Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Бакалаврська дипломна робота

на тему: «Розробка програмного забезпечення для знаходження оптимального шляху безпілотного літального апарату»

Виконав: студент 4 курсу
групи 4ПІ-20б
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Матейко Є.В.
(прізвище та ініціали)

Керівник: к.т.н., доц. каф. ПЗ Черноволик Г.О.
(прізвище та ініціали)

Рецензент: к.т.н., доц. каф. ОТ Томчук М.А.
(прізвище та ініціали)

Допущено до захисту

Зав. кафедри ПЗ _____
«10» червня 2024 р.

ВНТУ – 2024

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший (бакалаврський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітньо-професійна програма «Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
д.т.н., професор
О. Н. Романюк _____
«12» березня 2024 р.

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Матейку Євгенію Віталійовичу

1. Тема роботи – розробка програмного забезпечення для знаходження оптимального шляху безпілотного літального апарату.

Керівник роботи: Черноволик Галина Олександрівна, к.т.н., доц. кафедри ПЗ, затверджені наказом вищого навчального закладу від 11 березня 2024 р. №80.

2. Строк подання студентом роботи 3 червня 2024.

3. Вихідні дані до роботи: базові алгоритми – алгоритм Дейкстри, алгоритм A*; бібліотеки – osmdroid, osmdroid-bonuspack, GSON; середовище розробки – Android Studio; середовище збірки проєкту – Gradle Kotlin Script; мова програмування – Java; мінімальний рівень API операційної системи Android – 29.

4. Зміст текстової частини: вступ; аналіз стану систем знаходження оптимального шляху безпілотного літального апарату; порівняльний аналіз аналогів; вплив алгоритмів знаходження оптимального шляху на ефективність безпілотного літального апарату; постановка задач дослідження; аналіз вхідних та вихідних даних системи; розробка інтерфейсу мобільного застосунку; розробка моделі системи; розробка алгоритмів роботи системи; аналіз і обґрунтування вибору мови програмування; аналіз і обґрунтування вибору засобу відображення мапи; розробка модуля визначення оптимального шляху; розробка модуля локального збереження даних; тестування мобільного застосунку; розробка інструкції користувача; список використаних джерел; додатки.

5. Перелік ілюстративного матеріалу: титульний слайд, актуальність розробки, мета, об'єкт та предмет дослідження, завдання дослідження, новизна

та практична цінність отриманих результатів, аналоги розробленого застосу
структура мобільного застосунку, діаграма діяльності мобільного застосу
блок-схема методу пошуку найкоротшого шляху, блок-схема методу врахува
характеристик безпілотника, графічна схема інтерфейсу головної активно
технології використані при розробці, функціонал мобільного застосу
локалізація мобільного застосунку, апробація та публікація результатів робо
висновки.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийня
1-4	Черноволик Г.О., к.т.н., доцент кафедри ПЗ	13.03.24	03.06.24

7. Дата видачі завдання 13 березня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

Ч.ч.	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз розвитку систем знаходження оптимального шляху та постановка задач дослідження	14.03.24 – 07.04.24	Вик.
2	Розробка методів, моделі та алгоритмів мобільного застосунку	08.04.24 – 21.04.24	Вик.
3	Розробка структури та програмних модулів мобільного застосунку	22.04.24 – 05.05.24	Вик.
4	Тестування мобільного застосунку	06.05.24 – 19.06.24	Вик.
5	Розробка інструкції користувача	20.05.24 – 26.05.24	Вик.
6	Оформлення матеріалів до захисту БДР	27.05.24 – 30.05.24	Вик.

Студент _____ **Є. В. Матейк**
(підпис) (ініціали та прізвище)

Керівник бакалаврської дипломної роботи _____ **Г. О. Черноволи**
(підпис) (ініціали та прізвище)

АНОТАЦІЯ

Бакалаврська дипломна робота складається з 62 сторінок формату А4, на яких є 38 рисунків, 4 таблиці, список використаних джерел містить 23 найменування.

У бакалаврській дипломній роботі розроблено програмні засоби мобільної системи знаходження оптимального шляху безпілотного літального апарату. Застосунок призначений для мобільної системи, що покликана автоматизувати процес визначення найкращого шляху за одним з алгоритмів на вибір користувача.

Застосунок розроблено з використанням мови програмування Java, інтегрованого середовища розробки для мобільних систем Android Studio та бібліотек osmdroid та GSON. Для збірки проєкту використано Gradle Kotlin Script.

Отримані в бакалаврській роботі результати можуть бути використані розробниками для створення мобільних систем під керуванням ОС Android для вирішення задачі комівояжера та інших задач по оптимізації процесу пошуку оптимального шляху.

Ключові слова: мобільний застосунок, оптимальний шлях, безпілотний літальний апарат.

ABSTRACT

The bachelor's thesis consists of 62 A4 pages, with 38 figures, 4 tables, and a list of references containing 23 titles.

In the bachelor's thesis, the software of a mobile system for finding the optimal path of an unmanned aerial vehicle was developed. The application is intended for a mobile system designed to automate the process of determining the best path according to one of the algorithms of the user's choice.

The application was developed using the Java programming language, the Android Studio integrated development environment for mobile systems, and the osmdroid and GSON libraries. Gradle Kotlin Script was used to build the project.

The results obtained in the bachelor's thesis can be used by developers to create mobile systems running Android to solve the salesman problem and other tasks to optimise the process of finding the optimal path.

Keywords: mobile application, optimal path, unmanned aerial vehicle.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. Безпілотні літальні апарати (БПЛА) – це авіаційні засоби, які можуть виконувати різноманітні завдання без прямої участі людини. БПЛА можуть застосовуватися для військових, наукових, цивільних та комерційних цілей, таких як розвідка, спостереження, моніторинг, пошук і рятування, доставка, сільське господарство, фотографія тощо [1].

Однією з ключових проблем, яка виникає при використанні БПЛА, є забезпечення їх безпечної та ефективної навігації в динамічному та непередбачуваному середовищі [2]. Для вирішення цієї задачі використовується програмне забезпечення, яке може знаходити оптимальний шлях для БПЛА, враховуючи різні обмеження та фактори, такі як топографія місцевості, погодні умови, присутність перешкод, вимоги до завдання, енергетична ефективність тощо [3].

Враховуючи специфічність характеристик безпілотних літальних апаратів, актуальною є розробка власної мобільної системи для знаходження оптимального шляху безпілотного літального апарату.

Зв’язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження. Метою роботи є підвищення енергоефективності безпілотних літальних апаратів за рахунок збільшення якості визначення шляху. Робота передбачає розробку різних модулів, які будуть інтегровані в мобільну систему. Ці модулі включають модуль локального збереження даних, модуль взаємодії з картою OpenStreetMap та модулі графічного інтерфейсу.

У бакалаврській дипломній роботі необхідно виконати такі завдання:

– розробити загальний алгоритм системи;

– розробити метод знаходження оптимального шляху безпілотного літального апарату на основі алгоритмів Дейкстри та A*;

– розробити метод врахування характеристик безпілотника;

– розробити зручний та адаптивний графічний інтерфейс мобільного застосунку;

– розробити модуль для локального збереження маршрутів на пристрої користувача;

– здійснити тестування мобільного застосунку згідно поставлених задач;

– розробити інструкцію користувача.

Об’єктом дослідження є процес розробки програмних засобів системи знаходження оптимального шляху безпілотного літального апарату.

Предметом дослідження є алгоритми і засоби реалізації системи знаходження оптимального шляху безпілотного літального апарату.

Методи дослідження. У процесі дослідження використовувались:

– принципи об’єктно-орієнтованого програмування для реалізації взаємодії об’єктів системи;

– алгоритм Дейкстри, алгоритм A* для реалізації модулів визначення оптимального шляху;

– метод юніт-тестування для тестування функцій алгоритмів.

Новизна отриманих результатів:

1. Подальшого розвитку отримали метод знаходження оптимального шляху на основі алгоритмів Дейкстри та A*, який, на відміну від існуючих, враховує зони запобігання, що дозволило підвищити точність визначення оптимального шляху.

2. Подальшого розвитку отримав метод врахування характеристик безпілотників, який, на відміну від існуючих, надає рекомендації, що дозволило підвищити ефективність використання безпілотників.

Практична цінність отриманих результатів. Практична цінність одержаних результатів полягає у можливості використання розробниками для створення мобільних систем під керуванням ОС Android для вирішення задачі комівояжера.

Особистий внесок здобувача. Усі результати, викладені у бакалаврській

дипломній роботі, отримані автором особисто.

Апробація матеріалів бакалаврської дипломної роботи. Результати роботи були розглянуті та обговорені на конференції LIII Науково-технічної конференції факультету інформаційних технологій та комп'ютерної інженерії (2024) [4].

Публікації. Основні результати досліджень було опубліковано на конференції LIII Науково-технічної конференції факультету інформаційних технологій та комп'ютерної інженерії (2024) [4].

1 АНАЛІЗ РОЗВИТКУ СИСТЕМ ЗНАХОДЖЕННЯ ОПТИМАЛЬНОГО ШЛЯХУ ТА ПОСТАНОВКА ЗАДАЧ ДОСЛІДЖЕННЯ

1.1 Аналіз стану систем знаходження оптимального шляху безпілотного літального апарату

Сучасний світ стрімко розвивається, і з кожним днем технології стають все більш інноваційними та інтегрованими у повсякденне життя. Однією з таких інновацій є безпілотні літальні апарати (БПЛА), які відкривають нові можливості для різних сфер діяльності, від сільського господарства до моніторингу та доставки [5]. Однак, ефективне використання БПЛА потребує розробки надійних систем знаходження оптимального шляху, які б забезпечували безпеку, точність та ефективність польотів.

Основні переваги БПЛА включають зниження витрат на експлуатацію та можливість виконання завдань без ризику для життя людей. Вони також можуть мати різну ступінь автономності, від дистанційно керованих до повністю автоматичних [6], що робить їх гнучкими для різноманітних застосувань. Водночас, існують і виклики, такі як уразливість систем дистанційного управління, особливо важливі для БПЛА військового призначення.

Існуючі системи знаходження оптимального шляху для безпілотних літальних апаратів мають свої недоліки та виклики, які потребують уваги та подальшого вдосконалення. Одним із основних недоліків є недосконалість алгоритмів знаходження шляху, яка полягає у обмеженій адаптивності та неврахуванні обмежень самого апарату.

Енергоефективність є критичною у сфері безпілотних технологій. Алгоритми маршрутизації, які використовують велику кількість даних, можуть вимагати потужних обчислювальних ресурсів, що у свою чергу може викликати затримки у визначенні оптимального маршруту.

Визначення шляху відіграє ключову роль у забезпеченні безпеки операцій БПЛА. Це дозволяє уникати колізій з іншими повітряними об'єктами, інфраструктурою та іншими можливими небезпечними об'єктами. Ефективний

алгоритм маршрутизації допомагає попереджати аварії та збільшує загальну надійність функціонування системи.

Визначення оптимального шляху значно впливає на ефективність використання ресурсів. Завдяки правильно побудованому маршруту, БПЛА може економити паливо, збільшувати час польоту та покращувати продуктивність в конкретних завданнях [7]. Наприклад, у комерційних сферах, таких як доставка чи аграрний сектор, оптимальний шлях може значно зменшити витрати і збільшити прибуток.

Отже, розробка програмних засобів системи знаходження оптимального шляху безпілотного літального апарату є актуальною та важливою задачею, яка має багато практичних застосувань та перспектив: це дозволить підвищити якість та продуктивність виконання різних завдань БПЛА, зменшити ризики для людського життя та здоров'я, збільшити автономність та гнучкість БПЛА, а також розширити сфери їх застосування [8].

1.2 Порівняльний аналіз аналогів

Використання безпілотних літальних апаратів стає все більш популярним у різних сферах. Це пов’язано з їхньою здатністю виконувати широкий спектр завдань, від моніторингу та картографування до доставки вантажів і надання допомоги в надзвичайних ситуаціях. Безпілотники використовуються в сільському господарстві, будівництві, енергетиці та багатьох інших галузях. Розробка програмних засобів системи знаходження оптимального шляху безпілотного літального апарату може привернути увагу до проблематики оптимізації процесу пошуку найкоротшого маршруту для дрона.

До найбільш відомих рішень відносять:

- PathPlanner;
- PX4 Autopilot;
- Mission Planner Home;
- QGroundControl;
- DJI Gs Pro.

PathPlanner – це програмний застосунок для планування маршрутів для рухомих систем. Він розроблений Фраунгоферським інститутом (Fraunhofer IOSB) і призначений для оптимізації траєкторій руху індивідуальних мобільних систем або гетерогенних груп мобільних систем [9], які працюють разом.

Основні можливості PathPlanner:

1. Застосунок створює оптимальні траєкторії для рухомих систем, включаючи безпечний перехід від початкової до цільової позиції, а також траєкторії для повного огляду складних територій.

2. Планування враховує внутрішні та зовнішні параметри, такі як перешкоди, зони забороненого руху, характеристики обладнання та маневреність рухомих систем.

3. PathPlanner базується на алгоритмах планування маршрутів, розроблених Фраунгофером, і забезпечує швидкі результати при низькому навантаженні на систему.

Інтерфейс програмного застосунку «PathPlanner» зображено на рисунку 1.1.

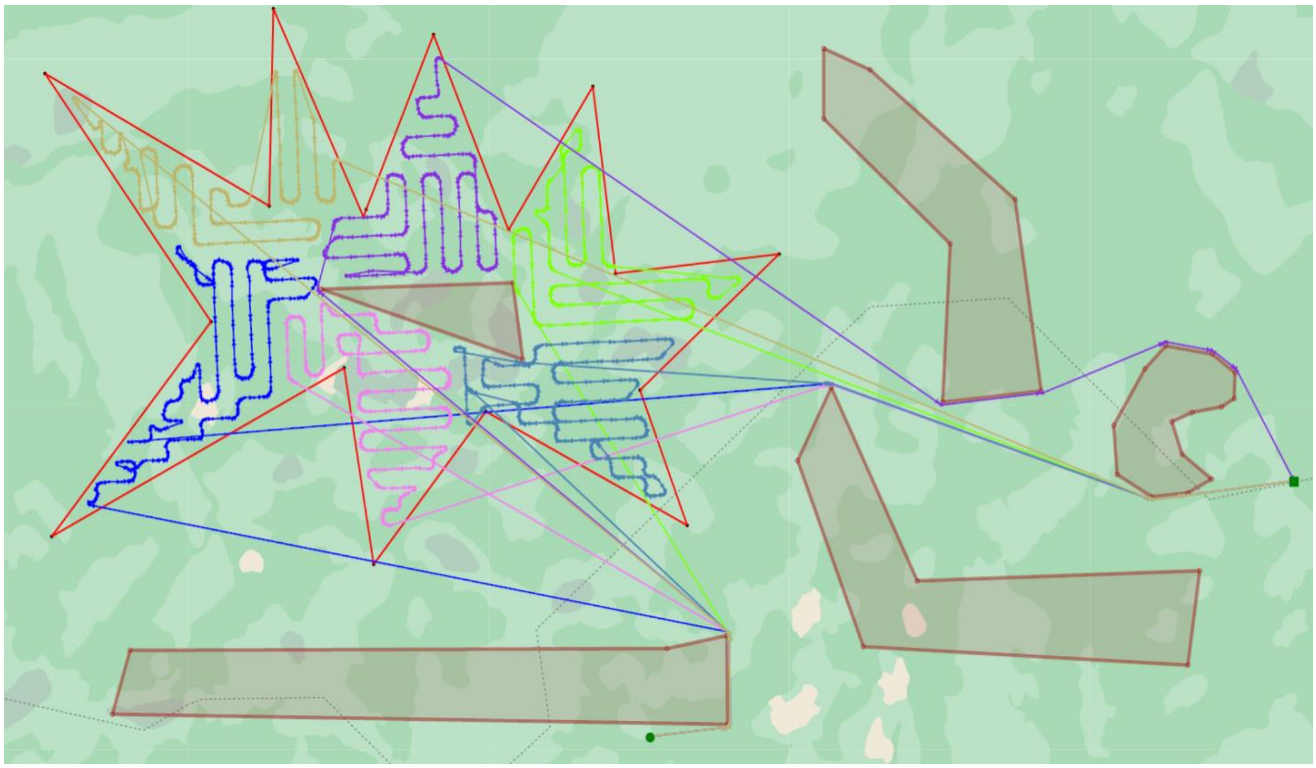


Рисунок 1.1 – Інтерфейс програмного застосунку «PathPlanner»

PX4 Autopilot – це відкрите програмне забезпечення для автономного керування безпілотними літальними апаратами. Воно розроблене спільнотою інженерів та розробників і представляє собою комплексну автопілотну систему [10], яка використовується в різних типах безпілотних апаратів, включаючи дрони, вертольоти та літальні апарати.

Основні характеристики PX4 Autopilot включають:

1. PX4 є відкритим програмним забезпеченням, що означає, що його вихідний код доступний для загального використання та редагування спільнотою, що сприяє швидшому розвитку системи завдяки участі великої кількості розробників.

2. PX4 надає користувачам велику кількість налаштувань та параметрів, що дозволяє їм налаштовувати систему відповідно до конкретних вимог та умов.

3. PX4 враховує аспекти безпеки, такі як системи перевірки стану апарата (health monitoring), автоматичне перехоплення та інші функції, що забезпечують безпечну експлуатацію.

Інтерфейс програмного застосунку «PX4 Autopilot» зображено на рисунку 1.2.

Рисунок 1.2 – Інтерфейс програмного застосунку «PX4 Autopilot»

Mission Planner – це програмне забезпечення для керування та планування місій для безпілотних літальних апаратів. Воно часто використовується в сполученні з платформою автопілота ArduPilot, такою як ArduCopter для дронів, ArduPlane для літаків та ArduRover для рухомих засобів на землі.

Можливості Mission Planner [11]:

1. Введення точок маршруту/обмеження за допомогою кліків мишею, використовуючи Google Maps/Bing/Open Street Maps.
2. Вибір команд місії з випадаючих меню.
3. Завантаження файлів журналів місій та їхній аналіз.
4. Налаштування параметрів автопілота для безпілотного літального апарату.

Інтерфейс програмного застосунку «Mission Planner» зображено на рисунку 1.3.

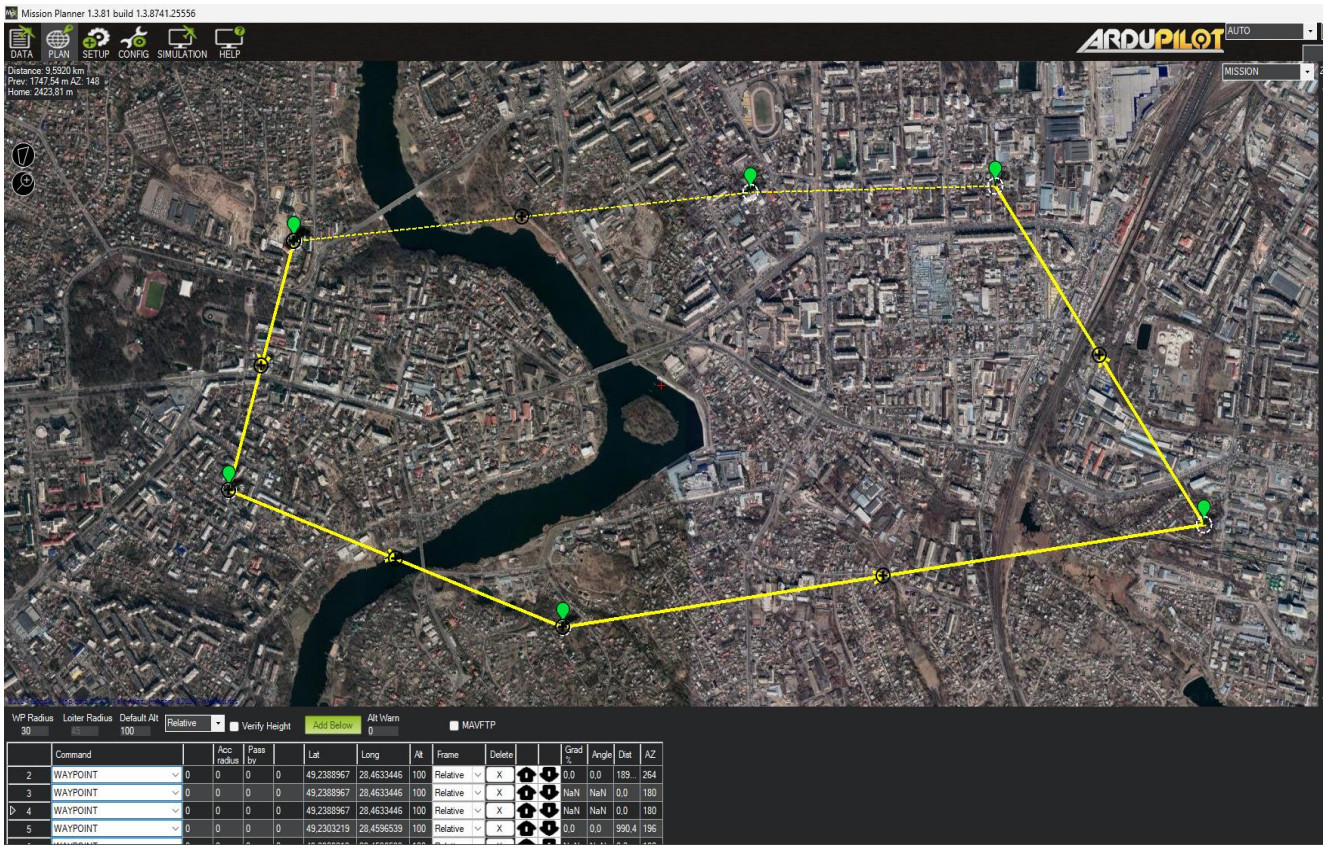


Рисунок 1.3 – Інтерфейс програмного застосунку «Mission Planner»

QGroundControl – це програмне забезпечення для керування та

моніторингу безпілотними літальними апаратами. Це відкрите програмне забезпечення, яке використовується в парі з різними автопілотами, зокрема з PX4 та ArduPilot [12].

Основні характеристики QGroundControl включають:

1. QGroundControl дозволяє користувачам планувати маршрути для безпілотних літальних апаратів, встановлюючи точки маршруту та області інтересу.

2. Користувачі QGroundControl можуть створювати місії та програми, які включають в себе різні команди та дії для автономного виконання.

3. QGroundControl надає інформацію про стан апарата в реальному часі.

4. QGroundControl має інтуїтивно зрозумілий графічний інтерфейс, що полегшує взаємодію з системою та використання різноманітних функцій.

5. QGroundControl розроблено для різних операційних систем, таких як Windows, macOS і Linux, що робить його доступним для широкого кола користувачів.

Інтерфейс програмного застосунку «QGroundControl» зображено на рисунку 1.4.

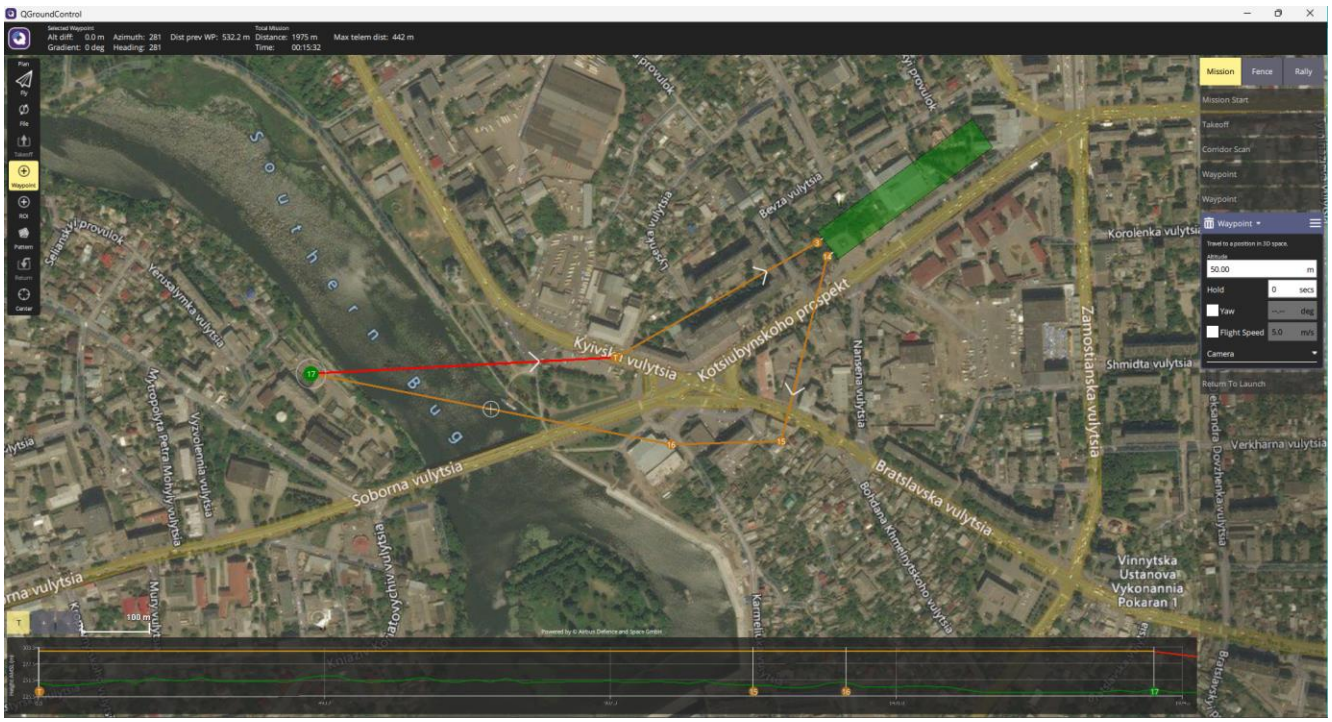


Рисунок 1.4 – Інтерфейс програмного застосунку «QGroundControl»

DJI GS Pro – це застосунок для iPad, який дозволяє користувачам планувати та контролювати складні місії для безпілотних літальних апаратів. Він включає такі функції, як автоматизовані польотні місії, управління даними польоту в хмарі та співпраця над проєктами [13].

Основні характеристики DJI GS Pro включають:

1. DJI GS Pro дозволяє користувачам створювати детальні плани польотів, встановлюючи точки шляху та параметри польоту.
2. Після планування місії, додаток може автоматично керувати польотом дрона, дозволяючи здійснювати зйомку або інспекцію без постійного втручання оператора.
3. Всі дані польоту можуть зберігатися в хмарі.
4. DJI GS Pro включає ряд функцій безпеки, таких як автоматичне повернення на базу при втраті сигналу або низькому заряді батареї.

Інтерфейс програмного застосунку «DJI GS Pro» зображено на рисунку 1.5.

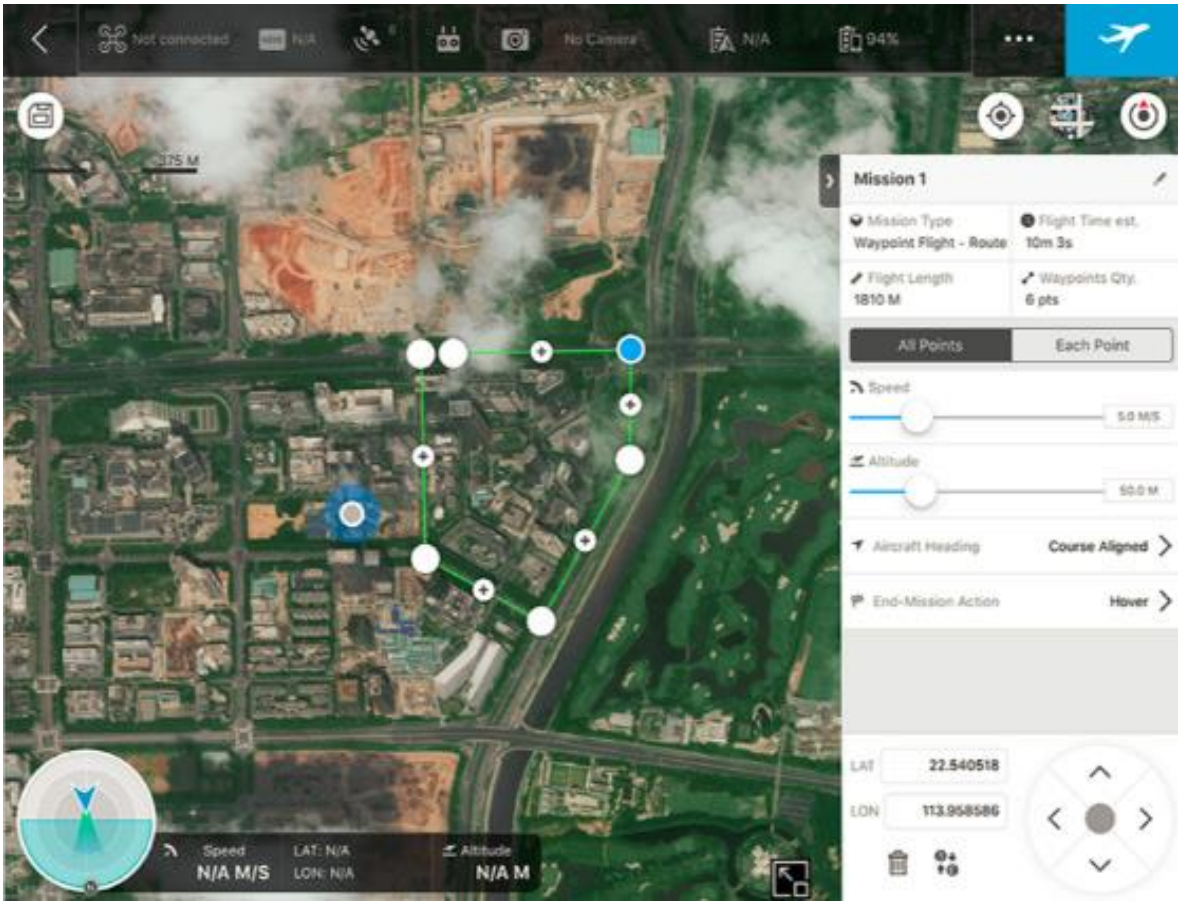


Рисунок 1.5 – Інтерфейс програмного застосунку «DJI GS Pro»

Розглянувши ці аналоги було створено таблицю 1.1 для порівняння їхнього функціоналу з програмним забезпеченням «DronePathfinder».

