_details and not coordinates\_data:

        print("Нет данных для визуализации.")

        return

    fig, ax = plt.subplots(figsize=(12, 8))

    ax.set\_aspect('equal', adjustable='datalim')

    all\_x\_coords = []

    all\_y\_coords = []

    colors = ['lightblue', 'lightgreen', 'lightcoral', 'lightyellow', 'lightpink', 'cyan', 'magenta']

    for i, zone in enumerate(zone\_details):

        zone\_coords = zone['coordinates']

        color = colors[i % len(colors)]

        polygon = patches.Polygon(zone\_coords, closed=True,

                                edgecolor='black', facecolor=color, alpha=0.6)

        ax.add\_patch(polygon)

        for x, y in zone\_coords:

            all\_x\_coords.append(x)

            all\_y\_coords.append(y)

        if zone\_coords:

            cx = sum(p[0] for p in zone\_coords) / len(zone\_coords)

            cy = sum(p[1] for p in zone\_coords) / len(zone\_coords)

            zone\_info\_from\_analysis = zone\_analysis.get(zone['id'], {})

            visit\_count = zone\_info\_from\_analysis.get('visit\_count', 0)

            total\_time\_in\_zone\_td = zone\_info\_from\_analysis.get('total\_time', timedelta(0))

            label\_text = (f"{zone['name']}\n({zone['role']})\n"

                        f"Посещений: {visit\_count}\nВремя: {format\_timedelta\_hhmmss(total\_time\_in\_zone\_td)}")

            ax.text(cx, cy, label\_text, ha='center', va='center',

                    color='black', fontsize=7, bbox=dict(boxstyle="round,pad=0.3",

                    facecolor='white', alpha=0.7))

    if coordinates\_data:

        x\_coords\_traj = [coord['x'] for coord in coordinates\_data]

        y\_coords\_traj = [coord['y'] for coord in coordinates\_data]

        all\_x\_coords.extend(x\_coords\_traj)

        all\_y\_coords.extend(y\_coords\_traj)

        ax.plot(x\_coords\_traj, y\_coords\_traj, 'r-', alpha=0.5, linewidth=1, label='Траектория движения')

    if all\_x\_coords and all\_y\_coords:

        padding = 20

        ax.set\_xlim(min(all\_x\_coords) - padding, max(all\_x\_coords) + padding)

        ax.set\_ylim(min(all\_y\_coords) - padding, max(all\_y\_coords) + padding)

    ax.set\_xlabel("Ось X")

    ax.set\_ylabel("Ось Y")

    ax.set\_title("Зоны и траектория движения")

    ax.grid(True, alpha=0.3)

    ax.legend()

    print("Отображение графика с зонами и траекторией движения...")

    plt.show()

class ConsoleRedirect:

    def \_\_init\_\_(self, text\_widget, queue\_obj):

        self.text\_widget = text\_widget

        self.queue = queue\_obj

    def write(self, string):

        self.queue.put(string)

    def flush(self):

        pass

class ZoneAnalysisGUI:

    def \_\_init\_\_(self, root):

        self.root = root

        self.root.title("Анализ зон - Система отслеживания")

        self.root.geometry("900x670")

        self.console\_queue = queue.Queue()

        self.is\_running = False

        self.create\_widgets()

        self.process\_console\_queue()

        self.console\_redirect = ConsoleRedirect(self.console\_text, self.console\_queue)

        self.old\_stdout = sys.stdout

        self.old\_stderr = sys.stderr

    def create\_widgets(self):

        main\_frame = ttk.Frame(self.root, padding="10")

        main\_frame.grid(row=0, column=0, sticky=(tk.W, tk.E, tk.N, tk.S))

        self.root.columnconfigure(0, weight=1)

        self.root.rowconfigure(0, weight=1)

        main\_frame.columnconfigure(0, weight=1)

        main\_frame.rowconfigure(1, weight=1)

        title\_label = ttk.Label(main\_frame, text="Система анализа зон",

                                font=('Arial', 16, 'bold'))

        title\_label.grid(row=0, column=0, pady=(0, 10), sticky=tk.W)