Таблиця 1.1 – Функціональні характеристики програмних застосунків

Критерії	Path Planner	PX4 Autopilot	Mission Planner	Qground Control	DJI GS Pro	Drone Pathfinder
Вибір алгоритму знаходження шляху	-	-	-	-	-	+
Збереження маршрутів, профілів безпілотників	-	+	+	+	+	+
Наявність мобільного застосунку	-	-	+	+	+	+
Врахування характеристик безпілотника	-	+	+	+	+	+
Додання зон забороненого руху	+	+	+	+	+	+
Загальна оцінка	1	3	4	4	4	5

Аналізуючи вищенаведені дані, «DronePathfinder» має вищий сумарний коефіцієнт відносно аналогів: «PathPlanner» на 80%, «PX4 Autopilot» на 40%, «Mission Planner» на 20%, «QgroundControl» на 20%, «DJI GS Pro» на 20%.

У результаті проведеного аналізу аналогів було створено таблицю з

порівнянням функціоналу, яка демонструє доводить актуальність розробки програмного забезпечення «DronePathfinder».

1.3 Вплив алгоритмів знаходження оптимального шляху на ефективність безпілотного літального апарату

Різноманітні алгоритми маршрутизації мають значний вплив на ефективність безпілотних літальних апаратів (БПЛА), що відображається на їхній точності, швидкості виконання завдань, а також на витратах палива та енергії. Новітні технології використовують алгоритми генетичного пошуку та методи машинного навчання для оптимізації маршрутів БПЛА. Ці методи дозволяють апаратам навчатися та адаптуватися до змін у середовищі, що сприяє зменшенню часу виконання місій і підвищенню їхньої точності.

Вибір оптимального алгоритму маршрутизації залежить від конкретних вимог місії та умов експлуатації. Наприклад, в умовах, де необхідно швидко реагувати на зміни в середовищі, алгоритми машинного навчання можуть бути більш ефективними, тоді як у статичних умовах такі алгоритми, як алгоритм Дейкстри, можуть бути прийнятними. Важливим аспектом є здатність алгоритмів маршрутизації адаптуватися до змінних умов. Наприклад, при зміні метеорологічних умов або виявленні перешкод на маршруті, ефективний алгоритм маршрутизації повинен швидко переключатися на альтернативний шлях або вирішувати проблему обхідним маневром [4].

Оптимізація витрат палива та ресурсів є ще одним важливим аспектом. Ефективний алгоритм маршрутизації повинен вибирати маршрут, який дозволяє зменшити споживання палива та мінімізувати витрати ресурсів, що є критичним у довготривалих місіях або у важкодоступних регіонах.

До інших факторів, що впливають на ефективність алгоритмів маршрутизації, належать інтеграція з системами позиціонування (GPS тощо), врахування обмежень маршруту (наприклад, обмеження висоти польоту або зони забороненого польоту), а також забезпечення безпеки польотів та уникнення зіткнень.

Загалом, ефективність безпілотних літальних апаратів значною мірою залежить від розроблених та впроваджених алгоритмів маршрутизації. Розуміння їхнього впливу та вибір оптимального підходу є ключовим для досягнення успішних місій та забезпечення надійної роботи апаратів у різних умовах експлуатації.

1.4 Постановка задач дослідження

Проаналізувавши переваги та недоліки існуючих системи знаходження шляху безпілотного літального апарату було визначено наступні завдання, які необхідно виконати для розробки мобільної застосунку:

– розробити загальний алгоритм системи;

– розробити метод знаходження оптимального шляху безпілотного літального апарату на основі алгоритмів Дейкстри та A*;

– розробити метод врахування характеристик безпілотника;

– розробити зручний та адаптивний графічний інтерфейс мобільного застосунку;

– розробити модуль для локального збереження маршрутів на пристрої користувача;

– здійснити тестування мобільного застосунку згідно поставлених задач;

– розробити інструкцію користувача.

1.5 Висновки

У першому розділі було розглянуто стан систем знаходження оптимального шляху безпілотного літального апарату, вплив алгоритмів знаходження оптимального шляху на ефективність безпілотного літального апарату. Було проведено аналіз аналогів програмного забезпечення «DronePathfinder», що довів доцільність і актуальність розробки.

2 РОЗРОБКА МЕТОДІВ, МОДЕЛІ ТА АЛГОРИТМІВ МОБІЛЬНОГО ЗАСТОСУНКУ

2.1 Аналіз вхідних та вихідних даних системи

Вхідними даними системи є два масив точок на мапі. Перший масив визначає точки, через які безпілотник повинен пройти, щоб виконати своє завдання Другий масив визначає зони запобігання, які визначаються як коло з координатами центру та радіусом. У випадку, якщо точка з першого масиву знаходиться у радіусі зони запобігання – вона ігнорується алгоритмом. Ці дані є важливими для алгоритму пошуку шляху, оскільки вони включають координати старту, місця призначення, проміжні точки, які потрібно відвідати, а також обмеження, такі як заборонені зони або області з високим ризиком.

Точки на земній поверхні задаються за допомогою координат, які включають широту та довготу. Широта визначає положення точки на північ або південь від екватора, тоді як довгота визначає положення на схід або захід від початкового меридіану, який проходить через Гринвіч у Лондоні.

Широта вимірюється у градусах (°), хвилинах (') та секундах ("). Вона може мати значення від 0° на екваторі до 90° на полюсах. Широти, що знаходяться на північ від екватора, позначаються як північна широта (N), а ті, що на південь — як південна широта (S).

Довгота також вимірюється у градусах, хвилинах та секундах. Вона може мати значення від 0° у Гринвічі до 180° на схід або захід. Довготи на схід від Гринвіча позначаються як східна довгота (E), а на захід — як західна довгота (W).

У програмуванні, особливо при роботі з картами та геолокаційними даними, координати часто представляються у вигляді об’єктів, які містять числові значення широти та довготи. Наприклад, у бібліотеці osmdroid для Android, клас GeoPoint використовується для створення об’єкта з цими двома параметрами.

Об’єкти GeoPoint можуть бути зібрані в List<GeoPoint>, щоб представити

послідовність точок, які безпілотник повинен відвідати під час польоту.

Вихідні дані системи представляються у вигляді графу, що є універсальним способом візуалізації маршрутів.

Оптимальний шлях — це такий маршрут, який задовольняє певні критерії, такі як найкоротша відстань, найменший час подорожі, або найменша вартість. У контексті osmdroid, оптимальний шлях може бути представлений як послідовність GeoPoint об’єктів, які визначають точки на карті, через які повинен пройти безпілотний літальний апарат. Граф на мапі може бути візуалізований за допомогою ліній, які з’єднують точки. Кожна лінія має певну вагу, яка може відображати відстань або час між точками.

2.2 Розробка загального алгоритму роботи системи

Для розробки мобільної застосунку для знаходження оптимального шляху необхідно розробити загальний алгоритм системи, згідно якого буде працювати мобільний застосунок (рисунки 2.1 – 2.2).

Після відкриття мобільного застосунку користувачу доступно 8 основних дій.

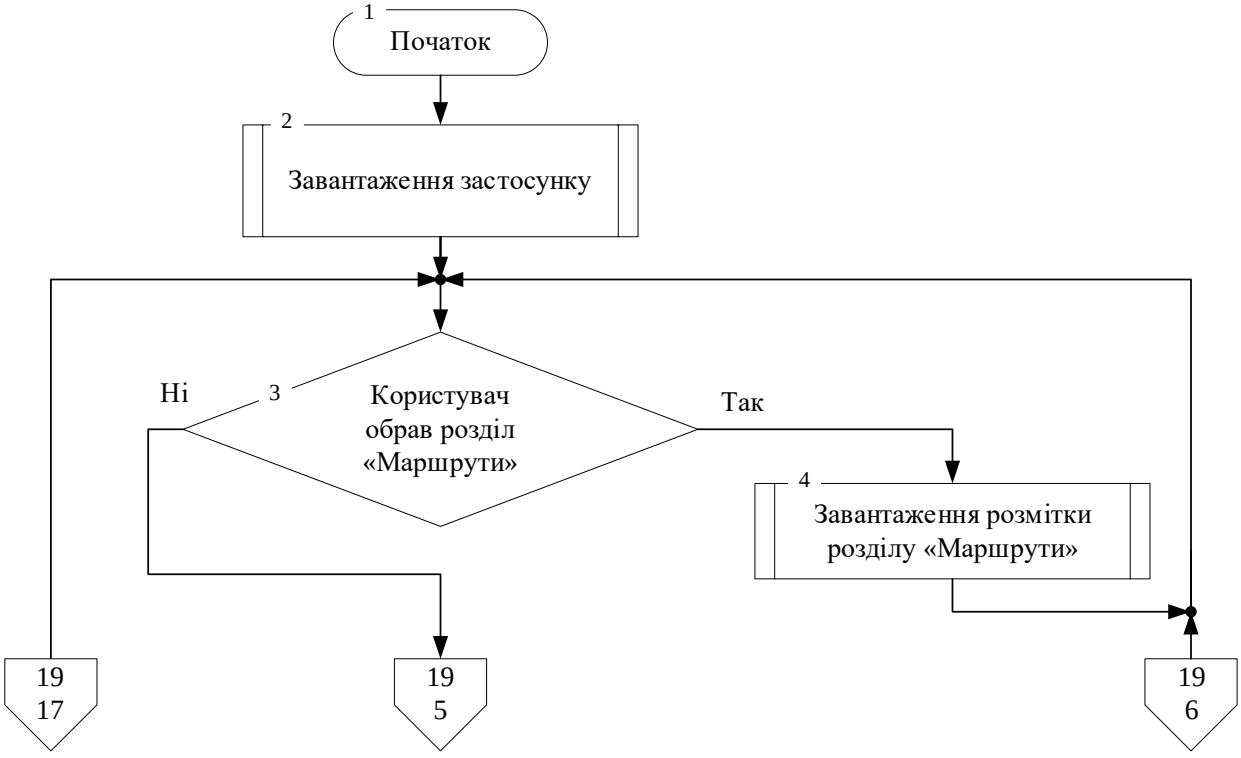


Рисунок 2.1 – Загальний алгоритм роботи застосунку «DronePathfinder»

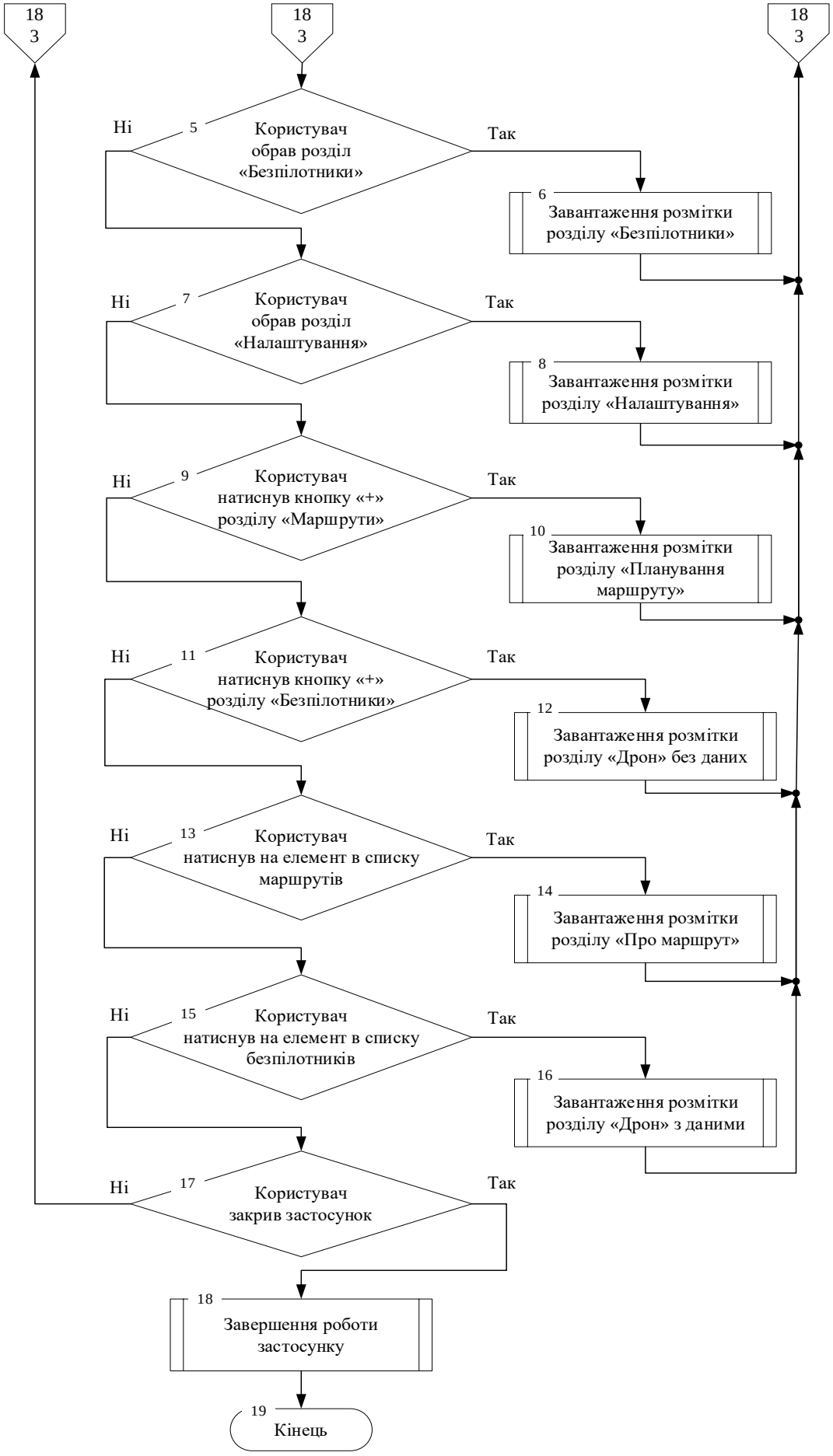


Рисунок 2.2 – Продовження загального алгоритму застосунку «DronePathfinder»

2.3 Розробка методу визначення оптимального шляху на основі алгоритмів Дейкстри та A*

Головною задачею мобільної застосунку є визначення та побудова оптимального шляху для безпілотного літального апарату.

Метод findShortestPath використовує алгоритм на вибір користувача – алгоритм Дейкстри або алгоритм A* – для знаходження найкоротшого шляху між точками на карті. Ці алгоритми є одними з найвідоміших алгоритмів пошуку шляху в графі, які знаходять шлях з найменшою сумарною вагою між двома вершинами.

Спочатку метод ініціалізує граф на основі переданих точок. Граф складається з вершин, які представляють точки, та ребер, які представляють відстані між цими точками.

Початкова та кінцева точки визначаються як перша та остання точки у списку відповідно.

Далі, метод встановлює відстань до початкової точки як нуль та до всіх інших точок як нескінченність. Це робиться для того, щоб під час роботи алгоритму можна було порівнювати відстані та визначати найкоротші шляхи.

Використання Log.d дозволяє вивести в лог інформацію про наявність початкової та кінцевої точок у графі.

PriorityQueue використовується для визначення порядку обробки точок на основі їх відстаней, що дозволяє ефективно знаходити найкоротші шляхи.

Використовуючи чергу з пріоритетом, метод обробляє кожну точку, починаючи з початкової. Для кожної точки перевіряються всі сусідні точки, і якщо знайдено коротший шлях до сусідньої точки, то відстань до неї оновлюється.

Коли поточна точка є кінцевою, метод відтворює шлях від кінця до початку, використовуючи інформацію про попередників, яка зберігається під час виконання алгоритму. Якщо шлях до кінцевої точки не знайдено, метод повертає порожній список.

Для ілюстрації роботи алгоритму було створено блок-схему, яка детально

показує кожен крок процесу, від ініціалізації до відтворення шляху (рисунки 2.3 – 2.5).

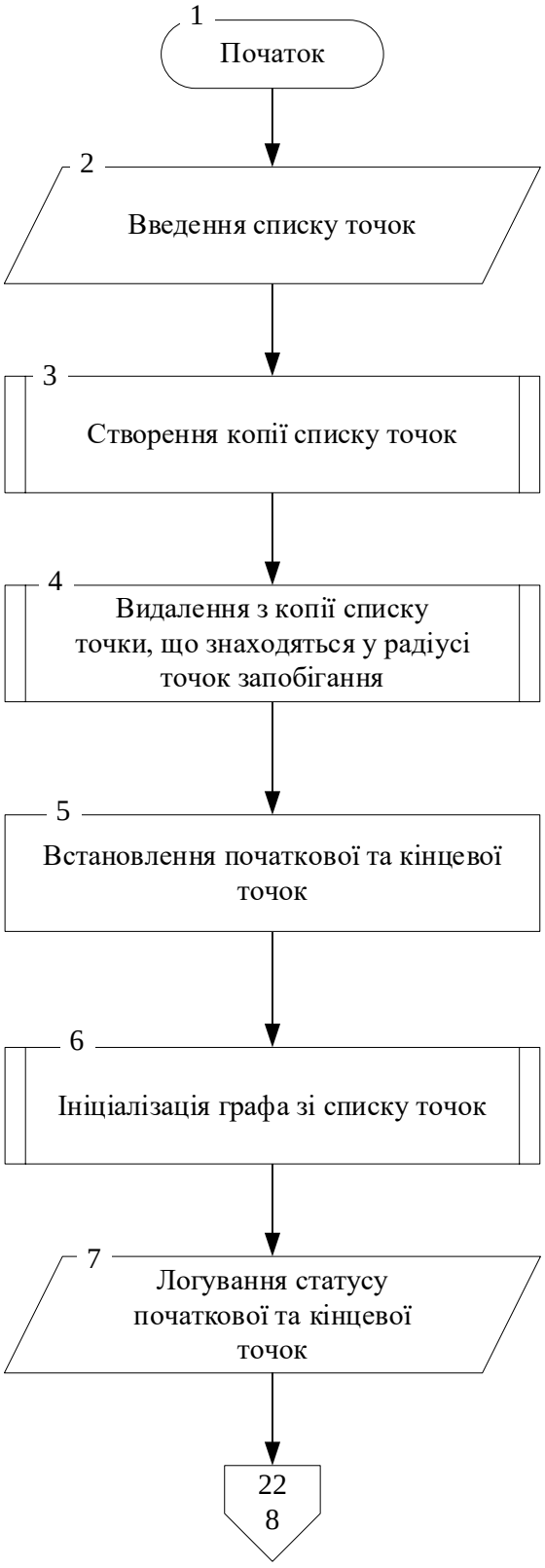


Рисунок 2.3 – Блок-схема методу визначення оптимального шляху на основі алгоритму Дейкстри (частина 1)

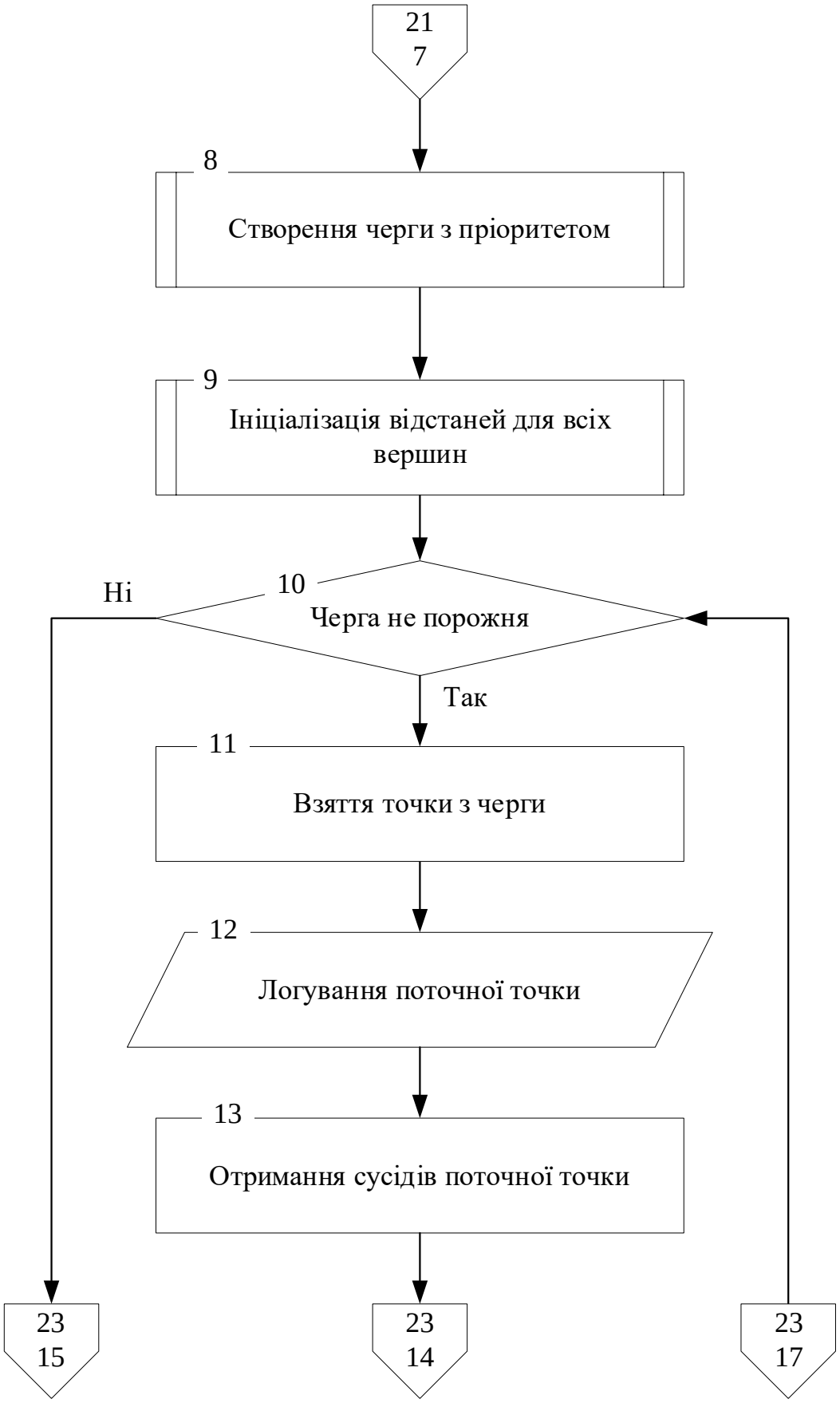


Рисунок 2.4 – Блок-схема методу визначення оптимального шляху на основі алгоритму Дейкстри (частина 2)

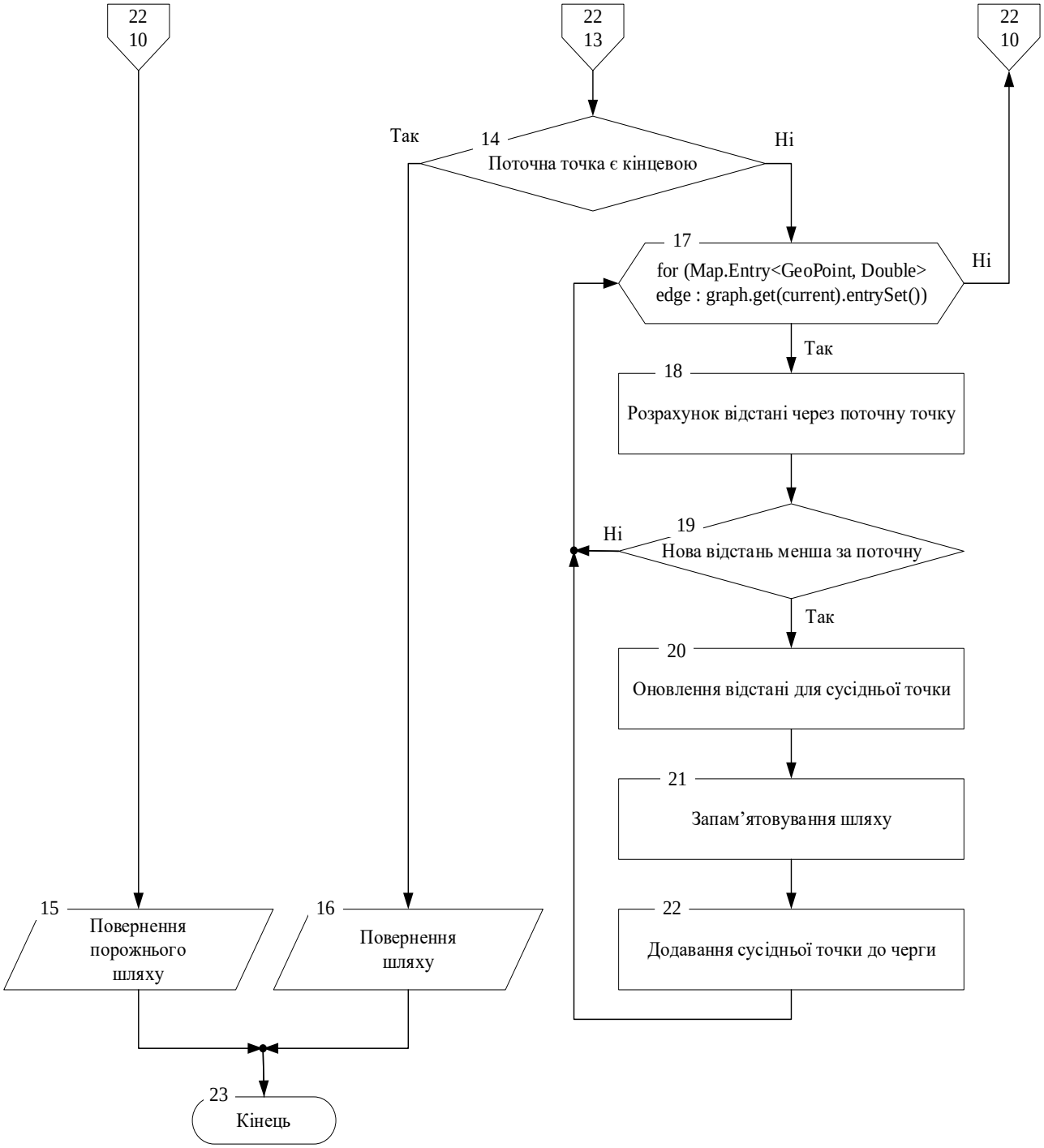


Рисунок 2.5 – Блок-схема методу визначення оптимального шляху на основі алгоритму Дейкстри (частина 3)

Цей алгоритм дозволяє повністю автоматизувати процес визначення оптимального шляху. Його ефективність робить придатним для використання в реальному часі, навіть для великих графів, а універсальність і масштабованість роблять його незамінним для вирішення широкого спектру задач, пов’язаних з маршрутизацією.

2.4 Розробка методу врахування характеристик безпілотника

Іншою не менш важливою задачею мобільної застосунку є врахування характеристик безпілотника та надання рекомендацій.