        # Консоль

        console\_frame = ttk.LabelFrame(main\_frame, text="Вывод консоли", padding="5")

        console\_frame.grid(row=1, column=0, sticky=(tk.W, tk.E, tk.N, tk.S), pady=(0, 10))

        console\_frame.columnconfigure(0, weight=1)

        console\_frame.rowconfigure(0, weight=1)

        self.console\_text = scrolledtext.ScrolledText(

            console\_frame,

            wrap=tk.WORD,

            width=100,

            height=28,

            font=('Consolas', 9),

            state=tk.DISABLED

        )

        self.console\_text.grid(row=0, column=0, sticky=(tk.W, tk.E, tk.N, tk.S))

        # Кнопки управления

        control\_frame = ttk.Frame(main\_frame)

        control\_frame.grid(row=2, column=0, sticky=(tk.W, tk.E))

        self.btn\_zones = ttk.Button(control\_frame, text="Ввод зон, ролей и имён", command=self.start\_zones)

        self.btn\_zones.grid(row=0, column=0, padx=(0, 10))

        self.btn\_analysis = ttk.Button(control\_frame, text="Анализ по координатам", command=self.start\_analysis)

        self.btn\_analysis.grid(row=0, column=1, padx=(0, 10))

        self.btn\_visual = ttk.Button(control\_frame, text="Визуализация зон и движения", command=self.visualize)

        self.btn\_visual.grid(row=0, column=2, padx=(0, 10))

        # --- Новая кнопка ---

        self.btn\_found\_zones = ttk.Button(control\_frame, text="Найденные зоны", command=self.show\_found\_zones)

        self.btn\_found\_zones.grid(row=0, column=3, padx=(0, 10))

        # --------------------

        self.btn\_clear = ttk.Button(control\_frame, text="Очистить консоль", command=self.clear\_console)

        self.btn\_clear.grid(row=0, column=4, padx=(0,10))

        self.progress = ttk.Progressbar(control\_frame, mode='indeterminate')

        self.progress.grid(row=0, column=5, sticky=(tk.W, tk.E), padx=(10, 0))

        control\_frame.columnconfigure(5, weight=1)

    def write\_to\_console(self, text, tag=None):

        self.console\_text.config(state=tk.NORMAL)

        self.console\_text.insert(tk.END, text)

        self.console\_text.see(tk.END)

        self.console\_text.config(state=tk.DISABLED)

    def process\_console\_queue(self):

        try:

            while True:

                message = self.console\_queue.get\_nowait()

                self.write\_to\_console(message)

        except queue.Empty:

            pass

        self.root.after(100, self.process\_console\_queue)

    def clear\_console(self):

        self.console\_text.config(state=tk.NORMAL)

        self.console\_text.delete(1.0, tk.END)

        self.console\_text.config(state=tk.DISABLED)

    def show\_found\_zones(self):

        """Показать найденные зоны из zone.txt без ролей и имён"""

        def task():

            sys.stdout = self.console\_redirect

            sys.stderr = self.console\_redirect

            self.progress.start()

            zones = read\_zones\_from\_file(ZONE\_FILE\_PATH)

            if zones:

                self.gui\_output(f"Обнаружено зон: {len(zones)}\n")

                display\_zones(zones)

            else:

                self.gui\_output("Зоны не найдены (src/zone.txt пуст или отсутствует)\n")

            self.progress.stop()

            sys.stdout = self.old\_stdout

            sys.stderr = self.old\_stderr

        threading.Thread(target=task, daemon=True).start()

    def gui\_input(self):

        """Запрашивает строку через диалоговое окно ввода."""

        popup = tk.Toplevel(self.root)

        popup.title("Ввод")

        popup.geometry("350x120")

        label = ttk.Label(popup, text="Введите значение:")

        label.pack(pady=7)

        entry\_var = tk.StringVar()

        entry = ttk.Entry(popup, textvariable=entry\_var, width=32)

        entry.pack(pady=5)

        entry.focus\_set()

        result = []

        def on\_ok():

            result.append(entry\_var.get())

            popup.destroy()

        def on\_cancel():

            result.append("")

            popup.destroy()

        btn\_ok = ttk.Button(popup, text="OK", command=on\_ok)

        btn\_ok.pack(side=tk.LEFT, padx=30, pady=5, expand=True)

        btn\_cancel = ttk.Button(popup, text="Отмена", command=on\_cancel)

        btn\_cancel.pack(side=tk.RIGHT, padx=30, pady=5, expand=True)

        popup.grab\_set()

        self.root.wait\_window(popup)

        return result[0] if result else ""

    def gui\_output(self, s):

        self.console\_queue.put(str(s))

    def start\_zones(self):

        def task():

            sys.stdout = self.console\_redirect

            sys.stderr = self.console\_redirect

            self.progress.start()

            zones = read\_zones\_from\_file(ZONE\_FILE\_PATH)

            previous = read\_previous\_zone\_details(ROLES\_FILE\_PATH)

            if not zones:

                self.gui\_output("Нет зон для ввода данных.\n")

                self.progress.stop()

                sys.stdout = self.old\_stdout

                sys.stderr = self.old\_stderr

                return

            details = get\_zone\_details\_from\_user(

                zones,

                gui\_input=self.gui\_input,

                gui\_output=self.gui\_output

            )

            save\_details\_to\_file(details, ROLES\_FILE\_PATH)

            self.progress.stop()

            sys.stdout = self.old\_stdout

            sys.stderr = self.old\_stderr

        threading.Thread(target=task, daemon=True).start()

    def start\_analysis(self):

        def task():

            sys.stdout = self.console\_redirect

            sys.stderr = self.console\_redirect

            self.progress.start()

            coordinates\_data = read\_coordinates\_from\_file(COORDINATES\_FILE\_PATH)

            zone\_details = read\_zone\_details(ROLES\_FILE\_PATH)

            if not coordinates\_data or not zone\_details:

                self.gui\_output("Для анализа необходимы координаты и зоны.\n")

                self.progress.stop()

                sys.stdout = self.old\_stdout

                sys.stderr = self.old\_stderr

                return

            zone\_analysis\_result = analyze\_zone\_presence(coordinates\_data, zone\_details)

            behavioral\_patterns\_result = calculate\_behavioral\_patterns(zone\_analysis\_result)

            save\_analysis\_to\_file(zone\_analysis\_result, behavioral\_patterns\_result, OUTPUT\_FILE\_PATH)

            self.gui\_output("\nАнализ завершён, результаты сохранены в файл.\n")

            self.progress.stop()

            sys.stdout = self.old\_stdout

            sys.stderr = self.old\_stderr

        threading.Thread(target=task, daemon=True).start()

    def visualize(self):

        def task():

            sys.stdout = self.console\_redirect

            sys.stderr = self.console\_redirect

            self.progress.start()

            coordinates\_data = read\_coordinates\_from\_file(COORDINATES\_FILE\_PATH)

            zone\_details = read\_zone\_details(ROLES\_FILE\_PATH)

            zone\_analysis\_result = analyze\_zone\_presence(coordinates\_data, zone\_details) if coordinates\_data and zone\_details else {}

            visualize\_zones\_and\_movement(zone\_details, coordinates\_data, zone\_analysis\_result)

            self.progress.stop()

            sys.stdout = self.old\_stdout

            sys.stderr = self.old\_stderr

        threading.Thread(target=task, daemon=True).start()

def main():

    root = tk.Tk()

    style = ttk.Style()

    try:

        style.theme\_use('clam')

    except:

        pass

    app = ZoneAnalysisGUI(root)

    root.mainloop()

if \_\_name\_\_ == "\_\_main\_\_":

    main()