Надання рекомендацій відбувається автоматично на основі статусу маршруту, що визначається при створенні маршруту і оновлюється при доданні безпілотників до маршруту. Кожен маршрут обов'язково має один із чотирьох статусів:

- NO_DRONE – вказує на відсутність дрона;
- ROUTE_TOO_LONG – вказує на те, що маршрут занадто довгий для дрона;
- OVERWEIGHT – означає, що дрон перевантажений;
- GOOD – вказує на те, що дрон готовий до польоту.

Спочатку відбувається перевірка на наявність дрону: якщо до маршруту не доданий дрон, то метод повертає маршруту статус NO_DRONE і надає користувачу повідомлення з рекомендацією: «Маршруту не призначений дрон! Призначте дрон натиснувши кнопку внизу екрану.».

В іншому випадку, коли до маршруту доданий дрон, то відбувається ще декілька перевірок:

1. Якщо максимальна дальність польоту дрона менша за обраховану довжину маршруту, метод повертає маршруту статус ROUTE_TOO_LONG і надає повідомлення з рекомендацією: «Довжина маршруту більша за дальність польоту дрону! Призначте інший дрон натиснувши кнопку внизу екрану.».
2. Якщо вантажопідйомність дрона менше за вказану вагу вантажу, метод повертає маршруту статус OVERWEIGHT і надає повідомлення з рекомендацією: «Вага вантажу більша за вантажопідйомність дрону! Призначте інший дрон натиснувши кнопку внизу екрану.».
3. Якщо жодна з попередніх умов не виконується, метод повертає маршруту статус GOOD.

На рисунку 2.6 зображено блок-схему методу врахування характеристик безпілотника.

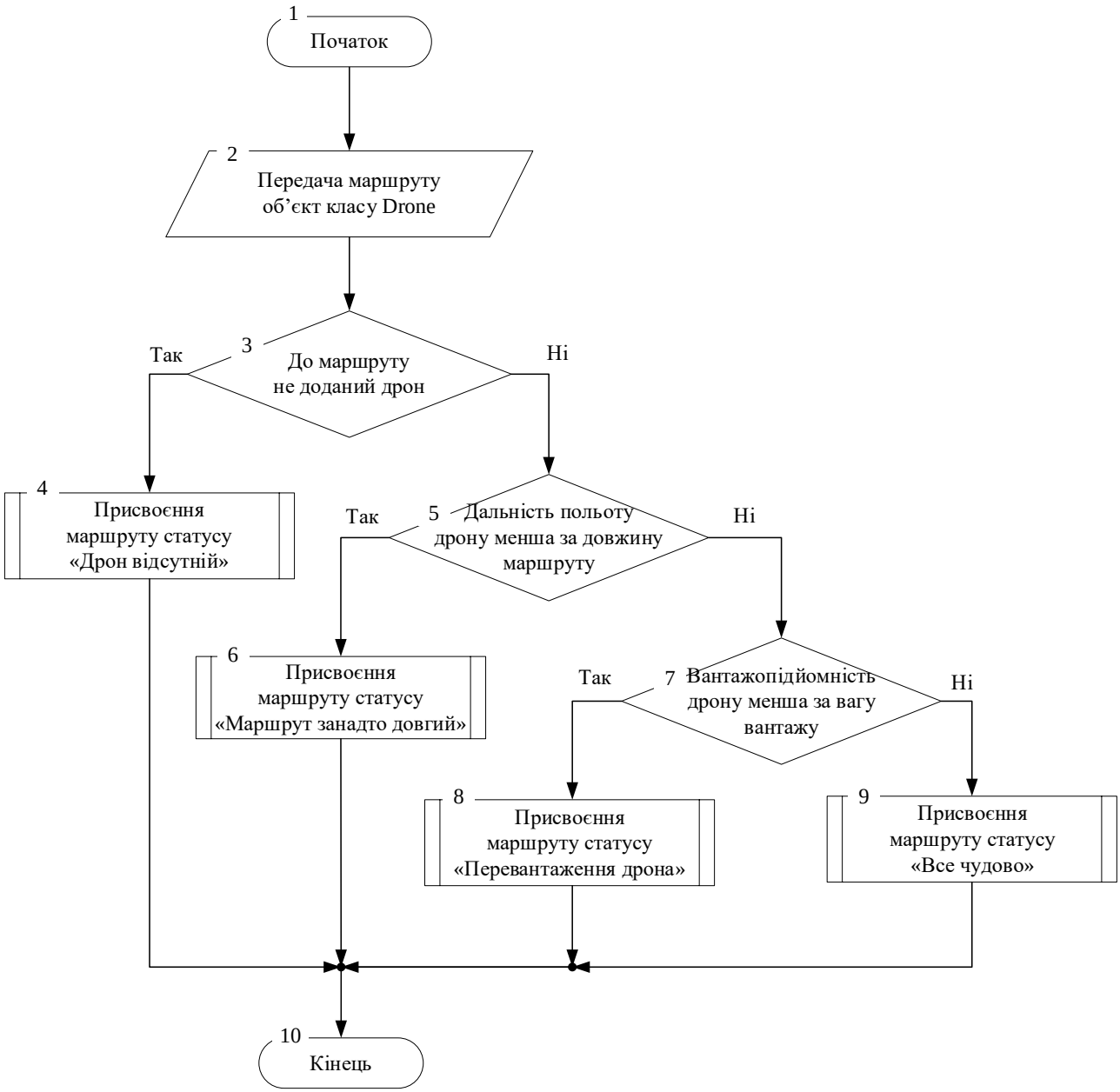


Рисунок 2.6 – Блок-схема методу врахування характеристик безпілотника

2.5 Розробка моделі системи

UML діаграма діяльності є ключовим інструментом для візуалізації та планування взаємодії користувача з мобільною системою. Вона допомагає розробникам зрозуміти послідовність дій та взаємозв’язки між різними модулями, а також сприяє ефективному проєктуванню користувацького інтерфейсу [14].

Для наочного представлення функціонування програмних модулів мобільної системи було розроблено UML діаграму діяльності (рисунок 2.7).

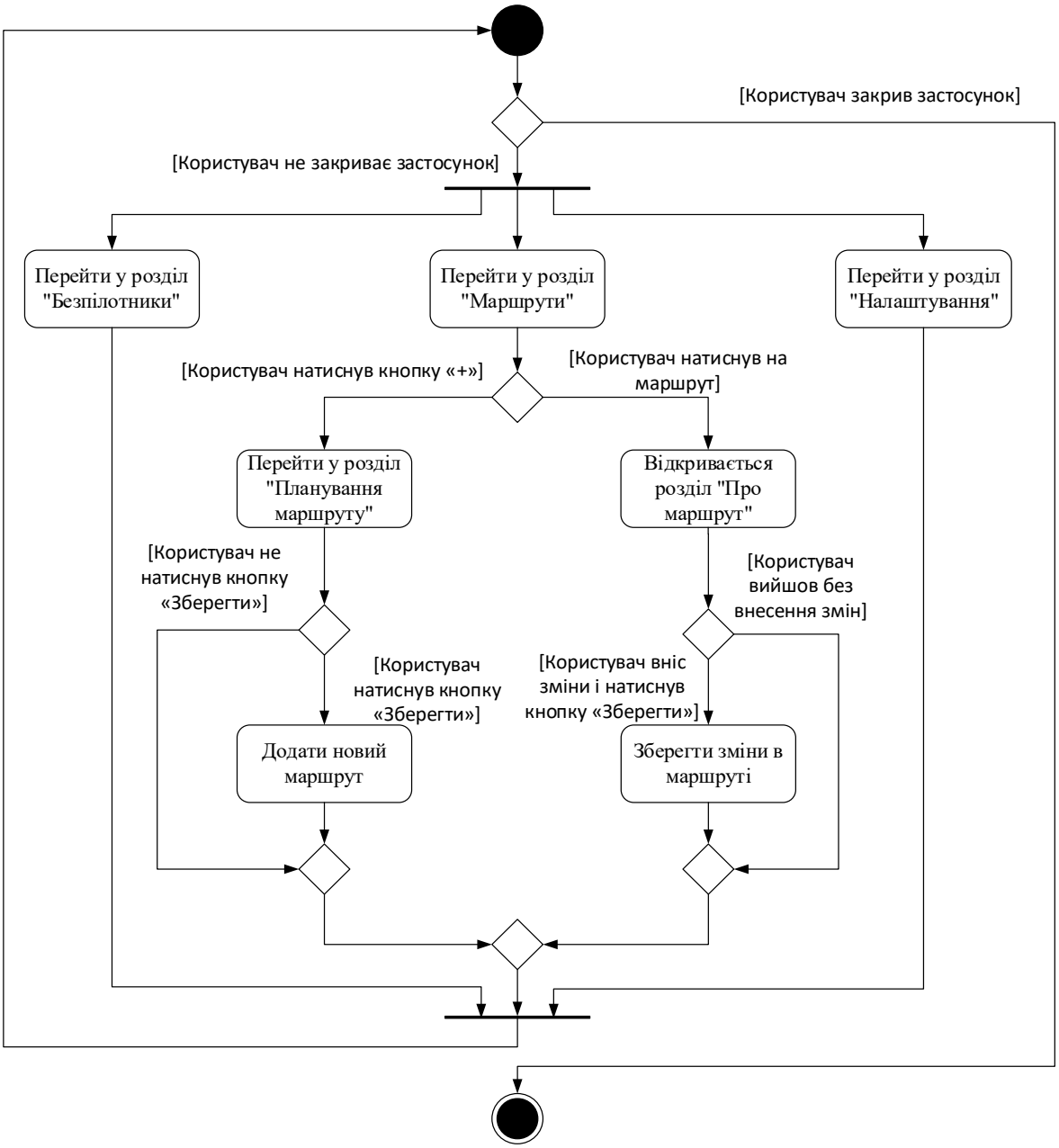


Рисунок 2.7 – Діаграма діяльності мобільного застосунку

2.6 Висновки

У другому розділі було проведено аналіз вхідних та вихідних даних мобільного застосунку. Розроблено загальний алгоритм роботи мобільного застосунку, метод знаходження оптимального шляху на основі алгоритму Дейкстри та A*, метод врахування характеристики безпілотника. Також описано модель роботи мобільного застосунку за допомогою UML діаграм діяльності.

3 РОЗРОБКА СТРУКТУРИ ТА ПРОГРАМНИХ МОДУЛІВ МОБІЛЬНОГО ЗАСТОСУНКУ

3.1 Аналіз і обґрунтування вибору мови програмування

У світі мобільної розробки, вибір мови програмування може мати значний вплив на процес створення застосунку. При розробці мобільних застосунків для платформи Android зазвичай використовуються дві мови, які часто порівнюються – це Java та Kotlin. Обидві мови мають свої переваги та недоліки, і вибір між ними залежить від багатьох факторів.

Одним з найпоширеніших виборів для розробки Android-застосунків є мова програмування Java. Вона відома своєю масштабованістю, надійністю та безпекою. Однак вона має складний синтаксис та детальний код, що може призвести до поганої продуктивності. Java є основною мовою для розробки на платформі Android, що дозволяє легко і ефективно створювати різноманітні застосунки, використовуючи різні бібліотеки та інструменти.

Kotlin, розроблена JetBrains, - це об’єктно-орієнтована мова для JVM та Android, яка зосереджена на безпеці, ясності та взаємодії. Вона є популярною для розробки застосунків Android через її легкість вивчення та ефективність.

Java має ряд переваг перед Kotlin [15]:

– Java існує вже більше двох десятиліть і має стабільну платформу з великою кількістю ресурсів для навчання та багатьма можливостями для розробників;

– Java має багато бібліотек та фреймворків, доступних для використання;

– Java має прямолінійний та зрозумілий синтаксис;

– у Java вищі стандарти безпеки, і виправлення помилок може бути простішим у порівнянні з Kotlin.

До переваг Kotlin перед Java відносять [16]:

– Kotlin має систему типів, яка запобігає помилкам через null, що зводить ризик виникнення NullPointerException до нуля;

– Kotlin повністю сумісний з Java, що дозволяє використовувати Kotlin

разом з існуючими Java бібліотеками та фреймворками;

– Kotlin включає сучасні особливості, такі як лямбда-вирази та розширені функції, які полегшують розробку;

– Kotlin дозволяє зменшити кількість шаблонного коду (boilerplate), що робить розробку швидшою та ефективнішою.

Порівняння швидкодії цих мов програмування у різних ситуаціях наведено в таблиці 3.1 [17].

Таблиця 3.1 – Порівняння швидкодії Java та Kotlin у різних ситуаціях

Тест-кейс	Java	Kotlin
Ітерація циклу	Швидше	Повільніше
Створення об’єктів	Повільніше	Швидше
Виклик методів	Швидше	Повільніше

Проаналізувавши переваги Java та Kotlin, було прийнято рішення використовувати мову програмування Java для розробки модулів мобільного застосунку «DronePathfinder».

3.2 Аналіз і обґрунтування вибору засобу відображення мапи

Однією з ключових функцій багатьох застосунків є відображення інформації на мапі, що дозволяє користувачам зручно навігуватися, знаходити місця та отримувати необхідну географічну інформацію. Вибір оптимального засобу для відображення мапи в Android-застосунку є важливим рішенням, яке впливає на функціональність, продуктивність та задоволення користувачів. Двома найпопулярнішими сервісами для інтеграції карт є Google Maps API та OpenStreetMap. Обидва пропонують унікальні можливості та обмеження, які слід розглянути при виборі.

Google Maps API – це набір інструментів та сервісів, які надаються Google для інтеграції та використання картографічних можливостей у розробці Android-

додатків. API дозволяє отримати доступ до різноманітних функцій Google Maps, таких як відображення карти, навігація, пошук місць, відображення дорожнього трафіку та багато інших [18].

До особливостей Google Maps API належать:

– має багатий функціонал, включаючи детальні карти, навігацію, відображення дорожньої трафіку, вбудовані сервіси пошуку місць та інші корисні функції;

– пропонує розширені можливості, такі як Street View, реальний час заторів, та інтеграцію з іншими сервісами Google;

– надає обширну документацію та API, які полегшують інтеграцію та розробку;

– має безкоштовний тарифний план, але вимагає ключа API для використання на комерційних проєктах з великим обсягом запитів.

OpenStreetMap (OSM) – це вільно доступний проєкт картографії, який забезпечує відкритий доступ до глобальних карт та географічних даних. У контексті розробки Android-додатків, OpenStreetMap надає API та інструменти для інтеграції картографічних можливостей у додатки [19].

До особливостей OpenStreetMap належать:

– надає відкритий доступ до своїх даних, що дозволяє розробникам використовувати, змінювати та розповсюджувати їх відповідно до потреб своїх проєктів;

– дозволяє кастомізувати карти, додаючи іконки, відстежуючи локації та малюючи форми, що робить її гнучкою для різних типів застосунків;

– дані доступні безкоштовно, але вартість може збільшуватися при використанні додаткових сервісів чи служб, таких як хмарне зберігання даних;

– є гнучкою системою, яка може бути адаптована до різних потреб та масштабуватися від невеликих проєктів до великих масштабів.

Використання OpenStreetMap у розробці Android-застосунків може бути вигідним завдяки його відкритості, гнучкості та активній спільноті.

Враховуючи особливості Google Maps API та OpenStreetMap було

прийнято рішення використовувати OpenStreetMap при розробці модуля відображення мапи у застосунку.

3.3 Розробка структури та інтерфейсу мобільного застосунку

Структурна схема — це візуальне представлення, яке окреслює ключові компоненти програмного продукту, їхні зв’язки та функції. Ця схема може включати елементи, такі як підсистеми, модулі, бібліотеки та інші ресурси, які спільно працюють для досягнення загальної мети програми.

Процес створення структурної схеми часто вимагає ітеративного підходу, де кожен крок додає більше деталей до загальної картини, дозволяючи розробникам краще розуміти та оптимізувати взаємодію між різними частинами програми [20].

Структурна схема застосунку «DronePathfinder» зображено на рисунку 3.1.

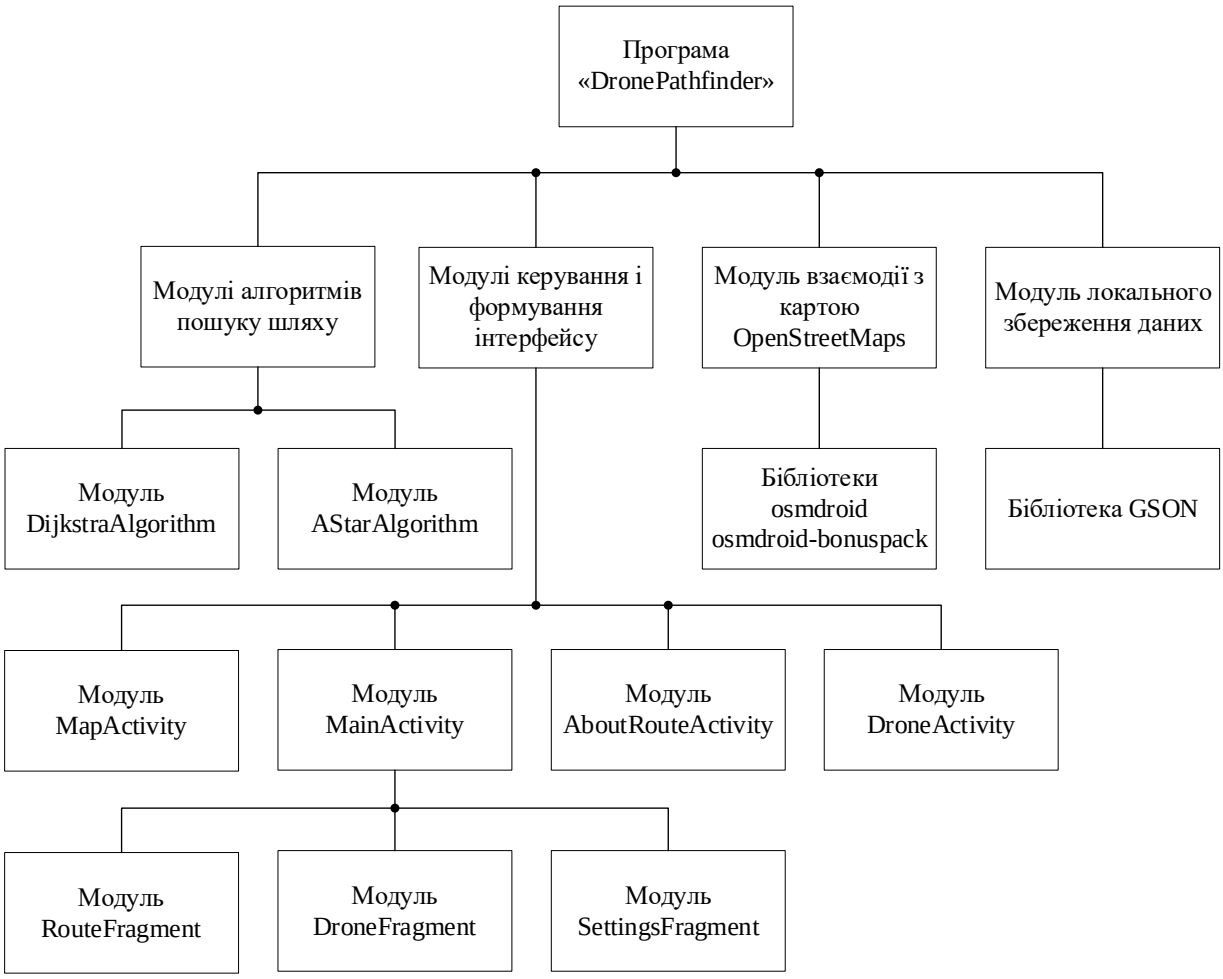


Рисунок 3.1 – Структурна схема застосунку «DronePathfinder»

Інтерфейс користувача складається з чотирьох активностей та трьох фрагментів, що входять у головну активність. До активностей застосунку відносяться:

– MainActivity – використовується для відображення головного розділу;

– MapActivity – використовується для відображення розділу планування маршруту;

– AboutRouteActivity – використовується для відображення розділу «Про маршрут»;

– DroneActivity – використовується для відображення розділу «Дрон».

На рисунку 3.2 зображено графічну схему головної активності застосунку «DronePathfinder».

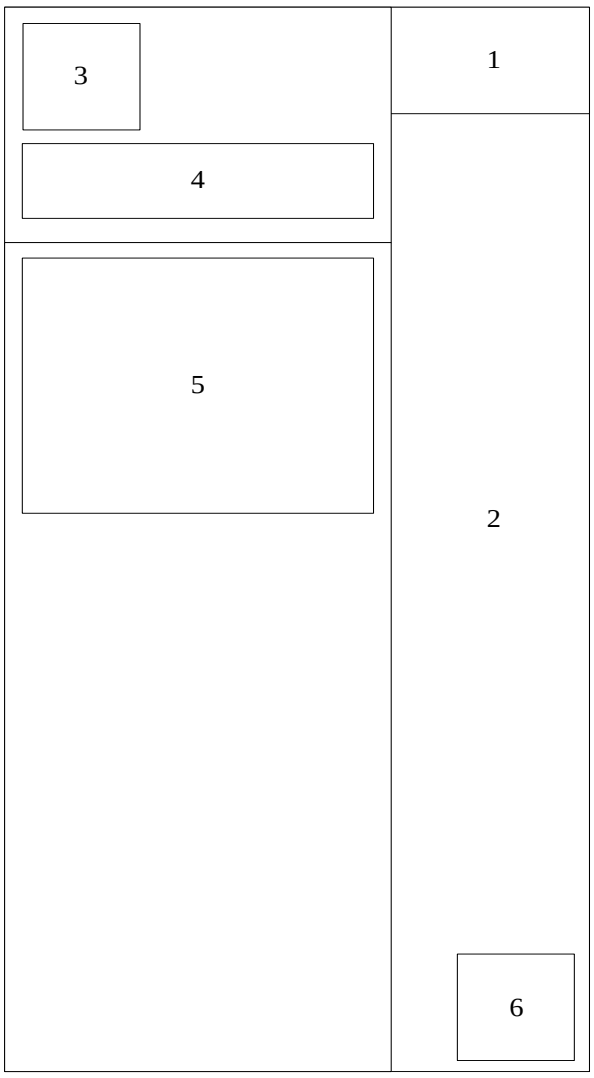


Рисунок 3.2 – Графічна схема головної активності застосунку «DronePathfinder»

До елементів головної активності належать:

1. Заголовок активності.
2. Робоча область активності.
3. Логотип мобільного застосунку «DronePathfinder».
4. Інформація про розробника мобільного застосунку «DronePathfinder».
5. Пункти навігаційного меню – містить пункти «Маршрути», «Безпілотники», «Налаштування».
6. Основна кнопка взаємодії з фрагментом.

Основними кольорами мобільного застосунку було обрано синій та блакитний.

Розробку графічного інтерфейсу користувача було проведено у середовищі Android Studio з використанням вбудованого редактора.

3.4 Використання принципів об’єктно-орієнтованого програмування при розробці мобільного застосунку

Для уникнення повторюваності коду використано один із найважливіших принципів об’єктно-орієнтованого програмування – наслідування. Враховуючи що класи Route та Drone міститимуть однакові поля: назва, статус; для цих класів було створено батьківський клас Object, який містить ці поля. При цьому клас Object є абстрактним, що унеможливлює створення об’єктів саме цього класу.

Використання іншого не менш важливого принципа об’єктно-орієнтованого програмування – інкапсуляції – реалізовано таким чином: усі поля класів Route, Drone, Object та AvoidancePoint мають модифікатор доступу private, а доступ до полів здійснюється з використанням мутаторів доступу.

У взаємозв’язку створених класів використано агрегацію: кожен об’єкт класу Route може містити об’єкт класу Drone та масив об’єктів AvoidancePoint.

AvoidancePoint – клас створений для зберігання інформації про точки запобігання – центр, що описується двома координатами, та його радіус.

На рисунку 3.3 зображено діаграму класів об’єктів призначених для зберігання мобільного застосунку.

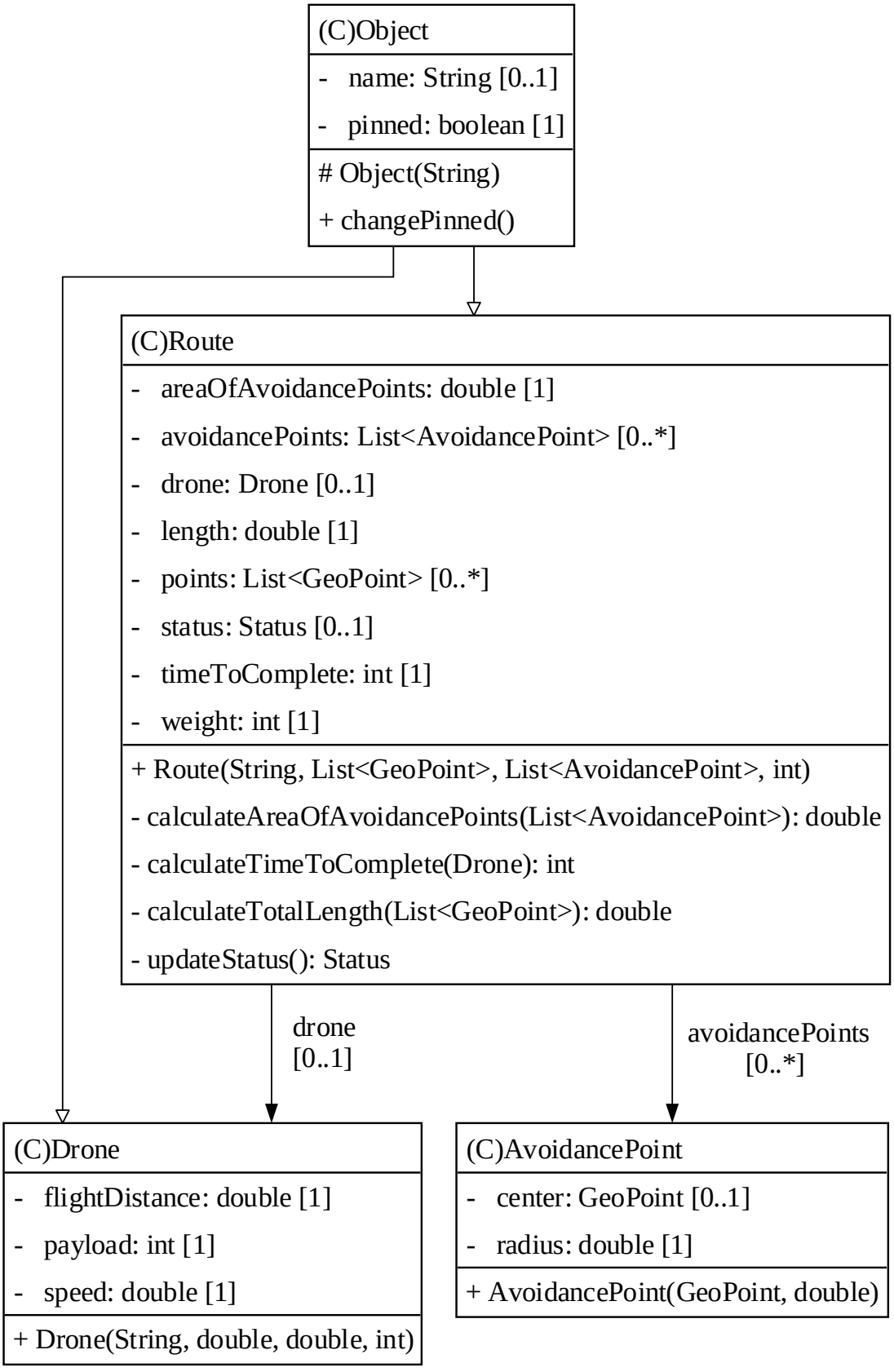


Рисунок 3.3 – Діаграма класів мобільного застосунку

Класи Route, Drone, Object та AvoidancePoint імплементують інтерфейс Serializable, що надає можливість виконувати серіалізацію/десереалізацію

об’єктів цих класів.

3.5 Розробка модуля локального збереження даних

Основна задача модуля локального збереження даних полягає у керуванні списком маршрутів у мобільному додатку. Цей модуль є важливою частиною застосунку, оскільки він дозволяє користувачам переглядати, зберігати та відновлювати маршрути, які вони створили або використовували. Завдяки цьому, користувачі можуть легко повертатися до своїх улюблених або часто використовуваних маршрутів.

Метод saveRoutesToSharedPreferences() використовується для збереження даних про маршрути в локальній пам’яті пристрою за допомогою SharedPreferences (рисунок 3.4).

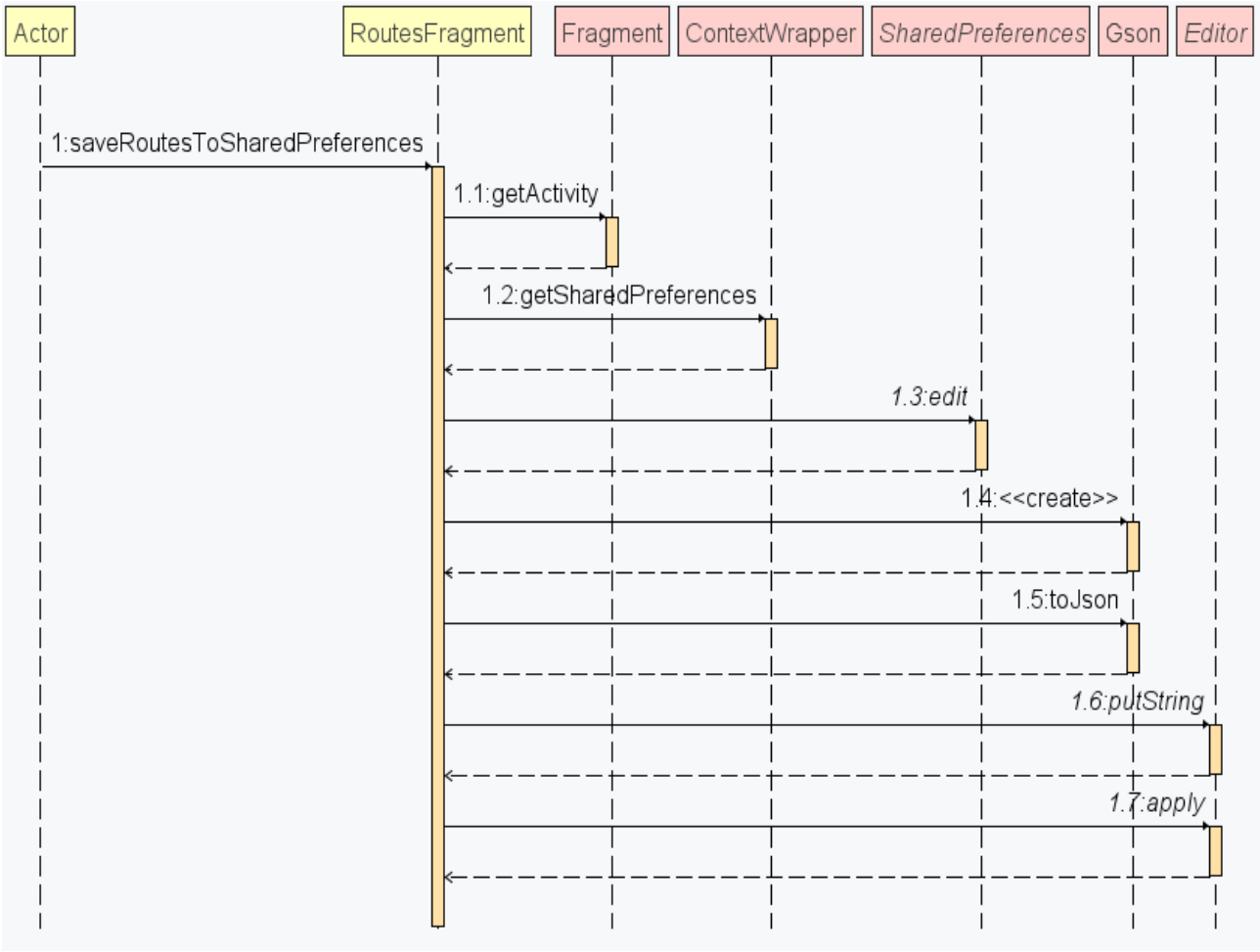


Рисунок 3.4 – Діаграма послідовності методу saveRoutesToSharedPreferences

Він перетворює список маршрутів у формат JSON, а потім зберігає цей JSON рядок під певним ключем. Це дозволяє додатку зберігати стан маршрутів, щоб користувачі могли доступити до своїх маршрутів навіть після перезапуску додатку або пристрою. Такий підхід є стандартною практикою для збереження невеликих обсягів даних, які не вимагають постійного доступу до мережі або бази даних.

Метод loadRoutesFromSharedPreferences() відновлює збережений список маршрутів з локальної пам’яті пристрою (рисунок 3.5).

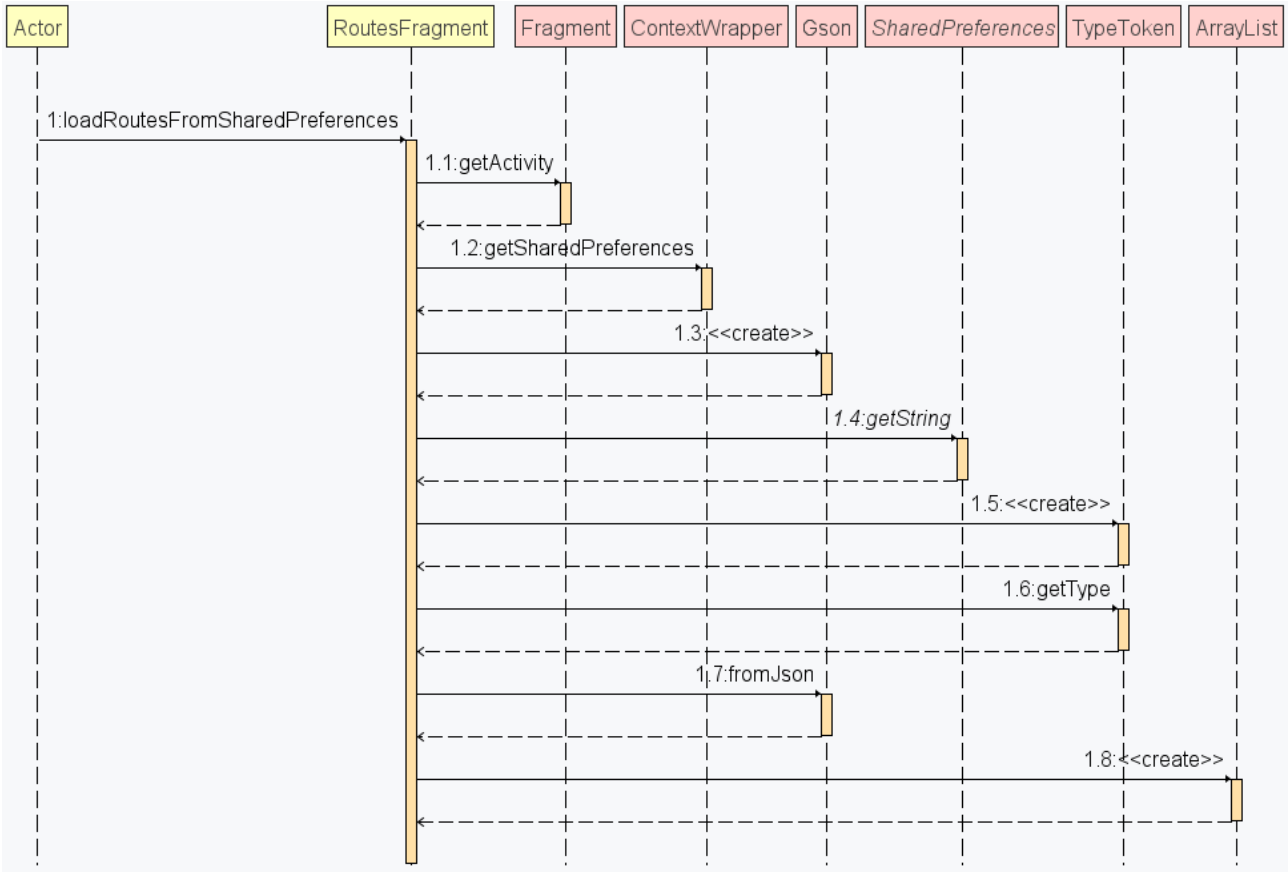


Рисунок 3.5 – Діаграма послідовності методу loadRoutesFromSharedPreferences

Він використовує SharedPreferences для отримання рядка у форматі JSON, який містить інформацію про маршрути. Потім цей рядок перетворюється назад у список об’єктів Route за допомогою бібліотеки Gson. Якщо збережені дані відсутні, метод створює новий порожній список. Таким чином, користувачі можуть продовжувати працювати зі своїми маршрутами після перезапуску

додатку або пристрою.

Для керування списком маршрутів використовується адаптер, який дозволяє відображати дані про маршрути у вигляді списку на екрані.

Він перевизначає метод getView, щоб кожен маршрут у списку мав своє представлення. Коли користувач переглядає список, для кожного маршруту відображається інформація, яка включає назву маршруту, координати початкової та кінцевої точок, а також його довжину. Ця інформація форматується у читабельний текст, який потім встановлюється як текст для відображення у компоненті TextView.

3.6 Висновки

У третьому розділі описано структуру мобільного застосунку, використання принципів об’єктно-орієнтованого програмування при його розробці; проаналізовано та обґрунтовано вибір мови програмування та технологій для розробки системи. Проаналізувавши вимоги роботи обрано мову програмування Java. У якості засобу відображення мапи обрано osmdroid OpenStreetMap через широкі можливості кастомізації. Розроблено інтерфейс мобільного застосунку, модуль локального збереження даних. Збереження даних у локальній пам’яті пристрою реалізовано за допомогою GSON.

4 ТЕСТУВАННЯ СИСТЕМИ

4.1 Тестування мобільного застосунку

Для моніторингу продуктивності застосунку «DronePathfinder», логування подій та знаходження виключних ситуацій використовується Logcat (рисунок 4.1).

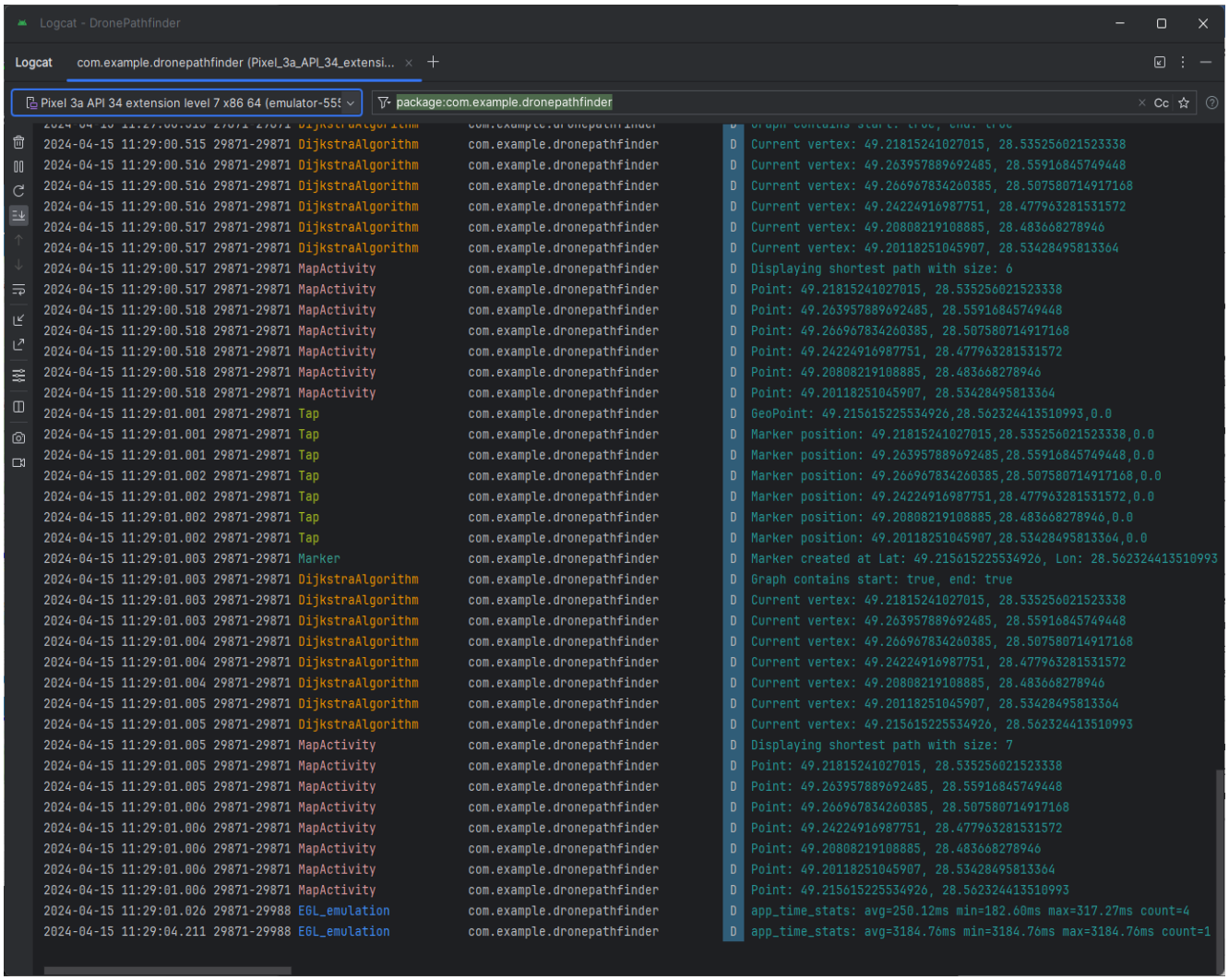


Рисунок 4.1 – Моніторинг з використанням Logcat

Logcat — це засіб логування, який виводить журнал системних повідомлень, при цьому можна додати свої повідомлення за допомогою класу Log [21]. Цей інструмент дозволяє відстежувати повідомлення, пов’язані з виконанням застосунка та операційної системи Android. Кожен лог містить дату,

часову мітку, ідентифікатор процесу та потоку, тег, ім’я пакета, пріоритет та повідомлення. При виникненні виняткової ситуації Logcat показує повідомлення та пов’язаний стек викликів з посиланнями на рядки коду.

Юніт тести дуже важливі для розробки програмного забезпечення, оскільки вони забезпечують кілька ключових переваг [22]:

1. Юніт тести перевіряють, що окремі частини коду (модулі, класи, функції) працюють належним чином.

2. Вони допомагають виявити помилки та проблеми на ранніх етапах розробки, що може значно зменшити витрати на виправлення.

3. Написання юніт тестів може спонукати до кращого дизайну програми, оскільки вони вимагають, щоб код був легко тестованим.

4. Юніт тести можуть бути автоматизовані та виконуватися часто, що забезпечує постійну перевірку коду на наявність помилок.

Для перевірки роботи правильності алгоритму Дейкстри і виявлення помилок було створено юніт тести.

Результат виконання юніт тестів зображено на рисунку 4.2

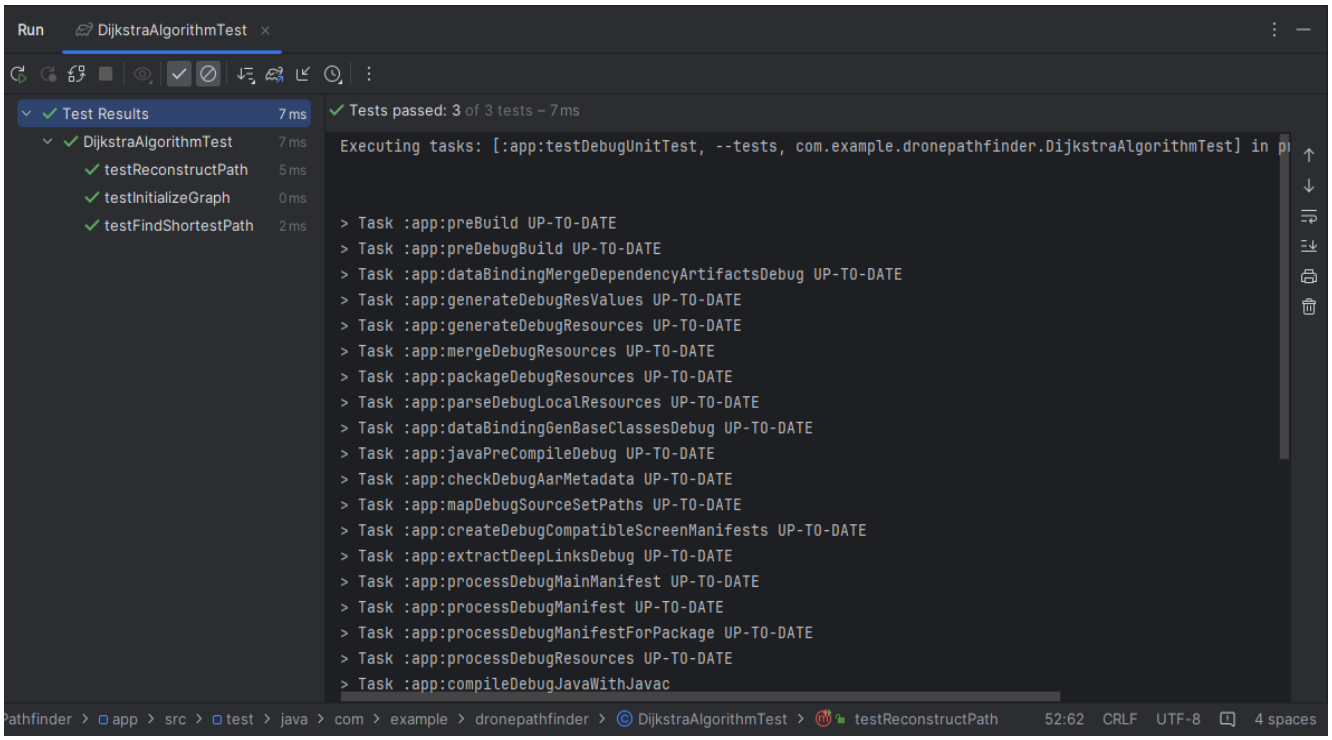


Рисунок 4.2 – Результат виконання юніт тестів

4.2 Розробка інструкції користувача

Інструкція користувача передбачає визначення технічних вимог для запуску мобільного застосунку. Деталі щодо мінімальної конфігурації мобільного пристрою наведено в таблиці 4.1.

Таблиця 4.1 – Мінімальна конфігурація

Операційна система	Android Q (10.0)
Процесор	Чотирьохядерний процесор з частотою 1.2 ГГц
Об’єм оперативної пам’яті	2 ГБ
Об’єм внутрішньої пам’яті	8 ГБ
Роздільна здатність екрану	720p (HD)
Ємність батареї	3000 mAh
Підтримка мережі	4G LTE

Деталі щодо рекомендованої конфігурації мобільного пристрою наведено в таблиці 4.2.

Таблиця 4.2 – Рекомендована конфігурація

Операційна система	Android R (11.0)
Процесор	Восьмиядерний процесор з частотою 2.0 ГГц
Об’єм оперативної пам’яті	4 ГБ або більше
Об’єм внутрішньої пам’яті	64 ГБ або більше
Роздільна здатність екрану	1080p (Full HD) або вище
Ємність батареї	4000 mAh або більше
Підтримка мережі	5G

Інсталяційний файл мобільного застосунку поширюється у форматі .APK файлу.

Для інсталяції застосунку «DronePathfinder» користувачу необхідно виконати такі дії:

1. Завантажити файл DronePathfinder.apk на свій мобільний пристрій.
2. Перейти до «Налаштувань» на пристрої.
3. Обрати розділ «Безпека» або «Приватність».
4. Знайти опцію «Встановлення з невідомих джерел» і активувати її.
5. Відкрити файловий менеджер на пристрої.
6. Знайти завантажений APK-файл.
7. Натиснути на файл, щоб почати встановлення. У випадку якщо система запитає дозвіл на встановлення застосунку, користувачу необхідно підтвердити свій вибір.
8. Дочекатись повідомлення про успішне встановлення.

Після завершення встановлення користувач отримує змогу відкрити застосунок через меню додатків або піктограму на головному екрані.

Щоб видалити застосунок з мобільного пристрою, користувачу необхідно виконати такі дії:

1. Відкрити застосунок «Налаштування» на пристрої.
2. Прокрутити до розділу «Додатки» або «Менеджер додатків».
3. Знайти та обрати застосунок «DronePathfinder».
4. Натиснути на кнопку «Видалити» (рисунок 4.3).

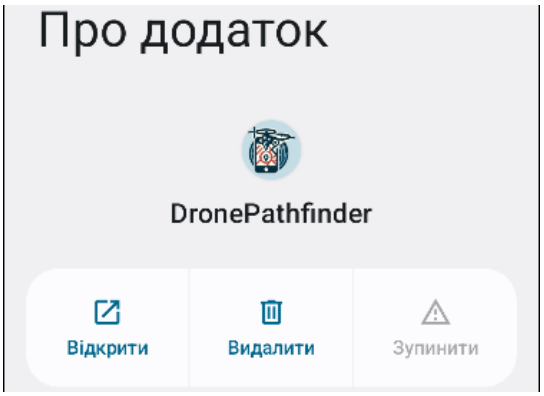


Рисунок 4.3 – Сторінка «Про додаток»

5. Необхідно підтвердити вибір, якщо система запитає про це (рисунок 4.4).

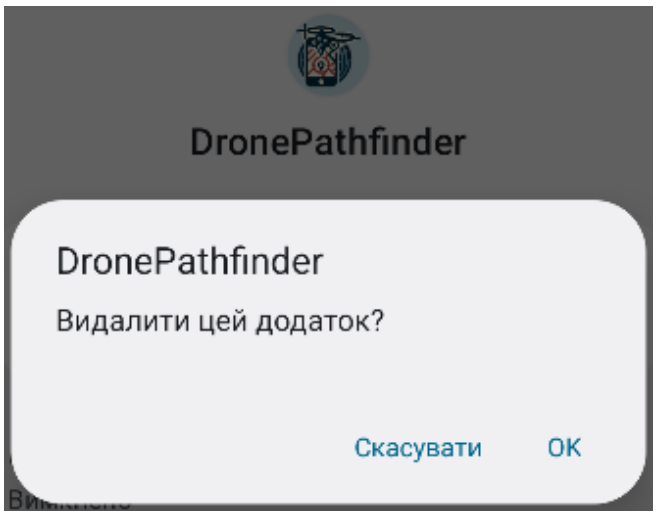


Рисунок 4.4 – Вікно підтведження видалення застосунку

Після видалення застосунку видаляються усі дані застосунку, в тому числі маршрути, профілі безпілотників та налаштування застосунку.

При першому відкритті застосунку користувача зустрічає екран завантаження з логотипом (рисунок 4.5).

Рисунок 4.5 – Екран завантаження застосунку

Після завантаження застосунку відкривається розділ «Маршрути» у якому

користувач може керувати переліком своїх маршрутів. При першому відкритті застосунку розділ «Маршрути» є порожнім (рисунок 4.6).

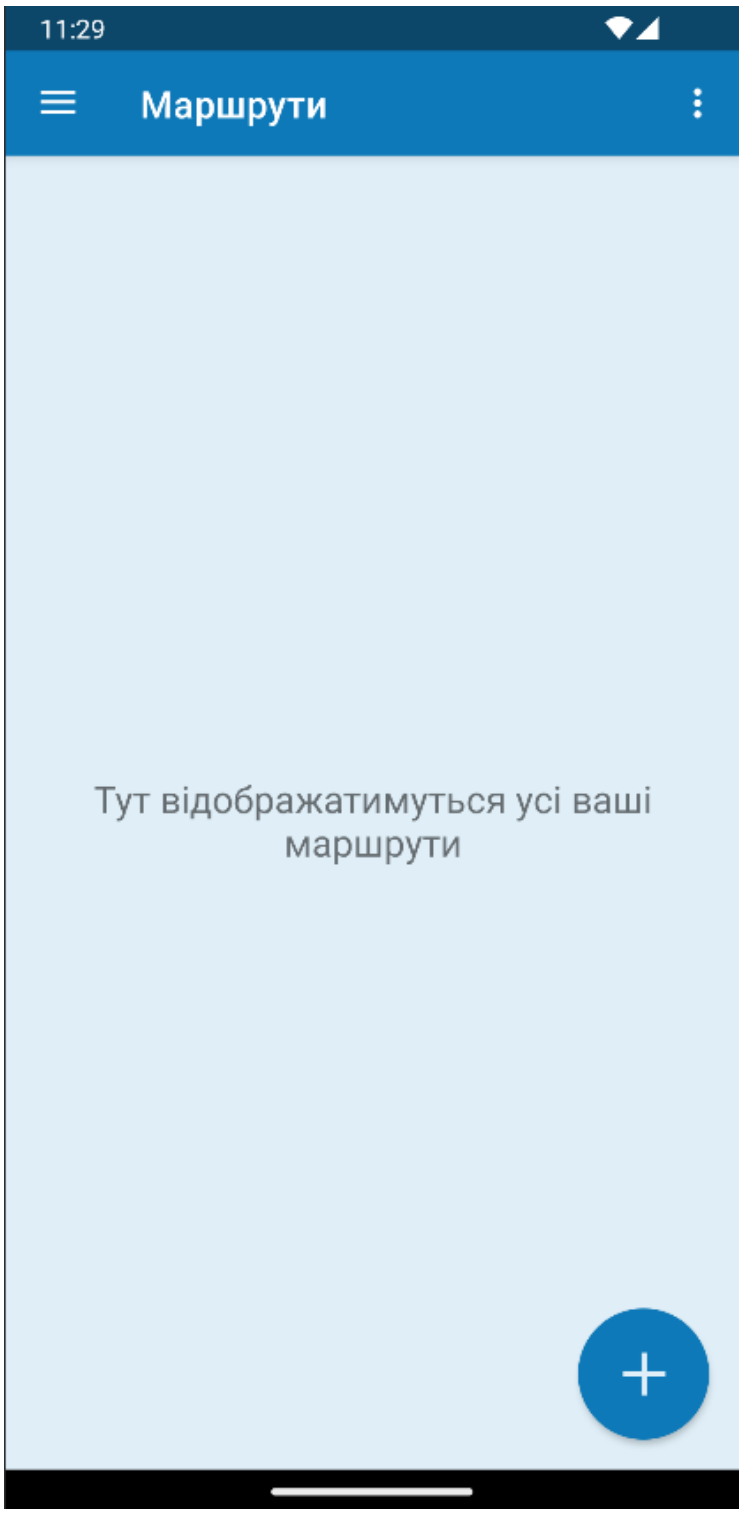


Рисунок 4.6 – Розділ «Маршрути»

Щоб створити новий маршрут користувачу необхідно натиснути на кнопку

«+» в правому нижньому куті екрану. Після натиснення кнопки відкривається розділ «Планування маршруту» з порожньою мапою (рисунок 4.7). Для масштабування мапи користувач може використати мультітач жести.

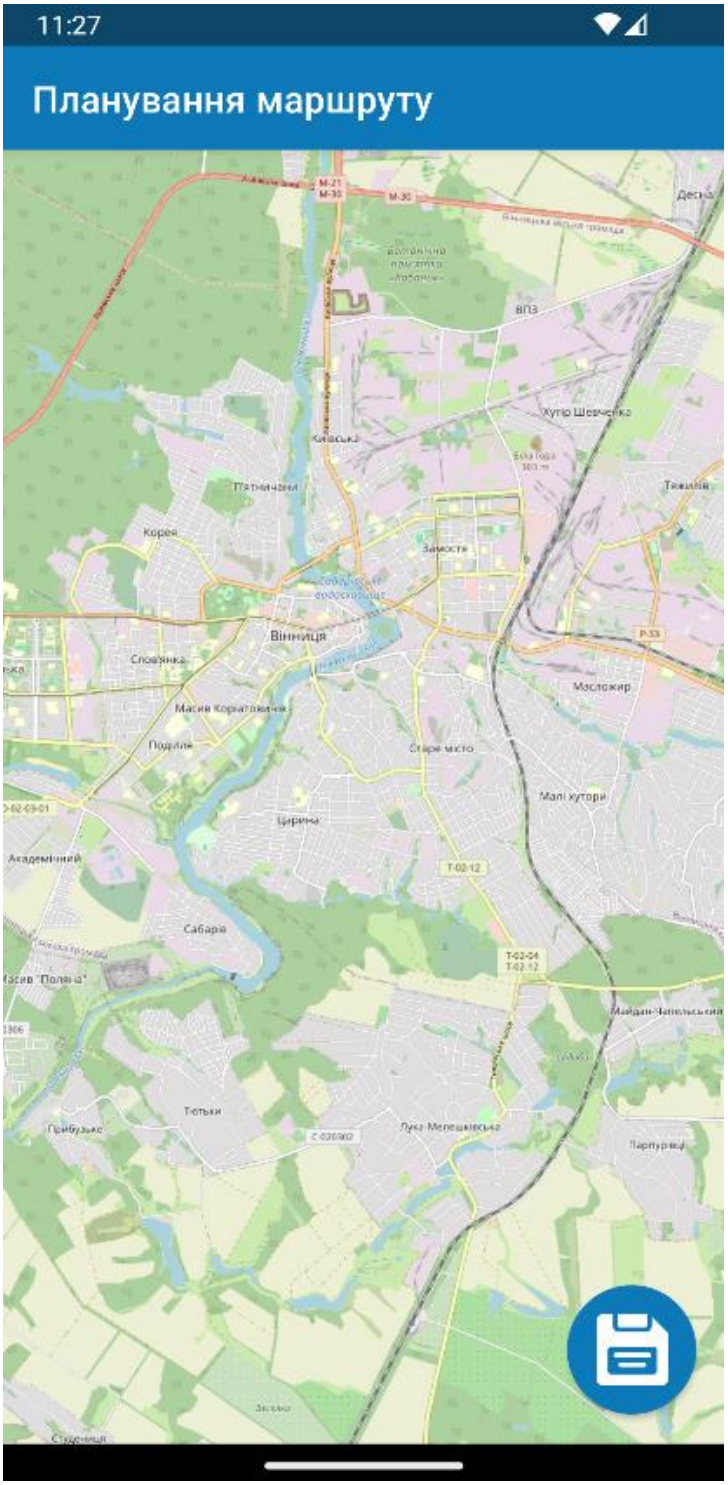


Рисунок 4.7 – Розділ «Планування маршруту»

Щоб додати нову мітку на мапу і розпочати процес створення маршруту

користувачу необхідно натиснути у будь-якому місці на мапу у розділі «Планування маршруту» (рисунок 4.8).

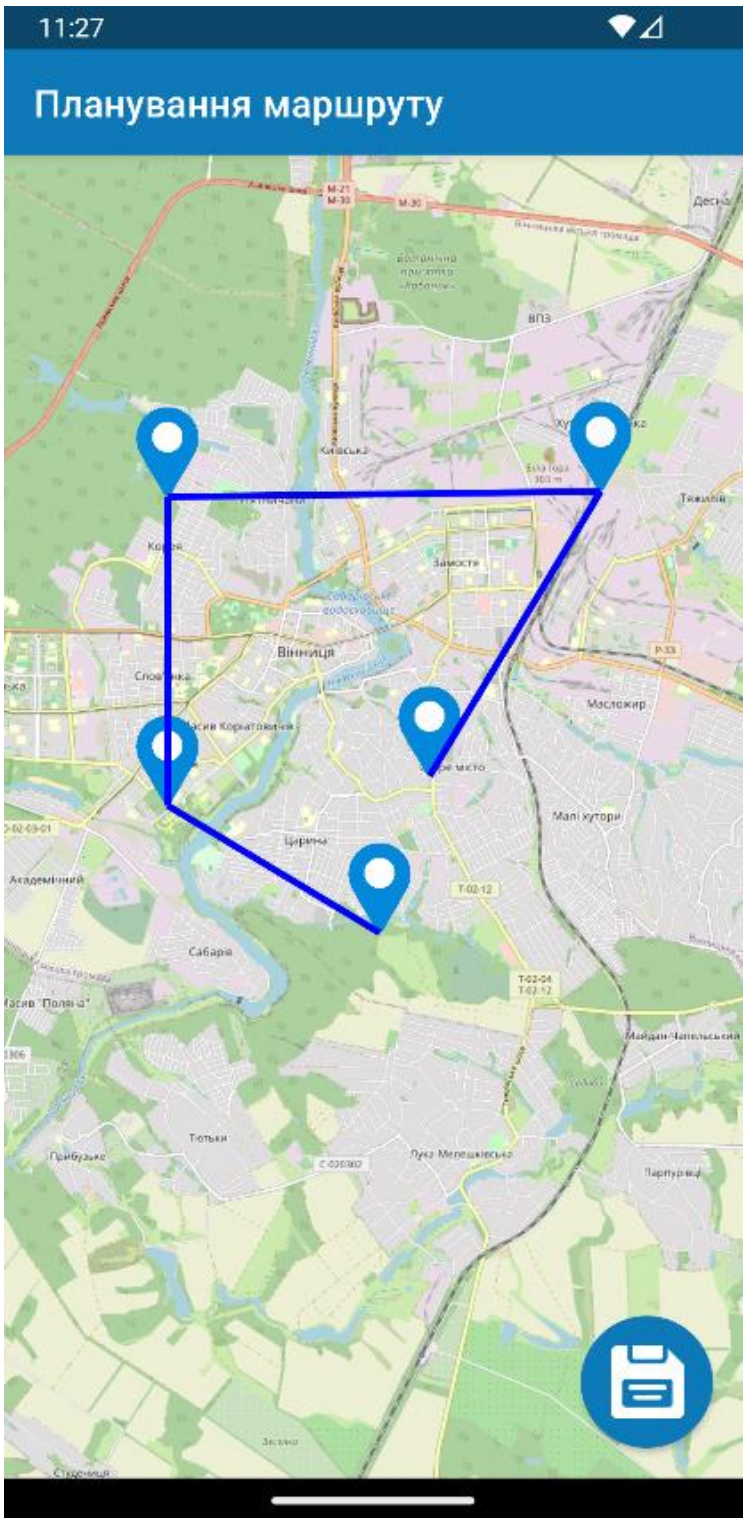


Рисунок 4.8 – Додання міток на мапу у розділі «Планування маршруту»

Щоб видалити мітку з мапи користувачу необхідно повторно натиснути на

існуючу мітку на мапі у розділі «Планування маршруту» (рисунок 4.9).

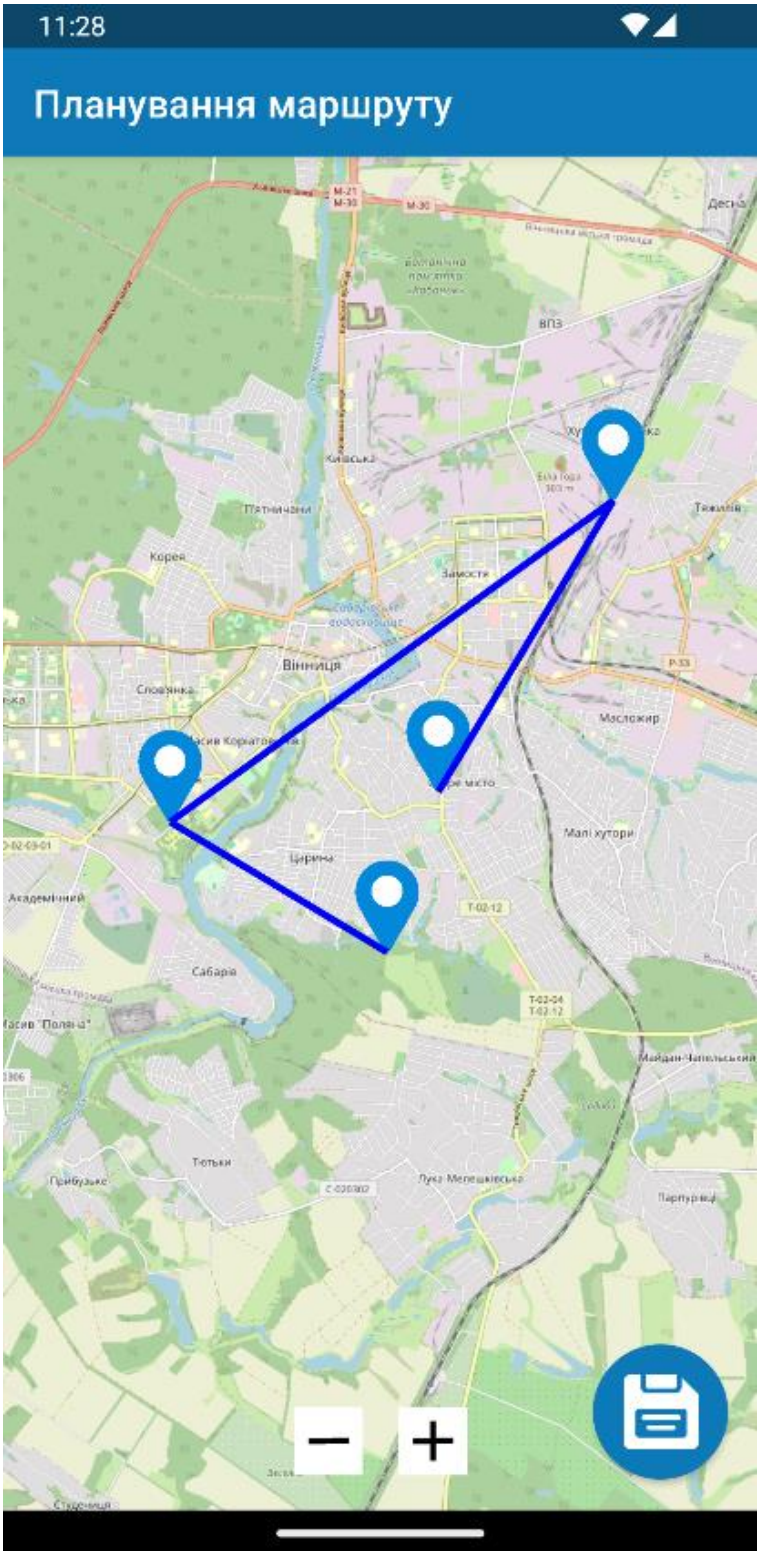


Рисунок 4.9 – Видалення міток з мапи у розділі «Планування маршруту»

Для того щоб додати точку запобігання, яка буде обмежувати точки, що використовуються алгоритмом для знаходження оптимального шляху,

користувачу необхідно використати жест «натиснути і утримати» у будь-якому місці на мапі. Після виконання цього жесту відкриється вікно (рисунок 4.10), у якому необхідно ввести радіус точки запобігання у метрах.

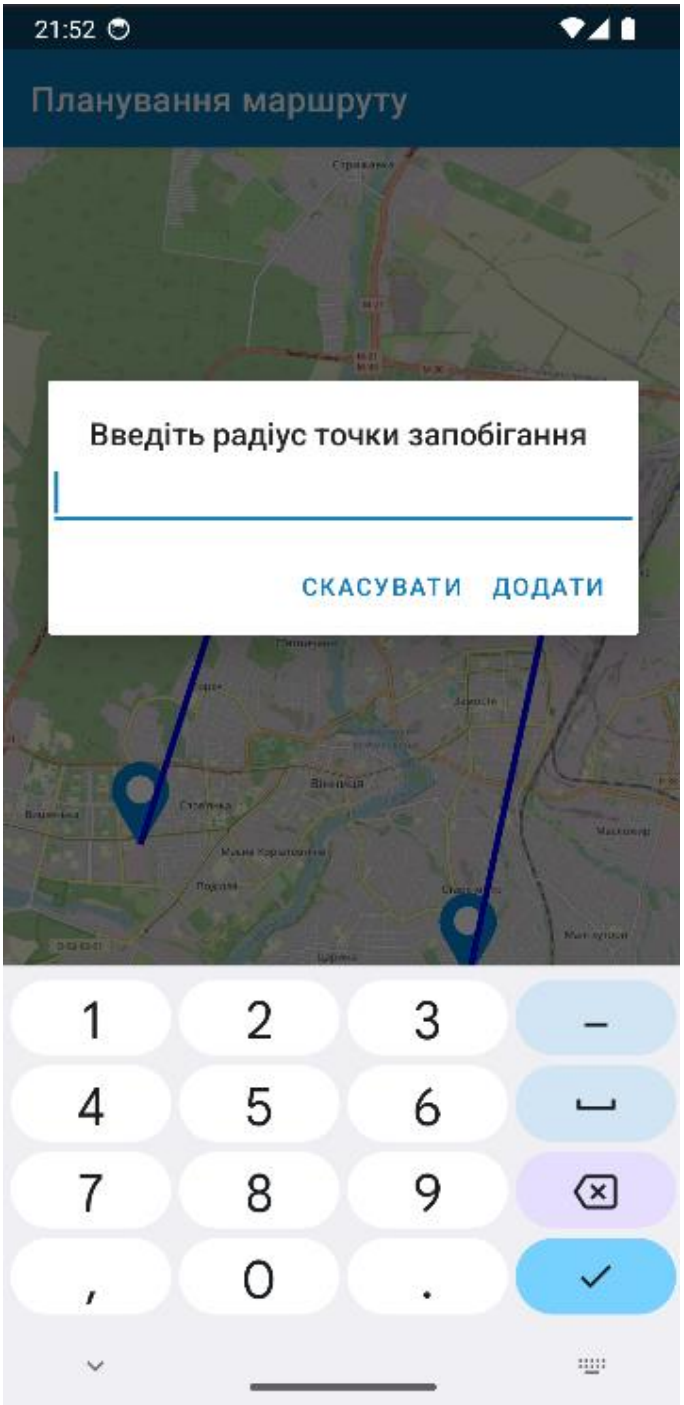


Рисунок 4.10 – Вікно додання точки запобігання у розділі «Планування маршруту»

Після підтвердження операції додання точки запобігання натисканням

кнопку «Додати» на мапі з’являється коло червоного кольору, що відображає додану зону (рисунок 4.11).

Рисунок 4.11 – Додана точка запобігання на мапі

Після завершення процесу планування маршруту користувач повинен натиснути кнопку з піктограмою дискети для його збереження. Користувача

автоматично повертає у розділ «Маршрути» і відображає у кінці списку збережений маршрут (рисунок 4.12).

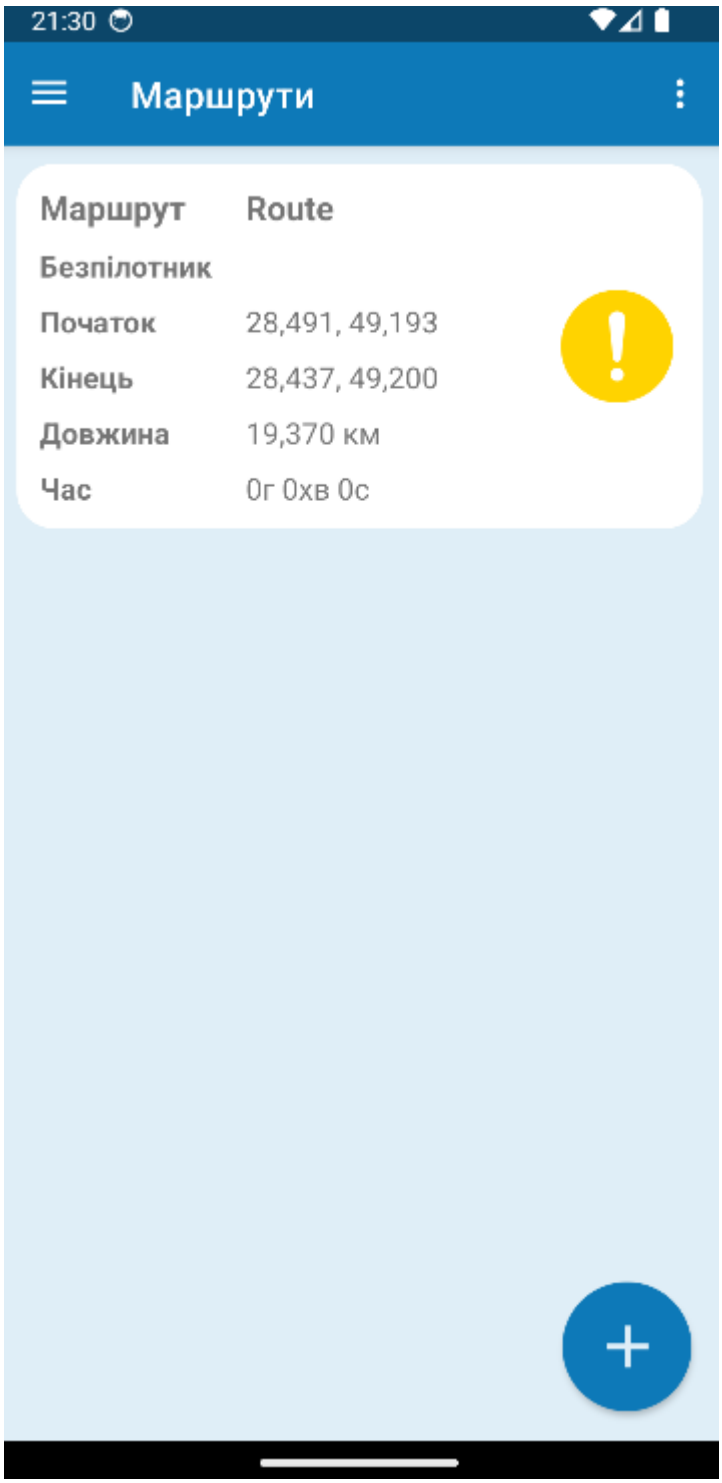


Рисунок 4.12 – Збережений маршрут у розділі «Маршрути»

У випадку, якщо до маршруту не додано безпілотник, відображається статус з жовтою піктограмою знака оклику, інакше, якщо його

вантажопідйомність та дальність польоту більша за вимоги маршруту – зеленою піктограмою «Checkmark», якщо ж якийсь із параметрів менший – червона піктограма (рисунок 4.13).

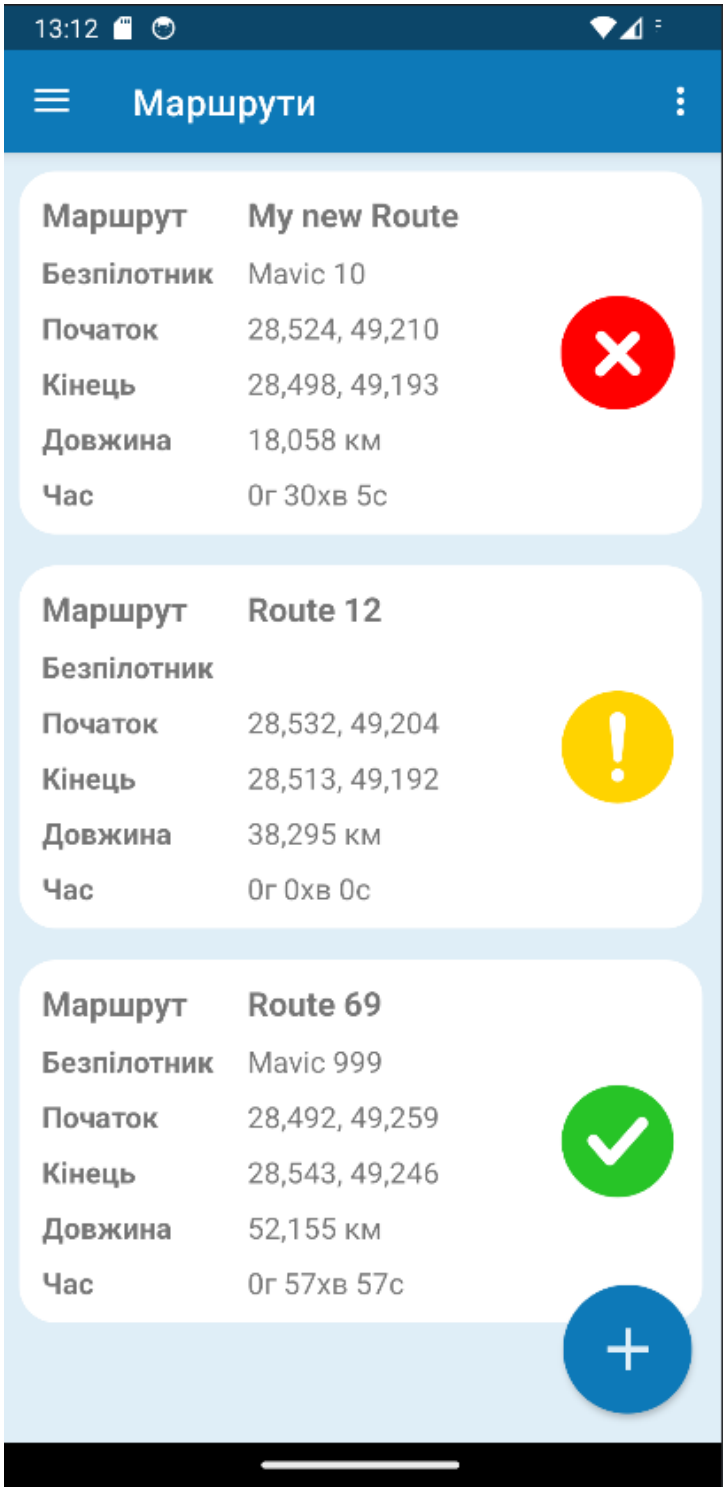


Рисунок 4.13 – Відображення статусу маршруту у списку

Інтерфейс та локалізація застосунку автоматично адаптується до мови

системи: українська у випадку використання української мови системи, у всіх інших випадках – англійська (рисунок 4.14).

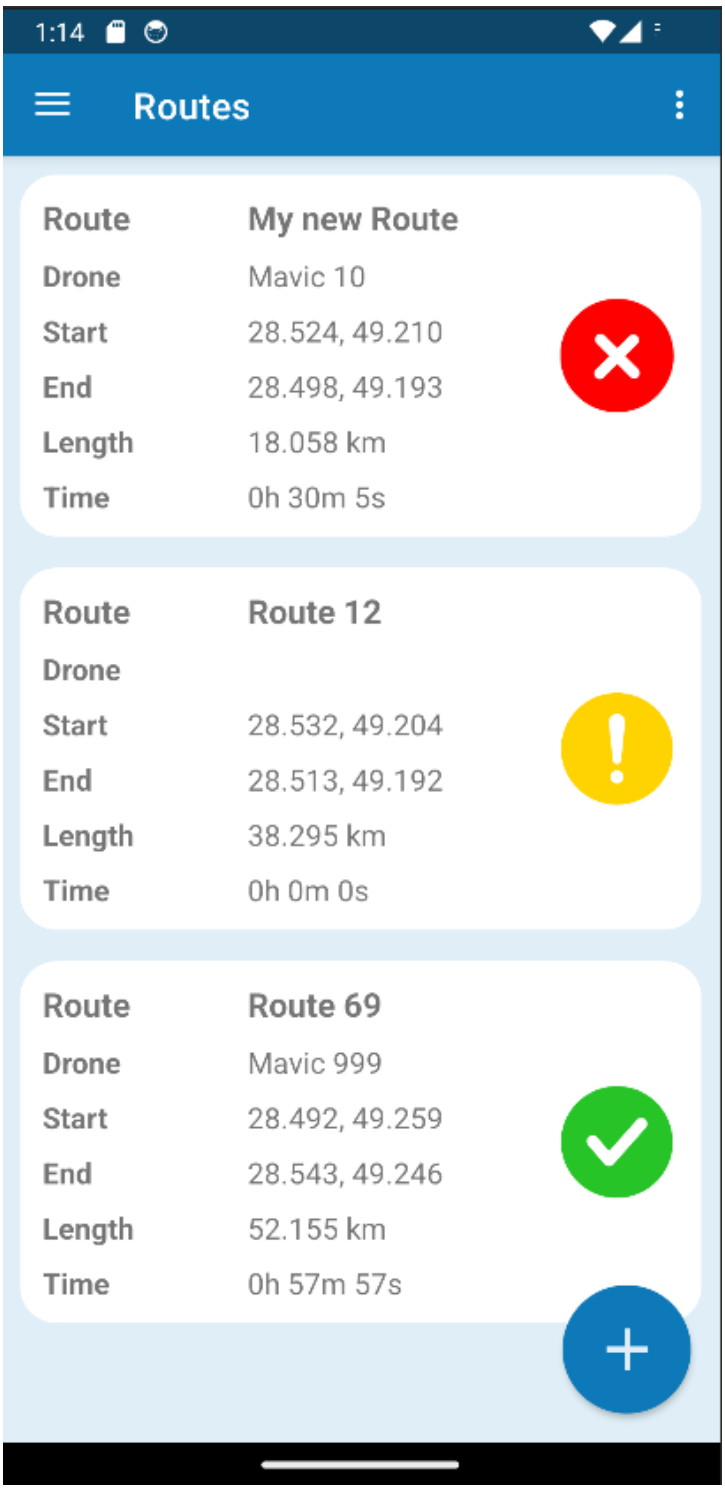


Рисунок 4.14 – Зміна мови застосунку

Щоб отримати можливість переглянути більше інформації про маршрут, змінити безпілотник, що використовується у маршруті, або видалити маршрут,

користувачу необхідно натиснути на маршрут у списку. Після чого відкривається активність «Про маршрут» (рисунок 4.15). Також у цій активності користувач може переглянути статус та рекомендації щодо маршруту.

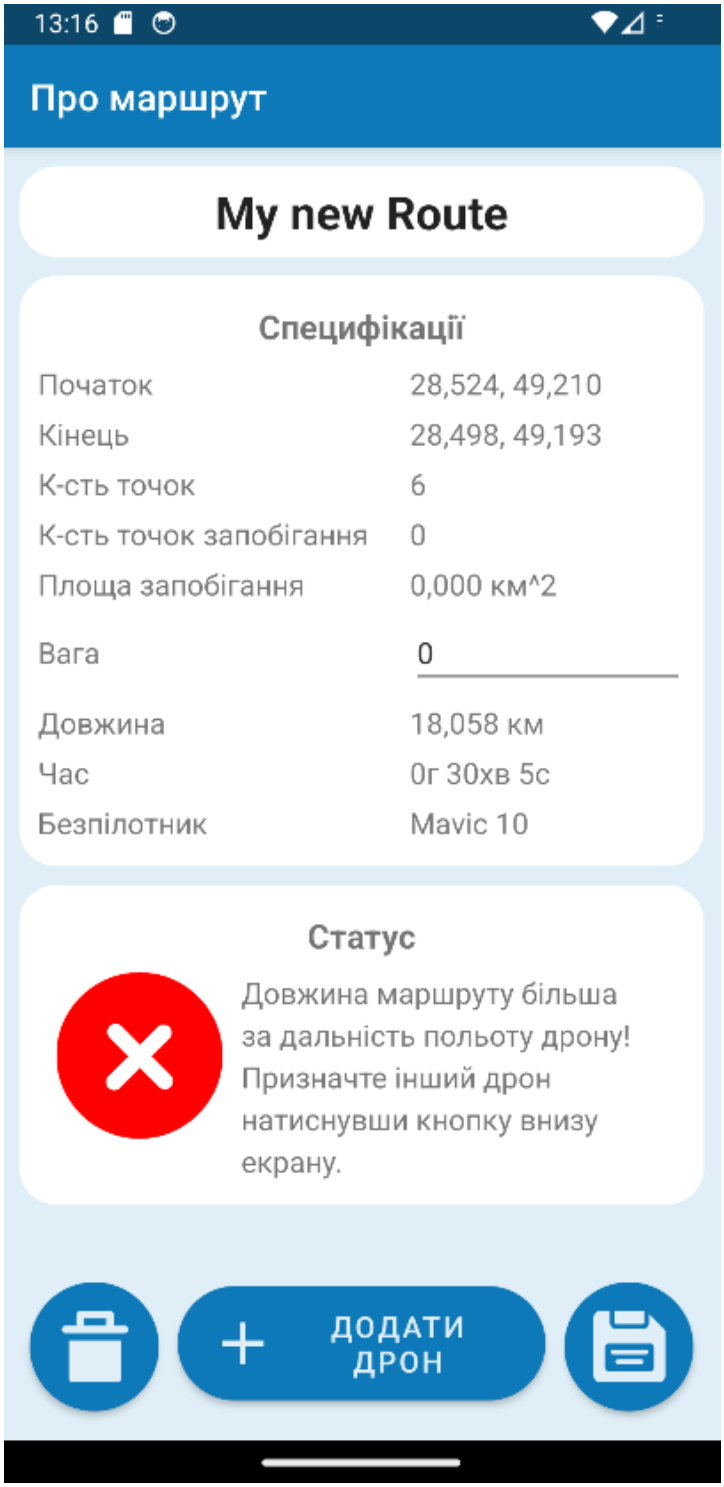


Рисунок 4.15 – Активність «Про маршрут»

При натисканні на кнопку «Додати дрон» з’являєьтся діалогове вікно

вибору безпілотників (рисунок 4.16), у якому користувач може обрати який саме безпілотник додати до маршруту.

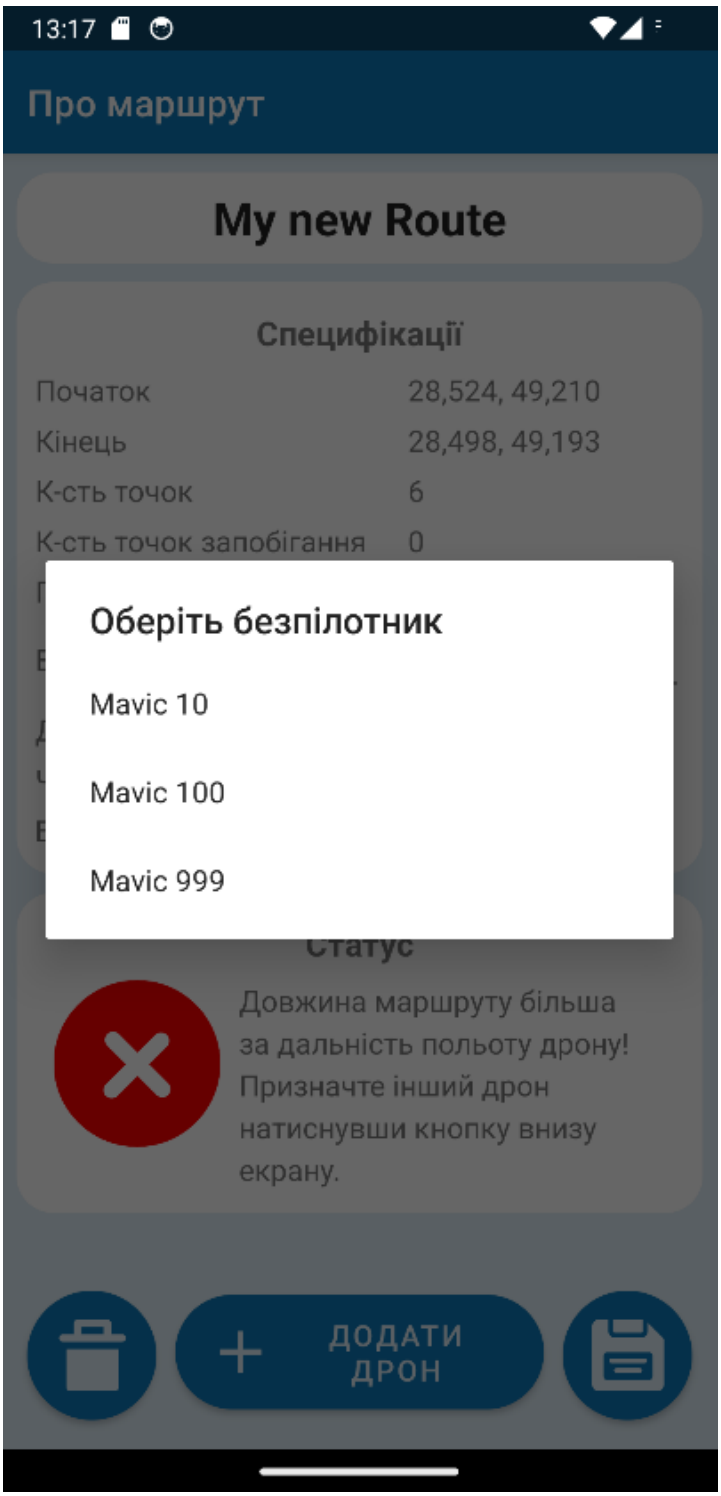


Рисунок 4.16 – Додання безпілотника до маршруту

Для видалення маршруту користувачу достатньо натиснути на кнопку з відповідною піктограмою і підтвердити операцію видалення у діалоговому вікні

що з’явиться (рисунок 4.17).

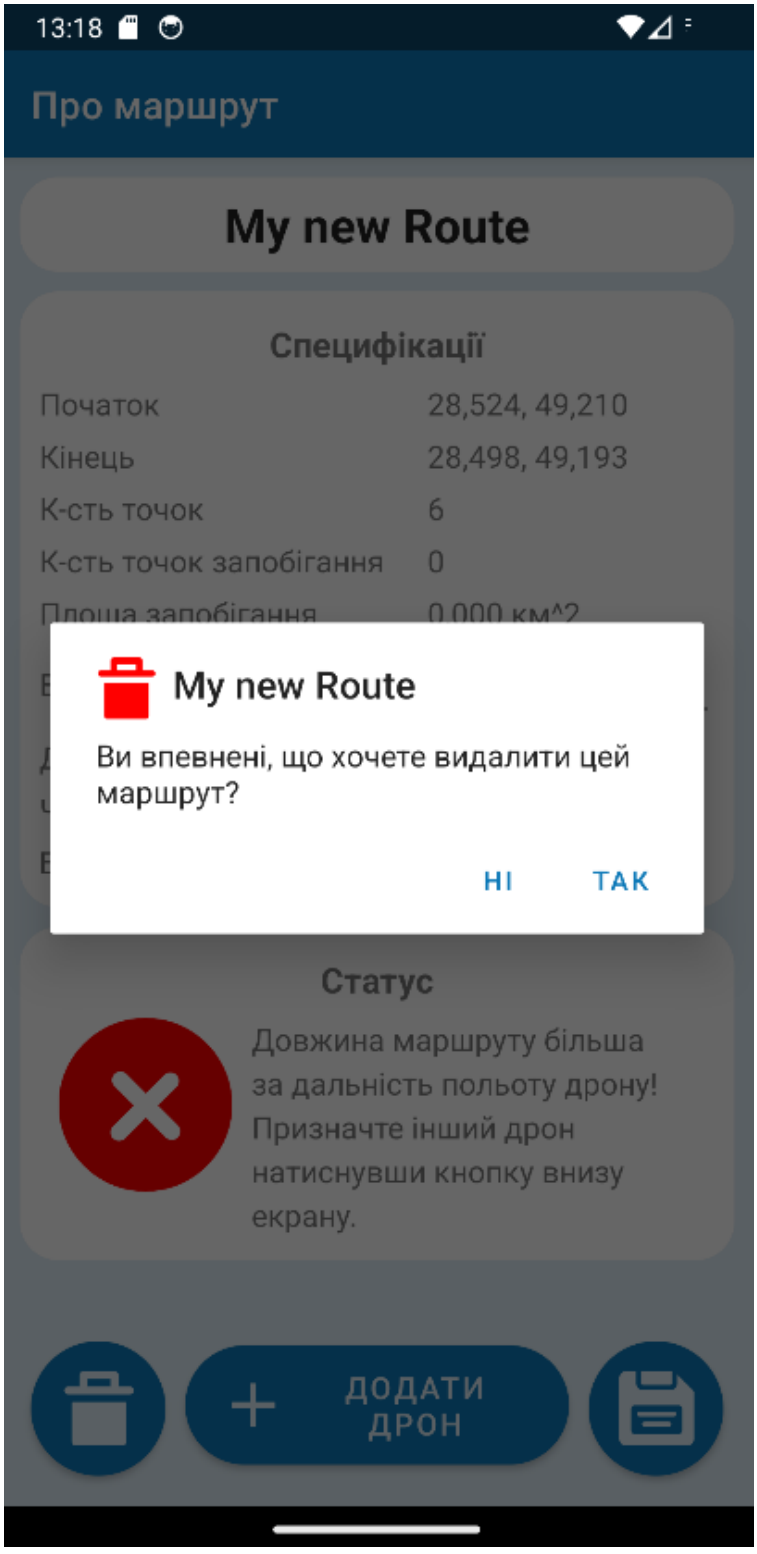


Рисунок 4.17 – Вікно підтвердження видалення маршруту

Для перегляду та керування списком безпілотників використовується активність «Безпілотники». Щоб додати новий дрон користувачу необхідно

натиснути на кнопку з піктограмою «+» в правій нижній частині активності (рисунок 4.18).

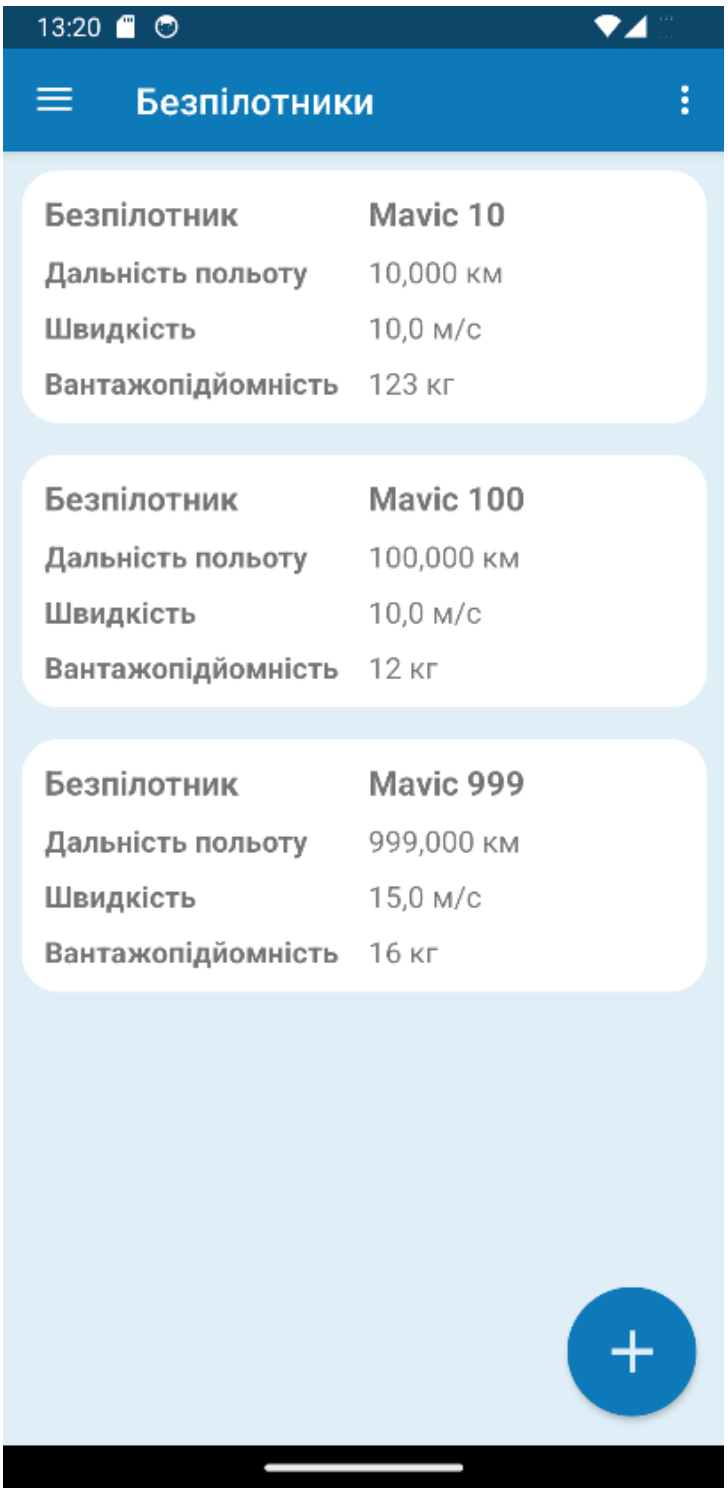


Рисунок 4.18 – Активність «Безпілотники»

Щоб отримати можливість переглянути більше інформації про безпілотник, змінити його характеристики або видалити, користувачу необхідно

натиснути на дрон у списку. Після чого відкривається активність «Безпілотник» (рисунок 4.19).

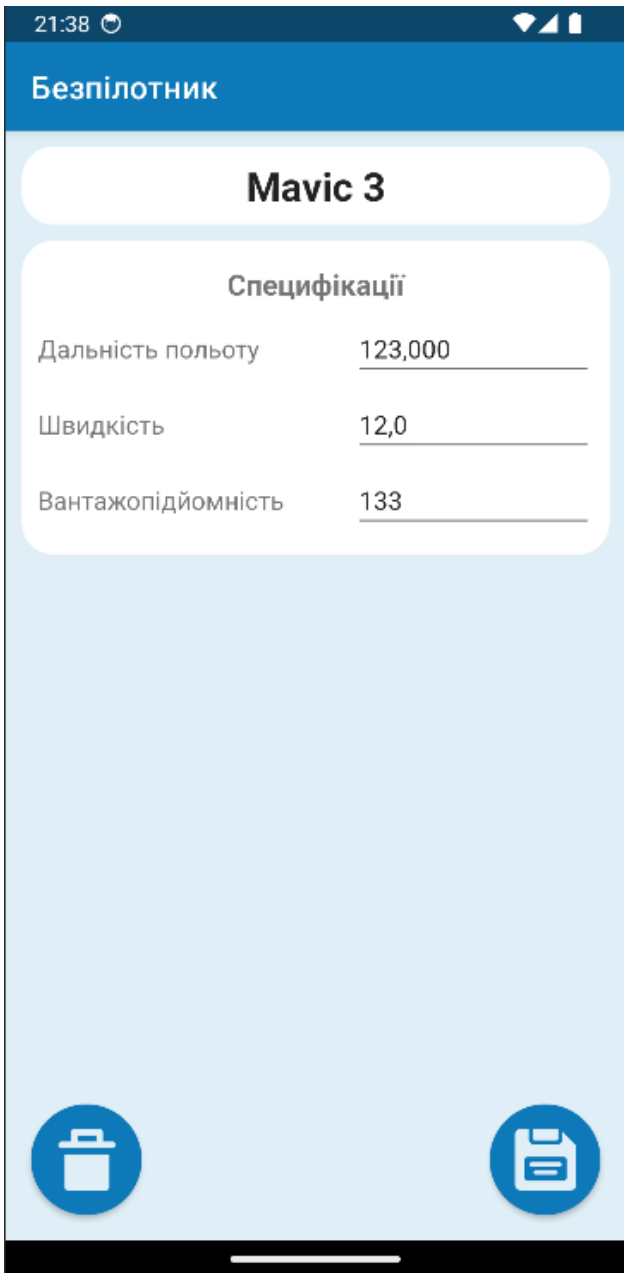


Рисунок 4.19 – Можливість перегляду та зміни характеристик дрона

В активності «Безпілотник» користувач може змінити таку інформацію про дрон як:

– назва;
– дальність польоту;
– швидкість;

– вантажопідйомність.

Для збереження змін користувачу необхідно натиснути на кнопку з піктограмою дискети. У випадку, якщо хоча б одне поле залишиться порожнім, користувач отримає відповідне повідомлення (рисунок 4.20)

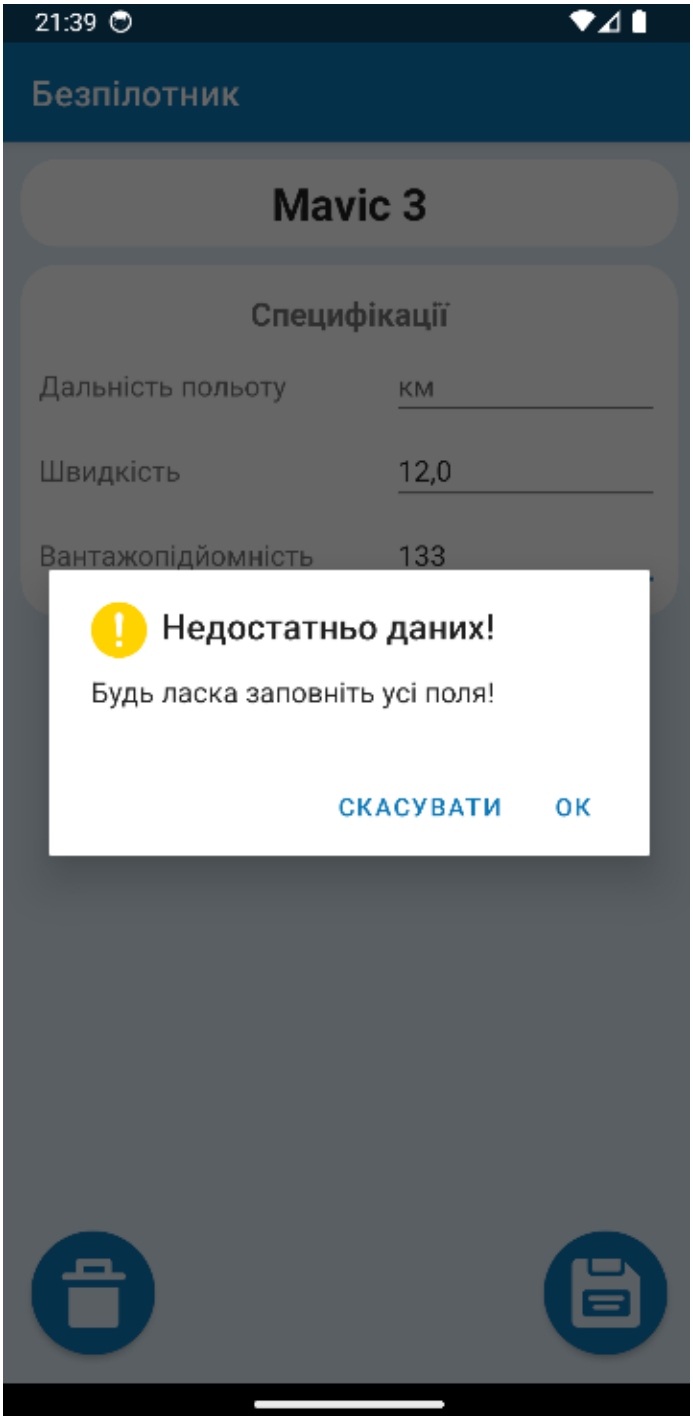


Рисунок 4.20 – Повідомлення про відсутність даних

Однією з головних функціональних характеристики застосунку є

можливість вибору алгоритму знаходження оптимального шляху (рисунок 4.21).

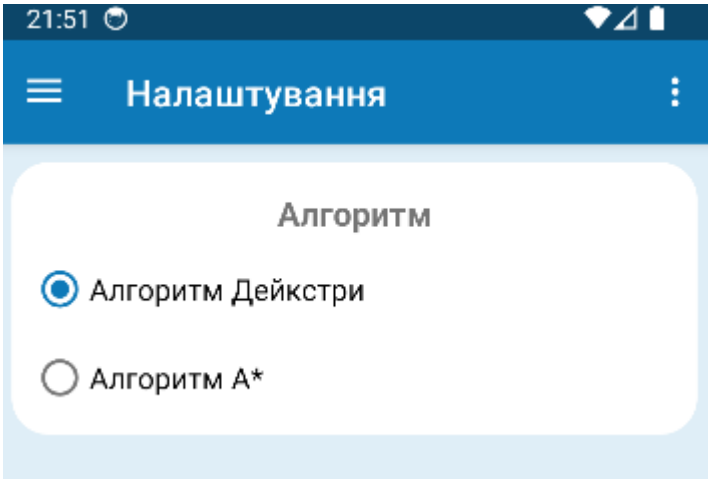


Рисунок 4.21 – Вибір алгоритму в налаштуваннях

Щоб змінити алгоритм користувачу необхідно в активності «Налаштування» обрати бажаний алгоритм.

4.3 Висновки

У четвертому розділі було описано спосіб логування подій, проведено юніт тестування програмних модулів мобільної системи. Також розроблено інструкцію користувача зі встановлення/видалення/використання мобільного застосунку.

ВИСНОВКИ

У ході виконання бакалаврської дипломної роботи було розроблено програмні засоби мобільного застосунку знаходження оптимального шляху безпілотного літального апарату.

Мобільний застосунок розроблено з використанням мови програмування Java, інтегрованого середовища розробки для мобільних систем Android Studio та бібліотек osmdroid, osmdroid-bonuspack та GSON. Для збірки проєкту використано Gradle Kotlin Script. Бакалаврська дипломна робота розроблено згідно методичних вказівок [23].

У бакалаврській дипломній роботі виконано такі завдання:

– розроблено загальний алгоритм системи;

– розроблено метод знаходження оптимального шляху безпілотного літального апарату на основі алгоритмів Дейкстри та A*;

– розроблено метод врахування характеристик безпілотника;

– розроблено зручний та адаптивний графічний інтерфейс мобільного застосунку;

– розроблено модуль для локального збереження маршрутів на пристрої користувача;

– здійснено тестування мобільного застосунку згідно поставлених задач;

– розроблено інструкцію користувача.

Розглянуто стан систем знаходження оптимального шляху безпілотного літального апарату, вплив алгоритмів знаходження оптимального шляху на ефективність безпілотного літального апарату. Проведено аналіз аналогів програмного забезпечення «DronePathfinder», що довів доцільність і актуальність розробки.

Проведено аналіз вхідних та вихідних даних мобільного застосунку. Також описано модель роботи мобільного застосунку за допомогою UML діаграм діяльності.

Описано структуру мобільного застосунку, використання принципів

об’єктно-орієнтованого програмування при його розробці; проаналізовано та обґрунтовано вибір мови програмування та технологій для розробки системи. Проаналізувавши вимоги роботи обрано мову програмування Java. У якості засобу відображення мапи обрано osmdroid OpenStreetMap через широкі можливості кастомізації. Збереження даних у локальній пам’яті пристрою реалізовано за допомогою GSON.

Описано спосіб логування подій, проведено юніт тестування програмних модулів мобільної системи. Також розроблено інструкцію користувача зі встановлення, видалення та використання мобільного застосунку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Applications of Unmanned Aerial Vehicles | Encyclopedia MDPI [Електронний ресурс] – Режим доступу до ресурсу: https://encyclopedia.pub/entry/25512

2. Simple and Efficient Algorithm for Drone Path Planning | IEEE Conference Publication | IEEE Xplore [Електронний ресурс] – Режим доступу до ресурсу: https://ieeexplore.ieee.org/document/9500370

3. Path Planning for Autonomous Drones: Challenges and Future Directions [Електронний ресурс] – Режим доступу до ресурсу: https://www.mdpi.com/2504-446X/7/3/169

4. Матейко Є. В. Вплив алгоритмів знаходження оптимального шляху на ефективність безпілотного літального апарату / Є. В. Матейко, Г. О. Черноволик // Матеріали LIII Науково-технічної конференції підрозділів Вінницького національного технічного університету, Вінниця, 2024. [Електронний ресурс] – Режим доступу до ресурсу: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2024/paper/view/20696/17212

5. Drones and their Applications in Different Sectors | by Webnx | Medium [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/@webnxoffice/drones-and-their-applications-in-different-sectors-d23f6e0fe087

6. Towards Fully Autonomous UAVs: A Survey [Електронний ресурс] – Режим доступу до ресурсу: https://www.mdpi.com/1424-8220/21/18/6223

7. UAV Path Planning Optimization Strategy: Considerations of Urban Morphology, Microclimate, and Energy Efficiency Using Q-Learning Algorithm [Електронний ресурс] – Режим доступу до ресурсу: https://www.mdpi.com/2504-446X/7/2/123

8. A Comprehensive Review of Recent Research Trends on Unmanned Aerial Vehicles (UAVs) [Електронний ресурс] – Режим доступу до ресурсу: https://www.mdpi.com/2079-8954/11/8/400

9. PathPlanner – Software application for path planning for mobile systems - Fraunhofer IOSB [Електронний ресурс] – Режим доступу до ресурсу: https://www.iosb.fraunhofer.de/en/projects-and-products/PathPlanner-Software_application_for_path_planning_for_mobile_systems.html

10. Software Overview - PX4 Autopilot [Електронний ресурс] – Режим доступу до ресурсу: https://px4.io/software/software-overview/

11. Mission Planner Home — Mission Planner documentation [Електронний ресурс] – Режим доступу до ресурсу: https://ardupilot.org/planner/

12. QGroundControl User Guide | QGC Guide (master) [Електронний ресурс] – Режим доступу до ресурсу: https://docs.qgroundcontrol.com/master/en/qgc-user-guide/index.html

13. DJI GS Pro – DJI [Електронний ресурс] – Режим доступу до ресурсу: https://www.dji.com/global/ground-station-pro

14. Інструкція, як будувати UML-діаграми | DOU [Електронний ресурс] – Режим доступу до ресурсу: https://dou.ua/forums/topic/40575/

15. Kotlin VS Java – What's the Difference? [Електронний ресурс] – Режим доступу до ресурсу: https://www.freecodecamp.org/news/kotlin-vs-java-whats-the-difference/

16. Java Vs Kotlin – Javatpoint [Електронний ресурс] – Режим доступу до ресурсу: https://www.javatpoint.com/java-vs-kotlin

17. Kotlin vs. Java: Which One is Better? – TechMagic [Електронний ресурс] – Режим доступу до ресурсу: https://www.techmagic.co/blog/kotlin-vs-java/

18. Google Maps Platform Documentation | Maps SDK for Android | Google for Developers [Електронний ресурс] – Режим доступу до ресурсу: https://developers.google.com/maps/documentation/android-sdk

19. osmdroid/osmdroid: OpenStreetMap-Tools for Android [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/osmdroid/osmdroid

20. Структурна схема [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%BD%D0%B0_%D1%81%D1%85%D0%B5%D0%B

C%D0%B0

21. logcat | Android Developers [Електронний ресурс] – Режим доступу до ресурсу: https://stuff.mit.edu/afs/sipb/project/android/docs/tools/help/logcat.html

22. 5 Benefits of Unit Testing and Why You Should Care [Електронний ресурс] – Режим доступу до ресурсу: https://cswsolutions.com/blog/posts/2022/december/5-benefits-of-unit-testing-and-why-you-should-care/

23. Методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів спеціальності 121 «Інженерія програмного забезпечення» / Уклад. О. Н. Романюк, Г. О. Черноволик – Вінниця: ВНТУ, 2022. – 52 с.

ДОДАТКИ

Додаток А – Технічне завдання

65

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
д.т.н., проф. О. Н. Романюк
" 12 " березня 2024 р.

Технічне завдання
на бакалаврську дипломну роботу «Розробка програмного забезпечення для знаходження оптимального шляху безпілотного літального апарату» за спеціальністю
121 Інженерія програмного забезпечення

Керівник бакалаврської дипломної роботи:
к.т.н., доцент Г.О. Черноволик
" 12 " березня 2024 р.
Виконав:
студент гр.4ПІ-20б Є.В. Матейко
" 12 " березня 2024 р.

Вінниця – 2024 року

1. Найменування та галузь застосування

Бакалаврська дипломна робота: «Розробка програмного забезпечення для знаходження оптимального шляху безпілотного літального апарату».

Галузь застосування – сфера логістики.

2. Підстава для розробки.

Підставою для виконання бакалаврської дипломної роботи (БДР) є індивідуальне завдання на БДР та наказ №80 від «11» березня 2024 р. ректора ВНТУ про закріплення тем БДР.

3. Мета та призначення розробки.

Метою роботи є підвищення енергоефективності безпілотних літальних апаратів за рахунок збільшення якості визначення шляху.

Призначення роботи – розробка та реалізація мобільного застосунку для знаходження оптимального шляху безпілотного літального апарату.

4. Вихідні дані для проведення НДР.

Перелік основних літературних джерел, на основі яких буде виконуватись БДР:

1. Simple and Efficient Algorithm for Drone Path Planning | IEEE Conference Publication | IEEE Xplore [Електронний ресурс] – Режим доступу до ресурсу: https://ieeexplore.ieee.org/document/9500370

2. Path Planning for Autonomous Drones: Challenges and Future Directions [Електронний ресурс] – Режим доступу до ресурсу: https://www.mdpi.com/2504-446X/7/3/169

3. Матейко Є. В. Вплив алгоритмів знаходження оптимального шляху на ефективність безпілотного літального апарату / Є. В. Матейко, Г. О. Черноволик // Матеріали LIII Науково-технічної конференції підрозділів Вінницького національного технічного університету, Вінниця, 2024. [Електронний ресурс] –

Режим доступу до ресурсу: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2024/paper/view/20696/17212

4. Drones and their Applications in Different Sectors | by Webnx | Medium [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/@webnxoffice/drones-and-their-applications-in-different-sectors-d23f6e0fe087

5. Технічні вимоги.

Базові алгоритми – алгоритм Дейкстри, алгоритм A*; бібліотеки – osmdroid, osmdroid-bonuspack, GSON; середовище розробки – Android Studio; середовище збірки проєкту – Gradle Kotlin Script; мова програмування – Java; мінімальний рівень API операційної системи Android – 29.

6. Конструктивні вимоги.

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт:

- пояснювальна записка до БДР;
- технічне завдання;
- лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації.

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки:

Ч.ч.	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи
1	Аналіз розвитку систем знаходження оптимального шляху та постановка задач дослідження	14.03.24 – 07.04.24
2	Розробка методів, моделі та алгоритмів мобільного застосунку	08.04.24 – 21.04.24
3	Розробка структури та програмних модулів мобільного застосунку	22.04.24 – 05.05.24
4	Тестування мобільного застосунку	06.05.24 – 19.06.24
5	Розробка інструкції користувача	20.05.24 – 26.05.24
6	Оформлення матеріалів до захисту БДР	27.05.24 – 30.05.24

10. Порядок контролю та прийняття.

Виконання етапів бакалаврської дипломної роботи контролюється керівником згідно з графіком виконання роботи. Захист бакалаврської дипломної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком.

Додаток Б – Протокол перевірки бакалаврської дипломної роботи на наявність текстових запозичень

70

ПРОТОКОЛ
ПЕРЕВІРКИ БАКАЛАВРСЬКОЇ ДИПЛОМНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: «Розробка програмного забезпечення для знаходження оптимального шляху безпілотного літального апарату»

Тип роботи: БДР

Підрозділ : кафедра програмного забезпечення, ФІТКІ

Науковий керівник: к.т.н., доц. Черноволик Г.О.

Оригінальність	95
Схожість	5

Аналіз звіту подібності

■ **Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.**

□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.

□ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку Черноволик Г. О.

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck

Матейко Є. В.

Автор роботи

Черноволик Г. О.

Керівник роботи

Додаток В – Лістинг програми

```
package com.example.dronepathfinder.algorithms;

import com.example.dronepathfinder.objects.AvoidancePoint;

import org.osmdroid.util.GeoPoint;

import java.util.List;

interface Algorithm
{
   List<GeoPoint> findShortestPath(List<GeoPoint> points, List<AvoidancePoint>
avoidancePoints);
}

package com.example.dronepathfinder.algorithms;

import android.util.Log;

import com.example.dronepathfinder.objects.AvoidancePoint;

import org.osmdroid.util.GeoPoint;

import java.util.ArrayList;
import java.util.Comparator;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.PriorityQueue;

public class AStarAlgorithm implements Algorithm
{
   private final Map<GeoPoint, List<GeoPoint>> adjacencyList = new HashMap<>();
   private final Map<GeoPoint, GeoPoint> cameFrom = new HashMap<>();
```

```
    private final Map<GeoPoint, Double> gScore = new HashMap<>();
    private final Map<GeoPoint, Double> fScore = new HashMap<>();
    private final PriorityQueue<GeoPoint> openSet = new
PriorityQueue<>(Comparator.comparingDouble(fScore::get));

    public List<GeoPoint> findShortestPath(List<GeoPoint> points, List<AvoidancePoint>
avoidancePoints)
    {
        Log.d("AStarAlgorithm", "Calling findShortestPath");

        adjacencyList.clear();
        cameFrom.clear();
        gScore.clear();
        fScore.clear();
        openSet.clear();

        List<GeoPoint> filteredPoints = new ArrayList<>(points);
        filteredPoints.removeIf(point -> isWithinAvoidancePoint(point, avoidancePoints));

        for (int i = 0; i < filteredPoints.size() - 1; i++)
        {
            GeoPoint current = filteredPoints.get(i);
            GeoPoint next = filteredPoints.get(i + 1);
            adjacencyList.computeIfAbsent(current, k -> new ArrayList<>()).add(next);
            adjacencyList.computeIfAbsent(next, k -> new ArrayList<>()).add(current);
            gScore.put(current, Double.MAX_VALUE);
            fScore.put(current, Double.MAX_VALUE);
        }

        GeoPoint start = filteredPoints.get(0);
        GeoPoint goal = filteredPoints.get(filteredPoints.size() - 1);
        gScore.put(start, 0.0);
        fScore.put(start, heuristicCostEstimate(start, goal));

        openSet.add(start);
```

```
        while (!openSet.isEmpty())
        {
            GeoPoint current = openSet.poll();

            Log.d("AStarAlgorithm", "Current vertex: " + current.getLatitude() + ", " +
current.getLongitude());

            if (current.equals(goal))
                return reconstructPath(cameFrom, current);

            for (GeoPoint neighbor : adjacencyList.getOrDefault(current, new ArrayList<>()))
            {
                double tentativeGScore = gScore.get(current) + current.distanceToAsDouble(neighbor);

                if (tentativeGScore < gScore.getOrDefault(neighbor, Double.MAX_VALUE))
                {
                    cameFrom.put(neighbor, current);
                    gScore.put(neighbor, tentativeGScore);
                    fScore.put(neighbor, tentativeGScore + heuristicCostEstimate(neighbor, goal));

                    if (!openSet.contains(neighbor))
                        openSet.add(neighbor);
                }
            }
        }

        return new ArrayList<>();
    }

    private boolean isWithinAvoidancePoint(GeoPoint point, List<AvoidancePoint>
avoidancePoints)
    {
        for (AvoidancePoint avoidancePoint : avoidancePoints)
        {
```

```
            if (point.distanceToAsDouble(avoidancePoint.getCenter()) <= avoidancePoint.getRadius()) {
                return true;
            }
        }
        return false;
    }

    private List<GeoPoint> reconstructPath(Map<GeoPoint, GeoPoint> cameFrom, GeoPoint current)
    {
        Log.d("AStarAlgorithm", "Calling reconstructPath");

        List<GeoPoint> totalPath = new ArrayList<>();
        totalPath.add(current);

        while (cameFrom.containsKey(current))
        {
            current = cameFrom.get(current);
            totalPath.add(0, current);

            Log.d("AStarAlgorithm", "Added step: " + current);
        }

        return totalPath;
    }

    private double heuristicCostEstimate(GeoPoint start, GeoPoint goal)
    {
        return start.distanceToAsDouble(goal);
    }
}
```

```
package com.example.dronepathfinder.algorithms;

import android.util.Log;
```

```
import com.example.dronepathfinder.objects.AvoidancePoint;

import org.osmdroid.util.GeoPoint;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.PriorityQueue;

public class DijkstraAlgorithm implements Algorithm
{
   private Map<GeoPoint, Map<GeoPoint, Double>> graph = new HashMap<>();
   private Map<GeoPoint, GeoPoint> predecessors = new HashMap<>();

   private void initializeGraph(List<GeoPoint> points, List<AvoidancePoint> avoidancePoints)
   {
      Log.d("DijkstraAlgorithm", "Calling initializeGraph");

      graph.clear();
      predecessors.clear();

      for (int i = 0; i < points.size() - 1; i++)
      {
         GeoPoint current = points.get(i);
         GeoPoint next = points.get(i + 1);

         double weight = current.distanceToAsDouble(next);

         graph.computeIfAbsent(current, k -> new HashMap<>()).put(next, weight);
         graph.computeIfAbsent(next, k -> new HashMap<>()).put(current, weight);
      }
   }
```

```
public List<GeoPoint> findShortestPath(List<GeoPoint> points, List<AvoidancePoint> avoidancePoints)
{
    Log.d("DijkstraAlgorithm","Calling findShortestPath");

    List<GeoPoint> filteredPoints = new ArrayList<>(points);
    filteredPoints.removeIf(point -> isWithinAvoidancePoint(point, avoidancePoints));

    GeoPoint start = filteredPoints.get(0);
    GeoPoint end = filteredPoints.get(filteredPoints.size() - 1);

    initializeGraph(filteredPoints, avoidancePoints);

    Log.d("DijkstraAlgorithm", "findShortestPath: points.size(): "
            + points.size());
    Log.d("DijkstraAlgorithm", "findShortestPath: filteredPoints.size(): "
            + filteredPoints.size());

    Map<GeoPoint, Double> distances = new HashMap<>();
    PriorityQueue<GeoPoint> queue = new PriorityQueue<>((v1, v2)
            -> distances.get(v1).compareTo(distances.get(v2)));

    for (GeoPoint vertex : graph.keySet())
        distances.put(vertex, Double.MAX_VALUE);
    distances.put(start, 0.0);

    queue.add(start);

    while (!queue.isEmpty())
    {
        GeoPoint current = queue.poll();

        Log.d("DijkstraAlgorithm", "findShortestPath: Current vertex: "
                + current.getLatitude() + ", " + current.getLongitude());
```

```
        Map<GeoPoint, Double> edges = graph.get(current);
        if (edges == null)
            continue;

        if (current.equals(end))
            return reconstructPath(end);

        for (Map.Entry<GeoPoint, Double> edge : edges.entrySet())
        {
            GeoPoint neighbor = edge.getKey();
            Double weight = edge.getValue();
            Double distanceThroughCurrent = distances.get(current) + weight;

            if (distanceThroughCurrent < distances.get(neighbor))
            {
                distances.put(neighbor, distanceThroughCurrent);
                predecessors.put(neighbor, current);
                queue.add(neighbor);
            }
        }
    }

    return new ArrayList<>();
}

private boolean isWithinAvoidancePoint(GeoPoint point, List<AvoidancePoint>
avoidancePoints)
{
    for (AvoidancePoint avoidancePoint : avoidancePoints)
    {
        if (point.distanceToAsDouble(avoidancePoint.getCenter()) <= avoidancePoint.getRadius()) {
            return true;
        }
    }
    return false;
```

```
    }

    private List<GeoPoint> reconstructPath(GeoPoint end)
    {
        Log.d("DijkstraAlgorithm", "Calling reconstructPath");

        List<GeoPoint> path = new ArrayList<>();
        GeoPoint step = end;

        if (predecessors.get(step) == null)
        {
            Log.d("DijkstraAlgorithm", "reconstructPath: End point has no predecessor");
            return path;
        }

        path.add(step);

        while (predecessors.get(step) != null)
        {
            step = predecessors.get(step);

            if (path.contains(step))
            {
                Log.d("DijkstraAlgorithm", "reconstructPath: Cycle detected. Exiting loop.");
                break;
            }

            path.add(0, step);
            Log.d("DijkstraAlgorithm", "reconstructPath: Added step: " + step);
        }

        Log.d("DijkstraAlgorithm", "reconstructPath: Path reconstruction complete");
        return path;
    }
}
```

```
package com.example.dronepathfinder.ui.routes;

import android.app.Activity;
import android.content.Context;
import android.content.Intent;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.ImageView;
import android.widget.TextView;

import androidx.activity.result.ActivityResultLauncher;
import androidx.activity.result.contract.ActivityResultContracts;
import androidx.annotation.NonNull;
import androidx.fragment.app.Fragment;

import com.example.dronepathfinder.objects.Drone;
import com.example.dronepathfinder.ui.AboutRouteActivity;
import com.example.dronepathfinder.ui.MapActivity;
import com.example.dronepathfinder.R;
import com.example.dronepathfinder.objects.Route;
import com.example.dronepathfinder.databinding.FragmentRoutesBinding;
import com.google.gson.Gson;
import com.google.gson.reflect.TypeToken;

import java.lang.reflect.Type;
import java.util.ArrayList;
import java.util.List;

public class RoutesFragment extends Fragment
```

```
{
    private ActivityResultLauncher<Intent> aboutRouteActivityLauncher;
    private ActivityResultLauncher<Intent> mapActivityLauncher;
    private View root;
    private FragmentRoutesBinding binding;
    private ArrayAdapter<Route> adapter;
    private List<Route> routesList;
    private TextView textView;

    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        loadRoutesFromSharedPreferences();
    }

    public View onCreateView(@NonNull LayoutInflater inflater,
                             ViewGroup container, Bundle savedInstanceState)
    {
        //RoutesViewModel routesViewModel = new
ViewModelProvider(this).get(RoutesViewModel.class);

        binding = FragmentRoutesBinding.inflate(inflater, container, false);
        root = binding.getRoot();

        textView = binding.textRoutes;
        updateRoutesDesc();

        handleActivityResults();
        handleListView();

        Button btnNavigateToMapActivity = root.findViewById(R.id.add_route);
        btnNavigateToMapActivity.setOnClickListener(new View.OnClickListener()
        {
            @Override
```

```
        public void onClick(View v)
        {
            launchMapActivity();
        }
    });

    textView.setText(getString(R.string.frgt_routes_desc));

    return root;
}

private void launchAboutRouteActivity(Route route, int position)
{
    Intent intent = new Intent(getActivity(), AboutRouteActivity.class);
    intent.putExtra("route", route);
    intent.putExtra("routePosition", position);
    aboutRouteActivityLauncher.launch(intent);
}

private void launchMapActivity()
{
    Intent intent = new Intent(getActivity(), MapActivity.class);
    mapActivityLauncher.launch(intent);
}

private void handleActivityResults()
{
    aboutRouteActivityLauncher = registerForActivityResult(
            new ActivityResultContracts.StartActivityForResult(),
            result -> {
                if (result.getResultCode() == Activity.RESULT_OK && result.getData() != null) {
                    boolean
                            routeDeleted = result.getData().getBooleanExtra("routeDeleted", false),
                            routeUpdated = result.getData().getBooleanExtra("routeUpdated", false);
                    int routePosition = result.getData().getIntExtra("routePosition", -1);
```

```
                if (routeDeleted && routePosition != -1)
                    routesList.remove(routePosition);

                if (routeUpdated && routePosition != -1)
                {
                    Route updatedRoute = (Route) result.getData().getSerializableExtra("route");

                    routesList.set(routePosition, updatedRoute);
                }

                adapter.notifyDataSetChanged();
                saveRoutesToSharedPreferences();
            }
        }
    );

    mapActivityLauncher = registerForActivityResult(
            new ActivityResultContracts.StartActivityForResult(),
            result -> {
                if (result.getResultCode() == Activity.RESULT_OK && result.getData() != null)
                {
                    Route route = (Route) result.getData().getSerializableExtra("route");
                    updateRoutesList(route);
                    updateRoutesDesc();
                }
            }
    );
}

private void handleListView()
{
    adapter = new ArrayAdapter<Route>(getActivity(), R.layout.route_list_item, routesList)
    {
        @NonNull
```

```
@Override
public View getView(int position, View convertView, @NonNull ViewGroup parent)
{
   if (convertView == null)
      convertView = LayoutInflater.from(getContext()).inflate(R.layout.route_list_item,
parent, false);

      TextView
         lv_item_name = convertView.findViewById(R.id.listview_route_name_value),
         lv_item_drone =
convertView.findViewById(R.id.listview_route_drone_name_value),
         lv_item_start = convertView.findViewById(R.id.listview_route_start_value),
         lv_item_end = convertView.findViewById(R.id.listview_route_end_value),
         lv_item_length = convertView.findViewById(R.id.listview_route_length_value),
         lv_item_time = convertView.findViewById(R.id.listview_route_time_value);
      ImageView lv_item_status =
convertView.findViewById(R.id.listview_route_status_value);

      Route route = getItem(position);
      if (route != null)
      {
         lv_item_name.setText(route.getName());
         lv_item_drone.setText(route.getDroneName());
         lv_item_start.setText(String.format("%.3f, %.3f", route.getStart().getLongitude(),
route.getStart().getLatitude()));
         lv_item_end.setText(String.format("%.3f, %.3f", route.getEnd().getLongitude(),
route.getEnd().getLatitude()));
         lv_item_length.setText(String.format("%.3f %s", route.getLength()/1_000,
getString(R.string.menu_km)));
         lv_item_time.setText(getFormatedTime(route.getTimeToComplete()));
         switch (route.getStatus())
         {
            case GOOD:
               lv_item_status.setImageResource(R.drawable.ic_checkmark);
               break;
```

```
                case ROUTE_TOO_LONG:
                case OVERWEIGHT:
                    lv_item_status.setImageResource(R.drawable.ic_cancel);
                    break;
                case NO_DRONE:
                    lv_item_status.setImageResource(R.drawable.ic_warning);
                    break;
                default:
                    break;
            }
        }

        return convertView;
    }
};
binding.lvRoutes.setAdapter(adapter);

binding.lvRoutes.setOnItemClickListener(new AdapterView.OnItemClickListener()
{
    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position, long id)
    {
        Route selectedRoute = adapter.getItem(position);
        if (selectedRoute != null)
        {
            launchAboutRouteActivity(selectedRoute, position);
        }
    }
});

/*binding.lvRoutes.setOnItemLongClickListener(new
AdapterView.OnItemLongClickListener()
{
    @Override
    public boolean onItemLongClick(AdapterView<?> parent, View view, int position, long id)
```

```
        {
        }
    });*/
}

private void updateRoutesList(Route newRoute)
{
    if (routesList != null)
    {
        routesList.add(newRoute);
        adapter.notifyDataSetChanged();
    }
}

private void updateRoutesDesc()
{
    if (!routesList.isEmpty())
        textView.setVisibility(View.GONE);
    else
        textView.setVisibility(View.VISIBLE);
}

private void saveRoutesToSharedPreferences()
{
    SharedPreferences sharedPreferences = getActivity().getSharedPreferences("RoutesPref",
Context.MODE_PRIVATE);
    SharedPreferences.Editor editor = sharedPreferences.edit();
    Gson gson = new Gson();
    String jsonRoutes = gson.toJson(routesList);
    editor.putString("SavedRoutes", jsonRoutes);
    editor.apply();
}

private void loadRoutesFromSharedPreferences()
{
```

```
        SharedPreferences sharedPreferences = getActivity().getSharedPreferences("RoutesPref",
Context.MODE_PRIVATE);
        Gson gson = new Gson();
        String jsonRoutes = sharedPreferences.getString("SavedRoutes", null);
        Type type = new TypeToken<ArrayList<Route>>() {}.getType();
        routesList = gson.fromJson(jsonRoutes, type);

        if (routesList == null)
            routesList = new ArrayList<>();
    }

    private String getFormatedTime(int totalSeconds)
    {
        int hours = totalSeconds / 3_600;
        int minutes = (totalSeconds % 3_600) / 60;
        int seconds = totalSeconds % 60;

        return String.format("%d%s %d%s %d%s",
                hours, getString(R.string.menu_hours),
                minutes, getString(R.string.menu_minutes),
                seconds, getString(R.string.menu_seconds));
    }

    @Override
    public void onPause()
    {
        super.onPause();
        saveRoutesToSharedPreferences();
    }

    @Override
    public void onDestroyView()
    {
        super.onDestroyView();
        binding = null;
```

```
    }
}

package com.example.dronepathfinder.ui;

import android.app.Activity;
import android.content.Context;
import android.content.DialogInterface;
import android.content.Intent;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ImageView;
import android.widget.TextView;
import android.widget.Toast;

import androidx.appcompat.app.AlertDialog;
import androidx.appcompat.app.AppCompatActivity;
import androidx.appcompat.app.AppCompatDelegate;

import com.example.dronepathfinder.R;
import com.example.dronepathfinder.objects.Drone;
import com.example.dronepathfinder.objects.Route;
import com.google.gson.Gson;
import com.google.gson.reflect.TypeToken;

import java.lang.reflect.Type;
import java.util.ArrayList;

public class AboutRouteActivity extends AppCompatActivity {

    private Route route;
    private int routePosition;
```

```
private Button
        deleteButton,
        updateButton,
        addDroneButton;

private EditText
        et_route_name,
        et_route_weight;

@Override
protected void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    AppCompatDelegate.setDefaultNightMode(AppCompatDelegate.MODE_NIGHT_NO);
    setContentView(R.layout.activity_about_route);

    route = (Route) getIntent().getSerializableExtra("route");
    routePosition = (int) getIntent().getIntExtra("routePosition", -1);

    et_route_name = findViewById(R.id.et_route_name_value);
    et_route_weight = findViewById(R.id.et_route_weight_value);

    updateSpecsUI(route);
    updateStatusUI(route);

    deleteButton = findViewById(R.id.delete_route);
    updateButton = findViewById(R.id.update_route);
    addDroneButton = findViewById(R.id.add_drone_to_route);

    deleteButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            showDeleteConfirmationDialog();
        }
    });
```

```
    updateButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            updateRoute();
        }
    });

    addDroneButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            showDroneSelectionMenu();
        }
    });

}

private String getFormatedTime(int totalSeconds)
{
    int hours = totalSeconds / 3_600,
            minutes = (totalSeconds % 3_600) / 60,
            seconds = totalSeconds % 60;

    return String.format("%d%s %d%s %d%s",
            hours, getString(R.string.menu_hours),
            minutes, getString(R.string.menu_minutes),
            seconds, getString(R.string.menu_seconds));
}

private boolean updateSpecsUI(Route route)
{
    TextView
            tv_route_start = findViewById(R.id.tv_route_start_value),
            tv_route_end = findViewById(R.id.tv_route_end_value),
            tv_route_points = findViewById(R.id.tv_route_points_value),
```

```
            tv_route_avoidpoints = findViewById(R.id.tv_route_avoidpoints_value),
            tv_route_avoidpointsarea = findViewById(R.id.tv_route_avoidpointsarea_value),
            tv_route_length = findViewById(R.id.tv_route_length_value),
            tv_route_time = findViewById(R.id.tv_route_time_value),
            tv_route_drone = findViewById(R.id.tv_route_drone_value);

        if (route != null)
        {
            et_route_name.setText(route.getName());
            tv_route_start.setText(String.format("%.3f, %.3f", route.getStart().getLongitude(),
route.getStart().getLatitude()));
            tv_route_end.setText(String.format("%.3f, %.3f", route.getEnd().getLongitude(),
route.getEnd().getLatitude()));
            tv_route_points.setText(String.valueOf(route.getNumberOfPoints()));
            tv_route_avoidpoints.setText(String.valueOf(route.getNumberOfAvoidancePoints()));
            tv_route_avoidpointsarea.setText(String.format("%.3f %s",
route.getAreaOfAvoidancePoints()/1_000_000, getString(R.string.menu_square_km)));
            et_route_weight.setText(String.format("%d", route.getWeight()));
            tv_route_length.setText(String.format("%.3f %s", route.getLength()/1_000,
getString(R.string.menu_km)));
            tv_route_time.setText(getFormatedTime(route.getTimeToComplete()));
            tv_route_drone.setText(route.getDroneName());

            return true;
        }

        return false;
    }

    private boolean updateStatusUI(Route route)
    {
        ImageView iv_status = findViewById(R.id.iv_status_value);
        TextView tv_status_msg = findViewById(R.id.tv_route_status_value);

        if (route != null)
```

```
    {
        switch (route.getStatus())
        {
            case GOOD:
                iv_status.setImageResource(R.drawable.ic_checkmark);
                tv_status_msg.setText(R.string.status_msg_fine);
                break;
            case ROUTE_TOO_LONG:
                iv_status.setImageResource(R.drawable.ic_cancel);
                tv_status_msg.setText(R.string.status_mgs_route_too_long);
                break;
            case OVERWEIGHT:
                iv_status.setImageResource(R.drawable.ic_cancel);
                tv_status_msg.setText(R.string.status_msg_overweight);
                break;
            case NO_DRONE:
                iv_status.setImageResource(R.drawable.ic_warning);
                tv_status_msg.setText(R.string.status_msg_no_drone);
                break;
            default:
                break;
        }

        return true;
    }

    return false;
}

private void updateRoute()
{
    if (et_route_name.getText().toString().trim().isEmpty()
            || et_route_weight.getText().toString().trim().isEmpty())
    {
        showEmptyFieldsAlert();
```

```
    }
    else
    {
        String name = et_route_name.getText().toString();
        int weight;

        try
        {
            weight = Integer.parseInt(et_route_weight.getText().toString());
        }
        catch (NumberFormatException e)
        {
            weight = 0;
        }

        Intent returnIntent = new Intent();

        route.setName(name);
        route.setWeight(weight);

        returnIntent.putExtra("routeUpdated", true);
        returnIntent.putExtra("routePosition", routePosition);
        returnIntent.putExtra("route", route);

        setResult(Activity.RESULT_OK, returnIntent);

        finish();
    }
}

private void showEmptyFieldsAlert()
{
    new AlertDialog.Builder(this)
            .setTitle(getString(R.string.alert_not_enough_data_label))
            .setMessage(getString(R.string.alert_msg_not_enough_data))
```

```
            .setPositiveButton(R.string.alert_answer_ok, null)
            .setNegativeButton(R.string.alert_answer_cancel, null)
            .setIcon(R.drawable.ic_warning)
            .show();
    }

    private void showDroneSelectionMenu()
    {
        SharedPreferences sharedPreferences = this.getSharedPreferences("DronesPref",
Context.MODE_PRIVATE);
        String jsonDrones = sharedPreferences.getString("SavedDrones", null);
        Gson gson = new Gson();
        Type type = new TypeToken<ArrayList<Drone>>() {}.getType();
        ArrayList<Drone> dronesList = gson.fromJson(jsonDrones, type);

        if (dronesList == null || dronesList.isEmpty())
        {
            Toast.makeText(this, "Список безпілотників порожній",
Toast.LENGTH_SHORT).show();
            return;
        }

        String[] droneNames = new String[dronesList.size()];
        for (int i = 0; i < dronesList.size(); i++)
            droneNames[i] = dronesList.get(i).getName();

        AlertDialog.Builder builder = new AlertDialog.Builder(this);
        builder.setTitle("Оберіть безпілотник");
        builder.setItems(droneNames, new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which) {
                Drone selectedDrone = dronesList.get(which);
                route.setDrone(selectedDrone);
                updateSpecsUI(route);
                updateStatusUI(route);
```

```
            //Toast.makeText(this, "Ви обрали: " + selectedDrone.getName(),
Toast.LENGTH_SHORT).show();
        }
    });
    builder.show();
  }


  private void showDeleteConfirmationDialog()
  {
    new AlertDialog.Builder(this)
        .setTitle(route.getName())
        .setMessage(getString(R.string.alert_msg_delete_route))
        .setPositiveButton(R.string.alert_answer_positive, new DialogInterface.OnClickListener()
{
          public void onClick(DialogInterface dialog, int which) {
            deleteRoute();
          }
        })
        .setNegativeButton(R.string.alert_answer_negative, null)
        .setIcon(R.drawable.ic_delete)
        .show();
  }

  private void deleteRoute()
  {
    Intent returnIntent = new Intent();
    returnIntent.putExtra("routeDeleted", true);
    returnIntent.putExtra("routePosition", routePosition);
    setResult(Activity.RESULT_OK, returnIntent);

    finish();
  }
}
```

Додаток Г – Ілюстративна частина

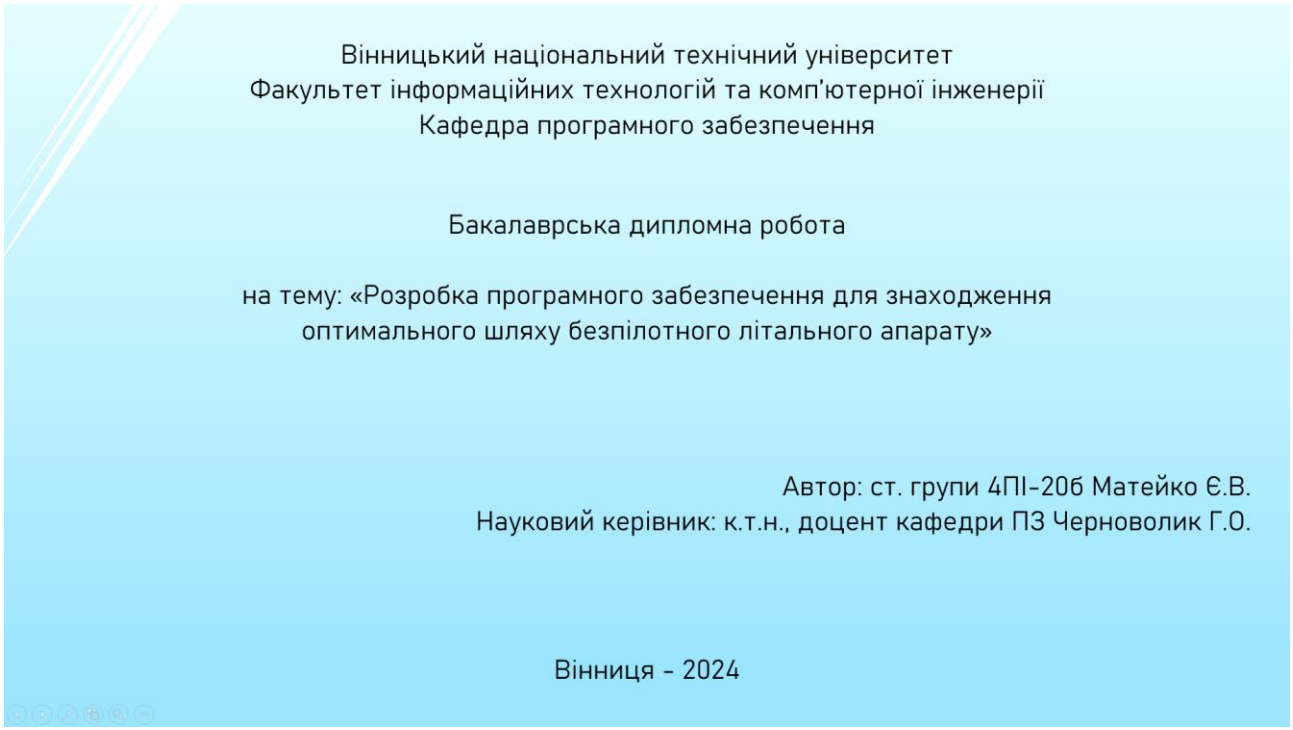


Рисунок Г.1 – Титульний слайд

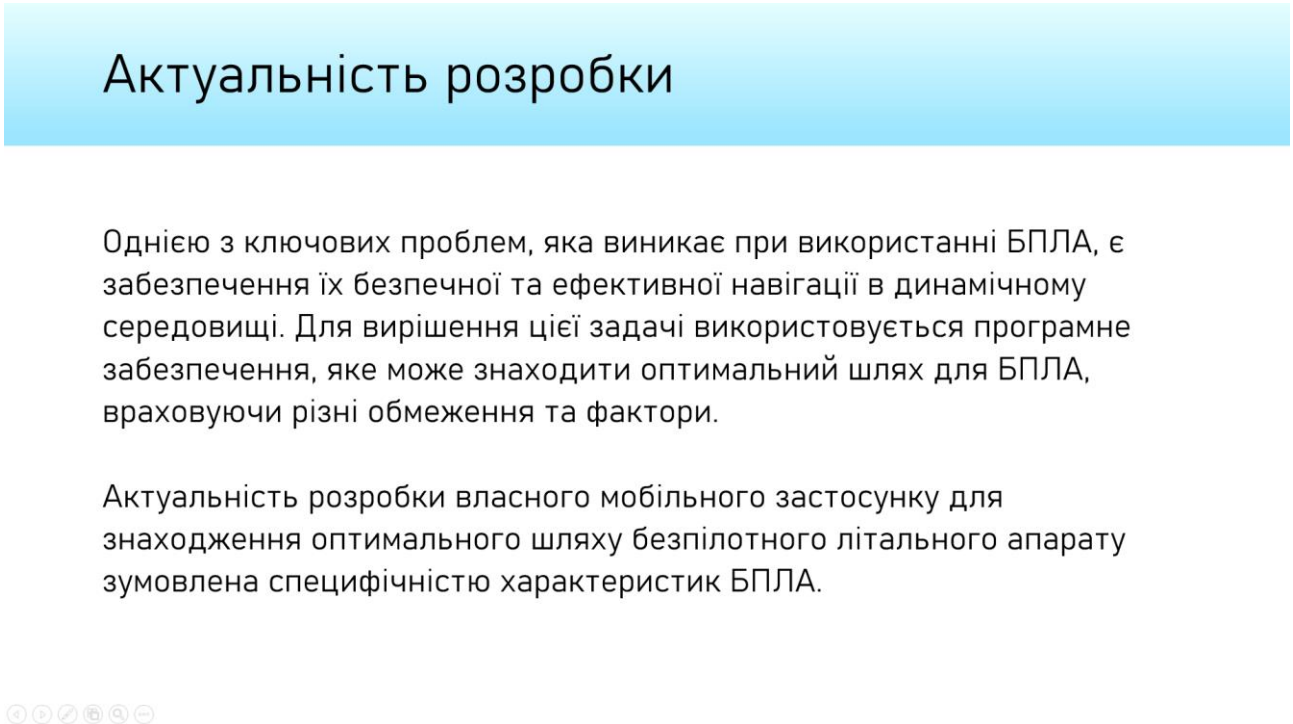


Рисунок Г.2 – Актуальність розробки

Мета, об’єкт та предмет дослідження

Метою роботи є підвищення енергоефективності безпілотних літальних апаратів за рахунок збільшення якості визначення шляху.

Об’єкт дослідження – процес розробки програмних засобів системи знаходження оптимального шляху безпілотного літального апарату.

Предмет дослідження – алгоритми і засоби реалізації системи знаходження оптимального шляху безпілотного літального апарату.

Рисунок Г.3 – Мета, об’єкт та предмет дослідження

Завдання дослідження

- розробити загальний алгоритм системи;
- розробити метод знаходження оптимального шляху безпілотного літального апарату на основі алгоритмів Дейкстри та A*;
- розробити метод врахування характеристик безпілотника;
- розробити зручний та адаптивний графічний інтерфейс мобільного застосунку;
- розробити модуль для локального збереження маршрутів на пристрої користувача;
- здійснити тестування мобільного застосунку згідно поставлених задач;
- розробити інструкцію користувача.

Рисунок Г.4 – Завдання дослідження

Новизна та практична цінність отриманих результатів

1. Подальшого розвитку отримали метод знаходження оптимального шляху на основі алгоритмів Дейкстри та A*, який, на відміну від існуючих, враховує зони запобігання, що дозволило підвищити точність визначення оптимального шляху.

2. Подальшого розвитку отримав метод врахування характеристик безпілотників, який, на відміну від існуючих, надає рекомендації, що дозволило підвищити ефективність використання безпілотників.

Практична цінність одержаних результатів полягає у можливості використання розробниками для створення мобільних систем під керуванням ОС Android для вирішення задачі комівояжера.

Рисунок Г.5 – Новизна та практична цінність отриманих результатів

Аналоги розробленого застосунку

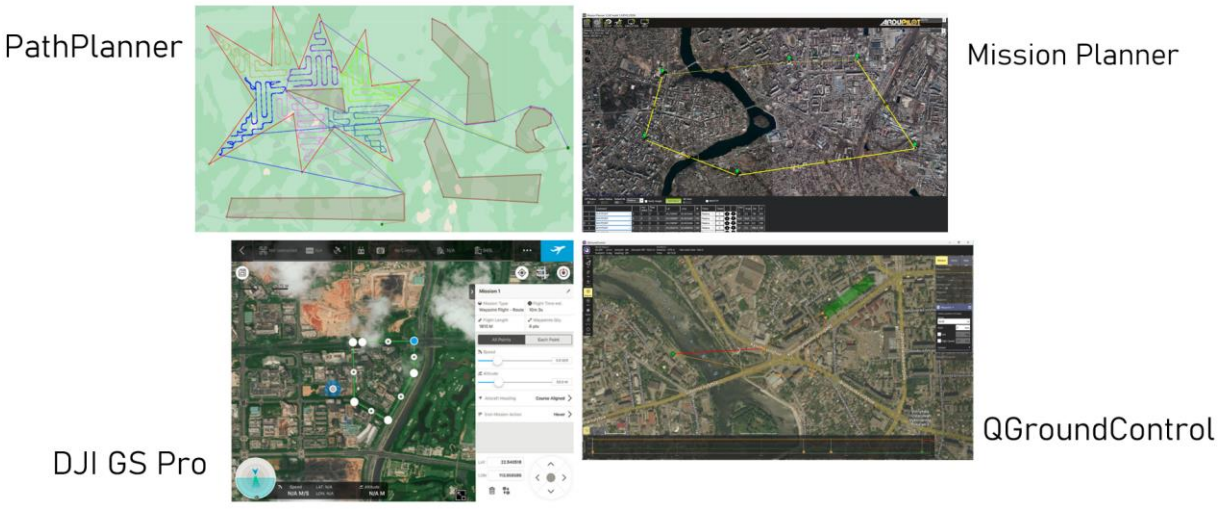
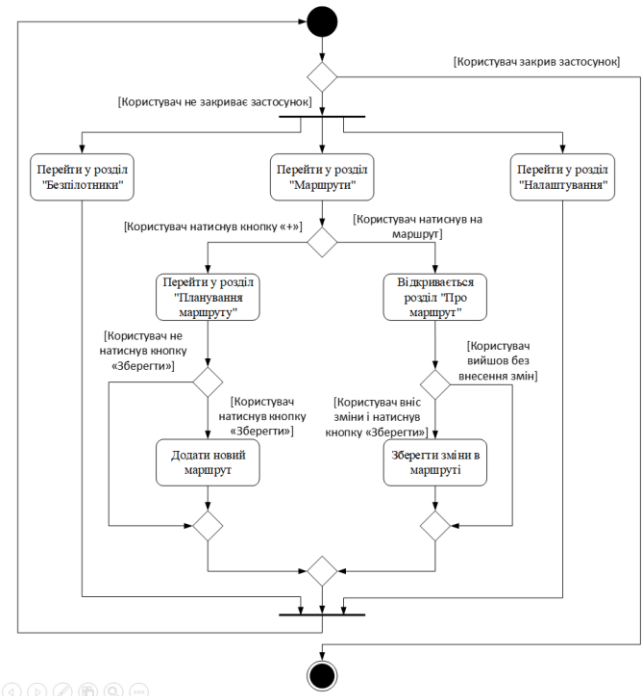


Рисунок Г.6 – Аналоги розробленого застосунку (частина 1)

Критерії	Path Planner	PX4 Autopilot	Mission Planner	Qground Control	DJI GS Pro	Drone Pathfinder
Вибір алгоритму знаходження шляху	–	–	–	–	–	+
Збереження маршрутів, профілів безпілотників	–	+	+	+	+	+
«Мобільність» застосунку	–	–	+	+	+/–	+
Врахування характеристик безпілотника	–	+	+	+	+	+
Додання зон забороненого руху	+	+	+	+	+	+
Загальна оцінка	1	3	4	4	3,5	5

Рисунок Г.7 – Аналоги розробленого застосунку (частина 2)

Рисунок Г.8 – Структура мобільного застосунку



Діаграма діяльності мобільного застосунку

Загалом користувачу мобільного застосунку доступно 7 розділів з якими він може взаємодіяти. На цих сторінках користувач може створювати/видаляти/змінювати маршрути або профілі безпілотників, переглядати рекомендації щодо маршрутів тощо.

Рисунок Г.9 – Діаграма діяльності мобільного застосунку

Блок-схема методу пошуку найкоротшого шляху

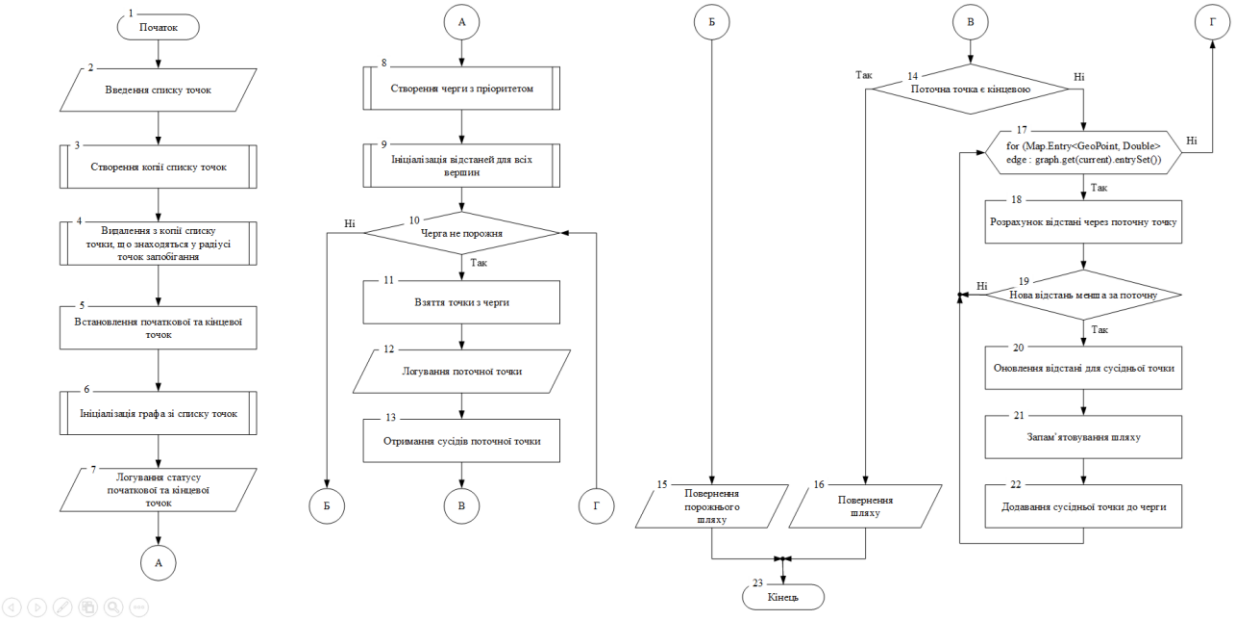


Рисунок Г.10 – Блок-схема пошуку найкоротшого шляху

Рисунок Г.11 – Блок-схема методу врахування характеристик безпілотника

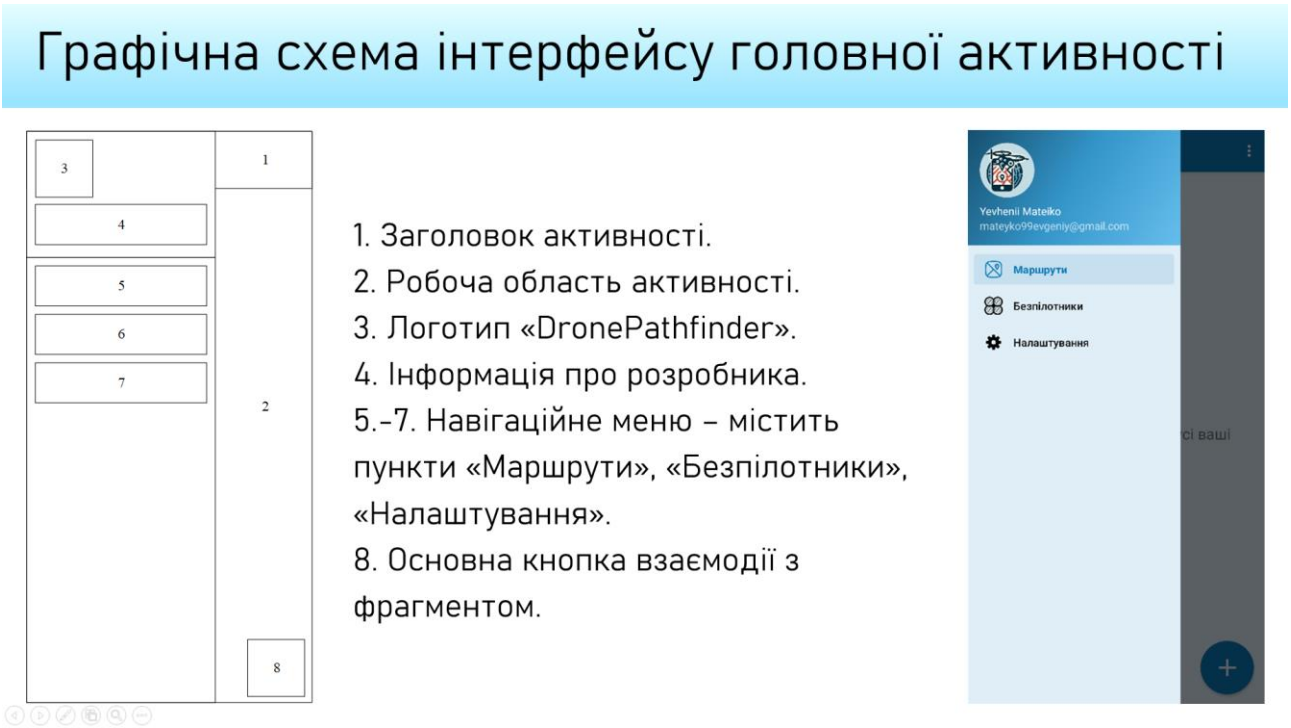


Рисунок Г.12 – Графічна схема інтерфейсу головної активності

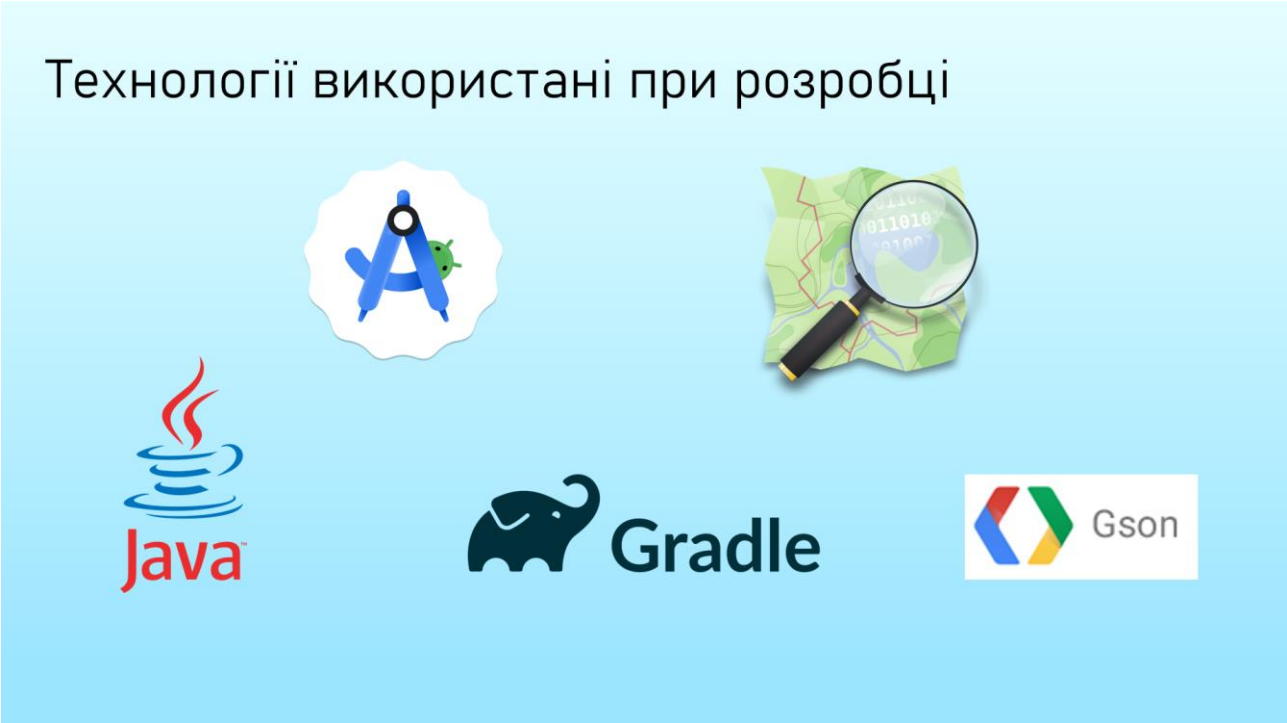


Рисунок Г.13 – Технології використані при розробці

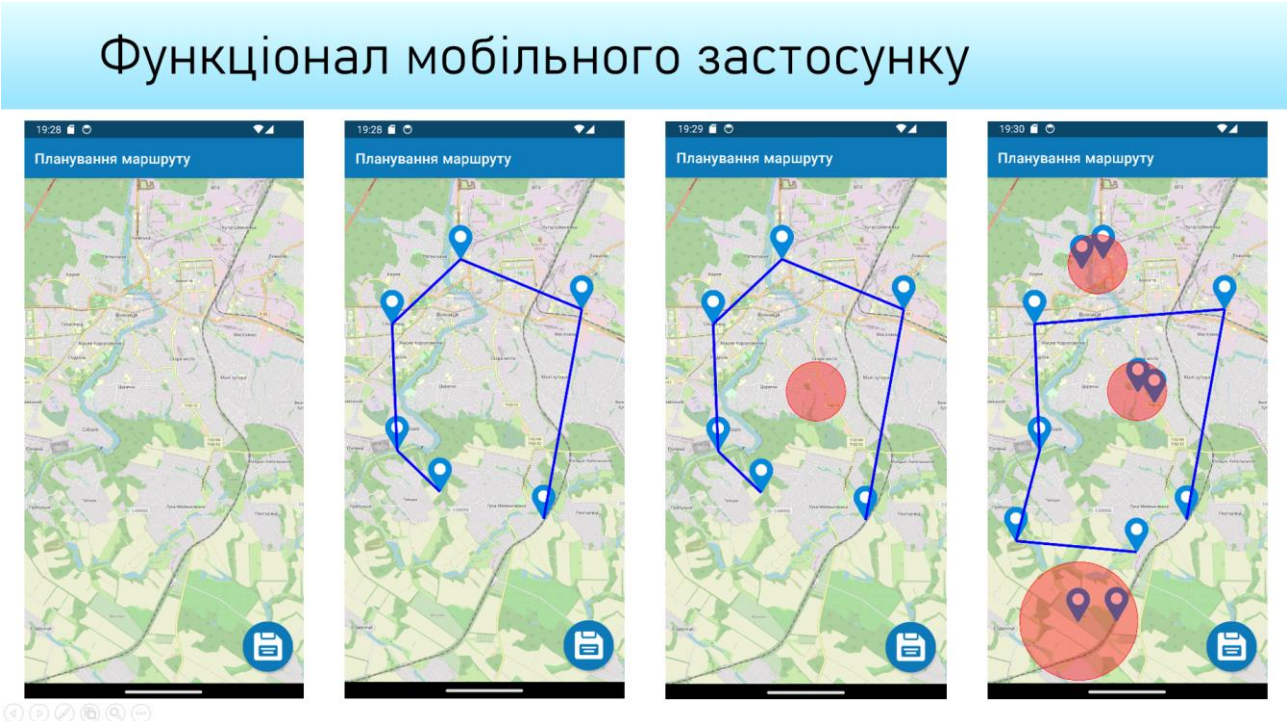


Рисунок Г.14 – Функціонал мобільного застосунку (частина 1)

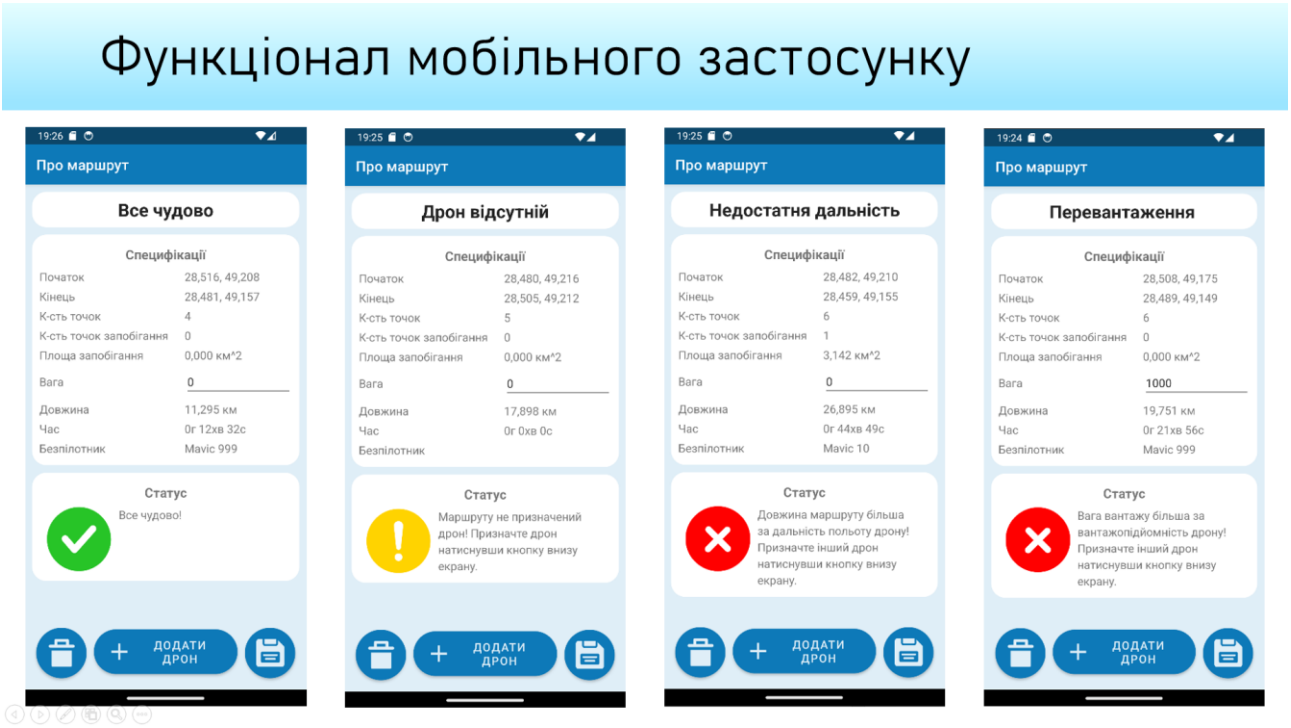
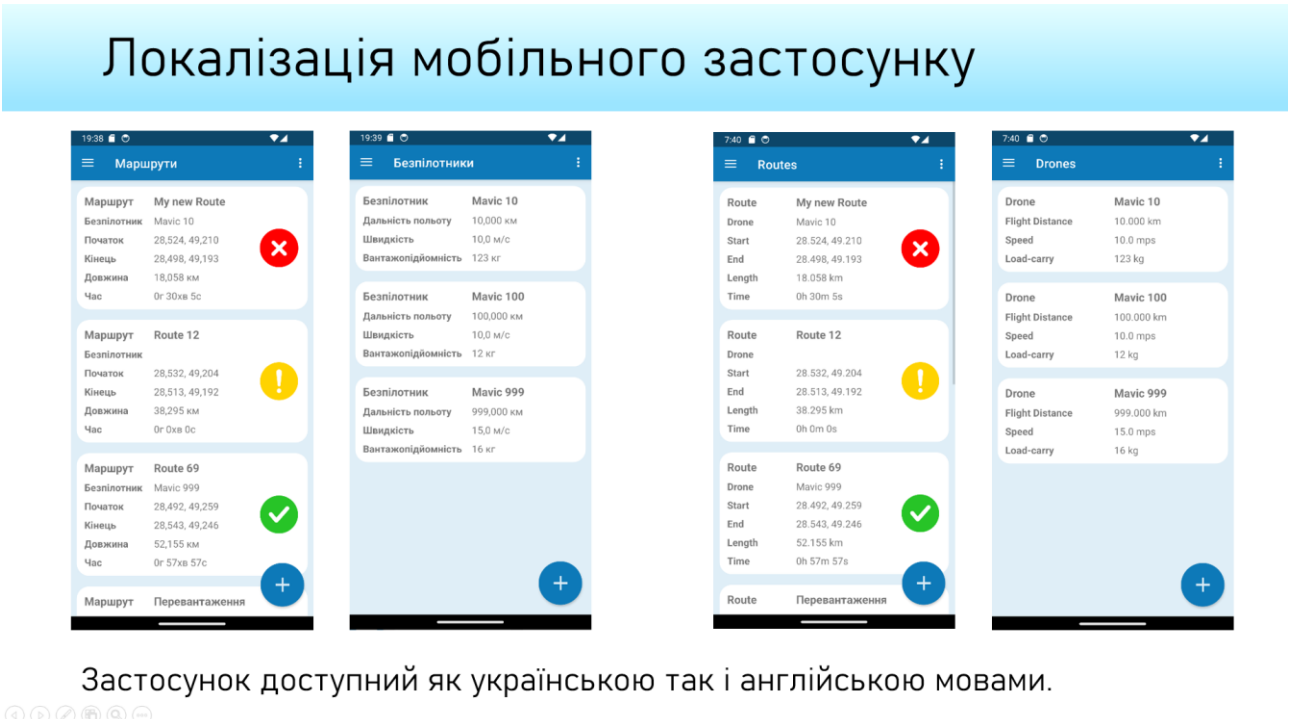


Рисунок Г.15 – Функціонал мобільного застосунку (частина 2)



Застосунок доступний як українською так і англійською мовами.

Рисунок Г.16 – Локалізація мобільного застосунку

Апробація та публікація результатів роботи

Основні результати досліджень було опубліковано на конференції LIII Науково-технічна конференція факультету інформаційних технологій та комп'ютерної інженерії (2024)

Матейко Є. В. Вплив алгоритмів знаходження оптимального шляху на ефективність безпілотного літального апарату / Є. В. Матейко, Г. О. Черноволик // Матеріали LIII Науково-технічної конференції підрозділів Вінницького національного технічного університету, Вінниця, 2024. [Електронний ресурс] – Режим доступу до ресурсу: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2024/paper/view/20696/17212

Рисунок Г.17 – Апробація та публікація результатів роботи

Висновки

Розглянуто стан систем знаходження оптимального шляху безпілотного літального апарату, вплив алгоритмів пошуку шляху на ефективність безпілотного літального апарату. Проведено аналіз аналогів програмного забезпечення «DronePathfinder», що довів доцільність і актуальність розробки.

Проведено аналіз вхідних та вихідних даних мобільного застосунку. Також описано модель роботи мобільного застосунку за допомогою UML діаграм діяльності.

Описано спосіб логування подій, проведено юніт тестування програмних модулів мобільної системи.

Рисунок Г.18 – Висновки (частина 1)

Висновки

У ході виконання бакалаврської дипломної роботи було виконано такі завдання:

- розроблено загальний алгоритм системи;
- розроблено метод знаходження оптимального шляху безпілотного літального апарату на основі алгоритмів Дейкстри та A*;
- розроблено метод врахування характеристик безпілотника;
- розроблено зручний та адаптивний графічний інтерфейс мобільного застосунку;
- розроблено модуль для локального збереження маршрутів на пристрої користувача;
- здійснено тестування мобільного застосунку згідно поставлених задач;
- розроблено інструкцію користувача.

Рисунок Г.19 – Висновки (частина 2)